SIAM J. COMPUT.
Vol. 38, No. 1, pp. 1–45

ON THE ALGORITHMIC ASPECTS OF DISCRETE AND LEXICOGRAPHIC HELLY-TYPE THEOREMS AND THE DISCRETE LP-TYPE MODEL*

NIR HALMAN†

Abstract. Helly's theorem says that, if every $d+1$ elements of a given finite set of convex objects in $\mathbb{R}^d$ have a common point, there is a point common to all of the objects in the set. In discrete Helly theorems the common point should belong to an a priori given set. In lexicographic Helly theorems the common point should not be lexicographically greater than a given point. Using discrete and lexicographic Helly theorems we get linear time solutions for various optimization problems. For this, we introduce the *DLP-type* (discrete linear programming–type) model, and provide new algorithms that solve in randomized linear time fixed-dimensional DLP-type problems. For variable-dimensional DLP-type problems, our algorithms run in time subexponential in the combinatorial dimension. Finally, we use our results in order to solve in randomized linear time problems such as the discrete p-center on the real line, the discrete weighted 1-center problem in $\mathbb{R}^d$ with either l_1 or l_∞ norm, the standard (continuous) problem of finding a line transversal for a totally separable set of planar convex objects, a discrete version of the problem of finding a line transversal for a set of axis-parallel planar rectangles, and the (planar) lexicographic rectilinear p-center problem for $p = 1, 2, 3$. These are the first known linear time algorithms for these problems. Moreover, we use our algorithms to solve in randomized subexponential time various problems in game theory, improving upon the best known algorithms for these problems.

Key words. Helly-type theorems, LP-type model, design and analysis of algorithms

AMS subject classifications. 68Q05, 68Q25, 68W20, 68R05, 52B55, 52C07

DOI. 10.1137/060656309

1. Introduction.

1.1. Helly-type theorems.

The classical theorem of Helly stands at the origin of what is known today as the combinatorial geometry of convex sets. It was discovered in 1913 and may be formulated as follows.

THEOREM 1.1 (Helly's theorem). *Let H be a family of closed convex sets in $\mathbb{R}^d$, and suppose either H is finite or at least one member of H is compact. If every $d+1$ or fewer members of H have a common point, then there is a point common to all members of H.*

A possible generalization of Helly's theorem is as follows. Let H be a family of objects, and let $\mathcal{P}$ be a predicate on subsets of H. A *Helly-type theorem* for H is of the form:

There is a constant k such that for every finite set G, $G \subseteq H$, $\mathcal{P}(G)$, if and only if, for every $F \subseteq G$ with $|F| \leq k$, $\mathcal{P}(F)$.

The minimal such constant k is called the *Helly number* of H with respect to the predicate $\mathcal{P}$. If no such constant exists, we say that the Helly number of H with

*Received by the editors April 5, 2006; accepted for publication (in revised form) August 21, 2007; published electronically March 28, 2008. An extended abstract containing parts of this work appeared in *Proceedings of the Forty-Fifth Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004 [18]. This work is part of the author's Ph.D. thesis, prepared at the School of Mathematical Sciences at Tel Aviv University, Tel Aviv, Israel, under the supervision of Professor Arie Tamir [17].

http://www.siam.org/journals/sicomp/38-1/65630.html

†Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139 (halman@mit.edu).

respect to $\mathcal{P}$ is *unbounded* or *infinite* (∞). In Helly's theorem, the Helly number is $d+1$, and $\mathcal{P}$ is the predicate of having a nonempty intersection.

Over the years, a vast body of application analogues and far-reaching generalizations of Helly's theorem has been assembled in the literature (see, for instance, the excellent surveys of [10, 12, 16]).

It is possible to give lexicographic versions to some of the Helly theorems. For instance, the following theorem is a lexicographic version of Helly's theorem. (Recall that, for every $x = (x_1, \ldots, x_d), y = (y_1, \ldots, y_d) \in \mathbb{R}^d$, x is said to be *lexicographically* smaller than y (*lsmaller*, in short, or $x <_L y$) if either $x_1 < y_1$ or there exists $d \geq k > 1$ such that $x_i = y_i$ for $i = 1, 2, \ldots, k-1$ and $x_k < y_k$.)

THEOREM 1.2 (lexicographic Helly's theorem [26, 20]). *Let H be a finite family of convex sets in $\mathbb{R}^d$. For every $x \in \mathbb{R}^d$, if every $d+1$ or fewer members of H have a common point which is not lexicographically greater than x, then there is a point common to all members of H which is also not lexicographically greater than x.*

This theorem is folklore. It derives directly from Helly's theorem and Lemma 8.1.2 in [26] and is proved independently in [20]. $d+1$ is called the *lexicographic Helly number* of H with respect to intersection (*lex-Helly number*, in short). The following theorem is a discrete version of Helly's theorem, due to Doignon.

THEOREM 1.3 (see [11]). *Let H be a finite family of at least 2^d convex sets in $\mathbb{R}^d$. If every 2^d or fewer members of H have a common point with integer coordinates, then there is a point with integer coordinates common to all members of H.*

2^d is called the *discrete Helly number* of H with respect to intersection. Halman [20] provides discrete versions to numerous known Helly theorems. For instance, a special case of Helly's theorems is when the given convex sets are axis-parallel boxes in $\mathbb{R}^d$. In this case the Helly number is just 2 [9]. A discrete version of this Helly theorem is as follows.

THEOREM 1.4 (Theorem 2.10 in [20]). *Let S be a finite set of points in $\mathbb{R}^d$, and let D be a finite family of closed boxes in $\mathbb{R}^d$ with edges parallel to the axes. If every $2d$ or fewer members of D have a common point in S, then there is a point in S common to all members of D.*

A combined discrete-lexicographic version of this Helly theorem is as follows.

THEOREM 1.5 (Theorem 2.10 in [20]). *Let S be a finite set of points in $\mathbb{R}^d$, and let D be a finite family of closed boxes in $\mathbb{R}^d$ with edges parallel to the axes. For every $x \in \mathbb{R}^d$, if every $2d$ or fewer members of D have a common point $x' \in S$, with $x' \leq_L x$, then there is a point $x^* \in S$ common to all members of D with $x^* \leq_L x$.*

1.2. Algorithmic aspects of finite Helly numbers. In this section we discuss two optimization models and show their relations to Helly numbers.

The LP-type model. Matoušek, Sharir, and Welzl [28] defined a model which generalizes linear programming (LP) and called it the LP-type model (see definitions in section 2). Fixed-dimensional LP-type problems can be solved efficiently by *LP-type algorithms* such as the ones of Matoušek, Sharir, and Welzl [28] or Kalai [22]. The algorithm of Clarkson [8], which was originally formulated to solve LP, fits the LP-type model as well [31, 7, 15]. This provides a tool for obtaining linear time algorithms to various (continuous) optimization problems, mainly in computational geometry and location theory, as shown in [2, 28].

The DLP-type model. In *continuous* optimization models related to LP-type problems, the feasible set is defined by a finite set of constraints. In the *discrete* versions, in addition to the above, there is also a prespecified set of *relaxations*. A

feasible solution is restricted to be in the set of relaxations as well as to satisfy the constraints. *Integer programming* (IP) is an example of a discrete optimization problem where the set of relaxations is the integer lattice. Another example for a discrete optimization problem is the *discrete point set width problem*, where we are given a finite set of points in the plane (i.e., constraints) and a finite set of permissible directions (i.e., relaxations). The goal is to find the minimal width of a band with a permissible direction which contains all of the points (see more detail about this problem in section 4). Many times discrete optimization problems are proved to be computationally harder to solve than their corresponding continuous versions (e.g., LP vs. IP and continuous planar Euclidean 1-center vs. the discrete version as proved in section 9). In this paper we propose the following framework for solving discrete optimization problems: We generalize integer programming by introducing the *discrete* LP-type (*DLP-type*) model. We provide randomized linear time algorithms to solve fixed-dimensional DLP-type problems satisfying a condition we call the violation condition (VC).

Helly numbers and the two optimization models. In [20] Halman defines the notion of discrete and lexicographic Helly theorems, provides lexicographic and discrete versions to numerous known Helly theorems, and studies the relations between the different types of Helly theorems. In this paper we show that discrete and lexicographic Helly theorems have interesting algorithmic aspects as well. In 1994, Amenta [2] showed that every *parameterized Helly system* satisfying a condition called the unique minimum condition (UMC) results in a fixed-dimensional LP-type problem (see definitions of the terms parameterized Helly theorems and UMC in section 2.3). In this paper we define lexicographic parameterized Helly systems and show that *every* such system results in a fixed-dimensional LP-type problem. Unlike in [2], no additional conditions are needed. Similarly to [2], this provides a framework for obtaining linear time algorithms (i.e., the LP-type algorithms mentioned above) for the optimization problems related to these Helly numbers. In this way the *existence* of finite lexicographic Helly numbers implies the *solvability* of their corresponding optimization problems by the linear time LP-type algorithms. Similarly to the above, we show that *every* lexicographic-discrete parameterized Helly system can be formulated as a fixed-dimensional DLP-type problem.

1.3. Applications. We improve upon the best known algorithms for the seven problems listed below. The problems differ in the way we solve them. The first three are solved by using the LP-type model and its connection to lexicographic Helly theorems. The next four problems are solved via the DLP-type model. While the first three of them are solved via lexicographic-discrete Helly theorems, the fourth is not. We solve in this paper the first five problems in linear time. Due to its length, we refer the reader to [17] for details of the solution of the sixth problem. The first six problems lie in the fields of research of either computational geometry or location theory. The seventh problem is solved in [19] and is different, since it lies in game theory and is solved in strongly subexponential time. We summarize the solutions we give to each of these problems in Table 1.1.

1. Planar lexicographic weighted rectilinear p-center optimization problem ($p = 1, 2, 3$). Given a finite set $H = \{h_1, \ldots, h_n\}$ of reference points in the plane and a set $W = \{w_1, \ldots, w_n\}$ of weights in $\mathbb{R}^+$, find the lexicographically smallest vector $(\lambda_1, x_1, y_1, x_2, y_2, \ldots, x_p, y_p) \in \mathbb{R}^+ \times \mathbb{R}^{2p}$ such that for every scaled square $\frac{\lambda_1}{w_i} h_i$, $h_i \in H$, centered at h_i with radius $\frac{\lambda_1}{w_i}$, there exists $1 \leq j \leq p$

TABLE 1.1
A comparison between the various problems solved.

Problem number	1	2	3	4	5	6	7
Running time	Linear	Linear	Linear	Linear	Linear	Linear	Subexponential
Model used	LP-type	LP-type	LP-type	DLP-type	DLP-type	DLP-type	LP/DLP-type
Type of Helly theorem used	lex	lex	lex	lex-discrete	lex-discrete	None	None

such that $\frac{\lambda_1}{w_i}h$ contains (i.e., is pierced by) point (x_j, y_j) (we call λ_1 the *radius* and $(x_1, y_1, \ldots, x_p, y_p)$ the *centers vector*).

For $p > 3$ [32] showed a lower bound of $\Omega(n \log n)$. [32, 21] solve the corresponding nonlexicographic problem in linear time.

2. Line transversal of axis-parallel rectangles optimization problem. Given a set B of axis-parallel rectangles, find the minimal scaling factor λ^* such that the set of scaled rectangles λ^* admits a line transversal.

For the next problem we use the following definitions. A set H of convex objects is called *totally separable* if there exists a direction such that each line in this direction intersects at most one convex object from H. We call the objects in H *simple* if they have a constant size storage description, the intersections and common tangents between any two objects can be found in constant time, and the minimal scaling factor for any 3 objects to admit a line transversal can be found in constant time.

3. Line transversal of totally separable set of convex planar objects decision problem. Given is a totally separable finite family H of simple convex objects (the direction of separation is not given). Decide whether H admits a line that intersects all of the objects in H.

Given the order in which any line transversal should meet the objects in H, the problem is solvable in linear time [13].

4. Lexicographic discrete line transversal of axis-parallel rectangles problem. Given a finite family D of axis-parallel rectangles and a finite set S of line *directions*, find a line transversal for D, $y = ax + b$, with the lexicographically smallest vector (a, b) satisfying $a \in S$.

We show in section 10 that a similar problem, where, instead of a finite family S of line directions, we are given a finite family S' of lines, and the goal is find the lexicographically smallest vector $(a, b) \in S'$ such that $y = ax + b$ is a line transversal for D, has a lower bound of $\Omega(n \log n)$ under the algebraic computation tree model.

5. Discrete weighted 1-center problem in $\mathbb{R}^d$ with an l_∞ norm. Given are sets $D = \{d_1, \ldots, d_n\}$ and $S = \{s_1, \ldots, s_m\}$ of *points* in $\mathbb{R}^d$ and a set $W = \{w_1, \ldots, w_n\}$ of *weights* in $\mathbb{R}^+$. Find a point $s \in S$ (*center*) which minimizes the real function $r(D, S) = \min_{s \in S} \max_i w_i \|s - d_i\|_\infty$ (the *optimal radius*). We solve the corresponding rectilinear problem (i.e., with an l_1 norm) in linear time as well.

It is folklore that the latter problem restricted to the case $S = D$ is solvable in $O(n \log n)$ time.

6. Discrete p-center problem on the real line. Given a finite set D of real numbers (*points*) and a finite set S of real numbers (*center locations*), find a subset $C \subseteq S$ of p points (*centers*) which minimizes the real function $r_p(D, S) = \min_{C \subseteq S, |C| \leq p} \max_{h \in D} dist(h, C)$ (the *optimal radius*). For every finite set of real numbers C and real h, $dist(h, C) = \min_{c \in C} |h - c|$. Due to space limitations we refer the

interested reader to Chapter 9 in [17] for a detailed description of our linear time solution for this problem.

Assuming the order of the points on the line is given, the discrete p-center problem on the real line is solvable in linear time by the fairly involved technique of Frederickson [14].

7. Simple stochastic games and infinite games. The first strongly subexponential algorithm for binary simple stochastic games (SSGs) was given by Ludwig [25] in 1995 by using ideas from the algorithms of [22] and [31] for LP-type problems. Halman [19] gives the first strongly subexponential solution for (nonbinary) SSGs by formulating the SSG as an LP-type problem and then calculating optimal strategies for both players by the LP-type algorithm of [31]. Since several infinite games are linearly reducible to nonbinary SSGs, this gives strongly subexponential algorithms to parity games (PGs) and the first strongly subexponential algorithms to mean payoff games (MPGs) and discounted payoff games. Halman notes in [19] that nonbinary SSGs can most naturally be formulated as discrete LP-type problems, because of the essentially primal-dual nature of the two-player game. We note that, independently, Björklund, Sandberg, and Vorobyovn [6] developed a (nonstrongly) subexponential algorithm for MPGs, a strongly subexponential algorithm for PGs [5], and a (nonstrongly) subexponential algorithm to nonbinary SSGs [4]. All of their algorithms are "tailored" to the specific game solved and "adapt" ideas from the algorithms of [25, 22, 31] (see formal definitions of all of these games in [19]).

Our contribution. In this paper we define a new model for solving discrete optimization problems, the DLP-type model. We develop for it several linear time (randomized) algorithms. We study the relations between discrete Helly theorems and the DLP-type model. We study also the relations between (nondiscrete) lexicographic Helly theorems and the LP-type model. We show that *every* lexicographic parameterized Helly system results in a fixed-dimensional LP-type problem. In this case the UMC stated in the main theorem of [2] is not needed. By incorporating these "ingredients" together we provide the first linear time algorithms for the first six problems defined above. All of these problems are related to computational geometry and location theory. By solving the seventh problem we show (for the first time, to the best of our knowledge) that the LP-type and DLP-type models have applications in other fields of research, such as game theory. Moreover, we show that these two models are also useful for solving non-fixed-dimensional problems in subexponential time.

Organization of the paper. In this paper we extensively use terms which are defined in [2], two tools for establishing linear time algorithms: the LP-type framework and Helly-type results, which are reviewed in [2, 32], and the two LP-type algorithms in [8, 32]. In order to make the paper self-contained we review these terms, models, and algorithms in section 2. In section 3 we define a dual version of the LP-type model, which we use in order to define the DLP-type model in section 4. In section 5 we develop algorithms which solve (fixed-dimensional) DLP-type problems in (randomized) linear time. The rest of the paper is dedicated to show the interrelations between discrete and lexicographic Helly theorems and DLP-type and LP-type models. In section 6 we study the relations between lexicographic Helly theorems and the LP-type model. By showing that every lexicographic parameterized Helly system results in a fixed-dimensional LP-type problem, we give a partial solution for the main open problem raised by Amenta [2], who asked to characterize the parameterized Helly systems which result in fixed-dimensional LP-type problems. In section 7

we demonstrate the applicability of these relations by solving the first problem discussed in this section—the planar lexicographic weighted rectilinear p-center problem ($p = 1, 2, 3$) in linear time. In section 8 we study the relations between discrete and lex-discrete Helly theorems and the DLP-type model. In sections 9 and 10 we solve in linear time the next four problems discussed in this section.

2. Literature review. In this section we review some of the definitions and results given in Amenta [1, 2], Sharir and Welzl [31], Matoušek, Sharir, and Welzl [28], and Clarkson [8]. The term used in the first two papers is GLP (general linear programming) rather than LP-type.

2.1. LP-type problems.

DEFINITION 2.1. *An* abstract problem *is a tuple (H, ω), where H is a finite set of elements (which we call* constraints*) and ω is an* objective function *from 2^H to some totally ordered set Λ which contains a special maximal (minimal) element ∞ ($-\infty$), respectively. The goal is to compute $\omega(H)$.*

DEFINITION 2.2. *Let (H, ω) be an abstract problem. For any subset $G \subseteq H$ we say that $F \subseteq G$* defines *the solution on G (F is a* solution-defining set *of G) if $\omega(F) = \omega(G)$.*

Clearly, for every $G \subset H$, G is a defining set for itself.

DEFINITION 2.3. *An* LP-type problem *is an abstract problem (H, ω) that obeys the following conditions (when we write $<$, $\leq$, $=$ etc., we mean under the ordered set Λ):*

1. Monotonicity: *For all $F \subseteq G \subseteq H : \omega(F) \leq \omega(G)$ (so the special element $-\infty$ is such that $\omega(\emptyset) = -\infty$).*
2. Locality: *For all $F \subseteq G \subseteq H$, with $\omega(G) = \omega(F) \neq -\infty$, and for each $h \in H$, if $\omega(G \cup \{h\}) > \omega(G)$ then $\omega(F \cup \{h\}) > \omega(F)$.*

Note that *lexicographic linear programming*, where the input is a finite set of closed half-planes in $\mathbb{R}^d$ and the output is the lsmallest point which lies in all half-planes, is an LP-type problem: H is the finite set consisting of these closed half-spaces, and the function $\omega(G)$ returns the coefficients of the lexicographic minimum point in $\bigcap G$. Adding half-planes to H cannot decrease the value of ω, so the monotonicity condition is satisfied. As for the locality condition, note that if $\omega(G) = \omega(F) \neq -\infty$, then $\omega(G)$ is realized in a single point $x^* \in \mathbb{R}^d$. The fact that $\omega(G \cup \{h\}) > \omega(G)$ implies that $x^* \notin h$. Therefore, $\omega(F \cup \{h\}) > \omega(F)$ as needed. An immediate consequence of the monotonicity and locality conditions is the following.

COROLLARY 2.4. *Let (H, ω) be an LP-type problem. For all $F \subseteq G \subseteq H$, with $\omega(G) = \omega(F) \neq -\infty$, and for each $h \in H$, $\omega(G \cup \{h\}) > \omega(G)$ if and only if $\omega(F \cup \{h\}) > \omega(F)$.*

We give now several definitions for every abstract problem (H, ω) which meets the monotonicity condition. Let $G \subseteq H$ be arbitrary, and let $n = |H|$. If $\omega(G) = \infty$, we say G is *infeasible*; otherwise we call G *feasible*. If $\omega(G) = -\infty$, we say G is *unbounded*; otherwise we call G *bounded*. We say a constraint $h \in H$ *violates* G when $\omega(G \cup \{h\}) > \omega(G)$. (Using this definition we note that the locality condition says that, for every bounded subset $G \subseteq H$, defining set F for G, and $h \notin G$, if h violates G, then h must violate the defining set F. Corollary 2.4 says that, for any such G, F, h, h violates G if and only if it violates its defining set F.) A *basis* B is a set $B \subseteq H$, with $\omega(B') < \omega(B)$ for all proper subsets B' of B. A *basis for* G is a basis $B \subseteq G$, with $\omega(B) = \omega(G)$. (In other words, a basis for G is a minimal (by inclusion) defining set of G.) We note that due to the monotonicity condition a basis for G, for any $G \subseteq H$, always exists. The basis for any unbounded set is the empty set.

Observation 2.5. Let (H, ω) be an LP-type problem, let $G \subset H$, and let $B \subseteq G$ be such that $\omega(B) = \omega(G)$. If $\omega(G) < \omega(H)$, then there exists a constraint $h \in H \setminus G$ which violates B.

To see this, it is sufficient to show that if no $h \in H \setminus G$ violates B, then $\omega(G) = \omega(H)$. We add to B and G an arbitrary constraint $h \in H \setminus G$. Since h does not violate B, the locality condition implies that h does not violate G and that $\omega(B) = \omega(G \cup \{h\})$. Repeating this argument $|H \setminus G|$ times, we get that $\omega(G) = \omega(H)$ as needed.

So B is a basis for G if and only if $B \subseteq G$ is a basis and no element in G violates it. We say that $h \in G$ is *extreme* in G if $\omega(G \setminus \{h\}) < \omega(G)$. Thus $h \in G$ is extreme in G if and only if h violates $G \setminus \{h\}$. From the minimality of a basis, every h in a basis B is extreme in B. From the monotonicity condition we get the following.

Observation 2.6. Let (H, ω) be an LP-type problem. Every $h \in G$ which is extreme in $G \subseteq H$ is contained in every basis B for G.

In other words, a basis B for G contains all of the constraints which are extreme in G. We note that not all of the elements in B are extreme in G as seen in Figure 2.1. Let G be the set of 5 lines. The two thick lines form a basis B for G, and each one of them is extreme in B. We note that the line with negative slope is extreme in G.

The terms "violates" and "extreme" are somewhat complementary: For $h \in G$ we may ask whether h is extreme in G (or, equivalently, whether it violates $G \setminus \{h\}$). Similarly, for $h \notin G$ we may test whether h violates G (or, equivalently, whether it is extreme in $G \cup \{h\}$). Using the monotonicity condition and the observation above we get the following.

Observation 2.7. Let (H, ω) be an LP-type problem. Let B be a basis for $G \subseteq H$. If $h \notin G$ violates B, then h is extreme in $G \cup \{h\}$ and is a member of every basis for $G \cup \{h\}$.

The *combinatorial dimension* d of (H, ω) is the maximum size of every basis for any feasible subset G. An abstract problem which meets the monotonicity condition and is of combinatorial dimension d, where d is independent of $|H|$, is called *fixed-dimensional*. A d-dimensional LP-type problem where the cardinality of every basis is exactly d is called a d-dimensional *basis-regular* LP-type problem. Note that if such a problem is feasible and bounded, then $\omega(H) = \max_{G \subset H,\, |G|=d} \omega(G)$.

For instance, in lexicographic linear programming, if $\bigcap G \neq \emptyset$, the lexicographically smallest point in $\bigcap G$ is determined by a basis of cardinality at most d (if G is unbounded, its basis is $\emptyset$). Notice that, although more than d half-spaces may have the minimum point on their boundary, a subfamily of at most d of them is sufficient to determine the minimum. In Figure 2.1 below, the thick two lines are a basis. Notice also that a subfamily G may have more than one basis.

2.2. LP-type algorithms. An LP-type algorithm takes a d-dimensional LP-type problem (H, ω) and returns a basis B for H. Several efficient randomized LP-type algorithms are known such as the ones of Clarkson [8], Matoušek, Sharir, and Welzl [28], or Kalai [22]. In the following two sections we review the first two algorithms. We develop in section 5 a DLP-type algorithm by combining these two algorithms together.

It is not clear, of course, what computational operations are possible on an abstract object such as (H, ω). We assume two computational primitives and analyze the various algorithms by counting the number of calls to these primitives. The running time for a specific LP-type problem then depends on how efficiently the primitives can be implemented. Let us now define the two primitive operations. A *basis computation* Basis(G) takes a family G of at most $d + 1$ constraints and finds a basis for

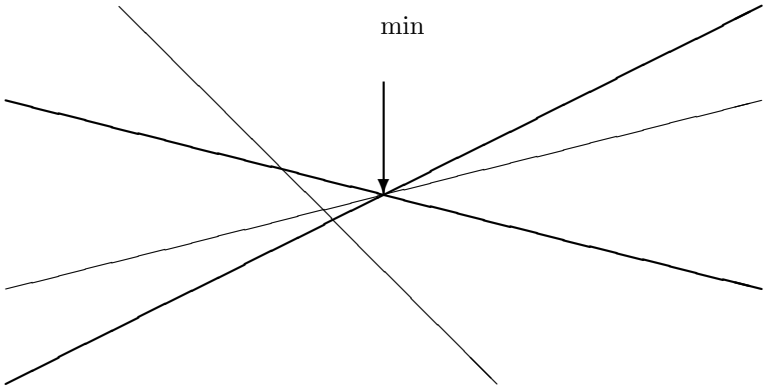


FIG. 2.1. *A basis for LP.*

G. A *violation test* Violation(B, h) takes a basis B and a constraint h and returns true if and only if h violates B (i.e., B is not a basis of $B \cup \{h\}$). Let t_b be the time required for a basis computation and t_v be the time required for a violation test.

2.2.1. Clarkson's algorithm. As originally presented, Clarkson's algorithm is aimed for linear programming. As Sharir and Welzl [31] note, the algorithm solves LP-type problems in the same time bound. We review the algorithm in the context of linear programming. Given a lexicographic linear programming problem in d variables with a set of constraints H ($|H| = n$) and objective function ω, we view it as the d-dimensional LP-type problem (ω, H).

Let x_s^* be an algorithm which gets input of size up to $9d^2$ (d is the dimension of the problem) and outputs a basis for H. The algorithm of Clarkson [8] is as follows:

Function $x_m^*(H)$ (Returns a basis for H)
1. Let $V^* := \emptyset$, let $V := H$
2. If $|H| \leq 9d^2$, then return $x_s^*(H)$
3. Else repeat the following until $V = \emptyset$:
   (a) Choose $R \subset H \setminus V^*$ uniformly at random, $|R| = d\sqrt{|H|}$
   (b) Let $B := x_i^*(R \cup V^*)$, and let $V := \{h \in H \mid \text{Violation}(B, h) = \text{TRUE}\}$
   (c) If $|V| \leq 2\sqrt{|H|}$, then let $V^* := V^* \cup V$
4. Return B

Function $x_i^*(H)$
1. Let $V := H$. For every $h \in H$ let $\nu_h := 1$
2. If $|H| \leq 9d^2$, then return $x_s^*(H)$
3. Else repeat the following until $V = \emptyset$:
   (a) Choose $R \subset H$ at random according to weights ν_h, $|R| = 9d^2$
   (b) Let $B := x_s^*(R)$.
   (c) Let $V := \{h \in H \mid \text{Violation}(B, h) = \text{TRUE}\}$
   (d) If $\nu(V) \leq 2\nu(H)/(9d - 1)$, then for every $h \in V$ let $\nu_h =: 2\nu_h$
4. Return B

As Amenta notes in [1], Clarkson's randomized algorithm for solving an LP-type problem (H, ω) improves the running time by separating the dependence on d and on n. He uses a three-level algorithm, with a "base-case" algorithm at the lowest level (x_s^*) solving subproblems of size up to $9d^2$.

The higher two levels x_m^* and x_i^* reduce the problem to smaller problems using the following idea. Take a sample $R \subseteq H$, find a basis B for R by calling the next lower level algorithm, and then find the subset $V \subseteq H$ of all of the constraints which violate B. If V is empty, Observation 2.5 tells us that B is a basis for H as well. Otherwise, by the monotonicity condition $\omega(H) > \omega(B)$. Let B' be a basis for H, and let $H' = B \cup B'$. Clearly $\omega(H) = \omega(H')$, so B' is a basis for H' as well. Applying Observation 2.5 for H' and B we get that there exists a constraint in B' which violates B. We've just proved the following lemma.

LEMMA 2.8 (Lemma 3.1 in [8]). *If the set V is nonempty, then it contains at least one constraint from every basis B for H.*

The purpose of the top level x_m^* is to get the number of constraints down so we can apply the second level (x_i^*), which is more efficient in d but less efficient in n. In the top level we take a random sample R, with $|R| = d\sqrt{n}$ so that $E[|V|] = O(\sqrt{n})$; that is, we take a big random sample which gives an expected small set of violators. This is a consequence from the following lemma.

LEMMA 2.9 (Lemma 3.2 in [8]). *Let $V^* \subset H$, and let $R \subset H \setminus V^*$ be a random subset of size r, with $|H \setminus V^*| = n$. Let $V \subset H$ be the set of constraints which violate $R \cup V^*$. Then the expected size of V is no more than $d(n - r + 1)/(r - d)$.*

We iterate, keeping the violators in a set V^* and finding a basis B' for $R \cup V^*$. At every iteration in the "repeat-until" loop of x_m^*, we add the violators to V^*, so that after d iterations V^* contains a basis B for H and $E[|V^*|] = d\sqrt{n}$. Solving the subproblem on V^* then gives the answer.

All recursive calls from the first level x_m^* call the second level algorithm x_i^*, which uses small random samples of size $9d^2$. Initially the sample R is chosen using the uniform distribution, but then we double the weights of elements in V and iterate. Since at least one basis element always ends up in V, eventually they all become so heavy that we get $B \subseteq R$. The analysis shows that the expected number of samples before $B \subseteq R$ is $O(d \log n)$. Since we need $O(n)$ work at each iteration to compare each constraint with the basis B' of R, without the first phase this algorithm alone would be $O(n \log n)$. All of the recursive calls from this reweighting algorithm are made to some "base-case" algorithm x_s^*.

Recall that t_v is the time required for a violation test, and let $t_s(n)$ be the time required for function x_s^* to run on n constraints. In his paper, Clarkson chooses x_s^* to be the simplex algorithm for linear programming on sets of $9d^2$ elements and estimates its running time by $t_s(9d^2) = d^{\frac{d}{2}+O(1)}$, using Stirling's approximation. Given a set H of n elements and a basis B (which in linear programming is equivalent to a point in $\mathbb{R}^d$), the time needed for a single call to function Violation(B, h) is d. Thus the time needed to execute the line

$$V \leftarrow \{h \in H \mid \text{Violation}(B, h) = \text{TRUE}\}$$

in the algorithm is dn, or nt_v.

In his time complexity analysis, Clarkson also uses a lemma to show that progress will be made during the execution of the algorithm. We say that an execution of the loop in x_m^* (x_i^*) is *successful* if the test $|V| \leq 2\sqrt{|H|}$ $(\nu(V) \leq 2\nu(H)/(9d-1))$ returns "true."

LEMMA 2.10 (Lemma 3.3 in [8]). *The probability that any given execution of a loop body is successful is at least* $1/2$*, and so on average two executions are required to obtain a successful one.*

Let $T_i(n)$ ($T_m(n)$) be the expected time required by x_i^* (x_m^*) for a problem with n constraints.

THEOREM 2.11 (Theorem 3.4 in [8]). *Given an LP-type problem, the iterative algorithm* x_i^* *requires*

$$T_i(n) = O(d \log n(nt_v + t_s(9d^2)))$$

expected time, where the constant factors do not depend on d*.*

THEOREM 2.12 (Theorem 3.5 in [8]). *Given an LP-type problem, algorithm* x_m^* *requires*

$$T_m(n) = O(d(T_i(d\sqrt{n}) + nt_v)) = O(d^2 \log n(\sqrt{n}t_v + t_s(9d^2)) + dnt_v)$$

expected time, where the constant factors do not depend on d*.*

We can see Clarkson's algorithm as a tool for reducing an LP-type problem with many constraints to a collection of small problems with a few constraints.

2.2.2. Sharir and Welzl's algorithm. As Amenta notes in [1], the algorithm of Sharir and Welzl [31] for solving an LP-type problem (H, ω) is a *monotone algorithm*; i.e., the sequence of values resulted by the calls the algorithm makes to the basis calculation primitive is monotone increasing. The idea is to select a random constraint $h \in H$ and recursively find a basis B for $H \setminus \{h\}$. If h doesn't violate B, then output B; otherwise solve the problem recursively starting from a basis for $B \cup \{h\}$. Although the statement of the algorithm does not include a set of tight constraints (i.e., the set of constraints which the current minimum must satisfy), Observation 2.7 demonstrates that every basis found in the recursive call will include h. So the dimension of the problem is effectively reduced. They show that the algorithm requires expected $O(n)$ calls to the Basis primitive on subproblems with $d+1$ constraints and $O(n)$ calls to the Violation primitive, when the constant depends exponentially on d.

For the sake of completeness we state their algorithm. Function lptype is called with an initial basis C which they call a *candidate basis*. C is not necessarily a basis for H. It can be viewed as some auxiliary information one gets for the computation of the solution. Note that C can have influence on the running time and output of the algorithm (e.g., when there are several optimal bases).

Function lptype(H, C)
1. If $H = C$, then return C
2. Else
   (a) Choose $h \in H \setminus C$ uniformly at random
   (b) Let $B :=$ lptype$(H \setminus \{h\}, C)$
   (c) If Violation$(B, h) =$ TRUE, then return lptype$(H,$ Basis$(B \cup \{h\}))$
   (d) Else return B

Matoušek, Sharir, and Welzl [28] cite explicitly all of the properties which are needed for the correctness and time analysis of their algorithm

LEMMA 2.13 (see [28]). *Let* (H, ω) *be an abstract problem. The correctness and time analysis of algorithm lptype applied on* (H, ω) *as described in* [28] *are valid, if for all* $F, G \subseteq H$*,* $F \subseteq G$*, and* $h \in H$*:*

1. $\omega(G) \geq \omega(F)$.
2. *If* $\omega(G) = \omega(F) > -\infty$, *then* h *violates* G *if and only if* h *violates* F.
3. *If* $\omega(G) < \infty$, *then any* $F \subseteq G$ *has at most* d *extreme constraints.*
4. *If* $\omega(G) < \infty$, *then every basis* $B \subseteq G$ *for* G *has exactly* d *constraints.*

We note that a d-dimensional basis-regular LP-type problem (H, ω) satisfies all of the above properties: The monotonicity condition yields property 1. Corollary 2.4 yields property 2, the d-dimensionality of the (H, ω) together with Observation 2.6 yield property 3, and property 4 (which is needed only for the time analysis) results because (H, ω) is basis-regular.

A simple inductive argument shows that the procedure returns the required answer. This happens after a finite number of steps, since the first recursive call decreases the number of constraints, while the second call increases the value of the candidate basis (and there are only finitely many different bases).

Recall that t_v denotes the time required for a violation test and t_b denotes the time required for the Basis primitive on subproblems with $d + 1$ constraints. Let n_v (n_b) be the number of violation tests (basis computations) performed throughout the execution of the algorithm. Matoušek, Sharir, and Welzl (see section 4 in [28]) show that $n_v \leq n_b n$, which implies a crude upper bound of $O(n_b(t_v n + t_b))$ for the running time of the algorithm. They [28] give a careful and complicated analysis of this algorithm for the case where n is not much larger than d (e.g., $d \leq n \leq \sqrt{d}e^{d/4}$) and show that $n_b = e^{O(\sqrt{d \ln d})}$. Hence, for this case, the algorithm of [31] runs in randomized $O(e^{O(\sqrt{d \ln d})}(t_v n + t_b))$ time, i.e., subexponential in the dimension d of the problem. (Actually they use property 4 only for showing the subexponential bound in d.) Since, for linear programming, both the violation test and the basis calculation can be performed in time polynomial in both n and d, this gives a subexponential randomized algorithm for linear programming. Using this as the base-case algorithm at the third level of Clarkson's algorithm (i.e., x_s^*) gives expected $O(e^{O(\sqrt{d \ln d})} \log n)$ basis computations and expected $O(dn + d^2 \log n e^{O(\sqrt{d \ln d})}))$ violation tests. When d is constant, the running time of the combined algorithm is $O(t_v n + t_b \log n)$. We will use this expression in the analysis of the running times of many of our applications.

2.3. Helly-type theorems and their relations to LP-type problems. The first works to systematically study the relations between Helly-type theorems and LP-type problem were those of Amenta [1, 2]. In this subsection we summarize her results.

An LP-type problem (H, ω) with combinatorial dimension k is an abstract problem with combinatorial dimension k such that ω obeys monotonicity. Therefore the theorem below implies that there is a Helly-type theorem corresponding to the constraint set of every fixed-dimensional LP-type problem.

THEOREM 2.14 (see [2]). *Let* (H, ω) *be an abstract problem with combinatorial dimension* k *such that* ω *obeys monotonicity, and let* $\lambda \in \Lambda$ *be arbitrary.* H *has the property* $\omega(H) \leq \lambda$ *if and only if every* $G \subseteq H$ *with* $|G| \leq k + 1$ *has the property* $\omega(G) \leq \lambda$.

The main theorem in [2] goes in the other direction. Before stating it we need some definitions.

A *set system* is a pair (X, H), where X is a set and H is a set consisting of subsets of X. We say (X, H) is a *Helly system* if there exists a finite integer k such that H has Helly number k with respect to the intersection predicate. Most Helly theorems can be restated in terms of the intersection predicate. For example, let us consider the following Helly-type theorem.

THEOREM 2.15 (radius theorem). *A family H of points in the Euclidean d-dimensional space E^d is contained in a unit ball if and only if every $d+1$ or fewer points from H are contained in a unit ball.*

Here the family of objects is the set of points in E^d, the predicate is that a subfamily is contained in a (closed) unit ball, and the Helly number is $d+1$. In order to restate this theorem in terms of the intersection predicate, we apply the following duality transformation. We transform every point $h \in H$ into the set $\mathcal{D}(h)$ of centers of unit balls containing h. In this way $\mathcal{D}(h)$ is a unit ball centered at h. Let $\mathcal{D}(H) = \{\mathcal{D}(h) \mid h \in H\}$. From the definition of this duality transformation we get that the points in H are contained in a unit ball if and only if the unit balls in $\mathcal{D}(H)$ have a nonempty intersection (see Figure 2.2). Since balls are a special case of convex sets, the radius theorem derives directly from Helly's theorem.

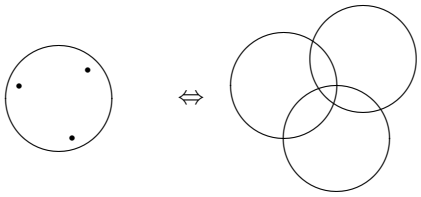


FIG. 2.2. *The 3 points on the left side are contained in a unit ball if and only if the 3 unit balls on the right side intersect.*

Recall that the range Λ of an LP-type problem can be any totally ordered set, and let (X, H) be a set system. We call $\omega' : X \to \Lambda$ a *ground set objective function*. We call $\omega : 2^H \to \Lambda$ the objective function *induced by* ω' on (X, H) if, for every $G \subseteq H$, $\omega(G)$ is the least value $\lambda^* \in \Lambda$ for which there exists $x^* \in \bigcap G$ such that $\omega'(x^*) = \lambda^*$, i.e., $\omega(G) = \min\{\omega'(x) \mid x \in \bigcap G\}$. If $\bigcap G = \emptyset$, we define $\omega(G) = \infty$. For example, when formulating lexicographic linear programming in the LP-type framework, the value of ω on a subset G of constraints is the minimum value that the ground set objective function ω' achieves on the points that are feasible with respect to G.

A *mathematical programming problem* is a triple (X, H, ω'), where X is a ground set, H is a set of subsets of X, and ω' is a ground set objective function to a totally ordered set Λ. We call the pair (H, ω), where ω is the objective function induced by ω' on (X, H), the *induced abstract problem*. If $|\{t \in \bigcap G \mid \omega'(t) = \omega(G)\}| = 1$ for all $G \subseteq H$, then we say that ω' satisfies the unique minimum condition (UMC).

Let $(X \times \Lambda, \bar{H})$ be a set system where Λ is a totally ordered set which contains a maximal element ∞. We call a ground set objective function ω' a *natural ground set objective function* if, for all $(x, \lambda) \in X \times \Lambda$, $\omega'(x, \lambda) = \lambda$. We call an objective function ω *natural* if it is induced by a natural ground set objective function. For every particular constraint $\bar{h} \in \bar{H}$ and $\lambda \in \Lambda$ we write $h_\lambda = \{x \in X \mid \exists \nu \leq \lambda \text{ s.t. } (x, \nu) \in \bar{h}\}$ for the projection into X of the part of $\bar{h}$ with Λ-coordinate no greater than λ. Also, for a subfamily of constraints $\bar{G} \subseteq \bar{H}$, we write G_λ as shorthand for $\{h_\lambda \mid \bar{h} \in \bar{G}\}$. We call an indexed family of subsets $\{h_\lambda \mid \bar{h} \in \bar{G}\}$, such that $h_\alpha \subseteq h_\beta$, for all $\alpha, \beta \in \Lambda$ with $\alpha < \beta$, a *nested family*.

Figure 2.3 (based upon Figure 1 in [2]) is a schematic diagram of a parameterized Helly system. The whole stack represents $X \times \Lambda$, and each of the cones represents a set $\bar{h} \in \bar{H}$. Each $\bar{h}$ is a subset of $X \times \Lambda$. Since all of the $\bar{h}$ are indexed with respect to Λ, the cross section at λ (represented by one of the planes) is equivalent to the Helly system (X, H_λ). Notice that if $\bar{G} \subseteq \bar{H}$ does not intersect at some value λ_2, then

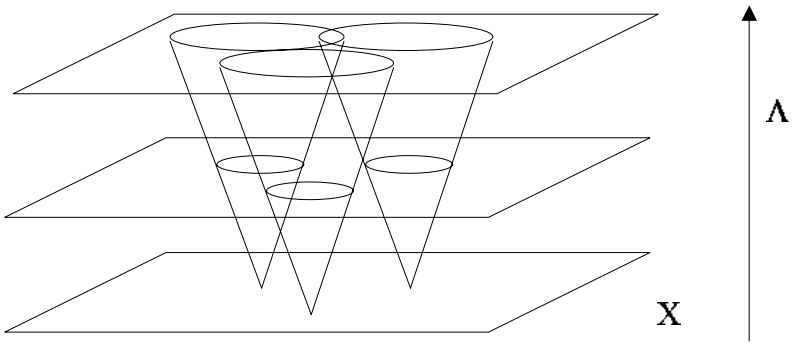


FIG. 2.3. *A parameterized Helly system.*

$\bar{G}$ also fails to intersects at all $\lambda_1 < \lambda_2$, and if $\bar{G} \subseteq \bar{H}$ intersects at λ_1, then $\bar{G}$ also intersects at all $\lambda_2 > \lambda_1$.

In her paper, Amenta [2] relates Helly-type theorems and LP-type problems by parameterization (a similar parameterization appears in [27] under the name "concrete LP-type problem").

DEFINITION 2.16. *A set system* $(X \times \Lambda, \bar{H})$ *is a* parameterized Helly system *with Helly number* k*, when*

1. $\{h_\lambda \mid \lambda \in \Lambda\}$ *is a nested family for all* $\bar{h} \in \bar{H}$*;*
2. (X, H_λ) *is a Helly system, with Helly number* k *for all* λ*.*

So the function ω' is just the projection into the Λ coordinate, and, for $\bar{G} \subseteq \bar{H}$, $\omega(\bar{G}) = \min\{\lambda \mid \bigcap G_\lambda \neq \emptyset\}$, or $\omega(\bar{G}) = \infty$ if $\bar{G}$ does not intersect at any value of λ.

Amenta [1] notes that it is almost always useful to think of Λ as time, so that a subfamily G_λ is a "snapshot" of the situation at time λ. Usually we can think of some initial time 0 at which G_0 does not intersect and then envision the h_λ growing greater with time, so that $\lambda^* = \omega(\bar{G})$ is the first "moment" at which G_λ intersects.

As an example, let us consider how the Helly system (X, H) for the radius theorem can be extended to a parameterized Helly system. (Recall that the ground set X of the Helly system representing the radius theorem is the set of centers of unit balls in E^d (which is equivalent to $\mathbb{R}^d$) and that each $h = h(p) \in H$ is the set of centers of unit balls which contain point p; i.e., $h(p)$ is a unit ball centered at p.) We define a parameterized Helly system $(X \times \Lambda, \bar{H})$, where $\Lambda = \mathbb{R}^+$ is the set radii, and each $h_\lambda = h(p)_\lambda \in H_\lambda$ is the set of centers at which a ball of radius at most λ contains a particular point p. The nested family $\bar{h} = \bar{h}(p)$ is the set of *all* balls containing p. The ground set $X \times \Lambda$ is the set of all balls in E^d, and $\bar{H}$ is the family of nested families for all points (see Figure 2.3).

The natural objective function for this parameterized Helly system $\omega(\bar{G})$ returns the smallest radius at which there is a ball containing all of the points corresponding to constraints $\bar{h} \in \bar{G}$. So $(X \times \Lambda, \bar{H}, \omega')$ is the following mathematical programming problem:

Problem: Smallest enclosing ball
Input: *A finite family* H *of points in* E^d*.*
Output: *The smallest ball enclosing* H*.*

In Figure 2.3 we see the parameterized Helly system corresponding to an instance H of the smallest enclosing ball problem consisting of 3 points. Each nested family $\bar{h}$ is a cone whose base is a point from H.

We say a ball is *realized* by points of H if it is the smallest volume ball enclosing

the points on its boundary. Assuming that the points in H are at general positions, such that no two different congruent balls are realized by points of H, the theorem below implies that the smallest enclosing ball in E^d problem can be formulated as a d-dimensional LP-type problem $(\bar{H}, \omega)$.

THEOREM 2.17 (main theorem in [2]). *Let $(X \times \Lambda, \bar{H})$ be a parameterized Helly system with Helly number k, natural ground set function ω', and natural objective function ω. If ω' meets the UMC, then $(\bar{H}, \omega)$ is an LP-type problem of combinatorial dimension k.*

Amenta showed that, without requiring the UMC, the theorem is not correct by giving an example of a Helly system with no fixed combinatorial dimension [2]. The theorem above is applied in [2] to get linear time solution algorithms for various geometric problems.

In her paper [2], Amenta investigates lexicographic objective functions. Let $(X \times \Lambda, \bar{H})$ be a parameterized Helly system with Helly number k and natural objective function ω. For all $\lambda \in \Lambda$, we assume a function $\nu_\lambda : 2^{H_\lambda} \to \Lambda'$, where Λ' is a totally ordered set containing a maximal element ∞, such that (H_λ, ν_λ) is an LP-type problem of combinatorial dimension at most d. The functions ν_λ may themselves be lexicographic. Amenta [2] imposes a lexicographic order on $\Lambda \times \Lambda'$ with $(\lambda, \kappa) > (\lambda', \kappa')$ if $\lambda > \lambda'$ or if $\lambda = \lambda'$ and $\kappa > \kappa'$. She defines a lexicographic objective function $\nu : 2^{\bar{H}} \to \Lambda \times \Lambda'$ in terms of ω and the functions ν_λ as seen in the following.

THEOREM 2.18 (see [2]). *Let Λ' be a totally ordered set. If $(X \times \Lambda, \bar{H})$ is a parameterized Helly system with Helly number k and natural objective function ω, and if, for every λ, (H_λ, ν_λ) is an LP-type problem of combinatorial dimension d, where $\nu_\lambda : 2^{H_\lambda} \to \Lambda'$, then $(\bar{H}, \nu)$ is an LP-type problem of combinatorial dimension $\leq k+d$, where $\nu : 2^{\bar{H}} \to \Lambda \times \Lambda'$ is defined as $\nu(\bar{G}) = (\omega(\bar{G}), \nu_{\omega(\bar{G})}(G_{\omega(\bar{G})}))$ for all $\bar{G} \subseteq \bar{H}$.*

Certainly, this bound on the combinatorial dimension is not always tight. For d-dimensional linear programming, for instance, this theorem gives an upper bound of $2d-1$ on the combinatorial dimension, since each H_λ is the constraint set of a $(d-1)$-dimensional linear program, and $(E^d, \bar{H})$ is a parameterized Helly system with Helly number d. Nonetheless, the theorem provides the best general bound as shown in [2].

3. Dual LP-type problems.

DEFINITION 3.1. *A* dual LP-type problem *is an abstract problem (H, ω) that obeys the following conditions (when we write $<$, $\leq$, $=$ etc., we mean under the totally ordered set Λ):*

1. Monotonicity: *For all $F \subseteq G \subseteq H : \omega(F) \geq \omega(G)$ (so the special element ∞ is such that $\omega(\emptyset) = \infty$).*
2. Locality: *For all $F \subseteq G \subseteq H$, with $\omega(G) = \omega(F) \neq \infty$, and for each $h \in H$, if $\omega(G \cup \{h\}) < \omega(G)$, then $\omega(F \cup \{h\}) < \omega(F)$.*

Let $G \subseteq H$ be arbitrary. If $\omega(G) = \infty$, we say G is *infeasible*; otherwise we call G *feasible*. If $\omega(G) = -\infty$, we say G is *unbounded*; otherwise we call G *bounded*. A *basis* B is a set $B \subseteq H$, with $\omega(B') > \omega(B)$ for all proper subsets B' of B. A *basis for* G is a basis $B \subseteq G$, with $\omega(B) = \omega(G)$. We note that due to the monotonicity condition a basis for G, for every $G \subseteq H$, always exists.

The *combinatorial dimension* d of a dual LP-type problem is the maximum cardinality of every basis for any *bounded* subfamily G. We note that the basis for every infeasible set is the empty set. A dual LP-type problem of combinatorial dimension d, where d is independent of $|H|$, is called *fixed-dimensional*. We choose the term dual LP-type (which should not be confused with the term dual in linear programming) because of the following.

Observation 3.2. The abstract problem (H, ω) is a dual LP-type problem if and only if $(H, -\omega)$ is an LP-type problem.

Looking at (H, ω), in order to prevent confusion between LP-type problems and their dual versions, we will denote by (D, ω) LP-type problems and by (S, ω) dual LP-type problems. The motivation for the choice of the letters "D" and "S" is as follows. We use the letter "D" in the LP-type problem (D, ω) since we look at D as a set of *demand elements (d-elements)*, or constraints on the feasible region, on which the minimum value is $\omega(D)$. Adding demand elements to D may increase the minimum solution of the problem and will never decrease its value. We use the letter "S" in the dual LP-type problem (S, ω) since we look at S as a set of *supply elements (s-elements)*, or relaxations on the feasible region, on which the minimum value is $\omega(S)$. Adding supply elements to S may decrease the minimum solution of the problem and will never increase its value. In the next section we define *discrete LP-type problems* by using the *same* ω in a primal and a dual LP-type problem.

4. Discrete LP-type problems.

DEFINITION 4.1. *A* discrete abstract problem *is a triple (D, S, ω), where D and S are finite sets of elements and ω is an* objective function *from $2^D \times 2^S \setminus \{(\emptyset, \emptyset)\}$ to some totally ordered set Λ which contains a special maximal (minimal) elements ∞ ($-\infty$). The goal is to compute $\omega(D, S)$.*

DEFINITION 4.2. *Let (D, S, ω) be a discrete abstract problem. For every $D', D'' \subseteq D$ and $S', S'' \subseteq S$ let $\alpha_{S'}(D'') = \omega(D'', S')$, and let $\beta_{D'}(S'') = \omega(D', S'')$. We say that (D, S, ω) is a* discrete LP-type problem *(DLP-type, in short) when $(D, \alpha_{S'})$ is an LP-type problem and $(S, \beta_{D'})$ is a dual LP-type problem for all $D' \subseteq D$ and $S' \subseteq S$. We say that $(D, \alpha_{S'})$ ($(S, \beta_{D'})$) is an* induced *LP-type (dual LP-type) problem of (D, S, ω).*

We note that we do not include $(\emptyset, \emptyset)$ in the domain of ω since this will result in the trivial ordered set $\Lambda = \{-\infty, \infty\}$, where $-\infty = \infty$: To see this, recall that the definition of LP-type problems implies that $\alpha_\emptyset(\emptyset) = -\infty$, the definition of dual LP-type problems implies that $\beta_\emptyset(\emptyset) = \infty$, and the definition of DLP-type problems implies that $\alpha_\emptyset(\emptyset) = \omega(\emptyset, \emptyset) = \beta_\emptyset(\emptyset)$.

Throughout this paper, whenever we call (D, α) $((S, \beta))$ *the induced* LP-type (dual LP-type) problem of (D, S, ω), we mean that $\alpha = \alpha_S$ ($\beta = \beta_D$). It is easy to see that the following definition for a DLP-type problem is equivalent to the former one.

DEFINITION 4.3. *A* DLP-type problem *is a discrete abstract problem (D, S, ω) which for all $S' \subseteq S$ and for all $D' \subseteq D$ obeys the following conditions (when we write $<$, $\leq$, $=$, etc., we mean under the ordered set Λ):*

1. Monotonicity of demand: *For all $D'' \subseteq D' \subseteq D : \omega((D'', S')) \leq \omega((D', S'))$.*
2. Monotonicity of supply: *For all $S'' \subseteq S' \subseteq S : \omega((D', S'')) \geq \omega((D', S'))$.*
3. Locality of demand: *For all $D'' \subseteq D' \subseteq D$ such that $\omega((D', S')) = \omega((D'', S')) > -\infty$ and for each $h \in D$, if $\omega((D' \cup \{h\}, S')) > \omega((D', S'))$, then $\omega((D'' \cup \{h\}, S')) > \omega((D'', S'))$.*
4. Locality of supply: *For all $S'' \subseteq S' \subseteq S$ such that $\omega((D', S')) = \omega((D', S'')) < \infty$ and for each $h \in S$, if $\omega((D', S' \cup \{h\})) < \omega((D', S'))$, then $\omega((D', S'' \cup \{h\})) < \omega((D', S''))$.*

Before continuing any further, we give an example of a DLP-type problem.

Problem: Discrete point set width

Input: *A finite set D of points in E^d and a finite set S of permissible directions.*

Output: *The minimal width of the set in the permissible directions (i.e., the minimal width of a band with a permissible direction which contains all the points in D).*

We assume general positions of the points and directions; that is, all $|S|\ \binom{|D|}{2}$ distances (in each one of the $|S|$ permissible directions) between pairs of points are different. For every set D of points and set S of permissible directions we define $\omega(D,S)$ to be the minimal width of the points in D in the permissible directions from S. Clearly (D,S,ω) is a discrete abstract problem. Let $S' \subseteq S$. We show now that $(D,\alpha_{S'})$ is an LP-type problem for every choice of S'. Since adding points to a set can only increase its width, $(D,\alpha_{S'})$ meets the monotonicity condition. Let $D'' \subset D' \subseteq D$ be such that $\alpha_{S'}(D'') = \alpha_{S'}(D')$. Due to the general position assumption there are unique $d_1, d_2 \in D''$ and $s \in S'$ such that the width of (D'',S') and of (D',S') is the distance between d_1 and d_2 in direction s. (In other words, the width of (D',S') and of (D'',S') is the width of the band in direction s between d_1 and d_2 in which all of the points of D' lie.) If for $h \notin D'$ $\alpha_{S'}(D' \cup \{h\}) > \alpha_{S'}(D')$, then point h is not inside this band, so there must be another triple of two points and one direction which realizes the width $\alpha_{S'}(D'' \cup \{h\})$. Due to the monotonicity condition, $\alpha_{S'}(D'' \cup \{h\}) \geq \alpha_{S'}(D'')$, and from the general position assumption we get that $\alpha_{S'}(D'' \cup \{h\}) > \alpha_{S'}(D'')$, so $(D,\alpha_{S'})$ meets the locality condition as well and thus is an LP-type problem.

Let $D' \subseteq D$. It remains to show that $(S,\beta_{D'})$ is a dual LP-type problem for every choice of D'. $(S,\beta_{D'})$ satisfies the monotonicity condition since adding directions to the set of permissible directions can only decrease the width. Let $S'' \subset S' \subseteq S$ be such that $\beta_{D'}(S'') = \beta_{D'}(S')$, and let $h \notin S'$. If $\beta_{D'}(S' \cup \{h\}) < \beta_{D'}(S')$, then the width $\beta_{D'}(S' \cup \{h\})$ must be realized by a band in direction h, that is, $\beta_{D'}(S' \cup \{h\}) = \beta_{D'}(\{h\})$. Hence we must have $\beta_{D'}(S'' \cup \{h\}) = \beta_{D'}(\{h\}) < \beta_{D'}(S') = \beta_{D'}(S'')$, so $(S,\beta_{D'})$ satisfies the locality condition as well and thus is a dual LP-type problem.

We now give more definitions. Let $G = (D',S') \in 2^D \times 2^S$ be arbitrary. Throughout this paper, if not explicitly specified otherwise, we choose G such that ω is defined on G, i.e., $G \neq (\emptyset,\emptyset)$. If $\omega(G) = \infty$, we say G is *infeasible*; otherwise we call G *feasible*. If $\omega(G) = -\infty$, we say G is *unbounded*; otherwise we call G *bounded*. We extend the terms "violates" and "extreme" in a natural way: We say that a d-element $h \in D \setminus D'$ (s-element $h \in S \setminus S'$) *violates* G if h violates D' (S') in the induced LP-type problem $(D,\alpha_{S'})$ (induced dual LP-type problem $(S,\beta_{D'})$). A d-element $h \in D'$ (s-element $h \in S'$) is *extreme* in G if h is extreme in D' (in S') in its induced LP-type problem $(D',\alpha_{S'})$ (induced dual LP-type problem $(S',\beta_{D'})$). We define bases in the following natural way.

DEFINITION 4.4. *Let (D,S,ω) be a DLP-type problem, let α and β be as defined in Definition* 4.2*, and let $G = (D',S') \in 2^D \times 2^S$. $B = (B_D,B_S) \in 2^{D'} \times 2^{S'}$ is a* basis for G in (D,S,ω) *if B_D is a basis for D' in its induced LP-type problem $(D,\alpha_{S'})$, and B_S is a basis for S' in its induced dual LP-type problem $(S,\beta_{D'})$.*

We note that there always exists a basis $B = (B_D,B_S)$ for any G.

Observation 4.5. Let (D,S,ω) be a DLP-type problem, and let $G = (D',S') \in 2^D \times 2^S$. If B is a basis for G, then $\omega(B) = \omega(G)$, and no $h \in D' \cup S'$ violates B.

This follows from both monotonicity conditions. $\omega(B) = \omega(G)$ since $\omega(G) = \omega(B_D,S') \leq \omega(B_D,B_S) \leq \omega(D',B_S) = \omega(G)$. $h \in D'$ doesn't violate B since $\omega(G) = \omega(B_D,B_S) \leq \omega(B_D \cup \{h\},B_S) \leq \omega(D',B_S) = \omega(G)$, and in a similar way $h \in S'$ doesn't violate B. In order to illustrate the term "basis" let us consider the following instance of the discrete point set width problem.

Example 4.6. Let $G = (D,S)$, where $D = \{(0,0);(2,1);(1,5)\}$ and $S = \{\text{horizontal, vertical}\}$, be an instance of the discrete point set width problem. The

minimal width is achieved by a vertical strip of width 2. Let (D, α) and (S, β) be its induced LP-type and induced dual LP-type problems, respectively. At first glance one may be tempted to suggest $B = (B_D, B_S) = (\{(0,0);(2,1)\}, \{\text{vertical}\})$ as a basis for G, since $\omega(B) = \omega(G)$. This B is not a basis for G, since B_D is not a basis for (D, α) (because of the horizontal direction: $\alpha(B_D) = \omega(B_D, \{\text{horizontal}\}) = 1 \neq \omega(B_D \cup \{(1,5)\}, \{\text{horizontal, vertical}\})$). The other subsets of (D, S) on which the value of ω is 2 are (D, S) and $(D, \{\text{vertical}\})$. (D, S) fails to be a basis for G because S is not a basis in (S, β) ($\beta(S \setminus \{\text{horizontal}\}) = \beta(S)$). It is easy to verify that D is a basis for D in (D, α) and $\{\text{vertical}\}$ is a basis for S in (S, β). Thus, $(D, \{\text{vertical}\})$ is a basis for G.

A "discrete" version of Observation 2.5 is as follows.

Observation 4.7. Let (D, S, ω) be a DLP-type problem. Let $G = (D', S') \in 2^D \times 2^S$, and let $B = (B_D, B_S) \in 2^{D'} \times 2^{S'}$ be such that $\omega(B) = \omega(G)$. If $\omega(B) \neq \omega(D, S)$, then there exists an element in either $D \setminus D'$ or $S \setminus S'$ which violates B.

To see this suppose first that the inequality is $\omega(B) < \omega(D, S)$. Considering the induced LP-type problem (D, α_{B_S}), and since $\omega(D, S) \leq \omega(D, B_S)$, this implies that $\alpha_{B_S}(B_D) < \alpha_{B_S}(D)$. Applying Observation 2.5 on (D, α_{B_S}), D', and B_D, we get that there exists $h \in D \setminus D'$ that violates B_D in (D, α_{B_S}). Hence h violates B. The case where the inequality is $\omega(B) > \omega(D, S)$ is treated similarly by considering the induced LP-type problem (S, β_{B_D}).

Corollary 4.8. *Let (D, S, ω) be a DLP-type problem. Let $G = (D', S') \in 2^D \times 2^S$, and let $B \in 2^{D'} \times 2^{S'}$ be a basis for G. If no $h \in (D \setminus D') \cup (S \setminus S')$ violates B, then B is a basis for (D, S) as well.*

Proof. We need to prove that B_D is a basis for D in the induced problem (D, α) and that B_S is a basis for S in the induced problem (S, β). We prove the first part. The proof of the second part is similar. We first show that B_D is a basis in (D, α). Let B'_D be a proper subset of B_D.

(4.1)
$$\alpha(B'_D) = \omega(B'_D, S) \leq \omega(B'_D, S') < \omega(B_D, S') = \omega(B_D, B_S) = \omega(B_D, S) = \alpha(B_D).$$

The first inequality follows from monotonicity of supply, the second (strict) inequality follows from the fact that B is a basis for G, and therefore B_D is a basis in $(D', \alpha_{S'})$, the following equality is due to the fact that B is a basis for G, and the next equality is due to Observation 2.5 applied on (S, β_{B_D}) (B_S is a basis for S' in this dual LP-type problem). It remains to show that $\alpha(B_D) = \alpha(D)$. From (4.1) we have $\alpha(B_D) = \omega(B_D, S) = \omega(B_D, B_S)$. We conclude by deriving from Observation 4.7 that $\omega(B_D, B_S) = \omega(D, S) = \alpha(D)$. □

We now define a condition sufficient for a DLP-type problem (D, S, ω) to satisfy a discrete version of Observation 2.7. This condition is used in the proof of correctness of our DLP-type algorithms.

Definition 4.9. *We say that the DLP-type problem (D, S, ω) satisfies the violation condition (VC) if for every $(D', S') \in 2^D \times 2^S$ and $(D'', S'') \in 2^{D'} \times 2^{S'}$ with $\omega(D', S') = \omega(D'', S'')$ the following properties hold:*

1. *For every $h \in D$, if $\omega(D'' \cup \{h\}, S'') > \omega(D'', S'')$, then $\omega(D' \cup \{h\}, S') > \omega(D', S')$;*
2. *for every $h \in S$, if $\omega(D'', S'' \cup \{h\}) < \omega(D'', S'')$, then $\omega(D', S' \cup \{h\}) < \omega(D', S')$.*

Note that due to Corollary 2.4 this condition is always satisfied whenever either $S' = S''$ or $D' = D''$. The lemma below is a discrete version of Observation 2.7.

LEMMA 4.10. *Let (D, S, ω) be a DLP-type problem which satisfies the violation condition. Let $G = (D', S') \in 2^D \times 2^S$, and let $B = (B_D, B_S) \in 2^{D'} \times 2^{S'}$ be a basis for G. If $h \in D$ ($h \in S$) violates B, then h is extreme in $(D' \cup \{h\}, S')$ ($(D', S' \cup \{h\})$) and is a member of every basis for this set.*

Proof. We will prove the case where $h \in D$. The proof for $h \in S$ is similar. If $h \in D$ violates B, i.e., $\omega(B_D \cup \{h\}, B_S) > \omega(B)$, then due to the VC

$$\omega(D' \cup \{h\}, S') > \omega(B) = \omega(D', S'), \tag{4.2}$$

so h is extreme in $(D' \cup \{h\}, S')$. To see that h is a member for every basis $B' = (B'_D, B'_S)$ for $(D' \cup \{h\}, S')$, we use the fact that B'_D is a basis for the induced LP-type problem of $(D' \cup \{h\}, S')$ (so $\omega(B') = \omega(B'_D, S')$) and (4.2) to get

$$\omega(B'_D, S') = \omega(D' \cup \{h\}, S') > \omega(D', S'). \tag{4.3}$$

We conclude the proof by noting that if h is not a member in B'_D, then $B'_D \subseteq D'$ and due to monotonicity of demand $\omega(B'_D, S') \leq \omega(D', S')$, in contradiction to (4.3). ∎

The *demand combinatorial dimension* k_D (d-dimension, in short) of (D, S, ω) is the combinatorial dimension of its induced LP-type problem. A DLP-type problem of d-dimension k_D, where k_D is independent of $|D| + |S|$, is called *fixed d-dimensional*. We define the terms *supply combinatorial dimension* (s-dimension, in short) and *fixed s-dimensionality* analogously. We call a DLP-type problem which is both fixed s-dimensional (of dimension k_S) and fixed d-dimensional (of dimension k_D) (k_D, k_S)-dimensional. A (k_D, k_S)-dimensional DLP-type problem where both its induced LP-type problem and induced dual LP-type problem are basis regular is called a (k_D, k_S)-dimensional *basis-regular* DLP-type problem.

We note that the discrete point set width problem is not fixed-dimensional. To see this, suppose by negation that it is k-d-dimensional. Consider an instance of the problem with $n = 2k$ d-elements, consisting of k pairs of antipodal points which are located on a unit circle. Let the s-elements be the n directions perpendicular to the one-unit length segments connecting the antipodal points. Clearly, each proper subset of the d-elements admits a width of less than one unit, whereas the width of the whole set is one unit. Therefore, the number of d-elements in any basis is at least $2k$, in contradiction to our assumption that the problem is k-d-dimensional.

If the problem were fixed-dimensional, the DLP algorithms stated in the next section would solve the problem in (randomized) linear time. The variable dimensionality of the problem is not surprising, since the problem admits an $\Omega(n \log n)$ (deterministic) lower bound under the algebraic computation tree model due to a linear time reduction from:

Problem: Set equality
Input: *Sets A and B of n real numbers each.*
Output: *"true" if and only if $A = B$.*

LEMMA 4.11 (see [3]). *Solving set equality requires $\Omega(n \log n)$ operations under the algebraic computation tree model.*

LEMMA 4.12. *Solving discrete point set width requires $\Omega(n \log n)$ operations under the algebraic computation tree model.*

Proof. Consider without loss of generality two sets A and B of n positive numbers each and a unit circle with center at the origin. We construct from A and B an instance (D, S) of discrete point set width. The numbers of A are transformed into points in D, and the numbers of B are transformed into directions in S as follows. We transform each number $a \in A$ into the two intersection points of the unit circle with the line

with slope a that passes through the origin. We transform each number $b \in B$ into the direction vertical to a line with slope b. It is easy to see that the solution of the instance (D, S) of the discrete point set width is 1 if and only if $A \equiv B$. □

We now define a condition sufficient for a DLP-type problem (D, S, ω) to be fixed s-dimensional.

DEFINITION 4.13. *Let (D, S, ω) be a discrete abstract problem, and let $p \in \mathbb{N}$. We say that (D, S, ω) is a* p-supply problem *if for every $G = (D', S') \in 2^D \times 2^S$ there exists $S'' \subseteq S'$ such that $|S''| \leq p$ and $\omega(D', S'') = \omega(G)$.*

LEMMA 4.14. *A DLP-type problem (D, S, ω) which is a p-supply problem is p-s-dimensional.*

Proof. Let (S, β) be its induced dual LP-type problem. Suppose by negation that there exists a bounded $S' \in 2^S$ and a basis B for S' with $|B| > p$. From the definition of a basis in dual LP-type problems, for every proper subset $B' \subset B$, $\beta(B') > \beta(B) = \beta(S')$. This contradicts the fact that (D, S, ω) is a p-supply problem. □

Integer programming can be formulated as a DLP-type problem where D is a set of half-hyperplanes and $S = \mathbb{Z}^k$. There is one problem with this formulation: The set S is not finite. We can overcome this by noting that, when given an instance of an IP problem, it is always possible to bound the integer lattice by a big box (whose radius depends exponentially on the input size), such that the solution of the IP problem, if it exists, is found inside the bounding box (see, for example, Theorem 17.2 in [30]). Because of the above, solving IP by the DLP-type model is not efficient.

5. DLP-type algorithms. Given an instance (D, S, ω) of a (k_D, k_S)-dimensional DLP-type problem, let $n = |D|$ and $m = |S|$. Similarly to the assumptions made with the LP-type model, we assume two primitive operations. A *basis computation* Basis(D', S') takes an ordered pair $G = (D', S')$, with $|D'| \leq k_D + 1$ and $|S'| \leq 9k_S^2$, and finds a basis for G. A *violation test* Violation(B, h) takes a basis B and a constraint h and returns true if and only if h violates B. Let t_b be the time required for a basis computation and t_v be the time required for a violation test.

We observe that, when changing (by deleting or adding elements) the set D (S) while keeping the set S (D) unchanged, the problem behaves like an LP-type (dual LP-type) problem. Thus, while "fixing" the set S (D) one can use LP-type algorithms in order to solve the induced LP-type (dual LP-type) problem on D (S).

In Chapter 6 in [17] we have developed several randomized algorithms that solve fixed-dimensional DLP-type problems that satisfy the VC in linear time. The algorithms differ in the choice of the LP-type algorithms used to solve the induced LP-type and dual LP-type problems and in the decision rules when and with which input to call these algorithms.

The 4-layer algorithm given below uses this observation. In the first layer, i.e., in Function DLP, the set of s-elements does not change, so Function DLP resembles Function x_m^* in Clarkson's algorithm [8] applied on the induced LP-type problem. In the second layer, i.e., in Function M, the set of d-elements does not change, so Function M (as well as its name) resembles Function x_m^* in Clarkson's algorithm [8] applied on the induced dual LP-type problem. The purpose of Function DLP (Function M) is to get the number of constraints (relaxations) down, so we can apply the third level Function I, which resembles (as well as its name) Function x_i^* in [8] and is more efficient in k_S but less efficient in $|D|$ and $|S|$. The fourth layer Function Demand is called only when the cardinality of the s-element set is bounded by $9k_S^2$ and it behaves similarly to Sharir and Welzl's algorithm [31], applied on the induced LP-type problem.

Function DLP(D, S)

1. Let $V^* := \emptyset$, let $V := D$, and find a candidate basis C_D for D in the induced LP-type problem of (D, S, ω)
2. If $|D| \leq 9k_D^2$, then return $\mathrm{M}(D, S, C_D)$
3. Else repeat the following until $V = \emptyset$:
   (a) Choose $R \subset D \setminus V^*$ uniformly at random, $|R| = k_D\sqrt{|D|}$
   (b) Find a candidate basis C_D for $R \cup V^*$ in the induced LP-type problem of $(R \cup V^*, S, \omega)$
   (c) Let $B := \mathrm{M}(R \cup V^*, S, C_D)$, and let $V := \{d \in D \mid \text{Violation}(B, d) = \text{TRUE}\}$
   (d) If $|V| \leq 2\sqrt{|D|}$, then let $V^* := V^* \cup V$
4. Return B

Function M(D, S, C_D)

1. Let $V^* := \emptyset$, let $V := S$
2. If $|S| \leq 9k_S^2$, then return $\text{Demand}(D, S, C_D)$
3. Else repeat the following until $V = \emptyset$:
   (a) Choose $R \subset S \setminus V^*$ uniformly at random, $|R| = k_S\sqrt{|S|}$
   (b) Let $B := \mathrm{I}(D, R \cup V^*, C_D)$, and let $V := \{s \in S \mid \text{Violation}(B, s) = \text{TRUE}\}$
   (c) If $|V| \leq 2\sqrt{|S|}$, then let $V^* := V^* \cup V$
4. Return B

Function I(D, S, C_D)

1. For every $s \in S$ let $\nu_s := 1$
2. If $|S| \leq 9k_S^2$, then return $\text{Demand}(D, S, C_D)$
3. Else repeat the following until $V = \emptyset$:
   (a) Choose $R \subset S$ at random according to weights ν_s, $|R| = 9k_S^2$
   (b) Let $B := \text{Demand}(D, R, C_D)$
   (c) Let $V := \{s \in S \mid \text{Violation}(B, s) = \text{TRUE}\}$
   (d) If $\nu(V) \leq 2\nu(S)/(9k_S - 1)$, then for every $s \in V$ let $\nu_s =: 2\nu_s$
4. Return B

Function Demand(D, S, C_D)

1. If $D = C_D$, then return $\text{Basis}(C_D, S)$
2. Else
   (a) Choose a random $d \in D \setminus C_D$
   (b) Let $B = (B_D, B_S) := \text{Demand}(D \setminus \{d\}, S, C_D)$
   (c) If $\text{Violation}(B, d) = \text{TRUE}$, then return $\text{Demand}(D, S,$ the first coordinate of $\text{Basis}(B_D \cup \{d\}, S))$
   (d) Else return B

We will first show that Function Demand returns the required answer by showing that all of the arguments of [28] apply here as well. We can view Function Demand applied on the DLP-type problem (D, S, ω) as a function applied on its induced LP-type problem (D, α). This is true since the s-elements set S does not change throughout the execution of Function Demand. In this way all of the conditions stated in Lemma 2.13 are satisfied.

Function Demand is similar, but not identical, to Function lptype of Sharir and Welzl, only because of line 2(c). Due to Corollary 2.4, the violation test in Function lptype, Violation(B, h), returns true if and only if B is not a basis for H. If in Function Demand we called Function Violation$((B_D, S), d)$ instead of calling Function Violation(B, d), then we would get *exactly* Function lptype applied on the induced LP-type problem (D, α) (but the running time would increase by a big constant depending on k_S). Because of this difference we need to prove Lemma 5.1.

LEMMA 5.1. *Let (D, S, ω) be a (k_D, k_S)-dimensional DLP-type problem which meets the VC, and suppose $\omega(D, S) = \omega(B) > -\infty$. Let $B = (B_D, B_S)$ be a basis for $(D \setminus \{d\}, S)$. Let (D, α) be the induced LP-type problem of (D, S, ω). The violation test Violation(B, d) in Function Demand applied on (D, S, ω) returns true if and only if B_D is not a basis for D in (D, α).*

Proof. If d does not violate B, then, due to Corollary 4.8, B is a basis for (D, S). Hence, B_D is a basis for D in (D, α). If d does violate B, then due to the VC we get that $\alpha(D) > \alpha(D \setminus \{d\}) = \alpha(B_D)$, which implies that B_D is not a basis for D. □

So this lemma implies that Function Demand correctly computes a basis for (D, S) whenever (D, S, ω) meets the VC. We now compute t_D, the time needed for Function Demand to run. Let t_{vD} (t_{bDS}) be the time required for the violation test Violation(B, d) (the basis calculation Basis$_{DS}$). Using the analysis in [28], Function Demand calls Functions Basis$_{DS}$ and Violation $O(|D|)$ times where the constant depends (exponentially) on k_D, so the running time of Function Demand is

$$t_D = O(|D|(t_{vD} + t_{bDS})). \tag{5.1}$$

If the violation test and basis calculation are done in constant time, Function Demand runs in $O(n)$ time.

We next show that Functions M and I return the required value. In order to prove this we need to show that Lemmas 2.8, 2.9, and 2.10 and Theorem 2.11 can be modified for the DLP-type framework. We also rely, of course, on the correctness of Function Demand. Lemma 2.10 and Theorem 2.11 are straightforwardly adapted to the DLP-type case. We provide proofs for the first 2 lemmas.

LEMMA 5.2 (adaptation of Lemma 3.1 in [8]). *In Functions M and I, if the set V is nonempty, and if (D, S, ω) satisfies the VC, then V contains an element from B'_S, where $B' = (B'_D, B'_S)$ is any basis of (D, S).*

Proof. We prove the correctness of Function M. The proof for Function I is similar. Let $S^* = R \cup V^*$, and let $B = (B_D, B_S)$ be a basis for (D, S^*). Let NV be the set of s-elements in $S \setminus S^*$ that do not violate B; i.e., S decomposes into $S = S^* \uplus NV \uplus V$. If V is not empty, then there exists $s \in V$ such that $\omega(B_D, B_S \cup \{s\}) < \omega(B)$. Hence, since (D, S, ω) satisfies the VC and s violates B, it also violates (D, S^*), that is, $\omega(D, S^* \cup \{s\}) < \omega(B)$. From the monotonicity of supply condition we get

$$\omega(D, S) < \omega(B). \tag{5.2}$$

None of $s \in NV$ violates B, so, by Corollary 4.8, B is a basis for $(D, S^* \cup NV)$. Let us consider an arbitrary basis $B' = (B'_D, B'_S)$ for (D, S). If B'_S does not contain an

element from V, then $B'_S \subseteq S^* \cup NV$, so by the monotonicity of supply condition we get

$$\omega(D,S) = \omega(B') = \omega(D,B'_S) \geq \omega(D, S^* \cup NV) = \omega(B),$$

in contradiction to (5.2). □

Note that the above lemma is the sole reason for which the VC is required to derive a linear time solution. (The discussion in the paragraph preceding Lemma 5.1 implies that, if the VC were not satisfied, it would still be possible to modify Function Demand to work correctly at an additional constant cost.)

LEMMA 5.3 (adaptation of Lemma 3.2 in [8]). *Let $R \subset S$ be a random subset of size r, where $|S| = m$. If $V \subset S$ is the set of elements violating a basis of (D,R), then its expected size is no more than $k_S(m-r)/(r+1)$.*

Proof. The probability that a random element $s \in S \setminus R$ violates a basis of (D,R) is not more than $k_S/(r+1)$, since $|B_S| \leq k_S$ for every basis (B_D, B_S) and the total size of the sample R with the element s is $r+1$. From the linearity of expectation the expected size of V is not more than $k_S(m-r)/(r+1)$. □

We now compute the complexity of Functions M and I. Theorem 2.12 tells us that Function M calls Function I $O(k_S)$ times (with an s-element set of size $O(\sqrt{|S|})$) and calls Function Violation $O(k_S|S|)$ times. Function I (when called with $|S|$ elements) calls Function Demand $O(k_S \log |S|)$ times and calls Function Violation $O(k_S|S| \log |S|)$ times.

If Function Demand runs in t_D time, then the total running time of Function M is $O(k_S|S|t_{vS} + k_S^2 \log |S| t_D))$, where the constant factors do not depend on k_D and k_S. Using (5.1), we get that the total running time of Function M is $O(k_S|S|t_{vS} + k_S^2(\log |S|)|D|(t_{vD} + t_{bDS}))$, where the constant depends exponentially on k_D.

After proving that Functions M, I, and Demand are correct and calculating their running times, it remains to consider Function DLP. In order to prove that Function DLP works correctly, we need to show that Lemmas 2.8, 2.9, and 2.10 and Theorem 2.11 can be modified for the DLP-type framework. This is done similarly to the way it was proved for Functions M and I.

It remains to consider the running time of Function DLP. Due to Theorem 2.12, Function DLP calls Function M (with a d-element set of size $k_D\sqrt{|D|}$) $O(k_D)$ times and calls Function Violation $O(k_D|D|)$ times. In this way Function Demand is called $O(k_D k_S^2 \log |S|)$ times, with an s-element set of constant size C and a d-element set of size $O(k_D\sqrt{|D|})$. If Function Demand is implemented in t_D time, then the total running time of Function DLP is $O(k_D(|D|t_{vD} + k_S|S|t_{vS} + k_S^2 \log |S| t_D))$, where the constant factors do not depend on k_D and k_S. Using (5.1), we get that the total running time of Function DLP is $O(k_D(|D|t_{vD} + k_S|S|t_{vS} + k_S^2 \log |S|\sqrt{|D|}(t_{vD} + t_{bDS})))$, where the constant depends exponentially on k_D. If the violation tests and basis calculations are done in constant time, this algorithm runs in $O(|D|+|S|)$ time. We have proved the following.

THEOREM 5.4. *Let (D,S,ω) be a (k_D,k_S)-dimensional DLP-type problem which meets the VC. Function DLP solves it in $O((|S|+|D|)t_v + \sqrt{|D|}\log|S| t_b)$ randomized time, where t_v (t_b) is the time needed for the violation test (basis calculation of a set consisting of k_D+1 d-elements and $9k_S^2$ s-elements) primitive.*

We summarize the structure of our algorithm in the following table (recall that $|D| = n$ and $|S| = m$):

The constants in the above algorithm may depend exponentially on k_D and k_S. We can get a linear time algorithm where the constants depend subexponentially on k_D

Function	*Input* $\lvert D\rvert$	*Input* $\lvert S\rvert$	*# iterations*	*Sample from*	*Sample size*
DLP	n	m	k_d	D	$\sqrt{n}$
M	$\sqrt{n}$	m	k_s	S	$\sqrt{m}$
I	$\sqrt{n}$	$\sqrt{m}$	$\log m$	S	const
Demand	$\sqrt{n}$	const	$\sqrt{n}$	D	1

and k_S when (D, S, ω) is basis regular. The idea is to call the modified algorithm of Sharir and Welzl only after the sizes of both the d-element set and the s-element set are reduced to constants and use the fact that this algorithm runs in linear time where the constants depend subexponentially on the dimension of the problem, when the problem is basis regular. Recall that Function I is a modified version of Function x_i^* in [8], applied on the s-element set. Instead of calling Function Demand in lines 2 and 3(b), we change it to call a new and similar Function I', which is a modified version of Function x_i^* in [8], applied on the d-element set. Function I' will call Function Demand in the lines corresponding to lines 2 and 3(b) in Function I. Thus Function Demand is called with both d-element and s-element sets of constant size. Recall that Function Demand is a modified version of Function lptype in [31], applied on the d-elements set. Instead of calling Function Basis in lines 1 and 2(c), we change it to call a new and similar Function Supply, which is a modified version of Function lptype in [31], applied on the s-element set. Function Supply will call Function Basis in the lines corresponding to lines 1 and 2(c) in Function Demand. Using similar arguments to the ones mentioned earlier in this section, we get that the resulting 6-layer algorithm proves the following theorem.

THEOREM 5.5. *Let (D, S, ω) be a (k_D, k_S)-dimensional DLP-type problem which meets the VC. (D, S, ω) is solved in $O((|S| + |D|)t_v + \sqrt{|D||S|}\log|D|\log|S|t_b)$ randomized time, where t_v (t_b) is the time needed for the violation test (basis calculation of a set consisting of at most $k_D + 1$ d-elements and $k_S + 1$ s-elements) primitive. If (D, S, ω) is basis regular, then the constants depend subexponentially on k_D and k_S.*

6. Continuous lexicographic Helly-type theorems and their relations to the LP-type model. Amenta [2] concludes her paper with "The major open problem is to characterize the Helly systems (X, H) for which there is an objective function ω that gives a fixed-dimensional LP-type[1] problem (H, ω)." We give a partial answer for her question in this section, by showing that *every* lexicographic Helly system (to be defined below) admits an objective function ω that gives a fixed-dimensional LP-type problem (H, ω).

Let $(X \times \Lambda, \bar{H})$ be a parameterized Helly system with Helly number k and ω be a natural objective function. If ω meets the UMC, then, by Theorem 2.17, $(\bar{H}, \omega)$ is an LP-type problem of combinatorial dimension k. If ω does not satisfy the UMC, in order to get a fixed-dimensional LP-type problem, one normally uses the following two "tricks." If possible, assume that the input is in such a general position that ω satisfies the UMC. Alternatively, explicitly change ω to be a lexicographic function ν whose first parameter is ω. The resulting LP-type problem $(\bar{H}, \nu)$ has usually combinatorial dimension greater than k (see [2, 32]).

Consider, for instance, LP. As noted in section 3 in [2], the parameterized Helly system corresponding to LP does not generally satisfy the UMC, but by using a lexicographic objective function, it does. As an additional example, consider the smallest

[1] Amenta uses the term GLP rather than LP-type.

enclosing ball problem defined in section 2.3. This problem does not necessarily satisfy the UMC. When we assume that the points in H are in general positions, such that no two different congruent balls are realized by points of H, this problem does satisfy the UMC.

Our approach is different. We provide a machinery which converts any parameterized lexicographic Helly system (to be defined below) into an LP-type problem. In this way, instead of extending the objective function, using (standard) Helly theorems, assuming UMC, and applying Theorem 2.17, we use lexicographic objective functions, lexicographic Helly theorems, and our framework. Unlike Theorem 2.17, this machinery does not require that the natural objective function meets the UMC.

We give some definitions first. For every totally ordered set Λ and $d \in \mathbb{N}$ we impose a lexicographic order on Λ^d such that for any $x = (x_1, \ldots, x_d), y = (y_1, \ldots, y_d) \in \Lambda^d$ we say that $x <_L y$ (x is lexicographically smaller than y (*lsmaller*, in short)) if $x_1 < y_1$ or there exists $d \geq k > 1$ such that $x_i = y_i$ for $i = 1, 2, \ldots, k-1$, and $x_k < y_k$. We say that $x \geq_L y$ if $x <_L y$ does not hold. For every $X \subseteq \Lambda^d$ and $x \in \Lambda^d$ we let $X_x = \{x' \in X \mid x' \leq_L x\}$ and let $X^x = \{x' \in X \mid x' \geq_L x\}$. We note that if X is a convex set, then for every $x \in X$, X_x and X^x are convex sets as well.

DEFINITION 6.1. *Let Λ be a totally ordered set. A* Helly system with lexicographic Helly number l *is a set system (X, H), where $X \subseteq \Lambda^d$ for some positive integer d, such that, for any $x \in X$, $(X, \{h \cap X_x \mid h \in H\})$ is a Helly system with Helly number l.*

This means that for any $x \in X$, whenever every l or less elements of H have a common point which is not lgreater than x, we get that all elements of H have a common point which is not lgreater than x.

In order to get LP-type problems from lexicographic Helly theorems, we impose a lexicographic order on the ground set X and parameterize the Helly system (X, H) with lexicographic Helly number l.

DEFINITION 6.2. *A set system $(X \times X, \bar{H})$ is a* parameterized Helly system with lexicographic Helly number l *if there exists a Helly system with lexicographic Helly number l, (X, H), such that, for all $h \in H$, $\bar{h} = \{(y, x) \mid x \in X,\ y \in h \cap X_x\}$ and $\bar{H} = \{\bar{h} \mid h \in H\}$.*

From the definitions it is easy to verify the following.

Observation 6.3. Let $(X \times X, \bar{H})$ be a parameterized Helly system with lexicographic Helly number l. For every $x, y \in X$ and $\bar{h} \in \bar{H}$ the following attributes hold:

1. $\{h_x \mid x \in X\}$ is a nested family for all $\bar{h} \in \bar{H}$.
2. (X, H_x) is a Helly system with lexicographic Helly number l.
3. $(X \times X, \bar{H})$ is a parameterized Helly system with Helly number l.
4. $(y, x) \in \bar{h} \rightarrow (y, y) \in \bar{h}$.
5. $(y, x) \in \bar{h} \rightarrow y \leq_L x$.

The importance of lexicographic Helly theorems follows partly from the following two results.

THEOREM 6.4. *Let $(X \times X, \bar{H})$ be a parameterized Helly system with lexicographic Helly number l. If ω is its natural objective function, then $(\bar{H}, \omega)$ is an LP-type problem of combinatorial dimension l.*

Proof. We show that all of the conditions of Theorem 2.17 are satisfied. Due to attribute 3 in Observation 6.3, $(X \times X, \bar{H})$ is a parameterized Helly system with Helly number at most l. It remains to show that the natural objective function ω meets the UMC. Suppose on the contrary that there is $\bar{G} \subseteq \bar{H}$, with $\omega(\bar{G}) = x$, such that there are two different points $x', x'' \in X$ such that both $(x', x), (x'', x) \in \bigcap \bar{G}$ realize $\omega(\bar{G})$. Due to attribute 5 in Observation 6.3, $x', x'' \leq_L x$. Without loss of

generality $x'' <_L x'$. Hence $x'' <_L x$, and from attribute 4 in Observation 6.3 we get that $(x'', x'') \in \bigcap \bar{G}$, so $\omega(\bar{G}) \leq_L x'' <_L x$ in contradiction. □

THEOREM 6.5. *Let $(X \times \Lambda, \bar{H})$ be a parameterized Helly system with Helly number k and natural objective function ω. If, for every $\lambda \in \Lambda$, (X, H_λ) is a Helly system with lexicographic Helly number l, then there is a function $\nu : 2^{\bar{H}} \to \Lambda \times X$ such that for all $\bar{G} \subseteq \bar{H}$ the first part of $\nu(\bar{G})$ is $\omega(\bar{G})$ and $(\bar{H}, \nu)$ is an LP-type problem of combinatorial dimension $\leq k + l$.*

Proof. For every $\lambda \in \Lambda$ we parameterize the Helly system (X, H_λ) such that $(X \times X, \bar{H_\lambda})$ is a parameterized Helly system with lexicographic Helly number l. If its natural objective function ν_λ is not well-defined, we symbolically compactify the space X by representing points at infinity. Due to Theorem 6.4 the resultant abstract problem (H_λ, ν_λ) is an LP-type problem of combinatorial dimension l. We conclude our proof by using Theorem 2.18. □

This theorem is useful when we want to omit general position assumptions. For instance, we reconsider the smallest enclosing ball problem. In the beginning of this section we represented this problem on the set H of points in $\mathbb{R}^d$ as a parameterized Helly system with Helly number $d+1$, $(X \times \Lambda, \bar{H})$, where $\Lambda = \mathbb{R}^+$ is the set of radii, and each $h_\lambda \in H_\lambda$ is the set of centers at which a ball of radius at most λ contains a particular point $h \in H$. The natural objective function ω is just the minimal radius of a ball which encloses all of the points in H. By assuming that the input points are in general positions, we caused the natural ground set function ω' to meet the UMC. In this way all of the conditions of Theorem 2.17 are met, and $(\bar{H}, \omega)$ is a d-dimensional LP-type problem.

Using the lexicographic version of Helly's theorem, Theorem 1.2, we note that the Helly system (X, H) representing the radius theorem (i.e., the ground set $X = \mathbb{R}^d$ is the set of centers of unit balls in E^d, and H is a family of unit balls) has lexicographic Helly number $d+1$. In this way we get that, for every $\lambda \in \Lambda = \mathbb{R}^+$, (X, H_λ) (i.e., $X = \mathbb{R}^d$ and H_λ is a family of balls of radius at most λ) is a Helly system with lexicographic Helly number $d+1$. Applying Theorem 6.5, we get that $(\bar{H}, \nu)$ is an LP-type problem of combinatorial dimension $\leq 2(d+1)$, where the first parameter of the objective function ν is the radius of the smallest enclosing ball of H.

It is possible to bound the combinatorial dimension of the resulting LP-type problem even further. We give some more definitions first. In the Helly system (X, H) representing the radius theorem, every $h = h(p) \in H$ is a unit ball centered at p. We call such p a *reference point*. For every positive *scaling factor* $\lambda \in \mathbb{R}^+$ we let $\lambda h = \lambda h(p)$ be the λ-units ball centered at p and $\lambda H = \{\lambda h \mid h \in H\}$ be the set of λ-units balls with the same centers as the balls in H.

THEOREM 6.6. *Let $d \in \mathbb{N}$, and let H be a finite family of compact subsets in $\mathbb{R}^d$ with a reference point for each one of them. If, for every scaling factor $\lambda_0 \in \mathbb{R}^+$, $(\mathbb{R}^d, \lambda_0 H)$ is a Helly system with lexicographic Helly number l, and $(\mathbb{R}^d, \lambda_0\, Int(H))$ is a Helly system with Helly number k, where $Int(H) = \{Int(h) \mid h \in H\}$ is the family of the interiors of the sets in H, then $(\mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d, \bar{H})$ is a parameterized Helly system with Helly number $m = \max\{k, l\}$, where, for all $h \in H$ and for all $\lambda = (\lambda_0, x) \in \mathbb{R}^+ \times \mathbb{R}^d$, $h_\lambda = (\lambda_0 h \cap X_x) \cup (\lambda_0\, Int(h))$. Moreover, if ω is its natural objective function, then $(\bar{H}, \omega)$ is an LP-type problem of combinatorial dimension $m = \max\{k, l\}$.*

In Figure 6.1 below, $d = 2$, h is a rectangle of length 2 and width 1 centered at the origin, and $\lambda = (1, 0, 0)$. $h_{(1,0,0)}$ is a rectangle whose closure is h itself. The dashed line and the open circles do not belong to $h_{(1,0,0)} = (h \cap \mathbb{R}^2_{(0,0)}) \cup Int(h)$, while the solid line and the black point do.

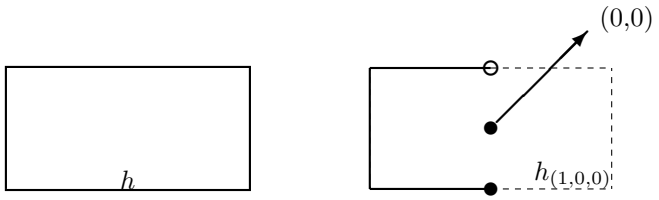


FIG. 6.1. *h and $h_{(1,0,0)}$.*

Proof. In order to prove that $(\mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d, \bar{H})$ is a parameterized Helly system with Helly number $m = \max\{k, l\}$, we need to show that $\{h_\lambda \mid \lambda \in \mathbb{R}^+ \times \mathbb{R}^d\}$ is a nested family and, for every $\lambda \in \mathbb{R}^+ \times \mathbb{R}^d$, $(\mathbb{R}^d, H_\lambda)$ is a Helly system with Helly number m.

Let $\alpha = (\lambda_0, x)$, $\beta = (\lambda_0', x') \in \mathbb{R}^+ \times \mathbb{R}^d$ be such that $\alpha <_L \beta$. If $\lambda_0 = \lambda_0'$, then $x <_L x'$, so $X_x \subset X_{x'}$, and, from the definition of h_λ, $h_\alpha \subseteq h_\beta$. Otherwise ($\lambda_0 < \lambda_0'$), $\lambda_0 h \subset \lambda_0' \, Int(h)$, so again $h_\alpha \subseteq h_\beta$. Hence $\{h_\lambda \mid \lambda \in \mathbb{R}^+ \times \mathbb{R}^d\}$ is a nested family.

We show now that, for every $\lambda = (\lambda_0, x) \in \mathbb{R}^+ \times \mathbb{R}^d$, $(\mathbb{R}^d, H_\lambda)$ is a Helly system with Helly number $m = \max\{k, l\}$. If every m elements in H_λ intersect in X_x, since $(\mathbb{R}^d, \lambda_0 H)$ is a Helly system with lexicographic Helly number $l \leq m$, then there is a point $x' \in X_x$ common to all of the sets in λH. Hence x' is common to all $h_\lambda \in H_\lambda$. If every m elements in H_λ intersect in $\mathbb{R}^d \setminus X_x$, then from the definition of h_λ every m elements in $\lambda_0 \, Int(H)$ intersect. Since $(\mathbb{R}^d, \lambda_0 \, Int(H))$ is a Helly system with Helly number $k \leq m$, all of the sets in $\lambda \, Int(H)$ have a point in common. Hence there is a point common to all $h_\lambda \in H_\lambda$. In this way we get that $(\mathbb{R}^d, H_\lambda)$ is a Helly system with Helly number m and $(\mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d, \bar{H})$ is a parameterized Helly system with Helly number m.

We will now apply Theorem 2.17 on the parameterized Helly system with Helly number m, $(\mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d, \bar{H})$. For this we need to show that the natural ground set objective function ω' meets the UMC. We observe that, due to the definition of h_λ, for every $\lambda_0 \in \mathbb{R}^+$, $\bar{h} \in \bar{H}$, and $x, y \in \mathbb{R}^d$

$$(y, \lambda_0, x) \in \bar{h} \rightarrow (y, \lambda_0, y) \in \bar{h} \tag{6.1}$$

holds. Second, we note that if $\lambda^* = (\lambda_0^*, x^*)$ is the value of the optimal solution over $\bar{G} \subseteq \bar{H}$, that is, $\omega(\bar{G}) = \omega'(x, \lambda^*) = \lambda^*$, then, for every point $(y, \lambda^*) \in \mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d$ realizing this value, there exists $h' \in H$ such that y lies on the boundary of $\lambda_0^* h' \cap X_x^*$. (Otherwise, y must be in $\cap_{h \in H} \lambda_0^* \, Int(h)$, and we can decrease λ_0^* slightly, say, to λ_0', and still have a nonempty intersection $(y, \lambda_0', x^*) \in \bigcap \bar{G}$, so $\omega(\bar{G}) \leq_L \omega'(y, \lambda_0', x^*) = (\lambda_0', x^*) <_L \lambda^*$ in contradiction to the optimality of λ^*.) Thus we have for every $y \in \mathbb{R}^d$

$$(y, \lambda_0^*, x^*) \in \bigcap \bar{G} \rightarrow y \leq_L x^*. \tag{6.2}$$

Suppose on the contrary that there exists $\bar{G} \subseteq \bar{H}$, with $\omega(\bar{G}) = \lambda^* = (\lambda_0^*, x^*)$, and there are two different points $y', y'' \in \mathbb{R}^d$ such that both $(y', \lambda^*), (y'', \lambda^*) \in \bigcap \bar{G}$ realize $\omega(\bar{G})$. Due to (6.2), $y', y'' \leq_L x^*$. Without loss of generality $y'' <_L y'$. Hence $y'' <_L x^*$, and (6.1) implies that $(y'', \lambda_0^*, y'') \in \bigcap \bar{G}$, so $\omega(\bar{G}) \leq_L (\lambda_0^*, y'') <_L (\lambda_0^*, x) = \lambda^*$ in contradiction to the optimality of λ^*. Hence ω' satisfies the UMC on $(\mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d, \bar{H})$, and, by Theorem 2.17, $(\bar{H}, \omega)$ is an LP-type problem of combinatorial dimension m. ∎

Considering once again the smallest enclosing ball problem, we note that, for every $\lambda_0 \in \mathbb{R}^+$, $(X, \lambda_0 H)$ ($X = \mathbb{R}^d$ and $\lambda_0 H$ is a family of balls of radius λ_0) is a Helly system with lexicographic Helly number $d+1$. Since Helly's theorem is valid for finite families of open convex sets, $(X, \lambda_0\, Int(H))$ is a Helly system with Helly number $d+1$. Applying the theorem above, we get that $(\bar{H}, \omega)$ is an LP-type problem of combinatorial dimension $d+1$, where the first coordinate (parameter) of ω is the radius of the smallest enclosing ball of the points in H, and the remaining parameters are its center location. We demonstrate the usage of Theorem 6.6 in the next section.

7. Solving the planar lexicographic rectilinear *p*-center problem. In the lexicographic rectilinear p-piercing *decision* problem (p-*lpiercing* decision problem, in short) we are given a finite set B of closed axis-parallel boxes in $\mathbb{R}^d$ and a p-tuple $A = (a_1, \ldots, a_p)$ of p points in $\mathbb{R}^d$, with $a_i \leq_L a_j$ for all $i < j$. We need to decide whether there exists a p-tuple $A' = (a'_1, a'_2, \ldots, a'_p)$ such that $\{a'_1, a'_2, \ldots, a'_p\}$ p-pierces B and $A' \leq_L A$. If such a p-tuple A' exists, we say that B is *A-p-pierceable* and call A' a *p-piercing vector of B*.

In the lexicographic rectilinear p-piercing *optimization* problem (p-*lpiercing* optimization problem, in short) we are given a finite set B of closed boxes in $\mathbb{R}^d$ with edges parallel to the coordinate axes and need to find the lexicographically least p-tuple A such that A p-pierces B. If no such p-tuple exists, we return a special symbol ∞.

The Helly-type theorem related to these problems is about the least $h_L(p)$ such that, for all A, B is A-p-pierceable if each $B' \subseteq B$, with $|B'| \leq h_L$, is A-p-pierceable.

THEOREM 7.1 (Theorem 2.7 in [20]). *Let B be a finite set of axis-parallel closed rectangles in the plane and $A = (a_1, \ldots, a_p)$ be a p-tuple of p points in $\mathbb{R}^d$, with $a_i \leq_L a_j$ for all $i < j$. For $p = 1, 2, 3$ the rectangles in B are A-p-pierceable if every subfamily $G \subset B$ of size at most $h_L(p)$ is A-p-pierceable, where $h_L(1) = 2$, $h_L(2) = 6$, and $16 \leq h_L(3) \leq 34$.*

Its corresponding nonlexicographic Helly-type theorem is the following.

THEOREM 7.2 (see [9]). *Let B be a finite set of axis-parallel rectangles in the plane such that all of the rectangles are either closed or open. For $p = 1, 2, 3$ the rectangles in B are p-pierceable if every subfamily $G \subset H$ of size at most $h(p)$ is p-pierceable, where $h(1) = 2$, $h(2) = 5$, and $h(3) = 16$.*

In this section we solve the planar lexicographic weighted p-center problem for $p = 1, 2, 3$ in randomized linear time by applying Theorem 7.2 on open rectangles, using its corresponding lexicographic version Theorem 7.1, and Theorem 6.6.

We start by defining the parameterized Helly system corresponding to our problem. Let the ground set of all possible p-centers be

$$X_p = \{(x_1, y_1, \ldots, x_p, y_p) \mid x_1, y_1, \ldots, x_p, y_p \in \mathbb{R}\} = \mathbb{R}^{2p},$$

where $(x_1, y_1), \ldots, (x_p, y_p)$ are the p centers. Let the range of the objective function be the radius, so $\Lambda = \mathbb{R}^+$.

We consider the $2p$-dimensional space X_p. For each reference point $h = h_j = (x_0, y_0) \in H$ we define $h_p = \cup_{i=1}^{p} h_p^i$, where

$$h_p^i = \left\{ (x_1, y_1, \ldots, x_p, y_p) \;\middle|\; |x_i - x_0| \leq \frac{1}{w_j};\ |y_i - y_0| \leq \frac{1}{w_j} \right\} \tag{7.1}$$

is the set of all points in X_p such that the ith center is at weighted distance at most 1 from h. We let $H_p = \{h_p \mid h \in H\}$. For every $\lambda_1 \in \mathbb{R}^+$ and $h = h_j \in H$ we define

$\lambda_1 h_p$ as h_p scaled by λ_1, that is, $\lambda_1 h_p = \cup_{i=1}^{p} \lambda_1 h_p^i$, where $\lambda_1 h_p^i = \{(x_1, y_1, \ldots, x_p, y_p) \mid |x_i - x_0| \leq \frac{\lambda_1}{w_j}; |y_i - y_0| \leq \frac{\lambda_1}{w_j}\}$.

Due to Theorem 7.1, for every scaling factor $\lambda_0 \in \mathbb{R}^+$, the set system $(X_p, \lambda_0 H_p)$ is a Helly system with lexicographic Helly number 2 (6, a constant bounded by 34) for $p = 1$ ($p = 2, 3$). Due to Theorem 7.2 (applied on open rectangles) $(X_p, \lambda_0\, Int(H_p))$ is a Helly system with Helly number 2 (5, 16) for $p = 1$ ($p = 2, 3$). Theorem 6.6 implies that $(X_p \times \mathbb{R}^+ \times X_p, \bar{H}_p)$, where for all $h_p \in H_p$ and for all $\lambda = (\lambda_0, x) \in \mathbb{R}^{2p+1}$, $h_{p\lambda} = (\lambda_0 h_p \cap X_x) \cup (\lambda_0\, Int(h_p))$ is a parameterized Helly system with Helly number 2 (6, a constant bounded by 34) for $p = 1$ ($p = 2, 3$). Moreover, if ω_p is its natural objective function, the theorem says that $(\bar{H}_p, \omega)$ is an LP-type problem of combinatorial dimension 2 (6, 34) for $p = 1$ ($p = 2, 3$).

THEOREM 7.3. *The lexicographic weighted planar p-center problem with an l_∞ norm is solvable in (randomized) linear time for $p = 1, 2, 3$.*

Proof. Until now we have shown that the lexicographic planar p-center problem with an l_∞ norm is an LP-type problem of dimension at most 2 (6, 34) for $p = 1$ ($p = 2, 3$). We solve this problem by using the LP-type randomized algorithms, such as the one of Sharir and Welzl (see section 2.2.2). In order to obtain a linear running time it remains to show how to implement the violation test and basis calculation primitives such that they run in constant time. We slightly change the structure of these two primitives: We implement the basis calculation primitives such that when called with input (B, h) it returns, in addition to a basis $B(B \cup \{h\})$ for $B \cup \{h\}$, also the value $\omega(B \cup \{h\})$ of the objective function on $B \cup \{h\}$ and the point $x(B \cup \{h\})$ which realizes this value (there is only such a point since the objective function is lexicographic). The input for the violation tests consists of $x(B)$ in addition to B (i.e., we call Violation$(B, h, x(B))$). The violation test primitive checks whether $x(B) \in \bar{h}_p(h)$. This is done in constant time since $\bar{h}_p(h)$ is of constant complexity. We implement the basis calculation primitive Basis(B, h) in constant time as follows. For any proper subset $B' \subset B \cup \{h\}$ we calculate explicitly $\omega(B')$ and the point $x(B')$ realizing this value. Then for every $h \in B \cup \{h\} \setminus B'$ we call Violation$(B', h, x(B'))$. B' is a basis for $B \cup \{h\}$ if and only if all of these calls return "false." ∎

We note that, since the optimal solution for the lexicographic planar p-center problem is an optimal solution for the nonlexicographic problem, we get an alternative solution to the one of [32]. We summarize as follows.

COROLLARY 7.4. *The planar p-center problem with an l_∞ norm is solvable in (randomized) linear time for $p = 1, 2, 3$.*

We note that the combinatorial dimension of the lexicographic problem is smaller than the combinatorial dimension given by [32] for the corresponding nonlexicographic problems (6 instead of 13 for the case $p = 2$ and 34 instead of 43 for the case $p = 3$).

8. Discrete Helly-type theorems and their relations to the DLP-type model.

8.1. DLP-type problems specialized to mathematical programming. In the DLP-type framework both D and S are sets of abstract objects, and the objective function applies to elements of $2^D \times 2^S$. We consider an extended version of mathematical programming which is a quadruple (X, D, S, ω'), where X is a *ground set* (usually $\mathbb{R}^d$), D is a set of d-elements, S is a set of s-elements (both of which are subsets of the ground set), and ω' is an objective function from X to some totally ordered set Λ. We call the elements of X points. For $G = (D', S') \in 2^D \times 2^S$ we write $\bigcap G$ for $(\bigcap D) \cap (\bigcup S)$. The points in $\bigcap (D, S)$ are called *feasible*. The goal is

to minimize ω' over the set of feasible points.

One can think of a discrete mathematical programming problem (X, D, S, ω') as a mathematical programming problem on a grid made by $\bigcup S$, that is, the mathematical programming problem $(X \cap (\bigcup S), D \cap (\bigcup S), \omega')$. However, our definition of a discrete mathematical programming problem enables us to solve fixed-dimensional DLP-type problems efficiently, as explained later.

To simplify our proofs later, we will make a few observations about the DLP-type framework specialized to mathematical programming.

DEFINITION 8.1. *Let (X, D, S, ω') be a discrete mathematical programming problem. For $G = (D', S') \in 2^D \times 2^S$, let $\omega(G) = \infty$ when $\bigcap G = \emptyset$ and $\omega(G) = \min\{\omega'(m) \mid m \in \bigcap G\}$ elsewhere. We call $\omega : 2^D \times 2^S \to \Lambda$ the* induced *subfamily objective function of (X, D, S, ω') and call the triple (D, S, ω) the* induced discrete abstract problem.

For instance, in the discrete 1-center problem on the real line we are given two finite sets of real numbers H_1 and H_2. We need to find a point $h \in H_2$ which minimizes the maximum distance between points in H_1 and h. We call this point a *center* and call the distance it realizes the *radius*. We formulate this problem as a discrete mathematical programming problem (X, D, S, ω'), where $X = \mathbb{R}^2$, D is the set of $\frac{\pi}{4}$ radians cones whose origins are the points of H_1, S is a set of vertical rays whose origins are the points of H_2, and, for all $(x, y) \in \mathbb{R}^2$, $\omega'(x, y) = y$. In Figure 8.1 we have an instance of the problem where $H_1 = \{5, 9\}$ (the black points) and $H_2 = \{4, 8\}$ (the white points). In the solution of this problem the center is 8, and the radius is 3. If the center is not restricted to be a point of H_2, the radius realized by choosing a center at 7 will be 2. In the next section we will discuss in detail other p-center problems such as the 1-center problem in $\mathbb{R}^d$ with either l_1 or l_∞ norm.

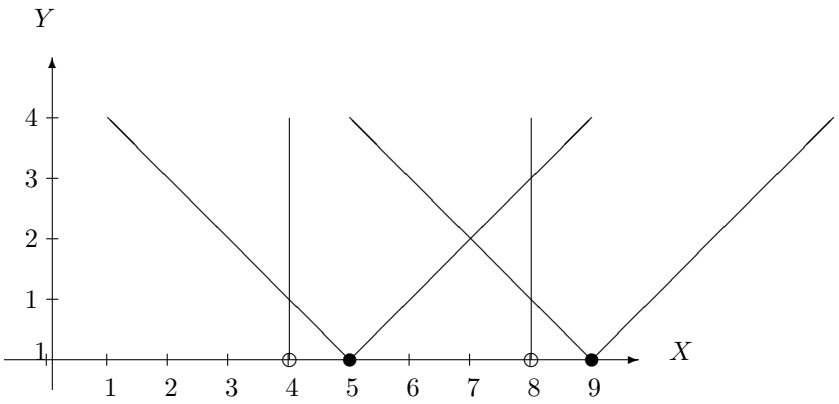


FIG. 8.1. *An instance of the general 1-center problem.*

Observation 8.2. Let (X, D, S, ω') be a discrete mathematical programming problem. The induced discrete abstract problem (D, S, ω) satisfies both monotonicity conditions of the DLP-type framework.

This follows from the fact that adding a d-element (i.e., a constraint) eliminates only feasible points, so the value of the minimum on the remaining feasible points cannot decrease. Adding an s-element (i.e., a relaxation) increases the set of feasible points, so the value of the minimum on the new enlarged set of feasible points cannot

increase.

Observation 8.3. Let (X, D, S, ω') be a discrete mathematical programming problem. Its induced discrete abstract problem (D, S, ω) is a 1-supply problem which satisfies both monotonicity conditions and the locality of supply condition.

Proof. In order to show that (D, S, ω) is a 1-supply problem, it is sufficient to show that for every feasible $G = (D', S') \in 2^D \times 2^S$ there exists $S'' \subseteq S$, with $|S''| = 1$ such that $\omega(G) = \omega(D', S'')$. Since (D, S, ω) is an induced discrete abstract problem, there exists $x \in \bigcap G$ such that $\omega(G) = \omega'(x)$. From the definition of $\bigcap G$, there is $h \in S'$ such that $x \in h$ so $x \in \bigcap(D', \{h\})$ and $\omega(D', \{h\}) = \omega(G)$. Hence we choose $S'' = \{h\}$.

By Observation 8.2, (D, S, ω) obeys both monotonicity conditions.

We now show that (D, S, ω) satisfies the locality of supply condition. Let $G = (D', S') \in 2^D \times 2^S$ be feasible, and let $S'' \subseteq S'$ such that $\omega(D', S') = \omega(D', S'')$. We need to show that, for every $h \in S$, $\omega(D', S' \cup \{h\}) < \omega(G)$ implies $\omega(D', S'' \cup \{h\}) < \omega(G)$. Since (D, S, ω) is a 1-supply problem, $\omega(D', S' \cup \{h\}) < \omega(G)$ only if $\omega(D', \{h\}) < \omega(G)$. From the monotonicity of supply condition we conclude that $\omega(D', S'' \cup \{h\}) \leq \omega(D', \{h\}) < \omega(G)$. ∎

Definition 8.4. *Let (X, D, S, ω') be a discrete mathematical programming problem, and let (D, S, ω) be a discrete abstract problem, where ω is the objective function induced by ω'. If, for all $G = (D', S') \in 2^D \times 2^S$, $|\{x \in \bigcap G \mid \omega'(x) = \omega(G)\}| = 1$, we say that ω' satisfies the* UMC.

This definition says that every subfamily not only has a minimum but that this minimum is achieved by a unique point. There is one simple sufficient condition to satisfy the UMC.

Observation 8.5. If $\omega'(x) \neq \omega'(y)$ for any two distinct points $x, y \in X$, then ω' satisfies the UMC.

Lemma 8.6. *Let (X, D, S, ω') be a discrete mathematical programming problem. If its ground set function ω' meets the UMC on (X, D, S), then its induced abstract problem (D, S, ω) is a 1 s-dimensional DLP-type problem.*

Proof. By Observation 8.3, (X, D, S) is a 1-supply problem which satisfies both monotonicity conditions as well as the locality of supply condition.

We prove now that the locality of demand condition is satisfied. Let $G = (D', S') \in 2^D \times 2^S$ be bounded, and let $D'' \subseteq D'$ such that $\omega(G) = \omega(D'', S')$. We need to show that for all $h \in D$, $\omega(D' \cup \{h\}, S') > \omega(G) \rightarrow \omega(D'' \cup \{h\}, S') > \omega(G)$. Due to the UMC, the value $\omega(D', S') = \omega(D'', S')$ is achieved at a single point $x \in X$. This means that $\omega(D' \cup \{h\}, S') > \omega(G)$ only if $x \notin h$, in which case $\omega(D'' \cup \{h\}, S') > \omega(G)$, so the locality of demand condition is satisfied, and (D, S, ω) is a DLP-type problem. By Lemma 4.14 it is 1 s-dimensional. ∎

We concentrate for a moment on lexicographic integer programming (lex IP, for short) in $\mathbb{Z}^d$. The corresponding discrete mathematical programming formulation is $(\mathbb{Z}^d, D, S, \omega')$, where D is a finite set of half-spaces in $\mathbb{Z}^d$, S is the (exponentially large) set of the integer lattice points inside a "bounding box" around the problem, and ω' is defined for every $x \in \mathbb{Z}^d$ as $\omega'(x) = x$. Since S is finite and ω' satisfies the UMC, we get from Lemma 8.6 that (D, S, ω) is a 1 s-dimensional DLP-type problem. It remains to consider its d-dimension. Alternatively, we consider the combinatorial dimension of its induced LP-type problem (D, α). Suppose its combinatorial dimension is k and that the optimal value is $\alpha(D) = x^*$. We will first show that $k \leq 2^d$. Suppose on the contrary that $k > 2^d$. This means that if B is a basis for D, then every proper subset of B, $B' \subset B$ has $\alpha(B') <_L x^*$. Let $x^{\max} = \max\{\alpha(B') \mid B' \subset B\}$ be the maximal value of α on proper subsets of B. Since B is finite, $x^{\max}$ is well-defined, and

$x^{\max} <_L x^*$. Since the lexicographic version of Theorem 1.3 has Helly number 2^d (see Theorem 2.5 in [20]), applying it on B and $x^{\max}$ implies that the half-spaces in B have a common point which is not lexicographically greater than $x^{\max}$, in contradiction.

We now give a lower bound of $2^d - 1$ on the combinatorial dimension k of lex IP. Theorem 8.7 below applied on lex IP tells us that the special case of the lexicographic version of Theorem 1.3 over half-spaces has a Helly number of at most $k + 1$. Since it is known (see section 2 in [20]) that this special case has Helly number 2^d, we get that $2^d \leq k + 1$ as needed.

8.2. (Nonlexicographic) discrete case. We first show that there is a discrete Helly theorem corresponding to the constraint set of every fixed-dimensional DLP-type problem.

THEOREM 8.7. *Let (D, S, ω) be a (k_D, k_S)-dimensional DLP-type problem. For every $\lambda \in \Lambda$, $H = (D, S)$ has the property $\omega(H) \leq \lambda$ if and only if every $B_D \subseteq D$, with $|B_D| \leq k_D + 1$, has the property $\omega(B_D, S) \leq \lambda$. Moreover, H has the property $\omega(H) \geq \lambda$ if and only if every $B_S \subseteq S$, with $|B_S| \leq k_S + 1$, has the property $\omega(D, B_S) \geq \lambda$.*

Proof. We prove the first part of the theorem. The proof of the second part of the theorem is analogous. Let $\omega(H) \leq \lambda$. By the monotonicity of demand condition, $\omega(B_D, S) \leq \omega(H) \leq \lambda$. Going in the other direction, H must contain a basis $B = (B_D, B_S)$, with $|B_D| \leq k_D + 1$, and $\omega(B_D, S) = \omega(H)$ (if H is feasible, $|B_D| \leq k_D$; otherwise, every subset of B_D is feasible, so $|B_D| \leq k_D + 1$). So if every subfamily (B_D, S), with $|B_D| \leq k_D + 1$, has $\omega(B_D, S) \leq \lambda$, then $\omega(H) = \omega(B_D, S) \leq \lambda$. ∎

We next show how to get fixed-dimensional DLP-type problems from discrete Helly-type problems.

We first "discretize" set systems and Helly systems. A *discrete set system* is a triple (X, D, S), where X is a set and D, S are families of subsets of X. A discrete set system (X, D, S) is a *discrete Helly system* if there exists a finite integer k such that the intersection of every k or less d-elements of D has a common element in $\bigcup S$ implies that $\bigcap D \cap (\bigcup S) \neq \emptyset$. Let $(X \times \Lambda, \bar{D}, \bar{S})$ be a discrete set system, where Λ is a totally ordered set which contains a maximal element ∞. For every $\lambda \in \Lambda$ and $\bar{h} \in \bar{D} \cup \bar{S}$, we write $h_\lambda = \{x \in X \mid \exists \nu \leq \lambda \text{ s.t. } (x, \nu) \in \bar{h}\}$ for the projection into X of the part of $\bar{h}$ with Λ-coordinate no greater than λ. Also, for $G \in 2^D \times 2^S$, we write G_λ as a shorthand for $\{h_\lambda \mid \bar{h} \in \bar{G}\}$.

We next discretize parameterized Helly systems.

DEFINITION 8.8. *A discrete set system $(X \times \Lambda, \bar{D}, \bar{S})$ is a* discrete parameterized Helly system *with Helly number k, when*

1. *$\{h_\lambda \mid \lambda \in \Lambda\}$ is a nested family for all $\bar{h} \in \bar{D} \cup \bar{S}$, and*
2. *$(X, D_\lambda, S_\lambda)$ is a discrete Helly system, with Helly number k for all λ.*

We say that $\bar{G} = (\bar{D}', \bar{S}') \in 2^{\bar{D}} \times 2^{\bar{S}}$ *intersects at* λ if $\bigcap D'_\lambda \cap (\bigcup S'_\lambda) \neq \emptyset$. $\omega(\bar{D}', \bar{S}')$ is then the least value in Λ at which $\bar{G} = (\bar{D}', \bar{S}')$ intersects, i.e., $\omega(\bar{D}', \bar{S}') = \lambda^* = \inf\{\lambda \mid \bigcap D'_\lambda \cap (\bigcup S'_\lambda) \neq \emptyset\}$, and $\omega(\bar{D}', \bar{S}') = \infty$ if $\bar{G}$ fails to intersect at all $\lambda \in \Lambda$.

Figure 8.2 is a schematic diagram of a discrete parameterized Helly system. The whole stack represents $X \times \Lambda$, each of the pyramids represents a set $\bar{h} \in \bar{D}$, and each of the vertical lines represents a set $\bar{h} \in \bar{S}$. Each $\bar{h}$ is a subset of $X \times \Lambda$. Since all of the $\bar{h}$ are indexed with respect to Λ, the cross section at λ (represented by one of the planes) is equivalent to the discrete Helly system $(X, D_\lambda, S_\lambda)$ corresponding to Theorem 1.4. The discrete parameterized Helly system drawn in this figure is related to the discrete weighted 1-center problem with an l_∞ norm, which we solve in the next section. $\omega(\bar{D}, \bar{S})$ is the smallest value in Λ at which the intersection of the pyramids

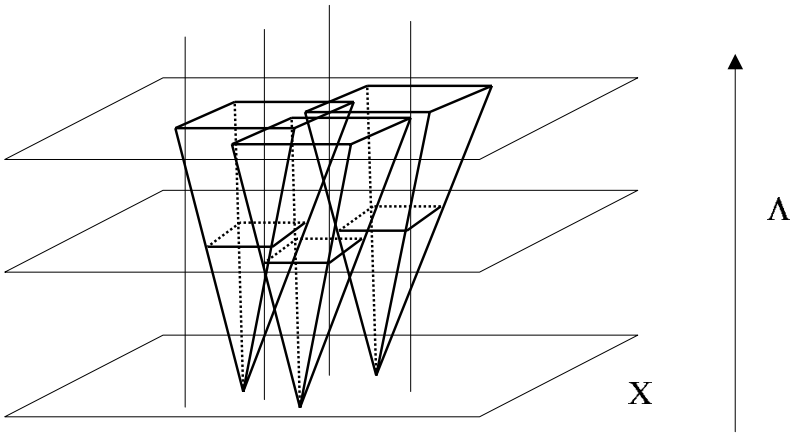


FIG. 8.2. *A discrete parameterized Helly system.*

in $\bar{D}$ "touches" a vertical line from $\bar{S}$.

We extend the main theorem in [2] to the discrete case and get the following.

THEOREM 8.9. *Let $(X \times \Lambda, \bar{D}, \bar{S})$ be a parameterized discrete Helly system with Helly number k, a natural ground set function ω', and a natural objective function ω. If ω' meets the UMC, then $(\bar{D}, \bar{S}, \omega)$ is a DLP-type problem of combinatorial dimension $(k, 1)$.*

Proof. Since ω is induced by the natural ground set objective function ω' on the space $X \times \Lambda$, and since ω' meets the UMC on $(X \times \Lambda, \bar{D}, \bar{S}, \omega')$ (Definition 8.4), by Lemma 8.6 $(\bar{D}, \bar{S}, \omega)$ is a 1 s-dimensional DLP-type problem.

It remains to prove that $(\bar{D}, \bar{S}, \omega)$ is k-d-dimensional. Consider any feasible $\bar{G} = (\bar{D}', \bar{S}') \in 2^{\bar{D}} \times 2^{\bar{S}}$ and a basis $\bar{B} = (\bar{B}_D, \bar{B}_S)$ for $\bar{G}$. Let $(\bar{D}', \alpha_{S'})$ be the induced LP-type problem of $(\bar{D}', \bar{S}', \omega)$. $\bar{B}_D$ is then a basis for $\bar{D}'$. We need to prove that $|\bar{B}_D| \leq k$. From the minimality of a basis we get that, for any $\bar{h} \in \bar{B}_D$, $\alpha_{S'}(\bar{B}_D \setminus \{\bar{h}\}) < \alpha_{S'}(\bar{B}_D)$. Let $\lambda^{\max} = \max\{\alpha_{S'}(\bar{B}_D \setminus \{\bar{h}\}) \mid \bar{h} \in \bar{B}_D\}$. Since $\bar{D}'$ is finite, so is $\bar{B}_D$, and this maximum is guaranteed to exist.

The basis $\bar{B}$ does not intersect at $\lambda^{\max}$, but for any $\bar{h} \in \bar{B}_D$, $\alpha_{S'}(\bar{B}_D \setminus \{\bar{h}\}) \leq \lambda^{\max}$, which means that $(\bar{B}_D \setminus \{\bar{h}\}, \bar{S}')$ intersects at $\lambda^{\max}$. Since $(X, D_{\lambda^{\max}}, S_{\lambda^{\max}})$ is a discrete Helly system with Helly number k, $\bar{B}_D$ must contain some subfamily $\bar{A}$, with $|\bar{A}| \leq k$, such that $(\bar{A}, \bar{S}')$ does not intersect at $\lambda^{\max}$. Every $\bar{h} \in \bar{B}_D$ must be in $\bar{A}$, since otherwise it would be the case that $\bar{A} \in (\bar{B}_D \setminus \{\bar{h}\})$ for some $\bar{h}$. This cannot be, because $(\bar{A}, \bar{S}')$ does not intersect at $\lambda^{\max}$, while every $(\bar{B}_D \setminus \{\bar{h}\}, \bar{S}')$ does. Therefore $\bar{B}_D = \bar{A}$ and $|\bar{B}_D| \leq k$. ∎

8.3. Lexicographic-discrete case. In this rather technical section we discretize the results in section 6. We start by "lexifying" discrete Helly systems.

DEFINITION 8.10. *A discrete Helly system with lexicographic Helly number ld is a discrete set system (X, D, S) such that, for every $x \in X$, $(X, \{d \cap X_x \mid d \in D\}, \{s \cap X_x \mid s \in S\})$ is a discrete Helly system with Helly number ld.*

This means that for every $x \in X$, whenever every ld or less elements of D have a common point in S which is not lgreater than x, we get that all elements of D have a common point in S which is not lgreater than x.

We next discretize Theorem 2.18. Let $(X \times \Lambda, \bar{D}, \bar{S})$ be a parameterized discrete Helly system with Helly number k and natural objective function ω. For all $\lambda \in \Lambda$, we assume a function $\nu_\lambda : 2^{D_\lambda} \times 2^{D_\lambda} \to \Lambda'$, where Λ' is a totally ordered set containing

a maximal element ∞ such that $(D_\lambda, S_\lambda, \nu_\lambda)$ is a DLP-type problem of d-dimension at most d and s-dimension 1. The functions ν_λ may themselves be lexicographic. Similarly to [2], we impose a lexicographic order on $\Lambda \times \Lambda'$ with $(\lambda, \kappa) > (\lambda', \kappa')$ if $\lambda > \lambda'$ or if $\lambda = \lambda'$ and $\kappa > \kappa'$. We define a lexicographic objective function $\nu : 2^{\bar{D}} \times 2^{\bar{S}} \to \Lambda \times \Lambda'$ in terms of ω and the functions ν_λ as seen in the following.

THEOREM 8.11. *Let Λ' be a totally ordered set. Let $(X \times \Lambda, \bar{D}, \bar{S})$ be a parameterized discrete Helly system with Helly number k and natural objective function ω. If, for all λ, $(D_\lambda, S_\lambda, \nu_\lambda)$ is a DLP-type problem of combinatorial dimension $(d, 1)$, where $\nu_\lambda : 2^{D_\lambda} \times 2^{S_\lambda} \to \Lambda'$, then $(\bar{D}, \bar{S}, \nu)$ is a DLP-type problem of d-dimension $\leq k+d$ and s-dimension 1, where $\nu : 2^{\bar{D}} \times 2^{\bar{S}} \to \Lambda \times \Lambda'$ is defined as $\nu(\bar{G}) = (\omega(\bar{G}), \nu_{\omega(\bar{G})}(G_{\omega(\bar{G})}))$ for all $\bar{G} \subseteq \bar{D} \times \bar{S}$.*

Proof. Due to Observation 8.2, $(\bar{D}, \bar{S}, \omega)$ obeys both monotonicity conditions. For every λ, $(D_\lambda, S_\lambda, \nu_\lambda)$ obeys both monotonicity conditions. Hence, since ν is a composition of monotone functions, we get that $(\bar{D}, \bar{S}, \nu)$ obeys both monotonicity conditions as well. We next show that $(\bar{D}, \bar{S}, \nu)$ obeys both locality conditions.

Consider $\bar{D}'' \subseteq \bar{D}' \subseteq \bar{D}$ and $\bar{S}' \subseteq \bar{S}$, with $\nu(\bar{D}'', \bar{S}') = \nu(\bar{D}', \bar{S}') = (\lambda^*, \kappa^*)$ and $\nu(\bar{D}' \cup \{\bar{h}\}, \bar{S}') = (\lambda, \kappa) > \nu(\bar{D}', \bar{S}')$. We must have either $\lambda > \lambda^*$ or $\kappa > \kappa^*$. If $\lambda > \lambda^*$, by the definition of ω, $\nu_{\lambda^*}(D'_{\lambda^*} \cup \{h_{\lambda^*}\}, S'_{\lambda^*}) = \infty$, so $\nu_{\lambda^*}(D'_{\lambda^*} \cup \{h_{\lambda^*}\}, S'_{\lambda^*}) > \nu_{\lambda^*}(D'_{\lambda^*}, S'_{\lambda^*})$. Otherwise, $\kappa > \kappa^*$, that is, $\nu_{\lambda^*}(D'_{\lambda^*} \cup \{h_{\lambda^*}\}, S'_{\lambda^*}) > \nu_{\lambda^*}(D'_{\lambda^*}, S'_{\lambda^*})$. In either case, by the locality of demand condition on ν_{λ^*}, $\nu_{\lambda^*}(D''_{\lambda^*} \cup \{h_{\lambda^*}\}, S'_{\lambda^*}) > \nu_{\lambda^*}(D''_{\lambda^*}, S'_{\lambda^*})$ and $\nu(\bar{D}'' \cup \{\bar{h}\}, \bar{S}') > \nu(\bar{D}'', \bar{S}')$. So the lexicographic function ν also satisfies the locality of demand condition.

We now consider the locality of supply condition. We note that, due to Observation 8.3, $(\bar{D}, \bar{S}, \omega)$ is a 1-supply problem which meets the locality of supply condition. For every λ, $(D_\lambda, S_\lambda, \nu_\lambda)$ obeys both locality conditions. We will show that, since ν is a composition of functions satisfying the locality of supply condition, $(\bar{D}, \bar{S}, \nu)$ meets the locality of supply condition. Let $\bar{S}'' \subseteq \bar{S}' \subseteq \bar{S}$ and $\bar{D}' \subseteq \bar{D}$, with $\nu(\bar{D}', \bar{S}'') = \nu(\bar{D}', \bar{S}') = (\lambda^*, \kappa^*)$ and $\nu(\bar{D}', \bar{S}' \cup \{\bar{h}\}) = (\lambda, \kappa) < \nu(\bar{D}', \bar{S}')$. We need to show that $\nu(\bar{D}', \bar{S}'' \cup \{\bar{h}\}) < (\lambda^*, \kappa^*)$. Clearly, we must have either $\lambda < \lambda^*$ or $\kappa < \kappa^*$. If $\lambda < \lambda^*$, since $(\bar{D}, \bar{S}, \omega)$ obeys the locality condition of supply, $\omega(\bar{D}', \bar{S}'' \cup \{\bar{h}\}) < \lambda^*$, so $\nu(\bar{D}', \bar{S}'' \cup \{\bar{h}\}) < (\lambda^*, \kappa^*)$, as needed. Otherwise, $\kappa < \kappa^*$, that is, $\nu_{\lambda^*}(D'_{\lambda^*}, S'_{\lambda^*} \cup \{h_{\lambda^*}\}) < \nu_{\lambda^*}(D'_{\lambda^*}, S'_{\lambda^*})$. In this case, by the locality of supply condition on ν_{λ^*}, $\nu_{\lambda^*}(D'_{\lambda^*}, S''_{\lambda^*} \cup \{h_{\lambda^*}\}) < \nu_{\lambda^*}(D'_{\lambda^*}, S''_{\lambda^*})$ and again $\nu(\bar{D}', \bar{S}'' \cup \{\bar{h}\}) < \nu(\bar{D}', \bar{S}'')$, as needed.

We now consider the combinatorial s-dimension. It is sufficient to show that, for every feasible $(\bar{D}', \bar{S}') \in 2^{\bar{D}} \times 2^{\bar{S}}$ and every basis $\bar{B} = (\bar{BD}, \bar{BS})$ for $(\bar{D}', \bar{S}')$, $|\bar{BS}| = 1$. Since $\bar{B}$ is a basis for $(\bar{D}', \bar{S}')$, we have $\nu(\bar{B}) = \nu(\bar{D}', \bar{S}') = (\lambda^*, \kappa^*)$. Let $B_{\lambda^*} = (BD'_{\lambda^*}, BS'_{\lambda^*})$ be a basis for $(D'_{\lambda^*}, S'_{\lambda^*})$ in the DLP-type problem $(D'_{\lambda^*}, S'_{\lambda^*}, \nu_{\lambda^*})$. Since the s-dimension of $(D'_{\lambda^*}, S'_{\lambda^*}, \nu_{\lambda^*})$ is 1, there is $\bar{h}' \in \bar{S}'$ such that $BS'_{\lambda^*} = \{h'_{\lambda^*}\}$. Let $\bar{S}'' \subseteq \bar{S}'$ be the set of all such $\bar{h}'$. Since $(\bar{D}', \bar{S}', \omega)$ is an induced discrete abstract problem, there is a feasible point $x \in \bigcap(\bar{D}', \bar{S}')$ such that $\omega(\bar{D}', \bar{S}') = \omega'(x) = \lambda^*$, $x \in h'_{\lambda^*}$ (so $x \in \bar{h}'$), and

$$\nu(\bar{D}', \bar{S}') = \nu(\bar{D}', \{\bar{h}'\}). \tag{8.1}$$

Let BD''_{λ^*} be a basis for D_{λ^*} in the induced LP-type problem of $(D_{\lambda^*}, S_{\lambda^*}, \nu_{\lambda^*})$ such that $BD''_{\lambda^*} \subseteq BD_{\lambda^*}$. It is possible to choose such a basis since $\nu_{\lambda^*}(BD_{\lambda^*}, S'_{\lambda^*}) = \kappa^*$. Similarly, let BS''_{λ^*} be a basis for S_{λ^*} in the induced dual LP-type problem of $(D_{\lambda^*}, S_{\lambda^*}, \nu_{\lambda^*})$ such that $BS''_{\lambda^*} \subseteq BS_{\lambda^*}$. It is possible to choose such a basis since $\nu_{\lambda^*}(D'_{\lambda^*}, BS_{\lambda^*}) = \kappa^*$. Due to the definition of a basis (Definition 4.4), $(BD''_{\lambda^*}, BS''_{\lambda^*})$

is a basis for $(D'_{\lambda^*}, S'_{\lambda^*})$. Hence $\bar{S}'' \cap \bar{BS}'' \neq \emptyset$, and consequently $\bar{S}'' \cap \bar{BS} \neq \emptyset$, so there exists

$$\bar{h}' \in \bar{S}'' \cap \bar{BS}. \tag{8.2}$$

We claim that $\bar{BS} = \{\bar{h}'\}$. Since $(\bar{BD}, \bar{BS})$ is a basis for $(\bar{D}', \bar{S}')$ we get

$$\nu(\bar{D}', \bar{S}') = \nu(\bar{D}', \bar{BS}). \tag{8.3}$$

Combining (8.1) and (8.3) together implies that $\nu(\bar{D}', \bar{BS}) = \nu(\bar{D}', \{\bar{h}'\})$. Since $\bar{BS}$ is a basis for the induced dual LP-type problem $(\bar{S}', \beta)$, the last equality implies that $\bar{BS} = \{\bar{h}'\}$, so $(\bar{D}, \bar{S}, \nu)$ has s-dimension 1.

Finally, we consider the combinatorial d-dimension. We note that, since $\bar{B}$ is a basis, $\nu(\bar{BD} \setminus \{h\}, \bar{S}') = (\lambda, \kappa) < \nu(\bar{B}) = (\lambda^*, \kappa^*)$ for any $\bar{h} \in \bar{BD}$. Let the subset $\bar{B}_1 = \{\bar{h} \in \bar{BD} \mid \nu(\bar{BD} \setminus \{\bar{h}\}, \bar{S}') = (\lambda, \kappa) \text{ and } \lambda < \lambda^*\}$. Since the d-dimension of $(B_{\lambda^*}, \nu_{\lambda^*})$ is d, $\bar{BD} \setminus \bar{B}_1$ contains at most d constraints. If $\bar{B}_1 = \emptyset$, we are done. Otherwise, we let

$$\lambda^{\max} = \max\{\lambda \mid \nu(\bar{BD} \setminus \{\bar{h}\}, \bar{S}') = (\lambda, \kappa),\ \bar{h} \in \bar{B}_1\}.$$

Since $\lambda^{\max} < \lambda^*$, $\bar{BD}$ fails to intersect with S' at $\lambda^{\max}$ and hence must contain a set $\bar{A}$ of size $\leq k$ that also fails to intersect. Every $\bar{h} \in \bar{B}_1$ must also be in $\bar{A}$, since $\bar{BD} \setminus \{\bar{h}\}$ intersects with S' at $\lambda^{\max}$ and $\bar{A}$ does not, so $\bar{A} \not\subseteq \bar{BD} \setminus \{\bar{h}\}$. So $|\bar{B}_1| \leq |\bar{A}| \leq k$ and $|\bar{BD}| \leq k + d$. □

In order to get DLP-type problems from lexicographic-discrete Helly theorems, we impose lexicographic order on the ground set X and parameterize the discrete Helly system (X, D, S) with lexicographic Helly number ld (see Definition 8.10) in the following way (recall that $X_x = \{x' \in X \mid x' \leq_L x\}$).

Definition 8.12. *A discrete set system $(X \times X, \bar{D}, \bar{S})$ is a* parameterized discrete Helly system with lexicographic Helly number *ld if there exists a discrete Helly system with lexicographic Helly number ld, (X, D, S), such that for all $h \in D$, $\bar{h} = \{(y, x) \mid x \in X,\ y \in h \cap X_x\}$, $\bar{D} = \{\bar{h} \mid h \in D\}$, and for all $h \in S$, $\bar{h} = \{(y, x) \mid x \in X,\ y \in h\}$, $\bar{S} = \{\bar{h} \mid h \in S\}$.*

From the definitions it is easy to verify the following.

Observation 8.13. Let $(X \times X, \bar{D}, \bar{S})$ be a parameterized discrete Helly system with lexicographic Helly number ld. For every $x, y \in X$ and $\bar{h} \in \bar{D} \cup \bar{S}$ the following attributes hold:

1. $\{h_x \mid x \in X\}$ is a nested family for all $\bar{h} \in \bar{D} \cup \bar{S}$.
2. (X, D_x, S_x) is a discrete Helly system with lexicographic Helly number ld.
3. $(X \times X, \bar{D}, \bar{S})$ is a parameterized discrete Helly system with Helly number ld.
4. $(y, x) \in \bar{h} \rightarrow (y, y) \in \bar{h}$.
5. $(y, x) \in \bar{h} \rightarrow y \leq_L x$.

We give the discrete versions of Theorems 6.4 and 6.5 and prove them similarly to the way we proved the continuous versions.

Theorem 8.14. *Let $(X \times X, \bar{D}, \bar{S})$ be a parameterized discrete Helly system with lexicographic Helly number ld and ω be its natural objective function. Then $(\bar{D}, \bar{S}, \omega)$ is a DLP-type problem of combinatorial dimension $(ld, 1)$.*

Proof. We show that all of the conditions of Theorem 8.9 are satisfied. Due to attribute 3 in Observation 8.13, $(X \times X, \bar{D}, \bar{S})$ is a parameterized discrete Helly system with Helly number at most ld. It remains to show that the natural objective function ω meets the UMC. Suppose on the contrary that there is $\bar{G} = (\bar{D}', \bar{S}') \in 2^{\bar{D}} \times 2^{\bar{S}}$,

with $\omega(\bar{G}) = x$, such that there are two different points $x', x'' \in X \cap (\bigcup S')$ so that both $(x', x), (x'', x) \in \bigcap \bar{G}$ realize $\omega(\bar{G})$. Due to attribute 5 in Observation 8.13, $x', x'' \leq_L x$. Without loss of generality $x'' <_L x'$. Hence $x'' <_L x$, and from attribute 4 in Observation 8.13 we get that $(x'', x'') \in \bigcap \bar{G}$, so $\omega(\bar{G}) \leq_L x'' <_L x$ in contradiction. □

THEOREM 8.15. *Let $(X \times \Lambda, \bar{D}, \bar{S})$ be a parameterized discrete Helly system with Helly number k and natural objective function ω. If, for every $\lambda \in \Lambda$, $(X, D_\lambda, S_\lambda)$ is a discrete Helly system with lexicographic Helly number ld, then there is a function $\nu : 2^{\bar{D}} \times 2^{\bar{S}} \to \Lambda \times X$ such that for all $\bar{G} \in 2^{\bar{D}} \times 2^{\bar{S}}$ the first part of $\nu(\bar{G})$ is $\omega(\bar{G})$ and $(\bar{D}, \bar{S}, \nu)$ is a DLP-type problem of d-dimension $\leq k + l$ and s-dimension 1.*

Proof. For every $\lambda \in \Lambda$ we parameterize the discrete Helly system $(X, D_\lambda, S_\lambda)$ such that $(X \times X, \bar{D_\lambda}, \bar{S_\lambda})$ is a parameterized discrete Helly system with lexicographic Helly number ld. If its natural objective function ν_λ is not well-defined, we symbolically compactify the space X by representing points at infinity. Due to Theorem 8.14 the resulted discrete abstract problem $(D_\lambda, S_\lambda, \nu_\lambda)$ is a DLP-type problem of combinatorial dimension $(ld, 1)$. We conclude our proof by using Theorem 8.11. □

It is possible to bound the combinatorial dimension of the resulting LP-type problem further by using the following discrete version of Theorem 6.6 (whose proof is similar to the one of Theorem 6.6).

THEOREM 8.16. *Let $d \in \mathbb{N}$, D be a finite family of compact subsets in $\mathbb{R}^d$ and S be a finite family of closed subsets in $\mathbb{R}^d$. If, for every scaling factor $\lambda_0 \in \mathbb{R}^+$, $(\mathbb{R}^d, \lambda_0 D, S)$ is a discrete Helly system with lexicographic Helly number l, and $(\mathbb{R}^d, \lambda_0\, Int(D), S)$ is a discrete Helly system with Helly number k, where $Int(D) = \{Int(h) \mid h \in D\}$ is the family of the interiors of the sets in D, then $(\mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^d, \bar{D}, \bar{S})$ is a parameterized discrete Helly system with Helly number $m = \max\{k, l\}$, where, for all $h \in D$ and for all $\lambda = (\lambda_0, x) \in \mathbb{R}^+ \times \mathbb{R}^d$, $h_\lambda = (\lambda_0 h \cap X_x) \cup (\lambda_0\, Int(h))$, and for all $h \in S$ and for all $\lambda = (\lambda_0, x) \in \mathbb{R}^{d+1}$, $h_\lambda = h$. Moreover, if ω is its natural objective function, then $(\bar{D}, \bar{S}, \omega)$ is a DLP-type problem of combinatorial dimension $(m, 1)$.*

9. Solving the discrete weighted 1-center problem in $\mathbb{R}^d$ with either l_1 or l_∞ norm. In this section we show how to solve the discrete weighted 1-center problem in $\mathbb{R}^d$ with an l_∞ norm (1-center problem, in short) in linear time by formulating it as a fixed-dimensional DLP-type problem which satisfies the VC.

Given an instance D, S, W of the 1-center problem, for every $G = (D', S') \in 2^D \times 2^S$ let $r(D', S')$ be the optimal radius of the 1-center problem on D', S', W, realized by making $s^*(D', S') \in S'$ the center.

Considering the set of boxes $r(D', S')D' = \{r(D', S')d_i \mid d_i \in D'\}$, where $r(D', S')d_i$ is the box with center at d_i and radius $\frac{r(D', S')}{w_i}$, we note that $s^*(D', S')$ intersects all of the boxes of $r(D', S')D'$. The proof of Theorem 1.5 applied on the set of boxes $r(D', S')D'$ and the set of points S' tells us that the following $2d$ boxes "define" the optimal solution:

For every $i = 1, \ldots, d$, let $L_i(D', S') \in D'$ be a d-element $d_{j(i)} \in D'$ such that the projection of box $r(D', S')d_{j(i)}$ on the ith coordinate results in an interval $[l_i, r_i]$ with the smallest r_i. Let $l_i(D', S') \in \mathbb{R}$ be the right end point of the projection of $r(D', S')d_{j(i)}$ on the ith coordinate.

For every $i = 1, \ldots, d$, let $G_i(D', S') \in D'$ be a d-element $d_{j(i)} \in D'$ such that the projection of box $r(D', S')d_{j(i)}$ on the ith coordinate results in an interval $[l'_i, r'_i]$ with the greatest l'_i. Let $g_i(D', S') \in \mathbb{R}$ be the left end point of the projection of $r(D', S')d_{j(i)}$ on the ith coordinate.

We also define $C(D', S') \in S'$ to be the lexicographically smallest optimal center. We let the range of the objective function be $\mathbb{R}^+ \times \mathbb{R}^{3d}$ and define the objective function to be

$$\omega(D', S') = (r(D', S'), C(D', S'), l_1(D', S'), -g_1(D', S'), \ldots, l_d(D', S'), -g_d(D', S')).$$

Clearly (D, S, ω) is a discrete abstract problem.

For every $(D', S') \in 2^D \times 2^S$ we let *Feasible*(D', S') denote the set of points that intersect all of the boxes $r(D', S')D'$. From the definitions of the variables and the optimality of the solution we get the following.

Observation 9.1. Let D, S, W be an instance of the discrete weighted 1-center problem in $\mathbb{R}^d$ with an l_∞ norm. For every $(D', S') \in 2^D \times 2^S$, *Feasible*$(D', S')$ is the minimal axis-parallel box containing the 2 points $(g_1(D', S'), \ldots, g_d(D', S'))$ and $(l_1(D', S'), \ldots, l_d(D', S'))$. Furthermore, $C(D', S')$ lies on its boundary.

We show now that (D, S, ω) is a $(2d, 1)$-dimensional DLP-type problem. (D, S, ω) obeys the monotonicity of demand condition since adding a new element h to D' cannot lexicographically decrease the value, i.e., $\omega(D', S') \leq_L \omega(D' \cup \{h\}, S)$. Similarly, (D, S, ω) obeys the monotonicity of supply condition since adding a new point h cannot lexicographically increase the objective function value.

We now show that (D, S, ω) obeys both locality conditions and that it obeys the VC. Let $G = (D', S') \in 2^D \times 2^S$ and $F = (D'', S'') \in 2^{D'} \times 2^{S'}$ be such that $\omega(G) =_L \omega(F)$ (so due to Observation 9.1 *Feasible*(G) = *Feasible*(F)). It suffices to show that the following 3 properties hold:

1. For all $h \in S$, $\omega(D', S' \cup \{h\}) <_L \omega(G)$ if and only if $\omega(D'', S'' \cup \{h\}) <_L \omega(F)$.
2. For all $h \in D$, $\omega(D' \cup \{h\}, S') >_L \omega(G) \rightarrow \omega(D'' \cup \{h\}, S') >_L \omega(F)$.
3. For all $h \in D$, $\omega(D'' \cup \{h\}, S'') >_L \omega(G) \rightarrow \omega(D' \cup \{h\}, S') >_L \omega(F)$.

Regarding the first property we note (for every set $X \in \mathbb{R}^d$, let *Int*(X) denote its interior, and let $\partial(X)$ denote its boundary) that

$$\begin{aligned}
\omega(D'', S'' \cup \{h\}) <_L \omega(F) &\iff C(D'', S'' \cup \{h\}) = h \\
&\iff (h \in \mathit{Int}(\mathit{Feasible}(F))) \vee ((h \in \partial(\mathit{Feasible}(F))) \\
&\qquad\qquad \wedge (h <_L C(F))) \\
&\iff (h \in \mathit{Int}(\mathit{Feasible}(G))) \vee ((h \in \partial(\mathit{Feasible}(G))) \\
&\qquad\qquad \wedge (h <_L C(G))) \\
&\iff C(D', S' \cup \{h\}) = h \\
&\iff \omega(D', S' \cup \{h\}) <_L \omega(G).
\end{aligned}$$

We now consider the remaining two properties. Let $d_i \in D$. $\omega(D'' \cup \{d_i\}, S'') >_L \omega(F)$ if and only if one of the following cases occurs:

1. $r(D'' \cup \{d_i\}, S'') > r(F)$, or
2. $r(D'' \cup \{d_i\}, S'') = r(F)$ and $C(D'' \cup \{d_i\}, S'') >_L C(F)$, or
3. $r(D'' \cup \{d_i\}, S'') = r(F)$, $C(D'' \cup \{d_i\}, S'') =_L C(F)$, and $(l_1(D'' \cup \{d_i\}, S''), -g_1(D'' \cup \{d_i\}, S''), \ldots, l_d(D'' \cup \{d_i\}, S''), -g_d(D'' \cup \{d_i\}, S'')) >_L (l_1(F), -g_1(F), \ldots, l_d(F), -g_d(F))$.

Regarding case 1, we have

$$\begin{aligned}
&r(D'' \cup \{d_i\}, S'') > r(F) \iff r(F)d_i \cap \mathit{Feasible}(F) \cap S'' = \emptyset \\
&\iff (r(G)d_i \cap \mathit{Feasible}(G) \cap S' = \emptyset) \vee ((r(G)d_i \cap \partial(\mathit{Feasible}(G) \cap (S' \setminus S'')) \neq \emptyset) \\
&\qquad\qquad \wedge C(G) \notin r(G)d_i) \\
&\iff (r(D' \cup \{d_i\}, S') > r(G)) \vee ((r(D' \cup \{d_i\}, S') = r(G)) \\
&\qquad\qquad \wedge (C(D' \cup \{d_i\}, S') >_L C(G))).
\end{aligned}$$

We now consider case 2.

$$\begin{aligned}
&(r(D'' \cup \{d_i\}, S'') = r(F)) \wedge (C(D'' \cup \{d_i\}, S'') >_L C(F)) \\
&\iff (C(F) \notin r(F)d_i) \wedge (r(F)d_i \cap \partial(\mathit{Feasible}(F)) \cap S'' \neq \emptyset) \\
&\Rightarrow (C(G) \notin r(G)d_i) \wedge (r(G)d_i \cap \partial(\mathit{Feasible}(G)) \cap S' \neq \emptyset) \\
&\iff (r(D' \cup \{d_i\}, S') = r(D', S')) \wedge (C(D' \cup \{d_i\}, S') >_L C(D', S')).
\end{aligned}$$

When $S' = S''$, the other direction of implications is also correct.

Case 3 occurs if and only $C(F) \in r(F)d_i$ and there exists j such that, among the projections of the boxes in $r(F)D'' \cup \{r(F)d_i\}$ on the jth coordinate, the projection of $r(F)d_i$ results in an interval $[l, r]$ with either the smallest r or the greatest l. This happens if and only if $C(G) \in r(G)d_i$, and among the projections of the boxes in $r(G)D' \cup \{r(G)d_i\}$ on the jth coordinate, the projection of $r(G)d_i$ results in an interval $[l, r]$ with either the smallest r or the greatest l. This occurs if and only if $r(D' \cup \{d_i\}, S') = r(G)$, $C(D' \cup \{d_i\}, S') =_L C(G)$, and $(l_1(D' \cup \{d_i\}, S'), -g_1(D' \cup \{d_i\}, S'), \ldots, l_d(D' \cup \{d_i\}, S'), -g_d(D' \cup \{d_i\}, S')) >_L (l_1(G), -g_1(G), \ldots, l_d(G), -g_d(G))$.

From the above analysis we get that the last two properties are indeed satisfied. Hence (D, S, ω) is a DLP-type problem which satisfies the VC.

It is easy to verify that $B(D, S) = (\{L_1(D, S), G_1(D, S), \ldots, L_d(D, S), G_d(D, S)\}, \{C(D, S)\})$ is a basis of a feasible and bounded (D, S) and that the problem is of d-dimension $2d$ and s-dimension 1.

The violation test can easily be implemented in constant time. For a basis $B = (B_D, B_S)$ and a d-element d_i, $\omega(B_D \cup \{d_i\}, B_S) > \omega(B)$ if and only if either $r(B)d_i$ does not contain $C(B)$ or there exists j such that, among the projections of the boxes in $r(B)D'' \cup \{r(B)d_i\}$ on the jth coordinate, the projection of $r(B)d_i$ results in an interval $[l, r]$ with either the smallest r or the greatest l. For an s-element h, $\omega(B_D, B_S \cup \{h\}) < \omega(B)$ if and only if either h lies in the interior of $\mathit{Feasible}(B)$ or h lies on the boundary of $\mathit{Feasible}(B)$ and $h \leq_L C(B)$. The basis calculation can be implemented in constant time by calling the violation test a constant number of times. Using a DLP algorithm such as the one stated in section 5, we conclude as follows.

THEOREM 9.2. *The discrete weighted* 1*-center problem in* $\mathbb{R}^d$ *with an* l_∞ *norm is solvable in (randomized) linear time for every fixed* d.

The rectilinear 1-center problem in $\mathbb{R}^d$ (i.e., with an l_1 norm) is solved similarly by using the rectilinear Helly-type versions of Theorems 1.4 and 1.5 (i.e., with rectilinear "balls" instead of axis-parallel boxes), which have Helly number 2^d instead of $2d$. We get the following theorem.

THEOREM 9.3. *The discrete weighted rectilinear* 1*-center problem in* $\mathbb{R}^d$ *is solvable in (randomized) linear time for every fixed* d.

We note that, while the Euclidean 1-center problem in $\mathbb{R}^d$ can be formulated as a $(d+1)$-dimensional LP-type problem and thus is solved in randomized linear time [31], the corresponding discrete problem admits an $\Omega(n \log n)$ lower bound under the algebraic computation tree model and is solved in the same time bound [24]. This demonstrates that sometimes the complexity of a discrete optimization version of a continuous optimization problem is strictly harder, as discussed also in the introduction.

10. Solving problems related to line transversals in the plane.

10.1. Continuous case. We first consider the *lexicographic (continuous) line transversal of axis-parallel rectangles problem.* The input is a family $D = \{d_1, \ldots, d_n\}$ of axis-parallel (closed) rectangles in the plane, together with a set of their reference points $C = \{c_1, \ldots, c_n\}$ such that c_i lies in the interior of d_i for every $i = 1, \ldots, n$. For a particular rectangle $d_i \in D$, let λd_i be the homothet of d_i that results from scaling d_i by a factor of λ, relatively to c_i (i.e., while keeping the point c_i fixed in the plane). Let $\lambda D = \{\lambda d \mid d \in D\}$. In the lexicographic (continuous) line transversal of axis-parallel rectangles *optimization* problem, we are interested in the smallest scaling factor λ^* and lexicographically smallest vector (a'', b'') such that the line $y = a''x + b''$ intersects each of the scaled rectangles in $\lambda^* D$. In the corresponding (nonlexicographic) *decision* problem (i.e., no scaling is allowed), we ask whether there exists a line transversal which intersects all of the rectangles in D. We note that this decision problem is solved in linear time via LP-type algorithms or by reducing it to linear programming [1, 2]. We are unaware of any linear time algorithms for the (nonlexicographic) optimization problem. We solve this problem by solving the (more general) lexicographic problem in linear time and noting that the optimal scaling factors of the lexicographic and nonlexicographic problems are equal. We solve the lexicographic problem by using the LP-type framework and the following two Helly-type theorems.

THEOREM 10.1 (see [29]). *Let D be a family of parallel open rectangles in the plane. If every subset of at most 6 rectangles admits a line transversal, then H does as well.*

THEOREM 10.2 (Theorem 2.12 in [20]). *Let D be a family of axis-parallel (closed) rectangles in the plane. For every pair of reals a' and b', if every subfamily of at most 6 rectangles admits a line transversal $y = ax + b$, with $(a, b) \leq_L (a', b')$, then D does as well.*

We note that, for every line direction (e.g., vertical to the x-axis), the restricted problem of finding the smallest scaling factor λ^*, such that there exists a line transversal for $\lambda^* D$ in this direction, is solvable in linear time by projecting the problem on the vertical direction (e.g., on the x-axis) and formulating it as a 2-dimensional LP problem. Hence it is enough to solve in linear time the problem where the line transversal must not be vertical to the x-axis.

We show that this problem is a 6-dimensional LP-type problem by formulating it as a parameterized Helly system with lexicographic Helly number 6 and using Theorem 6.6. Let the ground set $X = \mathbb{R}^2$ be the set of lines in the plane which are not vertical to the x-axis (i.e., $(a, b) \in X$ is the line $Y = aX + b$), and let $\Lambda = \mathbb{R}^+$. For every $d \in D$, let $t(d)$ be the set of lines intersecting d, let $\bar{d} = \{t(\lambda d) \mid \lambda \in \mathbb{R}^+\}$, and let $\bar{D} = \{\bar{d} \mid d \in D\}$.

Every line that intersects the homothet $\lambda_1 d$ also intersects $\lambda_2 d$ for any $\lambda_2 > \lambda_1$, so each $\bar{d}$ is a nested family of lines. Due to Theorem 10.2 every $(X, \lambda D)$ is a Helly system with lexicographic Helly number 6. Due to Theorem 10.1 every $(X, \lambda\, Int(D))$ is a Helly system with Helly number 6. Hence, Theorem 6.6 implies that $(\mathbb{R}^2 \times \mathbb{R}^+ \times \mathbb{R}^2, \bar{D})$ is a

parameterized Helly system with Helly number 6 and that $(\bar{D}, \omega)$ is a 6-dimensional LP-type problem. The natural objective function $\omega(\bar{D})$ is the lsmallest vector $\lambda = (\lambda_0, a, b)$ such that D_λ intersects at the point (a, b) (i.e., the line $aX + b$ intersects each of the scaled rectangles in $\lambda_0 D$). Recall that the algorithm in [28] runs in $O(t_v n + t_b \log n)$ time, where t_v is the time needed for a violation test and t_b is the time required for a basis calculation. Since both violation test and basis calculation primitives can easily be implemented in constant time, we have just proved the following theorem.

THEOREM 10.3. *The line transversal of axis-parallel rectangles optimization problem is solvable in (randomized) linear time.*

We conclude this section by considering several variants of the line transversal of a totally separable set of convex planar objects problem (problem 3 in the introduction). The input for the *lexicographic* version of this problem is a totally separable family $D = \{d_1, \ldots, d_n\}$ of simple convex objects, a family $C = \{c_1, \ldots, c_n\}$ of reference points such that c_i lies in the interior of d_i for every $i = 1, \ldots, n$, and a vector (a', b'). In the *decision problem* we want to decide whether there exists a line $Y = aX + b$, with $(a, b) \leq_L (a', b')$, which intersects all of the objects in D. In the *optimization problem* we are interested in the smallest scaling factor λ^* and lexicographically smallest vector (a, b) such that the line $y = ax + b$ intersects each of the scaled objects in $\lambda^* D$. Clearly, the answer for the decision problem is positive if and only if the solution of the optimization problem is at most $(1, a', b')$. We solve this problem in linear time using the LP-type framework and the following two Helly-type theorems.

THEOREM 10.4 (see [23]). *Let D be a totally separable finite family of open convex sets. If every subset of at most 3 sets admits a line transversal, then D does as well.*

THEOREM 10.5 (Corollary 2.20 in [20]). *Let D be a totally separable family of (closed) convex sets. For every pair of reals a' and b', if every subfamily of at most 3 sets admits a line transversal $y = ax + b$, with $(a, b) \leq_L (a', b')$, then H does as well.*

It is easy to show, using similar arguments to the ones mentioned earlier in this section, that the optimization problem is indeed a 3-dimensional LP-type problem and that it is solved in linear time. We thus have just proved the following theorem.

THEOREM 10.6. *The line transversal of totally separable set of convex planar objects decision problem is solvable in (randomized) linear time.*

10.2. Discrete case I—A finite number of permissible directions of line transversals. In this section we define a discrete version for the line transversal of axis-parallel rectangles optimization problem. We solve it in randomized linear time by using the DLP-type framework.

Problem 10.7. Given are a set $D = \{d_1, \ldots, d_n\}$ of (not necessarily pairwise disjoint) axis-parallel compact rectangles in the plane, together with a set of their reference points $C = \{c_1, \ldots, c_n\}$, such that c_i lies in the interior of d_i, for every $i = 1, \ldots, n$, and a set $S = \{s_1, \ldots, s_m\}$ of permissible line directions. Find the minimal scaling factor $\lambda_1^* = \lambda_1(D, S) \in \mathbb{R}^+$ such that $\lambda_1^* D$ admits a line transversal whose direction is in S.

If we choose S to be the (infinite) set of all possible directions, this problem coincides with the continuous one. We can assume that S does not contain the vertical direction and that the directions in S are such that the permissible lines are $\{y = ax + b \mid a \in S\}$. (If S does contain a vertical direction, we will take the minimal solution (i.e., scaling factor) among the ones of Problem 10.7 on S without

the vertical direction and on the vertical direction alone. As already mentioned in the previous section, the latter problem is solved in linear time by formulating it as an LP problem.)

A special case of Problem 10.7 is when the line transversal must be nondescending.

Problem 10.8. Given are a set $D = \{d_1, \ldots, d_n\}$ of (not necessarily pairwise disjoint) axis-parallel compact rectangles, together with a set of their reference points $C = \{c_1, \ldots, c_n\}$, such that c_i lies in the interior of d_i for every $i = 1, \ldots, n$, and a set $S = \{s_1, \ldots, s_m\}$ of permissible line directions. Find the minimal scaling factor $\lambda_1^* = \lambda_1(D, S) \in \mathbb{R}^+$ such that $\lambda_1^* D$ admits a nondescending line transversal whose direction is in S.

The solution of Problem 10.7 is the minimum scaling factor between the solution of Problem 10.8 and the analog problem where the line transversal must be nonascending.

10.2.1. A formulation as a discrete LP-type problem. In this section we formulate Problem 10.8 as a fixed-dimensional DLP-type problem by using Theorem 8.16 and the following Helly-type theorems.

THEOREM 10.9 (Theorem 2.13 in [20]). *Let D be a family of open rectangles in the plane with edges parallel to the axes, and let S be a set of nonnegative reals (line directions). If every subfamily of at most 4 rectangles admits a line transversal with a slope from S, then D does as well.*

THEOREM 10.10 (Theorem 5.8 in [20]). *Let D be a family of rectangles in the plane with edges parallel to the axes, and let S be a set of nonnegative reals (line directions). For every pair of reals $a' \geq 0$ and b', and a pair of nonnegative reals $sl_{\min} \leq sl_{\max}$, if every subfamily of at most 5 rectangles admits a line transversal $y = ax + b$, with $a \in S$, $sl_{\min} \leq a \leq sl_{\max}$, and $(a, b) \leq_L (a', b')$, then D does as well.*

Let $G = (D', S') \in 2^D \times 2^S$ be an arbitrary set such that $G \neq (\emptyset, \emptyset)$. We first look closely at an optimal solution for Problem 10.8 on G. Let λ_1^* be the optimal scaling factor. Due to Theorem 10.9 there is a direction $s^* \in S'$ and a set $D'' \subseteq D'$ of at most 4 rectangles such that the solution of Problem 10.8 on $(D'', \{s^*\})$ is λ_1^*. For this solution we define the following variables:

- $\lambda_1(D', S') \in \mathbb{R}^+$ is λ_1^*, the optimal scaling factor.
- $DIR(D', S') \in S$ is s^*, the minimal direction in S' in which there exists a nondecreasing line transversal for $\lambda_1(D', S')D'$.
- $LINE(D', S') = (DIR(D', S'), b(D', S'))$ is the line $y = DIR(D', S')x + b(D', S')$, which intersects every $\lambda_1(D', S')d \in \lambda_1(D', S')D$.

We note that, due to the optimality of λ_1^*, there exists only one line transversal to D' with direction s^*, and this line is tangent to at least 2 rectangles in $\lambda_1^* D'$.

In order to solve Problem 10.8, we first define and solve a lexicographic version of it, containing 4 more parameters. Let us consider the dual space $\mathbb{R}^2$ of all possible line transversals for $\lambda_1^* D'$. In this dual space, each nonvertical line $y = ax + b$ is represented by the point (a, b). We will use the following observation.

Observation 10.11 (Observation 5.7 in [20]). Let D be a family of axis-parallel rectangles in the plane, and, for every $d \in D$, let $\mathcal{L}(d)$ be the set of line transversals which d admits in the dual space of line transversals. Let $P(D) = \cap_{d \in D} \mathcal{L}(d)$ be the set of line transversals which D admits. The intersection of $P(D)$ with either the x nonnegative or x nonpositive half-planes is a convex polygon. The slopes of line transversals with nonnegative (nonpositive) slopes for D generate a slope range interval $[sl^{\min}, sl^{\max}]$ ($[sl_{\min}, sl_{\max}]$) resulted by the projection of $P(D)$ on the nonnegative (nonpositive) part of the x-axis, respectively. Each of the 4 end points of these two

intervals is determined by two rectangles.

Due to this observation, the range of slopes of the possible nondecreasing line transversals is a closed interval contained in $\mathbb{R}^+$. We call this interval the *slope range* corresponding to $\lambda_1^* D'$ and denote it by $[SL^{\min}(D', S'), SL^{\max}(D', S')]$, where

- $SL^{\min}(D', S')$ is the slope of the line transversal for $\lambda_1^* D'$ with a minimal nonnegative slope, and
- $SL^{\max}(D', S')$ is the slope of the line transversal for $\lambda_1^* D'$ with a maximal nonnegative slope.

(Due to the optimality of the scaling factor, the slope range does not contain any direction from S' in its interior.) We are ready to define a lexicographic version for Problem 10.8.

Problem 10.12. Given are a set $D = \{d_1, \ldots, d_n\}$ of (not necessarily pairwise disjoint) axis-parallel rectangles, together with a set of their reference points $C = \{c_1, \ldots, c_n\}$, such that c_i lies in the interior of d_i for every $i = 1, \ldots, n$, and a set $S = \{s_1, \ldots, s_m\}$ of permissible line directions. Find the lexicographically minimal vector $\lambda = (\lambda_1, s, b, sl^{\min}, -sl^{\max})$ such that the line $y = sx + b$ ($s \in S$) is nondescending, intersects all of the rectangles in $\lambda_1 D$, and the slope range corresponding to $\lambda_1 D$ is $[sl^{\min}, sl^{\max}]$.

Clearly, the optimal solution of Problem 10.12 is an optimal solution for Problem 10.8.

We now apply Theorem 10.10 in order to construct a parameterized discrete Helly system. Let the ground set be $X = \mathbb{R}^+ \times \mathbb{R} \times \mathbb{R}^+ \times \mathbb{R}^-$, the space of all nondecreasing lines and slope ranges. In this space, each point represents a line by the geometric duality transformation mentioned above and a slope range, as shown below. Let the range of the objective function be $\Lambda = \mathbb{R}^+$. For every $h \in D$ and $\lambda \in \mathbb{R}^+$ we define

(10.1)
$$h_\lambda = \left\{ x = (a, b, sl^{\min}, -sl^{\max}) \in X \;\middle|\; \begin{array}{l} y = ax + b \text{ is a line transversal for } \lambda h \text{ and} \\ a \in [sl^{\min}, sl^{\max}]. \end{array} \right\}.$$

As usual, we let $\bar{h} = \{h_\lambda \mid \lambda \in \Lambda\}$.

LEMMA 10.13. *For all $h \in D$, $\bar{h}$ is a nested family.*

Proof. We need to show that for all $\alpha, \beta \in \Lambda$, with $\alpha < \beta$, $h_\alpha \subseteq h_\beta$, i.e., for all $x \in h_\alpha$, x is also in h_β. This is true by monotonicity: A line transversal for αh is also a line transversal for βh. □

For every $h \in S$ and $\lambda \in \Lambda$ we let

$$h_\lambda = \{x \mid x \text{ is a line with direction } s\}.$$

Obviously, for every $h \in S$, $\bar{h} = \{h_\lambda \mid \lambda \in \Lambda\}$ is a nested family as well, and h_λ does not depend on λ.

From the definitions and Theorem 10.9 we get that, for all $\lambda \in \Lambda$, $(X, D_\lambda, S_\lambda)$ is a discrete Helly system with Helly number 4, and thus $(X \times \Lambda, \bar{D}, \bar{S})$ is a parameterized discrete Helly system with Helly number 4 (see Definition 8.8). From Theorem 10.10, we get that, for all $\lambda \in \Lambda$, $(X, D_\lambda, S_\lambda)$ is a discrete Helly system with lexicographic Helly number 5. Thus all of the conditions in Theorem 8.16 are fulfilled, and (D, S, ν) is a DLP-type problem of d-dimension at most 5 and s-dimension 1, where for all $G = (D', S') \in 2^D \times 2^S$

$$\nu(G) = (\lambda_1(G), DIR(G), b(G), SL^{\min}(G), -SL^{\max}(G)). \tag{10.2}$$

We have just proved the following lemma.

LEMMA 10.14. *(D, S, ν) is a $(5,1)$-dimensional DLP-type problem.*

Before we continue, we explain the values that the decision and optimization problems return. The decision problem returns "yes" if and only if $\lambda_1^* = \lambda_1(D,S) \leq 1$. The optimization problem returns the minimal scaling factor $\lambda_1(D,S)$, the minimal non-negative direction from S such that $\lambda_1(D,S)$ admits a line transversal with direction $DIR(D,S)$ and intersection point $b(D,S)$ with the y-axis, the slope range defined by $SL^{\min}(D,S), SL^{\max}(D,S)$, and a basis $B = (B_D, B_S)$ for (D,S). We note that there exist line transversals for $\lambda_1^* D$ with each of the slopes $SL^{\min}(D,S)$ and $SL^{\max}(D,S)$. We can view B and $LINE(D,S)$ as witnesses for the optimality of the scaling factor $\lambda_1(D,S)$. We need only to check that $\lambda_1(B_D,S) = \lambda_1^*$ (the monotonicity of demand condition implies $\lambda_1^*(D,S) \geq \lambda_1^*$) and that the line transversal $LINE(D,S)$ intersects each one of the rectangles $\lambda_1^* h$, $h \in D$ (the monotonicity of supply condition implies $\lambda_1^*(D,S) \leq \lambda_1^*$). The first test can be executed in $|S|$ time and the second in $|D|$ time.

10.2.2. A linear time algorithm. In this section we apply the linear time algorithm stated in section 5. We need to show that the conditions stated in Theorem 5.4 are satisfied and implement each of the violation test and basis calculation primitives in constant time. In the last section we formulated Problem 10.8 as a $(5,1)$-dimensional DLP-type problem. Thus, it remains to show the following.

LEMMA 10.15. *(D, S, ν) meets the VC (Definition 4.9).*

Proof. We need to show that for every $(D', S') \in 2^D \times 2^S$ and $(D'', S'') \in 2^{D'} \times 2^{S'}$ with $\nu(D', S') = \nu(D'', S'')$ the following properties hold:

1. For every $h \in D$, if $\nu(D'' \cup \{h\}, S'') > \nu(D'', S'')$, then $\nu(D' \cup \{h\}, S') > \nu(D', S')$.
2. For every $h \in S$, if $\nu(D'', S'' \cup \{h\}) < \nu(D'', S'')$, then $\nu(D', S' \cup \{h\}) < \nu(D', S')$.

Let $\lambda_1 = \lambda_1(D', S') = \lambda_1(D'', S'')$. We define the following functions related to $P(\lambda_1 D')$, the set of line transversal which $\lambda_1 D'$ admits (see Observation 10.11 for the definition and structure of $P(\lambda_1 D')$ in the dual space of line transversals). Let $l^{\min}(D', S')$ be the unique line transversal with direction $SL^{\min}(D', S')$ that $\lambda_1(D', S') D'$ admits. Let $b^{\min}(D', S')$ be its intersection point with the y-axis. In this way $(SL^{\min}(D', S'), b^{\min}(D', S'))$ is the leftmost point in $P(\lambda_1 D')$. We define $l^{\max}(D', S')$ and $b^{\max}(D', S')$ similarly, so $(SL^{\max}(D', S'), b^{\max}(D', S'))$ is the rightmost point in $P(\lambda_1 D')$.

The proof of both properties relies on the following observation which is true due to (10.2):

(10.3)

$$\nu(D', S') = \nu(D'', S'') \rightarrow \quad \text{the functions } SL^{\min},\ b^{\min},\ l^{\min},\ SL^{\max},\ b^{\max}, \text{ and } l^{\max} \text{ have the same values on } (D', S') \text{ and on } (D'', S'').$$

We first show that the first property holds. $h \in D$ violates (D'', S'') if and only if the set of line transversals which $\lambda_1 h$ admits does not contain both lines $l^{\min}(D'', S'')$ and $l^{\max}(D'', S'')$ (i.e., $\{(SL^{\min}(D'', S''), b^{\min}(D'', S'')); (SL^{\max}(D'', S''), b^{\max}(D'', S''))\} \not\subset P(\lambda_1 h)$). Using (10.3), the latter condition occurs if and only if the set of line transversals which $\lambda_1 h$ admits does not contain both lines $l^{\min}(D', S')$ and $l^{\max}(D', S')$, which in turn occurs if and only if $h \in D$ violates (D', S').

Regarding the second property, we observe that $h \in S$ violates (D'', S'') if and only if either the slope h lies in the interior of the slope range corresponding to $\lambda_1 D''$ (so the scaling factor decreases) or h is the left end point of the slope range with $h < DIR(D'', S'')$. We conclude the proof by using (10.3) again. ∎

We are ready to make the complexity calculations. Given a basis $B = (B_D, B_S)$ and its optimal scaling factor λ_1, we compute in constant time the functions $SL^{\min}(B)$, $b^{\min}(B), l^{\min}(B), SL^{\max}(B), b^{\max}(B), l^{\max}(B), DIR(B)$, and the set of all line transversals for $\lambda_1 B$, $P(\lambda_1 B_D) = \cap_{h \in B_D} \mathcal{D}(\lambda_1 h)$ (see Observation 10.11 for the notations). The following violation tests are implemented in constant time as follows:

t_{vS}: A new s-element h violates B if and only if either h lies in the interior of the slope domain $(SL^{\min}(B), SL^{\max}(B))$ or $h = SL^{\min}(B)$ and $DIR(B) = SL^{\max}(B)$.

t_{vD}: A new d-element h violates B if and only if $\mathcal{D}(\lambda_1 h)$ does not contain $\{(SL^{\min}(D'', S''), b^{\min}(D'', S'')); (SL^{\max}(D'', S''), b^{\max}(D'', S''))\}$.

Using the violation tests it is easy to see that the basis calculation for (D', S') where both $|D'|, |S'|$ are constants can be implemented in constant time. We have proved the following.

THEOREM 10.16. *Problem* 10.7 *is solvable in (randomized) linear time.*

COROLLARY 10.17. *The lexicographic discrete line transversal of axis-parallel rectangles problem is solvable in (randomized) linear time.*

10.3. Discrete case II—A finite number of permissible line transversals. In this section we show that the problem below has a lower bound of $\Omega(n \log n)$.

Problem 10.18. Given are a set $D = \{d_1, \ldots, d_n\}$ of (not necessarily pairwise disjoint) axis-parallel compact rectangles, together with a set of their reference points $C = \{c_1, \ldots, c_n\}$, such that c_i lies in the interior of d_i for every $i = 1, \ldots, n$, and a set $S = \{s_1, \ldots, s_m\}$ of permissible lines. Find the minimal scaling factor $\lambda_1^* = \lambda_1(D, S) \in \mathbb{R}^+$ such that $\lambda_1^* D$ admits a line transversal from S.

Clearly it is sufficient to show that the corresponding decision problem has that lower bound.

THEOREM 10.19. *Given a set D of axis-parallel rectangles and a set S of lines, deciding whether D admits a line transversal from S requires $\Omega(n \log n)$ time under the algebraic computation tree model (when $m = n$).*

Proof. We reduce in linear time the set equality problem (see definition in section 4) to this decision problem. Given an instance of the set equality problem, i.e., two sets A, B of n real numbers each, we act as follows. We find $\min_A$ and $\max_A$ ($\min_B$ and $\max_B$) the minimal and maximal elements in A (B), respectively. We define two new sets $A' = \{\frac{2(a-\min_A)}{\max_A - \min_A} - 1 \mid a \in A\}$ and $B' = \{\frac{2(b-\min_B)}{\max_B - \min_B} - 1 \mid b \in B\}$. All of the elements in A' and B' are numbers between -1 to 1, and $A = B$ if and only if $A' = B'$. For any $-1 \le r \le 1$ let $p(r)$ be the intersection point of the unit circle and the ray originating at the origin and having an angle of r radians with the positive part of the x-axis. We define two instances for the problem. The first instance, instance I, has a set of lines $S_{\mathrm{I}} = S(A')$ and a set of rectangles (intervals) $D_{\mathrm{I}} = D(B')$ defined as follows. We let $S(A') = \{s(a') \mid a' \in A'\}$, where $s(a')$ is the line tangent to the unit circle at point $p(a')$. We let $D(B') = \{i(b') \mid b' \in B'\}$, where $i(b')$ is a horizontal interval of length M, where M is a large number (e.g., 100), whose left end is slightly to the right of $p(b')$ (from a computation point of view we build the left end of the interval at exactly $p(b)$ but symbolically do not include this point in the interval). From the above construction we get that $D(B')$ admits a line transversal from $S(A')$ if and only if $A \not\subset B$. The second instance, instance II, has the set of lines $S_{\mathrm{II}} = S(B')$ and the set of rectangles $D_{\mathrm{II}} = D(A')$. We get that $D(A')$ admits a line transversal from $S(B')$ if and only if $B \not\subset A$. We conclude the proof by observing that $A = B$ if and only if both instances of the problem return negative responses. ∎

Acknowledgment. The author expresses his sincere gratitude to the referees for their valuable remarks, which helped to improve the exposition of the paper considerably.

REFERENCES

[1] N. Amenta, *Helly Theorems and Generalized Linear Programming*, Ph.D. dissertation, University of California, Berkeley, 1993.
[2] N. Amenta, *Helly-type theorems and generalized linear programming*, Discrete Comput. Geom., 12 (1994), pp. 241–261.
[3] M. Ben-Or, *Lower bounds for algebraic computation trees*, in Proceedings of the 15th ACM Annual Symposium on Theory of Algorithms, 1983, pp. 80–86.
[4] H. Björklund, S. Sandberg, and S. Vorobyov, *Randomized Subexponential Algorithms for Parity Games*, Technical report 2003-019, Department of Information Technology, Uppsala University, 2003; available online at http://www.it.uu.se/research/reports.
[5] H. Björklund, S. Sandberg, and S. Vorobyovn, *A discrete subexponential algorithm for parity games*, in Proceedings of the Twentieth Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2607, Springer, Berlin, 2003, pp. 663–674.
[6] H. Björklund, S. Sandberg, and S. Vorobyovn, *A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games*, Discrete Appl. Math., 155 (2007), pp. 210–229.
[7] B. Chazelle and J. Matoušek, *On linear-time deterministic algorithms for optimization problems in fixed dimension*, J. Algorithms, 21 (1996), pp. 579–597.
[8] K. L. Clarkson, *Las Vegas algorithms for linear and integer programming when the dimension is small*, J. ACM, 42 (1995), pp. 488–499.
[9] L. Danzer and B. Grünbaum, *Intersection properties of boxes in* $\mathbb{R}^d$, Combinatorica, 2 (1982), pp. 237–246.
[10] L. Danzer, B. Grünbaum, and V. Klee, *Helly's theorem and its relatives*, in Proc. Sympos. Pure Math. Vol VII, AMS, Providence, RI, 1963, pp. 101–180.
[11] J. Doignon, *Convexity in cristallographical lattices*, J. Geometry, 3 (1973), pp. 71–85.
[12] J. Eckhoff, *Helly, Radon and Carathéodory type theorems*, in Handbook of Convex Geometry, P. M. Gruber and J. M. Willis, eds., Elsevier Science Publishers B.V., Amsterdam, 1993.
[13] P. Egyed and R. Wenger, *Ordered stabbing of pairwise disjoint convex sets in linear time*, Discrete Appl. Math., 31 (1991), pp. 133–140.
[14] G. Frederickson, *Optimal algorithms for tree partitioning*, in Proceedings of the Second Annual ACM Symposium on Discrete Algorithms, 1991, pp. 168–177.
[15] B. Gärtner and E. Welzl, *Linear programming - Randomization and abstract frameworks*, in Proceedings of the Thirteenth Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1046, Springer, Berlin, 1996, pp. 669–687.
[16] J. Goodman, R. Pollack, and R. Wenger, *Geometric transversal theory*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., Springer, Berlin, 1993.
[17] N. Halman, *Discrete and Lexicographic Helly Theorems and Their Relations to LP-Type Problems*, Ph.D. dissertation, Tel Aviv University, 2004; available online at http://www.math.tau.ac.il/phd/dissertations/Halman_Nir.ps.
[18] N. Halman, *On the power of discrete and of lexicographic Helly theorems*, in Proceedings of the Forty-Fifth Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 463–472.
[19] N. Halman, *Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems*, Algorithmica, 49 (2007), pp. 37–50.
[20] N. Halman, *Discrete and lexicographic Helly theorems*, Discrete Comput. Geom., to appear.
[21] M. Hoffmann, *A simple linear algorithm for computing rectangular* 3*-centers*, in Proceedings of the Eleventh Annual Canadian Conference on Computational Geometry, 1999.
[22] G. Kalai, *A subexponential randomized simplex algorithm*, in Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computation, 1992, pp. 475–482.
[23] V. J. Klee, *Common secants for plane convex sets*, Proc. Amer. Math. Soc., 5 (1954), pp. 639–641.
[24] D. Lee and Y. F. Wu, *Geometric complexity of some location problems*, Algorithmica, 1 (1986), pp. 193–211.
[25] W. Ludwig, *A subexponential randomized algorithm for the simple stochastic game problem*, Inform. and Comput., 117 (1995), pp. 151–155.

[26] J. Matoušek, *Lectures on Discrete Geometry*, Springer, New York, 2002.
[27] J. Matoušek, B. Gärtner, L. Ruest, and P. Skovron, *Violator spaces: Structure and algorithms*, in 14th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 4168, Springer, Berlin, 2006, pp. 387–398.
[28] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.
[29] L. Santaló, *Un teorema sóbre conjuntos de paralelepipedos de aristas paralelas*, Publ. Inst. Mat. Univ. Nac. Litoral, 2 (1940), pp. 49–60.
[30] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, New York, 1986.
[31] M. Sharir and E. Welzl, *A combinatorial bound for linear programming and related problems*, in Proceedings of the Ninth Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 577, Springer, Berlin, 1992, pp. 569–579.
[32] M. Sharir and E. Welzl, *Rectilinear and polygonal p-piercing and p-center problems*, in Proceedings of the Twelfth Annual ACM Symposium on Computational Geometry, 1996, pp. 122–132.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 46–62

LOWER BOUNDS FOR RANDOMIZED AND QUANTUM QUERY COMPLEXITY USING KOLMOGOROV ARGUMENTS*

SOPHIE LAPLANTE† AND FRÉDÉRIC MAGNIEZ†

Abstract. We prove a very general lower bound technique for quantum and randomized query complexity that is easy to prove as well as to apply. To achieve this, we introduce the use of Kolmogorov complexity to query complexity. Our technique generalizes the weighted and unweighted methods of Ambainis and the spectral method of Barnum, Saks, and Szegedy. As an immediate consequence of our main theorem, it can be shown that adversary methods can only prove lower bounds for Boolean functions f in $O(\min(\sqrt{nC_0(f)}, \sqrt{nC_1(f)}))$, where C_0, C_1 is the certificate complexity and n is the size of the input.

Key words. quantum computing, lower bounds, query complexity, adversary method, Kolmogorov complexity

AMS subject classifications. 81P68, 68Q30, 68Q30

DOI. 10.1137/050639090

1. Introduction.

1.1. Overview. In this paper, we study lower bounds for randomized and quantum query complexity. In the query model, the input is accessed using oracle queries, and the query complexity of an algorithm is the number of calls to the oracle. Since it is difficult to obtain lower bounds on time directly, the query model is often used to prove concrete lower bounds, in classical as well as quantum computation.

The two main tools for proving lower bounds on randomized query complexity, the polynomial method [7] and the adversary method [2], were successfully extended to quantum computation. In the randomized setting, the adversary method is most often applied using Yao's minimax principle [21]. Using a different approach, which introduces the notion of quantum adversaries, Ambainis developed a general scheme in which it suffices to analyze combinatorial properties of the function in order to obtain a quantum lower bound. Recently, Aaronson [1] brought these combinatorial properties back to randomized computation, using Yao's minimax principle.

The most general method for proving lower bounds in quantum query complexity is the semidefinite programming method of Barnum, Saks, and Szegedy [5]. This method is in fact an exact characterization of the query complexity. However, the method is so general that it is very difficult to apply to obtain concrete lower bounds. Barnum, Saks, and Szegedy gave a weaker method derived from the semidefinite programming approach, using weight matrices and their largest eigenvalue. This spectral method can be thought of as a generalization of Ambainis's unweighted method. Other generalizations of Ambainis's unweighted method have been previously introduced [6, 3]. All of them use a weight function on the instances. The difficulty in applying these methods is finding a good weight function on the instances. Høyer,

*Received by the editors August 29, 2005; accepted for publication (in revised form) September 13, 2007; published electronically March 28, 2008. A preliminary version of this paper appeared in *Proceedings of the* 16*th Annual IEEE Conference on Computational Complexity*, 2004, pp. 294–304. Work was partially supported by the European Commission IST project QAP 015848 and by the ANR Blanc NT05-2_42239 AlgoQP.

http://www.siam.org/journals/sicomp/38-1/63909.html

†LRI, Univ. Paris-Sud, CNRS, Orsay, F-91405 France (laplante@lri.fr, magniez@lri.fr).

Neerbek, and Shi [15] were the first to use such weight assignments to prove lower bounds for searching in ordered lists and sorting.

This paper presents a new, very general adversary technique (Theorem 1.1) to prove lower bounds in quantum and randomized query complexity. We believed that this technique is simpler to prove and to apply. It is based on the framework of Kolmogorov complexity. This framework has proven to be very useful for proving negative results in other models of computation, for example, for the number of rounds and length of advice in random-self-reductions in [13, 4]. The techniques we use here are an adaptation of those techniques to the framework of query complexity. We expect that this framework will prove to be useful for negative results in other quantum models of computation, for instance, communication complexity, where we hope to give lower bounds for bounded round query complexity.

The proof of Theorem 1.1 is in two parts. The first part (divergence lemma) shows how fast the computations can diverge when they start on different inputs. This part depends on the model of computation (randomized or quantum). The quantum case of this lemma was first proven by Ambainis [2]. The second part (query information lemma) does not depend on the model of computation. It establishes the relationship between the Kolmogorov complexity of individual positions of the input and the probability that a given algorithm makes a query to this position. Whereas Aaronson [1] used a different approach to prove a version of Ambainis's method for randomized algorithms, here we use the same framework to establish lower bounds for both quantum and randomized query complexities (QQC and RQC).

We show that our method encompasses all previous adversary methods, including the quantum and randomized weighted methods [3, 1] (Theorem 4.2) and the spectral method [5] (Theorem 4.3). As an immediate consequence of our main theorem (observed by Troy Lee), our method can only prove lower bounds for arbitrary Boolean functions in $O(\min(\sqrt{nC_0(f)}, \sqrt{nC_1(f)}))$, where C_0 and C_1 is the certificate complexity of negative and positive instances, respectively, of f and n is the size of the input (Theorem 5.2). Prior to our work, it was known [3] that the unweighted Ambainis method [2, Theorem 5.1] could not prove bounds better than $\Omega(\sqrt{C_0(f)C_1(f)})$ for total functions; Szegedy [20] also proved independently that the semidefinite programming method could not prove lower bounds better than $O(\min(\sqrt{nC_0(f)}, \sqrt{nC_1(f)}))$, and Zhang [22] proved the same thing for Ambainis's weighted method.

We end the paper by giving some applications of our method to prove lower bounds for some graph properties: bipartiteness (Theorem 5.4) and connectivity (Theorem 5.3). The lower bound on connectivity was proven in [12] and the one on bipartiteness by Dürr and independently in [22]. We reprove it here to illustrate the simplicity of our method.

In recent developments, Špalek and Szegedy [19] showed that our method is equivalent to both the spectral method [5] as well as Ambainis's weighted method [3]. Subsequently, Laplante, Lee, and Szegedy showed that the square of the quantum adversary method was also a lower bound on formula size [16].

1.2. Main result. The conditional Kolmogorov complexity $\mathsf{K}(a|b)$ (defined formally in section 2.1) is the length of the shortest program which prints a given b as input. Our main result relates the query complexity of an algorithm A for f to the quantities $\{\mathsf{K}(i|x, A), \mathsf{K}(i|y, A) : x_i \neq y_i\}$ for any x, y such that $f(x) \neq f(y)$.

THEOREM 1.1. *There exists a constant $C > 0$ such that the following holds. Let Σ be a finite set, let $n \geq 1$ be an integer, and let $S \subseteq \Sigma^n$ and S' be sets. Let $f : S \to S'$. Let A be an algorithm that for all $x \in S$ computes f, with bounded error ε and at most T queries to the input. Then for every $x, y \in S$ with $f(x) \neq f(y)$:*

1. *if A is a quantum algorithm, then*
$$T \geq C \times \frac{1 - 2\sqrt{\varepsilon(1-\varepsilon)}}{\sum_{i:x_i \neq y_i} \sqrt{2^{-\mathsf{K}(i|x,A)-\mathsf{K}(i|y,A)}}};$$
2. *if A is a randomized algorithm, then*
$$T \geq C \times \frac{1 - 2\varepsilon}{\sum_{i:x_i \neq y_i} \min\left(2^{-\mathsf{K}(i|x,A)}, 2^{-\mathsf{K}(i|y,A)}\right)}.$$

We briefly describe the intuition behind the proof of Theorem 1.1. Consider an algorithm that purports to compute f, presented with two inputs x, y that lead to different outputs. The algorithm must query those positions where x and y differ with average probability of the order of $\frac{1}{T}$, or it will not successfully compute the function. On the other hand, the queries that are made with high average probability can be described succinctly given the input and the algorithm, by using the Shannon–Fano code. If we exhibit a pair of strings x, y for which there is no succinct description of any of the positions where x and y differ, then the number of queries must be large.

The same reasoning can be applied to classical and to quantum computing; the only difference is how fast two different input states cause the outputs to diverge to different outcomes.

To conclude the introduction we give a very simple application, for Grover search.

Example 1. Fix n and a quantum algorithm A for a Grover search for instances of length n. Let z be a binary string of length $\log n$, with $\mathsf{K}(z|A) \geq \log n$. Let j be the integer between 0 and $n-1$ whose binary expansion is z. Consider x, the all 0's string, and let y be everywhere 0 except at position $i = j + 1$, where it is 1. Then $\mathsf{K}(i|x, A) \geq \log n - O(1)$ and $\mathsf{K}(i|y, A) \geq 0$; therefore, $\mathsf{QQC}(\textsc{Search}) = \Omega(\sqrt{n})$.

2. Preliminaries.

2.1. Kolmogorov complexity. We use a few standard results in Kolmogorov complexity and information theory in this paper. We briefly review these here. The reader is invited to consult standard textbooks such as [17] for more background on Kolmogorov complexity and [9] for more on information theory. We denote the length of a finite string x by $|x|$. We assume that the Turing machine's alphabet is the same finite alphabet as the alphabet used to encode instances of the function under consideration. Letters x, y typically represent instances; i is an index into the binary representation of the instance; and p, q are probability distributions. Programs are denoted P, and the output of a Turing machine M on input x is written $M(x)$. When there are multiple inputs, we assume that a standard encoding of tuples is used.

Definition 2.1.

1. *A set of strings is* prefix-free *if no string is a prefix of another string in the set.*
2. *A universal Turing machine M is* prefix-free *if the set of programs $\{P : \exists x\, M(P, x) \neq \epsilon\}$, where ϵ is the empty string, is prefix-free.*
3. *Let M be a universal prefix-free Turing machine. Let x and y be finite strings. The* prefix-free Kolmogorov complexity *of x given y with respect to M is denoted $\mathsf{K}_M(x|y)$ and defined as follows:*
$$\mathsf{K}_M(x|y) = \min(|P| \text{ such that } M(P, y) = x).$$

In the rest of the paper M is a fixed universal prefix-free Turing machine, and we will write K instead of K_M. When y is the empty string, we write $\mathsf{K}(x)$ instead of $\mathsf{K}(x|y)$.

We first state standard bounds on conditional Kolmogorov complexity, where the last one is from [17, Theorem 3.9.1, p. 232].

PROPOSITION 2.2. *There exists a constant $c \geq 0$ such that, for every finite string σ,*

$$\mathsf{K}(x|\sigma) \leq \mathsf{K}(x) + c, \tag{2.1}$$

$$\mathsf{K}(x) \leq \mathsf{K}(\sigma) + K(x|\sigma) + c, \tag{2.2}$$

$$|\mathsf{K}(x,y) - \mathsf{K}(x) - \mathsf{K}(y|x,\mathsf{K}(x))| \leq c. \tag{2.3}$$

We shall also use the following bound.

PROPOSITION 2.3. *There is a constant $c \geq 0$ such that, for any three strings x, y, z,*

$$\mathsf{K}(z|x) \geq \mathsf{K}(x,y) - \mathsf{K}(x) - \mathsf{K}(y|z,x) + \mathsf{K}(z|x,y,\mathsf{K}(x,y)) - c.$$

Proof. Using the third bound of Proposition 2.2, there is a constant $c_1 \geq 0$ such that

$$|\mathsf{K}(a,b) - \mathsf{K}(a) - \mathsf{K}(b|a,\mathsf{K}(a))| \leq c_1.$$

Substituting x, y for a and z for b:

$$\mathsf{K}(x,y) + \mathsf{K}(z|x,y,\mathsf{K}(x,y)) - c_1 \leq \mathsf{K}(x,y,z) \leq \mathsf{K}(x) + \mathsf{K}(z|x) + \mathsf{K}(y|z,x) + c_2,$$

which gives the result, where the second inequality follows from the first and third bounds of Proposition 2.2. □

The main motivation for using prefix-free Kolmogrov complexity is the bound known as Kraft's inequality together with the last two bounds of Proposition 2.2.

PROPOSITION 2.4 (Kraft's inequality). *Let T be any prefix-free set of finite strings. Then $\sum_{P \in T} 2^{-|P|} \leq 1$. In particular, for any set of finite strings S and any finite string σ, $\sum_{x \in S} 2^{-\mathsf{K}(x|\sigma)} \leq 1$.*

A *source* $\mathcal{S}$ of finite strings is a pair (S, p), where S is a set of finite strings and p is a probability distribution over S.

PROPOSITION 2.5 (Shannon's coding theorem). *Consider a source $\mathcal{S}$ of finite strings where x occurs with probability $p(x)$. Then for any code for $\mathcal{S}$ the average code length is bounded below by the entropy of the source; that is, if x is encoded by the code word $c(x)$ of length $|c(x)|$, $H(\mathcal{S}) = \sum_{x:p(x)\neq 0} p(x) \log(\frac{1}{p(x)}) \leq \sum_{x:p(x)\neq 0} p(x)|c(x)|$.*

LEMMA 2.6. *Let $\mathcal{S}$ be a source as above. Then for any fixed finite string σ there exists a string x such that $p(x) \neq 0$ and $\mathsf{K}(x|\sigma) \geq \log(\frac{1}{p(x)})$.*

Proof. By Shannon's coding theorem,

$$H(\mathcal{S}) = \sum_{x:p(x)\neq 0} p(x) \log\left(\frac{1}{p(x)}\right) \leq \sum_{x:p(x)\neq 0} p(x)\mathsf{K}(x|\sigma),$$

because $\mathsf{K}(x|\sigma)$ is the length of an encoding of x. Therefore there exists x such that $p(x) \neq 0$ and $\mathsf{K}(x) \geq \log(\frac{1}{p(x)})$. □

The Shannon–Fano code is a prefix-free code that encodes each word x with $p(x) \neq 0$, using $\lceil \log(\frac{1}{p(x)}) \rceil$ bits. We will write $\log(\frac{1}{p(x)})$ to simplify notation. The code can easily be computed given a description of the probability distribution. We formalize this in the following proposition, letting $\mathsf{K}(x|\mathcal{S})$ denote the prefix-free Kolmogorov complexity of x given a finite description of $\mathcal{S}$.

PROPOSITION 2.7 (Shannon–Fano code). *There exists a constant $c \geq 0$ such that, for every source $\mathcal{S}$ as above, for all x such that $p(x) \neq 0$, $\mathsf{K}(x|\mathcal{S}) \leq \log(\frac{1}{p(x)}) + c$.*

2.2. Query models. The quantum query model was implicitly introduced by Deutsch, Jozsa, Simon, Bernstein, Vazirani, and Grover [11, 10, 18, 8, 14] and explicitly by Beals et al. [7]. In this model, as in its classical counterpart, we pay for accessing the oracle, but unlike the classical case, the machine can use the power of quantum parallelism to make queries in superposition. Access to the input $x \in \Sigma^n$, where Σ is a finite set, is achieved by way of a query operator O_x. The *query complexity* of an algorithm is the number of calls to O_x.

The state of a computation is represented by a register R composed of three subregisters: the *query register* $i \in \{0, \ldots, n\}$, the *answer register* $z \in \Sigma$, and the *work register* w. We denote a register using the ket notation $|R\rangle = |i\rangle|z\rangle|w\rangle$, or simply $|i, z, w\rangle$. In the quantum (resp., randomized) setting, the state of the computation is a complex (resp., nonnegative real) combination of all possible values of the registers. Let $\mathcal{H}$ denote the corresponding finite-dimensional vector space. We denote the state of the computation by a vector $|\psi\rangle \in \mathcal{H}$ over the basis $(|i, z, w\rangle)_{i,z,w}$. Furthermore, the state vectors are unit length for the ℓ_2 norm in the quantum setting and for the ℓ_1 norm in the randomized setting.

A *T-query algorithm* A is specified by a $(T+1)$-tuple $(U_0, U_1, \ldots, U_T)$ of matrices. When A is quantum (resp., randomized), the matrices U_i are unitary (resp., stochastic). The computation takes place as follows. The *query operator* is the unitary (resp., stochastic) matrix O_x that satisfies $O_x|i, z, w\rangle = |i, z \oplus x_i, w\rangle$ for every i, z, w, where by convention $x_0 = 0$. Initially the state is set to some fixed value $|0, 0, 0\rangle$. Then the sequence of transformations $U_0, O_x, U_1, O_x, \ldots, U_{T-1}, O_x, U_T$ is applied.

We say that the algorithm A *ε-computes* a function $f : S \to S'$, for some sets $S \subseteq \Sigma^n$ and S', if the observation of the last bits of the work register equals $f(x)$ with probability at least $1 - \varepsilon$ for every $x \in S$. Then $\mathsf{QQC}(f)$ (resp., $\mathsf{RQC}(f)$) is the minimum query complexity of quantum (resp., randomized) query algorithms that ε_0-compute f, where $\varepsilon_0 = 1/3$.

3. Proof of the main theorem. This section is devoted to the proof of the main theorem. We prove Theorem 1.1 in two main steps. Lemma 3.1 shows how fast the computations diverge when they start on different individual inputs, in terms of the query probabilities. This lemma depends on the model of computation. Lemma 3.2 establishes the relationship between the Kolmogorov complexity of individual positions of the input and the probability that a given algorithm makes a query to this position. This lemma is independent of the model of computation. Theorem 1.1 follows immediately by combining these two lemmas.

In the following two lemmas, let A be an ε-bounded error algorithm for f that makes at most T queries to the input. When A is a randomized algorithm, let $p_t^x(i)$ be the probability that A queries x_i at query t on input x. By analogy, when A is a quantum algorithm, the probability $p_t^x(i)$ is interpreted as the probability of observing i if the query register were measured at query t, that is, the square of the norm of the part of the state that queries x_i. Let $\overline{p}^x(i) = \frac{1}{T}\sum_{t=1}^{T} p_t^x(i)$ be the average query probability over all of the time steps up to time T. We assume henceforth without loss of generality that $\overline{p}^x(i) > 0$. (For example, we start by uniformly querying all positions and reverse the process.)

LEMMA 3.1 (divergence lemma). *For every input $x, y \in S$ such that $f(x) \neq f(y)$ the following hold.*

1. *For quantum algorithms:*

$$2T \sum_{i : x_i \neq y_i} \sqrt{\overline{p}^x(i)\overline{p}^y(i)} \geq 1 - 2\sqrt{\varepsilon(1-\varepsilon)}.$$

2. *For randomized algorithms:*

$$2T \sum_{i:x_i \neq y_i} \min\left(\overline{p}^x(i), \overline{p}^y(i)\right) \geq 1 - 2\varepsilon.$$

We defer the proof of Lemma 3.1 to the end of this section.

The next lemma relates the query probabilities to the Kolmogorov complexity of the strings. In this lemma and the results that follow, we assume that a finite description of the algorithm is given. Using the knowledge of A, we may assume without loss of generality that the function f that it computes is also given, as is the length n of the inputs. With additional care, the additive constants in all of the proofs can be made very small by adding to the auxiliary information made available to the description algorithms those constant-size programs that are described within the proofs.

LEMMA 3.2 (query information lemma). *There exists an absolute constant $c \geq 0$ such that, for every input $x \in S$ and position $i \in \{1, \ldots n\}$,*

$$\mathsf{K}(i|x, A) \leq \log\left(\frac{1}{\overline{p}^x(i)}\right) + c.$$

Proof. Let $\mathcal{S}_x$ be the source where i occurs with probability $\overline{p}^x(i)$. By Proposition 2.7, $\mathsf{K}(i|\mathcal{S}_x) \leq \log(\frac{1}{\overline{p}^x(i)}) + c$ for some absolute constant c. To complete the proof, it suffices to show that $\mathsf{K}(\mathcal{S}_x|x, A) = O(1)$ and apply the second bound of Proposition 2.2. Use x and A to compute the probabilities $(\overline{p}^x(i))_{1 \leq i \leq n}$. The probabilities can be computed in a finite number of steps because the dimension is finite, and the number of queries is bounded by T. □

From these two lemmas we derive the main theorem.

Proof of Theorem 1.1. By Lemma 3.2, there is a constant $c \geq 0$ such that, for any algorithm that makes at most T queries and any x, y, i,

$$\overline{p}^x(i) \leq 2^{-\mathsf{K}(i|x,A)+c} \quad \text{and} \quad \overline{p}^y(i) \leq 2^{-\mathsf{K}(i|y,A)+c}.$$

This is true in particular for all those i where $x_i \neq y_i$. Combining this with Lemma 3.1 concludes the proof of the main theorem with $C = 2^{-c-1}$. □

We now give the proof of Lemma 3.1. The proof of the quantum case is very similar to the proofs found in many papers which give quantum lower bounds on query complexity. To our knowledge, the randomized case is new despite the simplicity of its proof. Whereas Aaronson [1] used a different approach to prove a version of Ambainis's method for randomized algorithms, our lemma allows us to use the same framework to establish lower bounds for both quantum and randomized query complexities.

Proof of Lemma 3.1. Let $|\psi_t^x\rangle$ be the state of the ε-bounded error algorithm A just before the tth oracle query, on input x. By convention, $|\psi_{T+1}^x\rangle$ is the final state. When A is a quantum algorithm, $|\psi_t^x\rangle$ is a unit vector for the ℓ_2 norm; otherwise, it is a probabilistic distribution, that is, a nonnegative and unit vector for the ℓ_1 norm. Observe that the ℓ_1 distance is the total variation distance.

First we prove the quantum case. The starting state of A does not depend on the input, and thus before the first question we have $|\psi_1^x\rangle = |\psi_1^y\rangle$, so $\langle\psi_1^x|\psi_1^y\rangle = 1$. At the end of the computation, if the algorithm is correct with probability ϵ, then $|\langle\psi_{T+1}^x|\psi_{T+1}^y\rangle| \leq 2\sqrt{\epsilon(1-\epsilon)}$ [2]. At each time step, we consider how much the two states can diverge in the following claim, which we will prove after the end of this proof.

Claim 1 (quantum divergence).

$$|\langle\psi_t^x|\psi_t^y\rangle - \langle\psi_{t+1}^x|\psi_{t+1}^y\rangle| \leq 2 \sum_{i:x_i\neq y_i} \sqrt{p_t^x(i)p_t^y(i)}.$$

Over T time steps, the two states diverge as follows. The proof uses only Claim 1 and the Cauchy–Schwartz inequality.

$$\begin{aligned}
1 - 2\sqrt{\varepsilon(1-\varepsilon)} &\leq |\langle\psi_1^x|\psi_1^y\rangle - \langle\psi_{T+1}^x|\psi_{T+1}^y\rangle| \\
&\leq \sum_{t=1}^{T} |\langle\psi_t^x|\psi_t^y\rangle - \langle\psi_{t+1}^x|\psi_{t+1}^y\rangle| \\
&\leq \sum_{t=1}^{T} 2 \sum_{i:x_i\neq y_i} \sqrt{p_t^x(i)p_t^y(i)} \\
&\leq 2 \sum_{i:x_i\neq y_i} \sqrt{\sum_{t=0}^{T-1} p_t^x(i) \sum_{t=0}^{T-1} p_t^y(i)} \\
&= 2T \sum_{i:x_i\neq y_i} \sqrt{\overline{p}^x(i)\overline{p}^y(i)}.
\end{aligned}$$

Now we prove the randomized case. We use the ket notation for real-valued normalized vectors, for consistency in notation. Again, initially $|\psi_1^x\rangle = |\psi_1^y\rangle$. At the end of the computation, if the algorithm is correct with probability ϵ, then $\| \, |\psi_{T+1}^x\rangle - |\psi_{T+1}^y\rangle \, \|_1 \geq 1 - 2\epsilon$. At each time step, the distribution states now diverge according to the following claim, which we will prove after the end of this proof.

Claim 2 (randomized divergence).

$$\begin{aligned}
&\| \, |\psi_{t+1}^x\rangle - |\psi_{t+1}^y\rangle \, \|_1 \\
&\leq \| \, |\psi_t^x\rangle - |\psi_t^y\rangle \, \|_1 + 2 \sum_{i:x_i\neq y_i} \min\left(p_t^x(i), p_t^y(i)\right).
\end{aligned}$$

We now conclude the proof.

$$\begin{aligned}
1 - 2\varepsilon &\leq \sum_{t=1}^{T} \| \, |\psi_{t+1}^x\rangle - |\psi_{t+1}^y\rangle \, \|_1 - \| \, |\psi_t^x\rangle - |\psi_t^y\rangle \, \|_1 \\
&\leq \sum_{t=1}^{T} 2 \sum_{i:x_i\neq y_i} \min\left(p_t^x(i), p_t^y(i)\right) \\
&\leq 2T \sum_{i:x_i\neq y_i} \min\left(\overline{p}^x(i), \overline{p}^y(i)\right). \qquad \square
\end{aligned}$$

Proof of Claim 1. Let

$$\begin{aligned}
|\psi_t^x\rangle &= \sum_{i,z,w} \alpha_{i,z,w}|i,z,w\rangle, \text{ and} \\
|\psi_t^y\rangle &= \sum_{i,z,w} \beta_{i,z,w}|i,z,w\rangle.
\end{aligned}$$

After the tth query is made, the states $|\psi_t'^x\rangle = O_x|\psi_t^x\rangle$ and $|\psi_t'^y\rangle = O_y|\psi_t^y\rangle$ are

$$|\psi_t'^x\rangle = \sum_{i,z,w} \alpha_{i,z,w}|i, z \oplus x_i, w\rangle, \text{ and}$$
$$|\psi_t'^y\rangle = \sum_{i,z,w} \beta_{i,z,w}|i, z \oplus y_i, w\rangle.$$

Now, since the inner product is invariant under unitary transformations, we get

$$\langle\psi_{t+1}^x|\psi_{t+1}^y\rangle = \langle\psi_t'^x|\psi_t'^y\rangle,$$

and therefore

$$\begin{aligned}
&|\langle\psi_t^x|\psi_t^y\rangle - \langle\psi_{t+1}^x|\psi_{t+1}^y\rangle| \\
&= \left|\sum_{i,z,w} \overline{\alpha_{i,z,w}}\beta_{i,z,w} - \sum_{i,z,w} \overline{\alpha_{i,z\oplus x_i,w}}\beta_{i,z\oplus y_i,w}\right| \\
&= \left|\sum_{\substack{i,z,w \\ x_i \neq y_i}} \overline{\alpha_{i,z,w}}\beta_{i,z,w} - \overline{\alpha_{i,z\oplus x_i,w}}\beta_{i,z\oplus y_i,w}\right| \\
&\leq \sum_{i:x_i\neq y_i} \left(\left|\sum_{z,w} \overline{\alpha_{i,z,w}}\beta_{i,z,w}\right| + \left|\sum_{z,w} \overline{\alpha_{i,z\oplus x_i,w}}\beta_{i,z\oplus y_i,w}\right|\right) \\
&\leq 2 \sum_{i:x_i\neq y_i} \sqrt{\left(\sum_{z,w}|\alpha_{i,z,w}|^2\right)\left(\sum_{z,w}|\beta_{i,z,w}|^2\right)} \\
&\leq 2 \sum_{i:x_i\neq y_i} \sqrt{p_t^x(i)p_t^y(i)}. \qquad \square
\end{aligned}$$

Proof of Claim 2. Let us write the distributions using the same formalism as above, that is,

$$|\psi_t^x\rangle = \sum_{i,z,w} \alpha_{i,z,w}|i, z, w\rangle, \text{ and}$$
$$|\psi_t^y\rangle = \sum_{i,z,w} \beta_{i,z,w}|i, z, w\rangle.$$

Note that now the vectors are unit for the ℓ_1 norm. After the tth query is made, the states $|\psi_t'^x\rangle = O_x|\psi_t^x\rangle$ and $|\psi_t'^y\rangle = O_y|\psi_t^y\rangle$ are

$$|\psi_t'^x\rangle = \sum_{i,z,w} \alpha_{i,z,w}|i, z \oplus x_i, w\rangle, \text{ and}$$
$$|\psi_t'^y\rangle = \sum_{i,z,w} \beta_{i,z,w}|i, z \oplus y_i, w\rangle.$$

Now, since the ℓ_1 distance does not increase under stochastic matrices, we get

$$\| \, |\psi_{t+1}^x\rangle - |\psi_{t+1}^y\rangle \, \|_1 \leq \| \, |\psi_t'^x\rangle - |\psi_t'^y\rangle \, \|_1,$$

and therefore

$$\begin{aligned}
&\| \, |\psi_{t+1}^x\rangle - |\psi_{t+1}^y\rangle \, \|_1 \\
&= \left\| \sum_{i,z,w} (\alpha_{i,z,w}|i, z \oplus x_i, w\rangle - \beta_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 \\
&= \sum_i \left\| \sum_{z,w} (\alpha_{i,z,w}|i, z \oplus x_i, w\rangle - \beta_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 .
\end{aligned}$$

We now bound each term of the last sum separately. Fix any i. If $x_i = y_i$, then

$$\left\| \sum_{z,w} (\alpha_{i,z,w}|i, z \oplus x_i, w\rangle - \beta_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 = \sum_{z,w} |\alpha_{i,z,w} - \beta_{i,z,w}|.$$

If $x_i \neq y_i$, then

$$\begin{aligned}
&\left\| \sum_{z,w} (\alpha_{i,z,w}|i, z \oplus x_i, w\rangle - \beta_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 \\
&\leq \left\| \sum_{z,w} (\alpha_{i,z,w}|i, z \oplus y_i, w\rangle - \beta_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 \\
&\quad + \left\| \sum_{z,w} (\alpha_{i,z,w}|i, z \oplus x_i, w\rangle - \alpha_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 \\
&\leq \sum_{z,w} |\alpha_{i,z,w} - \beta_{i,z,w}| + 2 \sum_{z,w} |\alpha_{i,z,w}| \\
&= \sum_{z,w} |\alpha_{i,z,w} - \beta_{i,z,w}| + 2p_t^x(i).
\end{aligned}$$

In the same way we can prove that

$$\begin{aligned}
&\left\| \sum_{z,w} (\alpha_{i,z,w}|i, z \oplus x_i, w\rangle - \beta_{i,z,w}|i, z \oplus y_i, w\rangle) \right\|_1 \\
&\leq \sum_{z,w} |\alpha_{i,z,w} - \beta_{i,z,w}| + 2p_t^y(i).
\end{aligned}$$

We group together these upper bounds and conclude that

$$\begin{aligned}
&\| \, |\psi_{t+1}^x\rangle - |\psi_{t+1}^y\rangle \, \|_1 \\
&\leq \sum_{i,z,w} |\alpha_{i,z,w} - \beta_{i,z,w}| + 2 \sum_{i : x_i \neq y_i} \min \left(p_t^x(i), p_t^y(i) \right) \\
&= \| \, |\psi_t^x\rangle - |\psi_t^y\rangle \, \|_1 + 2 \sum_{i : x_i \neq y_i} \min \left(p_t^x(i), p_t^y(i) \right) . \qquad \square
\end{aligned}$$

4. Comparison with previous adversary methods. In this section, we reprove, as a corollary of Theorem 1.1, the previously known adversary lower bounds. Our framework also allows us to obtain somewhat stronger statements for free.

To obtain the previously known adversary methods as a corollary of Theorem 1.1, we must give a lower bound on terms $\mathsf{K}(i|x,A)$ and $\mathsf{K}(i|y,A)$. To this end, we apply Proposition 2.3 and give a lower bound on $\mathsf{K}(x,y)$ and upper bounds on $\mathsf{K}(x|i,y)$ and $\mathsf{K}(y|i,x)$. The lower bound on $\mathsf{K}(x,y)$ is obtained by applying Lemma 2.6, a consequence of Shannon's coding theorem, for an appropriate distribution. The upper bounds on $\mathsf{K}(x|i,y)$ and $\mathsf{K}(y|i,x)$ are obtained using the Shannon–Fano code for appropriate distributions.

The following lemma is the general formulation of the sketch above.

LEMMA 4.1. *There exists a constant $C>0$ such that the following holds. Let Σ be a finite set, let $n\ge 1$ be an integer, and let $S\subseteq\Sigma^n$. Let q be a probability distribution on S^2, let p be a probability distribution on S, and let $\{p'_{x,i} : x\in S, 1\le i\le n\}$ be a family of probability distributions on S. Assume that whenever $q(x,y)\neq 0$, then $p(x)$, $p(y)$, $p'_{y,i}(x)$, and $p'_{x,i}(y)$ are nonzero for every i such that $x_i\neq y_i$. Then for every finite string σ there exist $x,y\in S$, with $q(x,y)\neq 0$, such that*

$$\frac{1}{\sum_{i:x_i\neq y_i}\sqrt{2^{-\mathsf{K}(i|x,\sigma)-\mathsf{K}(i|y,\sigma)}}} \ge C\times\min_{i:x_i\neq y_i}\left(\frac{\sqrt{p(x)p'_{x,i}(y)\ p(y)p'_{y,i}(x)}}{q(x,y)}\right),$$

and (for the same $x,y\in S$)

$$\frac{1}{\sum_{i:x_i\neq y_i}\min\left(2^{-\mathsf{K}(i|x,\sigma)},2^{-\mathsf{K}(i|y,\sigma)}\right)} \ge C\times\min_{i:x_i\neq y_i}\left(\max\left(\frac{p(x)p'_{x,i}(y)}{q(x,y)},\frac{p(y)p'_{y,i}(x)}{q(x,y)}\right)\right).$$

Proof. In this proof, $c_1,\dots,c_5$ are some appropriate nonnegative constants. By Lemma 2.6, there exists a pair (x,y) such that $q(x,y)\neq 0$ and

$$\mathsf{K}(x,y|\sigma,p,p')\ge\log\left(\frac{1}{q(x,y)}\right),$$

where p' stands for a complete description of all of the $p'_{x,i}$.

Fix x and y so that this holds. By using the Shannon–Fano code (Proposition 2.5),

$$\mathsf{K}(x|p)\le\log\left(\frac{1}{p(x)}\right)+c_1$$

and

$$\mathsf{K}(y|x,i,p'_{x,i})\le\log\left(\frac{1}{p'_{x,i}(y)}\right)+c_1$$

for any i such that $x_i \neq y_i$. By Proposition 2.3,

$$\begin{aligned}
&\mathsf{K}(i|x,\sigma)\\
&\geq \mathsf{K}(i|x,\sigma,p,p') - c_3\\
&\geq \mathsf{K}(x,y|\sigma,p,p') - \mathsf{K}(x|p) - \mathsf{K}(y|i,x,p'_{x,i})\\
&\quad + \mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p') - c_4\\
&\geq \log\left(\frac{1}{q(x,y)}\right) - \log\left(\frac{1}{p(x)}\right) - \log\left(\frac{1}{p'_{x,i}(y)}\right)\\
&\quad + \mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p') - c_5\\
&= \log\left(\frac{p(x)p'_{x,i}(y)}{q(x,y)}\right) + \mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p') - c_5.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathsf{K}(i|y,\sigma) \geq \log&\left(\frac{p(y)p'_{y,i}(x)}{q(x,y)}\right)\\
&+ \mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p') - c_5.
\end{aligned}$$

To conclude, consider the sum

$$\begin{aligned}
&\frac{1}{\sum_{i:x_i\neq y_i}\sqrt{2^{-\mathsf{K}(i|x,\sigma)-\mathsf{K}(i|y,\sigma)}}}\\
&\geq \frac{1}{\sum_{i:x_i\neq y_i}\sqrt{2^{-\log\left(\frac{p(x)p'_{x,i}(y)}{q(x,y)}\right)-\log\left(\frac{p(y)p'_{y,i}(x)}{q(x,y)}\right)-2\mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p')+2c_5}}}\\
&\geq \frac{1}{\sum_{i:x_i\neq y_i} 2^{c_5}\sqrt{\frac{q(x,y)}{p(x)p'_{x,i}(y)}\frac{q(x,y)}{p(y)p'_{y,i}(x)}}2^{-\mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p')}}\\
&\geq 2^{-c_5}\min_{i:x_i\neq y_i}\left(\frac{\sqrt{p(x)p'_{x,i}(y)p(y)p'_{y,i}(x)}}{q(x,y)}\right)\frac{1}{\sum_{i:x_i\neq y_i}2^{-\mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p')}}.
\end{aligned}$$

We apply Kraft's inequality (Proposition 2.4) to show that $\sum_{i:x_i\neq y_i} 2^{-\mathsf{K}(i|x,y,\mathsf{K}(x,y),\sigma,p,p')} \leq 1$. This concludes the proof of the first part of the lemma using Kraft's inequality and letting $C = 2^{-c_5}$. The second part is similar. □

4.1. Ambainis's weighted scheme.

THEOREM 4.2 (Ambainis's weighted method). *Let Σ be a finite set, let $n \geq 1$ be an integer, and let $S \subseteq \Sigma^n$ and S' be sets. Let $f : S \to S'$. Consider a weight scheme as follows:*

- *Every pair $(x,y) \in S^2$ is assigned a nonnegative weight $w(x,y)$ such that $w(x,y) = 0$ whenever $f(x) = f(y)$.*
- *Every triple (x,y,i) is assigned a nonnegative weight $w'(x,y,i)$ such that $w'(x,y,i) = 0$ whenever $x_i = y_i$ or $f(x) = f(y)$.*

For all x, i, let

$$wt(x) = \sum_y w(x,y) \quad \textit{and}$$
$$v(x,i) = \sum_y w'(x,y,i).$$

If $w'(x,y,i)w'(y,x,i) \geq w^2(x,y)$ *for all* x,y,i *such that* $x_i \neq y_i$*, then*

$$\mathsf{QQC}(f) = \Omega\left(\min_{\substack{x,y,i \\ w(x,y)\neq 0, x_i \neq y_i}} \left(\sqrt{\frac{wt(x)wt(y)}{v(x,i)v(y,i)}}\right)\right).$$

Furthermore, if $w'(x,y,i), w'(y,x,i) \geq w(x,y)$ *for all* x,y,i *such that* $x_i \neq y_i$*, then*

$$\mathsf{RQC}(f) = \Omega\left(\min_{\substack{x,y,i \\ w(x,y)\neq 0, x_i \neq y_i}} \left(\max\left(\frac{wt(x)}{v(x,i)}, \frac{wt(y)}{v(y,i)}\right)\right)\right).$$

The relation in Ambainis's original statement is implicit in this formulation, since it corresponds to the nonzero-weight pairs. A weaker version of the randomized case was proven independently by Aaronson [1] using a completely different approach. We show that Theorem 4.2 follows from Theorem 1.1.

Proof. We derive probability distributions q, p, p' from the weight schemes as follows. Let $W = \sum_{x,y} w(x,y)$. Define

$$\begin{aligned} q(x,y) &= \frac{w(x,y)}{W}, \\ p(x) &= \frac{wt(x)}{W}, \\ p'_{x,i}(y) &= \frac{w'(x,y,i)}{v(x,i)} \quad \text{for any } x,y,i. \end{aligned}$$

It is easy to check that, by construction and hypothesis, these distributions satisfy the conditions of Lemma 4.1. We may now rearrange and simplify the terms as follows:

$$\begin{aligned} \frac{\sqrt{p(x)p'_{x,i}(y)\ p(y)p'_{y,i}(x)}}{q(x,y)} &= \frac{\sqrt{\frac{wt(x)}{W}\frac{w'(x,y,i)}{v(x,i)}\ \frac{wt(y)}{W}\frac{w'(y,x,i)}{v(y,i)}}}{\frac{w(x,y)}{W}} \\ &= \frac{\sqrt{\frac{wt(x)}{v(x,i)}\ \frac{wt(y)}{v(y,i)}w'(x,y,i)w'(y,x,i)}}{w(x,y)} \\ &\geq \sqrt{\frac{wt(x)}{v(x,i)}\ \frac{wt(y)}{v(y,i)}}. \end{aligned}$$

The final line follows from the hypothesis $w'(x,y,i)w'(y,x,i) \geq w^2(x,y)$. The second part of the theorem is obtained by similar rearrangement and simplification. ∎

We conclude this section by sketching the proof of the unweighted version of Ambainis's adversary method, as it affords a simpler combinatorial proof that does not require Lemma 4.1. To simplify notation we omit additive constants and the usual auxiliary strings including A.

Let $R \subseteq S \times S$ be a relation on pairs of instances, where $(x,y) \in R \Longrightarrow f(x) \neq f(y)$, and let R_i be the restriction of R to pairs x,y for which $x_i \neq y_i$. Viewing the relation R as a bipartite graph, let l, l', m, m' be as follows:

- m is a lower bound on the degree of all $x \in X$,
- m' is a lower bound on the degree of all $y \in Y$,
- for any fixed x and $i, 1 \leq i \leq n$, the number of y adjacent to x for which $x_i \neq y_i$ is at most l,

- for any fixed y and $i, 1 \leq i \leq n$, the number of x adjacent to y for which $x_i \neq y_i$ is at most l'.

We make the following observations:

1. $|R| \geq \max\{m|X|, m'|Y|\}$, so $\exists x, y \in R$ such that $\mathsf{K}(x,y) \geq \max(\log(m|X|), \log(m'|Y|))$.
2. For all $x \in X, \mathsf{K}(x) \leq \log(|X|)$, and $\mathsf{K}(y) \leq \log(|Y|)$ for all $y \in Y$.
3. For all x, y, i with $(x,y) \in R_i, \mathsf{K}(y|i,x) \leq \log(l)$ and similarly $\mathsf{K}(x|i,y) \leq \log(l')$.

For any i with $x_i \neq y_i$, by Proposition 2.3,

$$\begin{aligned}\mathsf{K}(i|x) &\geq \mathsf{K}(x,y) - \mathsf{K}(x) - \mathsf{K}(y|i,x) \\ &\quad + \mathsf{K}(i|x,y,\mathsf{K}(x,y)) \\ &\geq \log(m|X|) - \log(|X|) - \log(l) \\ &\quad + \mathsf{K}(i|x,y,\mathsf{K}(x,y)) \\ &= \log\left(\frac{m}{l}\right) + \mathsf{K}(i|x,y,\mathsf{K}(x,y)).\end{aligned}$$

The same proof works to show that $\mathsf{K}(i|y) \geq \log(\frac{m'}{l'}) + \mathsf{K}(i|x,y,\mathsf{K}(x,y))$. By Theorem 1.1 and Kraft's inequality,

$$\mathsf{QQC}(f) = \Omega\left(\sqrt{\frac{mm'}{ll'}}\right).$$

4.2. Spectral lower bound. We now show how to prove the spectral lower bound of Barnum, Saks, and Szegedy [5] as a corollary of Theorem 1.1. Recall that for any matrix Γ, $\lambda(\Gamma)$ is the largest eigenvalue of Γ.

THEOREM 4.3 (Barnum–Saks–Szegedy spectral method). *Let Σ be a finite set, let $n \geq 1$ be an integer, and let $S \subseteq \Sigma^n$ and S' be sets. Let $f : S \to S'$. Let Γ be an arbitrary $S \times S$ nonnegative real symmetric matrix that satisfies $\Gamma(x,y) = 0$ whenever $f(x) = f(y)$. For $i = 1, \ldots, n$ let Γ_i be the matrix:*

$$\Gamma_i(x,y) = \begin{cases} 0 & \text{if } x_i = y_i, \\ \Gamma(x,y) & \text{otherwise.} \end{cases}$$

Then

$$\mathsf{QQC}(f) = \Omega\left(\frac{\lambda(\Gamma)}{\max_i \lambda(\Gamma_i)}\right).$$

Proof. Since Γ and Γ_i are nonnegative real symmetric matrices, they have an eigenvector with only nonnegative real entries for their respective largest eigenvalues. Let $|\alpha\rangle$ (resp., $|\alpha_i\rangle$) be this unit eigenvector of Γ (resp., Γ_i). We define the probability distributions q, p, p' as follows:

$$\begin{aligned} q(x,y) &= \frac{\Gamma(x,y)\langle x|\alpha\rangle\langle y|\alpha\rangle}{\langle\alpha|\Gamma|\alpha\rangle}, \\ p(x) &= \langle x|\alpha\rangle^2, \\ p'_{x,i}(y) &= \frac{\Gamma_i(x,y)\langle y|\alpha_i\rangle}{\langle x|\Gamma_i|\alpha_i\rangle}, \quad \text{for any } x, y, i. \end{aligned}$$

First we check that these are probability distributions. Distribution p also has weight 1 because $|\alpha\rangle$ is a unit vector. Since $|\alpha\rangle$ and $|y\rangle$ have real entries, $\langle y|\alpha\rangle = \langle\alpha|y\rangle$.

Then the distribution q has weight $\frac{1}{\langle\alpha|\Gamma|\alpha\rangle}\sum_{x,y}\langle\alpha|y\rangle\Gamma(x,y)\langle x|\alpha\rangle$, which is 1 since $\sum_x \Gamma(x,y)\langle x|\alpha\rangle = \langle y|\Gamma|\alpha\rangle$. Using the same argument, $p'_{x,i}$ also has weight 1.

Now, fix any x, y, i such that $x_i \neq y_i$ and $q(x,y) \neq 0$. Note that $\langle\alpha|\Gamma|\alpha\rangle = \lambda(\Gamma)$, $\Gamma_i|\alpha_i\rangle = \lambda(\Gamma_i)|\alpha_i\rangle$, and $\Gamma(x,y) = \Gamma_i(x,y)$. Then the fractions $\frac{p(x)p'_{x,i}(y)}{q(x,y)}$ and $\frac{p(y)p'_{y,i}(x)}{q(x,y)}$ are, respectively, $\frac{\lambda(\Gamma)}{\lambda(\Gamma_i)}\frac{\langle y|\alpha_i\rangle}{\langle x|\alpha_i\rangle}\frac{\langle x|\alpha\rangle}{\langle y|\alpha\rangle}$ and $\frac{\lambda(\Gamma)}{\lambda(\Gamma_i)}\frac{\langle x|\alpha_i\rangle}{\langle y|\alpha_i\rangle}\frac{\langle y|\alpha\rangle}{\langle x|\alpha\rangle}$. Taking the square root of their product gives the result using Lemma 4.1. □

5. Certificate complexity and adversary techniques. Let f be a Boolean function. For any positive instance $x \in \Sigma^n$ of f ($f(x)$=1), a *positive certificate* for $f(x)$ is the smallest subset of indices $I \subseteq [n]$ of x such that, for any y with $x_i = y_i$ for all $i \in I$, $f(y)$=1.

The 1-*certificate complexity* of f, denoted $C_1(f)$, is the size of the largest positive certificate for $f(x)$, over all positive instances x. The 0-*certificate complexity* is defined similarly for negative instances x of f ($f(x) = 0$).

Prior to our work, it was known that the best possible bound that could be proven using the unweighted adversary technique for total functions [2, Theorem 5.1] is $O(\sqrt{C_0(f)C_1(f)})$. Independently, Szegedy [20] showed that the best possible lower bound using the spectral method is $O(\min(\sqrt{nC_0(f)}, \sqrt{nC_1(f)}))$ for arbitrary functions, and Zhang [22] proved the same for Ambainis's weighted method.

The following lemma, due to Troy Lee, results in a very simple proof of the fact that our method and, hence, all of the known variants of the adversary method have lower bounds larger than $\min(\sqrt{nC_0(f)}, \sqrt{nC_1(f)})$ for arbitrary functions.

LEMMA 5.1. *There exists a constant $c \geq 0$ such that the following holds. Let Σ be a finite set, let $n \geq 1$ be an integer, and let $S \subseteq \Sigma^n$ be a set. Let $f : S \to \{0,1\}$. For every $x, y \in S$ with $f(x) = 0$ and $f(y) = 1$, there is an i_0 with $x_{i_0} \neq y_{i_0}$ for which $\mathsf{K}(i_0|x,f) \leq \log(C_0(f)) + c$, and similarly there is an i_1 with $x_{i_1} \neq y_{i_1}$ such that $\mathsf{K}(i_1|y,f) \leq \log(C_1(f)) + c$.*

Proof. Among the negative certificates for $f(x)$, let I be the lexicographically smallest one. By definition of the 0-certificate complexity, the size of I is at most $C_0(f)$. Since $f(x) \neq f(y)$, x and y must differ on some $i_0 \in I$. To describe i_0 given x, it suffices to give an index into I, which requires at most $\log(C_0(f)) + c$ bits. The same can also be done for y and $C_1(f)$. □

THEOREM 5.2. *Let Σ be a finite set, let $n \geq 1$ be an integer, and let $S \subseteq \Sigma^n$ be a set. Let $f : S \to \{0,1\}$. Then any quantum query lower bound for f given by Theorem* 1.1 *is in $O(\min(\sqrt{nC_0(f)}, \sqrt{nC_1(f)}))$.*

Proof. Let A be a quantum algorithm that computes f with bounded error by making at most T queries to the input and $x, y \in S$ such that $f(x) = 0$ and $f(y) = 1$. Then a description of f can be obtained from a description of A, so $\mathsf{K}(i|x,A) \leq \mathsf{K}(i|x,f) + O(1)$. By Lemma 5.1, there exists i_0 such that $x_{i_0} \neq y_{i_0}$, and $\mathsf{K}(i_0|x,f) \leq \log(C_0(f)) + O(1)$. For any $i, 1 \leq i \leq n$, $\mathsf{K}(i|y,A) \leq \log(n) + O(1)$. Therefore $K(i_0|x,A) + K(i_0|y,A) \leq \log(nC_0(f)) + O(1)$.

The lower bound given by Theorem 1.1 is $O\big(\frac{1}{\sum_{i:x_i\neq y_i}\sqrt{2^{-\mathsf{K}(i|x,A)-\mathsf{K}(i|y,A)}}}\big)$. Since $\sum_{i:x_i\neq y_i}\sqrt{2^{-\mathsf{K}(i|x,A)-\mathsf{K}(i|y,A)}} \geq \sqrt{2^{-\mathsf{K}(i_0|x,A)-\mathsf{K}(i_0|y,A)}}$, the bound is $O(\sqrt{nC_0(f)})$ Similarly, it can be shown that the bound is $O(\sqrt{nC_1(f)})$. □

In recent work, Špalek and Szegedy showed that, for total functions, the best lower bound one can achieve with any of the adversary methods is $\sqrt{C_0(f)C_1(f)}$ for any total function [19].

5.1. Applications to graph properties. Theorem 1.1 provides a simple and intuitive method to prove lower bounds for specific problems. We illustrate this by giving lower bounds for two graph properties: connectivity and bipartiteness. These are direct applications of Theorem 1.1 in that we analyze directly the complexity $\mathsf{K}(i|x, A)$ without defining relations or weights or distributions: We need only to consider a "typical" hard pair of instances. In this section, we omit additive and multiplicative constants that result from using small, constant-size programs, as well as the constant length auxiliary string A to simplify the proofs.

We consider graphs over n vertices $\{0, 1, \ldots, n-1\}$, where the graph is represented as an adjacency matrix.

5.1.1. Graph connectivity.

THEOREM 5.3 (see [12]). $\mathsf{QQC}(\textsc{GraphConnectivity}) = \Omega(n^{3/2})$.

Proof. We construct one negative and one positive instance of graph connectivity, using the incompressibility method, using the ideas of [12]. Let S be an incompressible string of length $\log(n-1)! + \log\binom{n}{2}$, chopped into two pieces S_1 and S_2 of length $\log(n-1)!$ and $\log\binom{n}{2}$, respectively. We think of S_1 as representing a Hamilton cycle $C = (\pi(0), \pi(1) \ldots \pi(n-1), \pi(0))$ through the n vertices, where π is a permutation over $\{0, 1, \ldots, n-1\}$ such that $\pi(0) = 0$. Let G contain the cycle C, so that $\mathsf{K}(G) = \mathsf{K}(\pi)$. We also think of S_2 as representing a pair of distinct vertices s, t. Let H be obtained from G by breaking the cycle into two cycles at s and t, that is, $H = G \setminus \{(\pi(s), \pi(s+1)), (\pi(t), \pi(t+1))\} \cup \{(\pi(s), \pi(t+1)), (\pi(s+1), \pi(t))\}$, where addition is modulo n.

We show that, for the four edges e where G and H differ, $\mathsf{K}(e|G) + \mathsf{K}(e|H) \geq 3\log n - 4$. Let e_-, e'_- be the edges removed from G and e_+, e'_+ be the edges added to G. Observe that, up to an additive constant, $\mathsf{K}(e_+|G) = \mathsf{K}(e'_+|G)$ and $\mathsf{K}(e_-|H) = \mathsf{K}(e'_-|H)$.

Assume without loss of generality that $e_- = (\pi(s), \pi(s+1))$ and that the smallest cycle of H contains $\pi(s)$. Let l be the length of this cycle. Observe that $\mathsf{K}(s|G) = \mathsf{K}(e_-|G)$ and $\mathsf{K}(e_-|H) = \mathsf{K}(\pi, s, t|H)$.

$$\begin{aligned}
\log(n-1)! + \log\binom{n}{2} &\leq \mathsf{K}(S) \\
&\leq \mathsf{K}(G) + \mathsf{K}(s|G) + \mathsf{K}(t|G) \\
&\leq \mathsf{K}(G) + \mathsf{K}(e_-|G) + \log n, \\
\mathsf{K}(e_-|G) &\geq \log\binom{n}{2} - \log n = \log\frac{n-1}{2}.
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\mathsf{K}(H) &\leq \mathsf{K}(l) + \log\frac{(n-1)!}{(n-l)!} + \log(n-l-1)! \\
&\leq \log\left(\frac{n}{2}\right) + \log(n-1)! - \log(n-l) \\
&\leq \log(n-1)!\,.
\end{aligned}$$

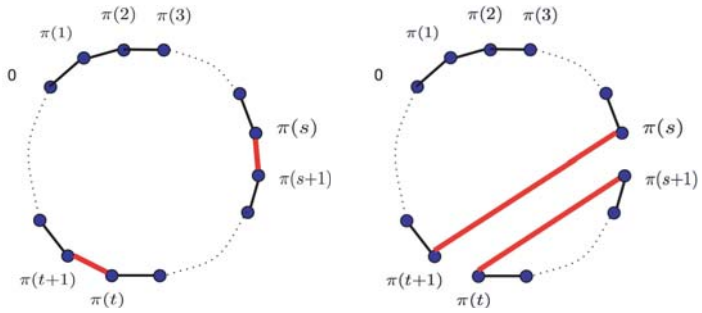


FIG. 5.1. *Graphs G, H for the graph lower bounds.*

Therefore,

$$\begin{aligned}\log(n-1)! + \log\binom{n}{2} &\le \mathsf{K}(S)\\ &\le \mathsf{K}(H) + \mathsf{K}(\pi, s, t|H)\\ &\le \mathsf{K}(H) + \mathsf{K}(e_-|H)\\ &\le \log(n-1)! + \mathsf{K}(e_-|H),\\ \mathsf{K}(e_-|H) &\ge \log\binom{n}{2}.\end{aligned}$$

For the added edges, e_+, e'_+, consider without loss of generality $e_+ = (\pi(s), \pi(t+1))$. Since S is incompressible, $\mathsf{K}(e_+|G) = \mathsf{K}(s,t|G) \ge \log\binom{n}{2}$. Furthermore, $\mathsf{K}(S) \le \mathsf{K}(H) + \mathsf{K}(e_+|H) + \mathsf{K}(e'_+|H)$ and $\mathsf{K}(e'_+|H) \le \log n$, so $\mathsf{K}(e_+|H) \ge \log\binom{n}{2} - \log n = \log\frac{n-1}{2}$. The same proof shows that $\mathsf{K}(e'_+|H) \ge \log\frac{n-1}{2}$. ∎

5.1.2. Bipartiteness. The following lower bound was proven by Dürr and independently in [22].

THEOREM 5.4. $\mathsf{QQC}(\text{BIPARTITENESS}) = \Omega(n^{3/2})$.

Proof. The proof is similar to the one of Theorem 5.3 except that we construct G to be an even cycle on $n = 2m$ vertices and H will be composed of two odd cycles on the same vertex set (see Figure 5.1).

Let S be an incompressible string of length $\log(n-1)! + \log(\binom{n}{2} - 1)$, chopped into two pieces S_1 and S_2 of length $\log(n-1)!$ and $\log(\binom{n}{2} - 1)$, respectively. We think of S_1 as representing a Hamilton cycle $C = (\pi(0) = 0, \pi(1) \dots \pi(n-1), \pi(0))$ through the n vertices and S_2 as representing a pair of distinct vertices s, t, with $s \not\equiv t \pmod 2$. Let G contain the cycle C, and let H be obtained from G by breaking the cycle into two odd cycles at s and t, that is, $H = G \setminus \{(\pi(s), \pi(s+1)), (\pi(t), \pi(t+1))\} \cup \{(\pi(s), \pi(t+1)), (\pi(s+1), \pi(t))\}$.

The same analysis as Theorem 5.3 yields the lower bound $\mathsf{QQC}(\text{BIPARTITENESS}) = \Omega(n^{3/2})$, as claimed. ∎

Acknowledgments. We thank Troy Lee, Christoph Dürr for many useful discussions, and Andris Ambainis for his helpful answers to our questions.

REFERENCES

[1] S. Aaronson, *Lower bounds for local search by quantum arguments*, SIAM J. Comput., 35 (2006), pp. 804–824.

[2] A. Ambainis, *Quantum lower bounds by quantum arguments*, J. Comput. System Sci., 64 (2002), pp. 750–767.

[3] A. Ambainis, *Polynomial degree vs. quantum query complexity*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, 2003, pp. 230–239.

[4] L. Babai and S. Laplante, *Stronger separations for random-self-reducibility, rounds, and advice*, in Proceedings of the IEEE Conference on Computational Complexity, 1999, pp. 98–104.

[5] H. Barnum, M. Saks, and M. Szegedy, *Quantum query complexity and semi-definite programming*, in Proceedings of the 18th IEEE Conference on Computational Complexity, 2003, pp. 179–193.

[6] H. Barnum and M. Saks, *A lower bound on the quantum query complexity of read-once functions*, J. Comput. System Sci., 69 (2004), pp. 244–258.

[7] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. Wolf, *Quantum lower bounds by polynomials*, J. ACM, 48 (2001), pp. 778–797.

[8] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.

[9] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience, New York, 1991.

[10] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci., 439 (1992), pp. 553–558.

[11] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. R. Soc. of Lond. Ser. A Math. Phys. Eng. Sci., 400 (1985), pp. 97–117.

[12] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla, *Quantum query complexity of some graph problems*, SIAM J. Comput., 35 (2006), pp. 1310–1328.

[13] J. Feigenbaum, L. Fortnow, S. Laplante, and A. V. Naik, *On coherence, random-self-reducibility, and self-correction*, Comput. Complexity, 7 (1998), pp. 174–191.

[14] L. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th ACM Symposium on Theory of Computing, 1996, pp. 212–219.

[15] P. Høyer, J. Neerbek, and Y. Shi, *Quantum complexities of ordered searching, sorting, and element distinctness*, Algorithmica, 34 (2002), pp. 429–448.

[16] S. Laplante, T. Lee, and M. Szegedy, *The quantum adversary method and classical formula size lower bounds*, Comput. Complexity, 16 (2006), pp. 163–196.

[17] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications*, in Graduate Texts in Computer Science, 2nd ed., Springer, New York, 1997.

[18] D. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483.

[19] R. Špalek and M. Szegedy, *All quantum adversary methods are equivalent*, Theory Comput., 2 (2006), pp. 1–18.

[20] M. Szegedy, *On the quantum query complexity of detecting triangles in graphs*, Technical report quant-ph/0310107, arXiv archive, 2003.

[21] A. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 222–227.

[22] S. Zhang, *On the power of Ambainis lower bounds*, Theoret. Comput. Sci., 339 (2005), pp. 241–256.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 63–84

MINIMIZING DISJUNCTIVE NORMAL FORM FORMULAS AND AC^0 CIRCUITS GIVEN A TRUTH TABLE*

ERIC ALLENDER†, LISA HELLERSTEIN‡, PAUL MCCABE§, TONIANN PITASSI§, AND MICHAEL SAKS¶

Abstract. For circuit classes R, the fundamental computational problem *Min-R* asks for the minimum R-size of a Boolean function presented as a truth table. Prominent examples of this problem include *Min-DNF*, which asks whether a given Boolean function presented as a truth table has a k-term disjunctive normal form (DNF), and *Min-Circuit* (also called the minimum circuit size problem (MCSP)), which asks whether a Boolean function presented as a truth table has a size k Boolean circuit. We present a new reduction proving that *Min-DNF* is NP-complete. It is significantly simpler than the known reduction of Masek [*Some NP-Complete Set Covering Problems*, manuscript, 1979], which is from *Circuit-SAT*. We then give a more complex reduction, yielding the result that *Min-DNF* cannot be approximated to within a factor smaller than $(\log N)^\gamma$, for some constant $\gamma > 0$, assuming that NP is not contained in quasi-polynomial time. The standard greedy algorithm for *Set Cover* is often used in practice to approximate *Min-DNF*. The question of whether *Min-DNF* can be approximated to within a factor of $o(\log N)$ remains open, but we construct an instance of *Min-DNF* on which the solution produced by the greedy algorithm is $\Omega(\log N)$ larger than optimal. Finally, we turn to the question of approximating circuit size for slightly more general classes of circuits. DNF formulas are depth-two circuits of AND and OR gates. Depth-d circuits are denoted by AC^0_d. We show that it is hard to approximate the size of AC^0_d circuits (for large enough d) under cryptographic assumptions.

Key words. machine learning theory, complexity theory, approximation algorithms, truth table minimization

AMS subject classifications. 68Q17, 68Q32, 03D15

DOI. 10.1137/060664537

1. Introduction.

A fundamental computational problem is to determine the minimum size of a Boolean function in some representation, given a truth table for the function. Two prominent examples are *Min-DNF*, which asks whether a Boolean function presented as a truth table has a k-term disjunctive normal form (DNF), and *Min-Circuit* (also called the minimum circuit size problem), which asks whether a Boolean function presented as a truth table has a size k Boolean circuit. By varying the representation class, we can obtain a hierarchy of problems between *Min-DNF* and *Min-Circuit*, including such problems as *Min-*AC^0, *Min-*TC^0, and *Min-*NC^1.

The main focus of this paper is the *Min-DNF* problem. *Min-DNF* is the decision version of finding the smallest DNF formula consistent with a truth table, where

*Received by the editors July 10, 2006; accepted for publication (in revised form) September 19, 2007; published electronically March 28, 2008. A preliminary version of this paper appeared in *Proceedings of the* 21*st Annual IEEE Conference on Computational Complexity* Prague, Czech Republic, 2006.

http://www.siam.org/journals/sicomp/38-1/66453.html

†Computer Science Department, Rutgers University, Piscataway, NJ 08854 (allender@cs.rutgers.edu). This author's research was supported in part by NSF grant CCF-0514155.

‡Computer Science Department, Polytechnic University, Brooklyn, NY 11201 (hstein@cis.poly.edu). Part of this research was conducted while this author was on sabbatical at the University of Wisconsin, Madison.

§Computer Science Department, University of Toronto, Toronto, ON, M5S 3G4, Canada (pmccabe@cs.toronto.edu, toni@cs.toronto.edu). The third author's research was supported by NSERC. The fourth author's research was supported by NSERC and PREA.

¶Department of Mathematics, Rutgers University, Piscataway, NJ 08854 (saks@math.rutgers.edu). This author's research was supported in part by NSF grant CCR-0515201.

the size of a DNF formula is considered to be the number of terms in it. This is a classic problem in computer science and circuit design. Heuristic approaches to solving this problem range from the Karnaugh maps of the 1960s to state-of-the-art software packages (cf. [14]).

Masek proved *Min-DNF* to be NP-complete in the 1970s [30]. This result was cited by Garey and Johnson [21] and is widely known, but Masek never published his proof. More recently, Czort presented a modernized, more readable version of Masek's proof [15] (see also [40]). Masek's proof is by direct reduction from *Circuit-SAT*, using gadget constructions, and even in Czort's version it is long and involved. We present a new, simple NP-completeness proof for *Min-DNF* by reduction from 3-*Partite Set Cover* (or, more particularly, from 3*D-Matching*).

It is well known that *Min-DNF* can be viewed as a special case of *Set Cover* and that the greedy *Set Cover* algorithm can be applied to *Min-DNF* to produce a DNF with $O(\log N)$ times as many terms as the optimal, where N is the size (number of entries) of the input truth table. This prompts the question of whether a better approximation factor can be achieved. Czort considered this question but showed only that, unless P = NP, the size of the smallest DNF cannot be approximated to within an *additive* constant k [15]. We also give a more complicated reduction (again from a restricted version of *Set Cover*) that allows us to prove the following inapproximability result for *Min-DNF*: If NP is not contained in quasi-polynomial time, then *Min-DNF* cannot be approximated to within a factor smaller than $(\log N)^\gamma$ for some constant $\gamma > 0$, where N is the size of the input truth table.

There is a gap between our $\Omega((\log N)^\gamma)$ inapproximability lower bound for *Min-DNF* and the $O(\log N)$ upper bound of the greedy *Set Cover* algorithm. Closing this gap remains an open question. We do, however, construct an instance of *Min-DNF* for which the greedy *Set Cover* algorithm produces a DNF formula that has $\Omega(\log N)$ times as many terms as the optimal. The greedy *Set Cover* algorithm is commonly used as a heuristic for solving *Min-DNF* in practice. We also prove an $\Omega(\sqrt{\log N})$ inapproximability lower bound for *Min-DNF* under the additional assumption that a restriction of *Set Cover* is $\Omega(\log n)$-hard to approximate.

Although the general *Min-DNF* problem is NP-hard, for $k = O(\sqrt{\log N})$ it is tractable [22]. Using a simple padding argument, we show hardness results for *Min-DNF* where $k = \omega(\log N)$. The question of whether *Min-DNF* is tractable for $k = \log N$ remains open. This question was posed in [22]; a negative result would imply that $\log n$-term DNF cannot be learned with membership and proper equivalence queries.

In addition to our results for *Min-DNF*, we also prove a result for *Min-*AC^0_d for all sufficiently large d. Under cryptographic assumptions, it is known that *Min-Circuit*, *Min-*NC^1, and *Min-*TC^0_d are not polynomial-time approximable [4]. (This is stated explicitly for *Min-Circuit* and *Min-*NC^1 in [4], while it is only implicit for *Min-*TC^0_d– because the argument presented in [4] makes use of the TC^0 pseudorandom function generator of [31]. All of this can also be viewed as being implicit in the work of Razborov and Rudich [36], and related results were also presented by Kabanets and Cai [25].) We extend the hardness results for *Min-*TC^0_d to obtain new hardness results for *Min-*AC^0_d, under cryptographic assumptions. This still leaves open the interesting question of whether *Min-Circuit* (or the other problems) are NP-complete. Kabanets and Cai [25] give evidence that such a reduction will not be straightforward.

The organization of this paper is as follows. In section 2 we define the relevant minimization problems and present necessary background. In section 3 we present our new proof that *Min-DNF* is NP-hard. In section 4 we present our hardness results

for approximating *Min-DNF*. In section 5 we give our construction of the instance of *Min-DNF* on which the greedy *Set Cover* algorithm produces an $\Omega(\log N)$ factor approximation. Section 6 concerns the fixed parameter versions of *Min-DNF*. Our hardness results for *Min-*AC^0_d appear in section 8. Conclusions are in section 9.

A preliminary version of this paper appeared in [3]. Feldman independently proved an $\Omega((\log N)^\delta)$ factor inapproximability result for *Min-DNF* [20] using related techniques. Feldman's result is based on the assumption P $\neq$ NP, rather than on the assumption that NP is not contained in quasi-polynomial time. Feldman also proved new results on proper learning of DNF, which are discussed in section 7.

2. Preliminaries. We begin with a few definitions. The set $\{1, \ldots, n\}$ is denoted by $[n]$. We use the bitwise ordering on vectors: For $v, w \in \{0,1\}^n$, we write $v \leq w$ if $v_i \leq w_i$ for all $i \in [n]$. Let $V_n = \{x_1, \ldots, x_n\}$. A *prime implicant* T of a function $f(x_1, \ldots, x_n)$ is a conjunction of literals over the variables V_n such that $T = 1 \Rightarrow f = 1$, and removing any literal from T violates this property. (In the literature, prime implicants are sometimes called *minterms*.) A *DNF formula* over the variables V_n is a formula $\phi = T_1 \vee T_2 \vee \ldots T_k$ for some k, where $T_1, \ldots, T_k$ are each conjunctions of literals over V_n. Each T_i in ϕ is a *term* of ϕ. Every Boolean function f can be expressed by a DNF formula in which every term is a prime implicant of f. The *size* of a DNF formula is the number of terms in it; for a Boolean function or partial function f, *dnf-size*(f) denotes the size of the smallest DNF formula consistent with f. The class of Boolean circuits AC^0_d consists of all depth-d circuits of AND and OR gates with arbitrary fan-in.

The classic *Set Cover* optimization problem is, given input $(\mathcal{S},\mathcal{U})$, where $\mathcal{U}$ is a finite universe and $\mathcal{S}$ is a collection of subsets of $\mathcal{U}$, find a smallest subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that that the union of the sets in $\mathcal{C}$ equals $\mathcal{U}$. It is NP-hard to approximate *Set Cover* to within a factor smaller than $c \log n$, where c is a constant and n is the size of the input (cf. [6]). On the other hand, there is a simple greedy algorithm that achieves an $O(\log n)$ approximation for *Set Cover* [24, 28, 13].

For r a positive integer, the r-*Uniform Set Cover* problem is as follows: On input $(n, k, \mathcal{S})$, where n and k are positive integers and $\mathcal{S}$ is a set of subsets of $[n]$, each subset having size r, determine whether there is a subcollection $\mathcal{C} \subseteq \mathcal{S}$ of size at most k whose union is $[n]$. The r-*Partite Set Cover* problem is a restriction: On input $(n, k, \Pi, \mathcal{S})$, where n and k are positive integers, Π is a partition of $[n]$ into r sets, and $\mathcal{S}$ is a collection of subsets of $[n]$, where every subset contains exactly one element from each of the sets of Π, determine whether there is a subcollection $\mathcal{C} \subseteq \mathcal{S}$ of size at most k whose union is $[n]$. The 3*D-Matching* problem is the NP-complete restriction of 3-*Partite Set Cover* where $k = n/3$ (cf. [21]).

We consider a general family of computational problems of the form *Min-R(S)* where the input is a Boolean function with input representation from S and the output should be a minimum representation of the function from R. For example, *Min-DNF(tt)* is the problem of determining a smallest DNF representation of a Boolean function f on n variables, if f is presented as a truth table of size $N = 2^n$. Our default input representation will be the truth table representation, and when we write *Min-R*, rather than *Min-R(S)*, we will assume the default input representation.

We focus primarily on DNF minimization. We consider the following four variations:

Min-DNF(A): The input is a total Boolean function, specified by explicitly listing all 1's of the function. That is, $A \subseteq \{0,1\}^n$ is the input, and we look for a minimum DNF that realizes the total function f_A, where $f_A(a) = 1$ for $a \in A$ and $f_A(b) = 0$ for $b \in \{0,1\}^n - A$.

Min-DNF: In the full-truth table version, the input is the entire truth table of $f : \{0,1\}^n \to \{0,1\}$, and we look for a minimum DNF that realizes the function f.

Min-DNF(A,B): The input is a partial Boolean function, specified by listing the 1's and 0's of the function, and we look for a minimum DNF that is consistent with the input. That is, $A, B \subseteq \{0,1\}^n$ is the input, and we look for a minimum DNF that realizes a function $f : \{0,1\}^n \to \{0,1\}$, where $f(a) = 1$ for $a \in A$ and $f(b) = 0$ for $b \in B$.

Min-DNF(*): The input is a partial Boolean function, specified by the entire truth table of $f : \{0,1\}^n \to \{0,1,*\}$, where $f(a) = *$ means that the value of f is not defined on a. We look for a minimum DNF that realizes a function $f' : \{0,1\}^n \to \{0,1\}$, where $f'(a) = 1$ for $a \in f^{-1}(1)$ and $f'(b) = 0$ for $b \in f^{-1}(0)$. Note that, as in the (A, B) version, the input here also specifies a partial function, but now the partial function is specified by a 2^n-sized input, regardless of the size of the domain of the partial function.

The decision versions of the above problems ask, given a function f and a natural number k, whether or not there is a DNF formula realizing f that has at most k terms. All decision versions are easily seen to lie in NP. It is also easy to see that *Min-DNF* is a special case of *Min-DNF(*)* and therefore reduces to *Min-DNF(*)*, and *Min-DNF(*)* reduces to *Min-DNF(A,B)*. Also *Min-DNF* reduces to *Min-DNF(A)*. Thus NP-hardness of *Min-DNF* implies NP-hardness of all other versions. The first three of the above problems are covered by Czort [15] in an excellent survey of previous related work. There is a hodgepodge of interesting but incomparable hardness results that are known for versions of DNF minimization, dating back to the 1960s. The simplest of these is the NP-hardness of the (A, B) version due to Pitt and Valiant [34]. As shown by Czort, there is also a clean NP-hardness proof of the A version that follows from a reduction of Gimpel. Masek [30] proved the NP-completeness of *Min-DNF*. In terms of inapproximability, Pitt and Valiant's proof of the (A, B) hardness result preserves solution values and thus shows the NP-hardness of achieving a factor n^ϵ approximation. Neither of the other two NP-hardness proofs (for the A version or for *Min-DNF*) give much in the way of inapproximability results.

A starting point for this paper is the well-known observation that *Min-DNF* easily reduces to *Set Cover* and in fact can be viewed as a special case of *Set Cover*. Given the truth table of a Boolean function f over n variables, all prime implicants of f can be generated in time $2^{O(n)}$. Each prime implicant can then be viewed as a subset of $\{0,1\}^n$ (corresponding to those inputs that satisfy the prime implicant). Thus given all of the prime implicants, finding a smallest DNF is equivalent to finding a smallest cover for these prime implicant sets. Applying the standard greedy algorithm for *Set Cover*, it follows that *Min-DNF* can be approximated to within a factor of $O(\log N)$, where N is the size (number of entries) of the truth table.

For a partial Boolean function f, the prime implicants of f are the prime implicants of the total function f' that satisfies $f'(\vec{x}) = 1$ iff $f(\vec{x}) = 1 \vee f(\vec{x}) = *$. Every partial function f has a smallest consistent DNF whose terms are prime implicants of f. The greedy *Set Cover* algorithm can also be used to approximate *Min-DNF(*)* in the same way that it is applied to *Min-DNF*, except that it chooses sets that cover the maximum number of 1's of the input function (i.e., it ignores $*$'s when greedily choosing sets).

The pseudocode for applying the greedy *Set Cover* algorithm to *Min-DNF* and *Min-DNF(*)* is shown in Figure 2.1. The input is the full truth table of a Boolean function or partial function f.

1: $\mathcal{T} := \{T \mid T \text{ is a prime implicant of } f\}$
2: $\varphi := \bot$
3: **while** φ does not cover all 1's of f **do**
4: let $T \in \mathcal{T}$ cover the most uncovered 1's of f
5: $\varphi := \varphi \vee T$
6: **end while**
7: return φ

FIG. 2.1. *Greedy Min-DNF and Min-DNF(*) algorithm.*

3. Simple proof that *MinDNF* is NP-complete. Our new proof that *Min-DNF* is NP-complete is a modification of the reduction of Gimpel mentioned above, which was was used by Czort to prove the NP-completeness of the A version of DNF minimization [15].

We start by briefly describing Gimpel's reduction. It can be viewed as consisting of two phases. In the first phase, an instance $(\mathcal{S},\mathcal{U})$ of *Set Cover* over the ground set $\mathcal{U} = [n]$ is mapped to a partial function f, as follows. First, both the sets as well as the ground elements are mapped to truth assignments in $\{0,1\}^n$ such that a set covers a ground element in $[n]$ iff the assignment corresponding to the ground element is less than the assignment corresponding to the set (where comparison of assignments is with respect to the bitwise ordering of the vectors). Each ground element $i \in [n]$ is mapped to the assignment that is all zero except for bit i, which is 1. Each set is mapped to the assignment corresponding to the characteristic function of the set. The 1's of f are those assignments corresponding to ground elements; the $*$'s of f are those assignments α such that $\alpha \leq \beta$ for some β corresponding to a set; and the remaining truth assignments are zeros of f. It can be shown that the size of the minimum DNF consistent with the partial function f is equal to the minimum size of the cover for the input instance of *Set Cover*.

In the second phase of Gimpel's reduction, the partial function f is mapped to a total function, g. We give the details of g below in section 3.2. The truth table sizes of f and g are exponential in the size of the *Set Cover* instance from the first phase. Thus Gimpel's reduction does not give a hardness result for *Min-DNF*. As Czort notes, it does, however, give a hardness result for *Min-DNF(A)*, provided that we begin the reduction not from the general *Set Cover* problem but from 3-*Uniform Set Cover*.

Our reduction proving that *Min-DNF* is NP-complete also has two phases. The first phase is similar to that of Gimpel. The main difference is that we need a much more compact mapping from the sets and ground elements of the *Set Cover* instance onto truth assignments, to ensure that the size of the truth table for the resulting function is only polynomial in the size of the input *Set Cover* instance. To do such a compact mapping in a simple way, we reduce from 3-*Partite Set Cover* rather than from 3-*Uniform Set Cover*. The second phase of our reduction is essentially identical to Gimpel's.

3.1. Reducing 3-*Partite Set Cover* to *Min-DNF*(*). In the first phase of our reduction, we reduce 3-*Partite Set Cover* to *Min-DNF(*)*. We note that our reduction from 3-*Partite Set Cover* would also work from 3*D-Matching*. We use the following lemma, which is implicit in Gimpel's reduction.

LEMMA 3.1. *Let $\mathcal{S}$ be a set of subsets of $[n]$. Let $t > 0$, and let $V = \{v^i : i \in [n]\}$ and $W = \{w^A : A \in \mathcal{S}\}$ be sets of vectors from $\{0,1\}^t$ satisfying*

$$(*) \qquad \text{For all } A \in \mathcal{S} \text{ and } i \in [n],\ i \in A \text{ iff } v^i \leq w^A.$$

Let $R = \{x \in \{0,1\}^t \mid x \notin V \text{ and for some } w \in W,\ x \leq w\}$. Let f be a partial

function with domain $\{0,1\}^t$ *such that* $f(x) = 1$ *if* $x \in V$, $f(x) = *$ *if* $x \in R$, *and* $f(x) = 0$ *otherwise. Then* $\mathcal{S}$ *has a cover of size* m *iff there is an* m*-term DNF consistent with* f.

Proof. For $u \in \{0,1\}^t$, let $D(u) = \{w : w \leq u\}$, and let $\tau(u)$ denote the DNF term $\bigwedge_{i:u_i=0} \neg x_i$. Note that $D(u)$ is exactly the set of satisfying assignments of $\tau(u)$. For a set U of vectors $D(U) = \bigcup_{u \in U} D(u)$. By (*), we have that $V \subseteq D(W)$. Also, $f(x) = *$ iff $x \in D(W) - V$.

Given a cover $\mathcal{C} \subseteq \mathcal{S}$ of size m, the m-term DNF whose terms are $\{\tau(w^C) \mid C \in \mathcal{C}\}$ is easily seen to be consistent with f. Conversely, suppose ϕ is an m-term DNF consistent with f. For each term $\tau \in \phi$, let $u(\tau)$ be the maximal vector satisfying τ. Since ϕ is consistent with f, we have that $u(\tau) \in D(W)$, so there must be a set $S(\tau) \in \mathcal{S}$ for which $u(\tau) \leq w^{S(\tau)}$. We claim that $\{S(\tau) : \tau \in \phi\}$ is a cover of $\mathcal{S}$. Let $j \in [n]$. We must show that j is covered. The consistency of ϕ implies that v^j is satisfied by some term $\tau_j \in \phi$. This implies $v^j \leq u(\tau_j)$. Thus $v^j \leq w^{S(\tau_j)}$, which by (*) implies $j \in S(\tau_j)$. ∎

The reduction from 3-*Partite Set Cover* to *Min-DNF(*)* is given in the following lemma.

LEMMA 3.2. *There is an algorithm that takes as input an instance* $(n, k, \Pi, \mathcal{S})$ *of* 3*-Partite Set Cover and outputs an instance of Min-DNF(*). The instance of Min-DNF(*) defines a partial function* f *on* $O(\log n)$ *variables such that the size of the smallest DNF consistent with* f *is equal to the size of the smallest cover for the input* 3*-Partite Set Cover instance. The algorithm runs in time polynomial in* n.

Proof. Given an input instance $(n, k, \Pi, \mathcal{S})$ of 3-*Partite Set Cover*, the algorithm produces an indexed set of vectors $V = \{v^i : i \in [n]\}$ and $W = \{w^A : A \in \mathcal{S}\}$ all of the same (small) length t satisfying the condition (*) of Lemma 3.1. We will specify V and then define W according to the rule that, for $A \in \mathcal{S}$, w^A is the bitwise OR of $\{v^i : i \in A\}$. This guarantees the forward implication of condition (*) for any choice of V; it is the backward implication that requires some care in choosing V.

Let q be the smallest integer such that $\binom{q}{q/2} \geq n$. Thus $q = O(\log n)$. Assign to each $i \in [n]$ a unique q-bit Boolean vector $b(i)$ containing exactly $q/2$ 1's. For $i \in [n]$, write $\Pi(i)$ for the index of the block of Π that contains i. Let $t = 3q$. We will consider the t-bit vectors in V and W as being divided into 3 blocks of size q. For $i \in [n]$, let v^i be equal to 0 on all blocks but block $\Pi(i)$; on block $\Pi(i)$ it is $b(i)$. To see that the backward implication of $(*)$ holds, let $A \in \mathcal{S}$ and $i \in [n]$, and assume that $v^i \leq w^A$. Then A contains one element i' with $\Pi(i') = \Pi(i)$, and so we must have $b(i) \leq b(i')$, which implies $i = i'$.

V and W can be generated in time $n^{O(1)}$. The partial function f will have domain $\{0,1\}^t$. The lemma then follows immediately from Lemma 3.1. ∎

3.2. Reducing *Min-DNF(*)* to *Min-DNF*. As mentioned above, the second phase of our reduction is taken from Gimpel. We describe the phase here and will build on it later in order to prove inapproximability results. The second phase of Gimpel's reduction maps a partial function f to a total function g. The variables underlying g are V (the variables of f) plus two additional variables y_1 and y_2. The total function g is defined as follows:

$$g(\vec{x}\, y_1 y_2) = \begin{cases} 1 & \text{if } f(\vec{x}) = 1 \text{ and } y_1 = y_2 = 1, \\ 1 & \text{if } f(\vec{x}) = * \text{ and } y_1 = y_2 = 1, \\ 1 & \text{if } f(\vec{x}) = *,\ y_1 = p(\vec{x}), \text{ and } y_2 = \neg p(\vec{x}), \\ 0 & \text{otherwise,} \end{cases}$$

where $p(\vec{x}) = 0$ if the parity of $\vec{x}$ is even, and $p(\vec{x}) = 1$ if the parity of $\vec{x}$ is odd. Let $s = |f^{-1}(*)|$. The following lemma is implicit in Gimpel's reduction (cf. [15]).

LEMMA 3.3. *dnf-size*(g) = *dnf-size*$(f) + s$.

Proof. The idea behind the proof is as follows. The key observation is that every DNF for g requires s distinct terms to cover the inputs of the third type in the definition of g above; these terms can simultaneously cover all inputs of the second type but not those of the first type. The remaining terms of the DNF must therefore cover the terms of the first type and may optionally cover the terms of the second type; they thus constitute a solution to the *Min-DNF(*)* problem for f. It follows that *dnf-size*(g) = *dnf-size*$(f) + s$. We now prove this formally.

We first show that *dnf-size*$(g) \leq$ *dnf-size*$(f) + s$. Suppose φ is a minimum-size DNF consistent with f. Define a DNF ψ with terms of two types: First, for every input $\vec{x} \in f^{-1}(*)$, ψ contains the term $\left(\bigwedge_{i:\vec{x}_i=1} x_i\right) \wedge \left(\bigwedge_{i:\vec{x}_i=0} \neg x_i\right) \wedge y_{2-p(\vec{x})}$. These terms cover all inputs of the second and third types in the definition of g. Second, for every term T of φ, ψ contains the term $T \wedge y_1 \wedge y_2$. These terms cover all inputs of the first type in the definition of g.

Finally, suppose that $\vec{x}y_1y_2$ satisfies ψ. Then one of the following three conditions holds: (1) $\vec{x} \in f^{-1}(*)$, $y_1 = p(\vec{x})$, and $y_2 = \neg p(\vec{x})$; (2) $\vec{x} \in f^{-1}(*)$ and $y_1 = y_2 = 1$; (3) $\vec{x}$ satisfies φ (and thus $\vec{x} \in f^{-1}(1) \cup f^{-1}(*)$) and $y_1 = y_2 = 1$. In all three cases we have $g(\vec{x}y_1y_2) = 1$, and thus ψ is consistent with g. The number of terms in ψ is $|f^{-1}(*)| + |\varphi|$ = *dnf-size*$(f) + s$.

We now show that *dnf-size*$(g) \geq$ *dnf-size*$(f) + s$. Suppose that ψ is a smallest DNF for g. We assume without loss of generality that each term of ψ is a prime implicant of g. We begin by proving that, for every $\vec{x} \in f^{-1}(*)$, ψ contains the term $t(\vec{x}) \wedge y_{2-p(\vec{x})}$, where $t(\vec{x}) = \left(\bigwedge_{i:\vec{x}_i=1} x_i\right) \wedge \left(\bigwedge_{i:\vec{x}_i=0} \neg x_i\right)$. The proof is as follows. Let $\vec{x} \in f^{-1}(*)$, and suppose that the parity of $\vec{x}$ is odd: The case of even parity is symmetric. Let T be a term of ψ that is satisfied by $\vec{x}10$ (where 1 and 0 are the values of y_1 and y_2, respectively). If, for some index i, T does not contain the variable x_i, let $\vec{x}'$ be obtained by flipping the ith bit of $\vec{x}$. Then $\vec{x}'10$ falsifies g (since $\vec{x}'$ has even parity) but satisfies T, contradicting the assumption that ψ is consistent with g. Thus T contains each of the variables $x_1, \ldots, x_n$. In addition, T contains the variable y_1, as otherwise $\vec{x}00$ would satisfy T. Finally, since T is a prime implicant of g, we have that $T = t(\vec{x})y_1$.

We now prove that there exists a subformula $\hat{\psi}$ of ψ and a DNF ψ' over the $\vec{x}$ variables that is consistent with f such that $\hat{\psi} = \bigvee_{T \in \psi'} (T \wedge y_1 \wedge y_2)$. Let $\hat{\psi}$ be the subformula of ψ consisting of those terms that are satisfied by $\vec{x}11$ for some $\vec{x} \in f^{-1}(1)$. Each term of $\hat{\psi}$ contains $y_1 \wedge y_2$, since flipping y_1 or y_2 produces an input that falsifies g. It follows that $\hat{\psi} = \bigvee_{T \in \psi'} (T \wedge y_1 \wedge y_2)$, where ψ' is a DNF. It remains to show that ψ' is consistent with f. For every $\vec{x} \in f^{-1}(1)$, there is a term of ψ satisfied by $\vec{x}11$, and thus there is a corresponding term of ψ' satisfied by $\vec{x}$. On the other hand, every $\vec{x} \in f^{-1}(0)$ must falsify ψ', as otherwise $\vec{x}11$ would satisfy ψ.

It follows from the above that ψ consists of the terms $t(\vec{x}) \wedge y_{2-p(\vec{x})}$ for each $\vec{x} \in f^{-1}(*)$, together with the subformula $\hat{\psi}$. These components are pairwise disjoint. Since ψ' is consistent with f, $\hat{\psi}$ contains at least *dnf-size*(f) terms, and thus the size of ψ is at least *dnf-size*$(f) + s$. □

It follows that there is a polynomial-time reduction from *Min-DNF(*)* to *Min-DNF*. Combining this with the previous reduction from 3-*Partite Set Cover* to *Min-DNF(*)*, it follows that *Min-DNF* is NP-complete.

4. On the approximability of *Min-DNF*. Although the two-phase reduction above proves the NP-completeness of *Min-DNF*, it does not give us inapproximability results. There are two problems. First, the reduction begins with an instance of 3-*Partite Set Cover*, a problem that can be approximated in polynomial time to within a factor of 4/3 [18]; to obtain inapproximability results we need to reduce from a problem that is difficult to approximate. Also, the second phase of the reduction, from *Min-DNF(*)* to *Min-DNF*, is not approximation-preserving.

We replace the first phase of the reduction with a reduction that exploits properties of the *Set Cover* instance obtained by the inapproximability results of Lund–Yannakakis [29] that are based on probabilistically-checkable proofs (PCPs). We then modify the second phase to make it approximation-preserving. The final two-phase reduction gives an inapproximability factor of $\Omega((\log N)^{\gamma})$ assuming that NP is not contained in quasi-polynomial time.

In Appendix A we also present a modified version of the first phase of the reduction, which reduces from *r-Uniform Set Cover* rather than from 3-*Partite Set Cover*. This allows us to obtain an inapproximability result for *Min-DNF* by applying known inapproximability results for *r-Uniform Set Cover*. However, the result we obtain for *Min-DNF* is weak (inapproximability to within a factor of $\Omega(\log\log N)$). Nevertheless, the reduction itself may be of independent interest, since it requires a different technique to reduce from *r-Uniform Set Cover* rather than from *r-Partite Set Cover*.

4.1. New reduction to *Min-DNF*(*). In this section we present a reduction that follows the PCP-based inapproximability results for *Set Cover* [29, 19]. We will closely follow the Lund–Yannakakis reduction, as presented by Khot [26].

An instance of *Label Cover* is denoted by $\mathcal{L} = (G, L_1, L_2, \Pi)$, where $G = (V, W, E)$ is a regular bipartite graph, L_1 and L_2 are sets of labels, and $\Pi = \{\pi_{vw}\}_{(v,w)\in E}$ denotes the constraints on each edge. For every edge $(v, w) \in E$ we have a map $\pi_{vw} : L_1 \to L_2$. A labeling $l : V \to L_1$, $W \to L_2$ satisfies the constraint on an edge (v, w) if $\pi_{vw}(l(v)) = l(w)$. Given an instance $\mathcal{L}$, the output should be a labeling that satisfies the maximum fraction $\mathrm{OPT}(\mathcal{L})$ of edge constraints.

THEOREM 4.1 (see [29, 26]). *There is a constant $c < 1$ such that it is NP-hard to solve the following gap version of Label Cover. The input is an instance $\mathcal{L} = (G = (V, W, E), [7], [2], \{\pi_{vw}\}_{(v,w)\in E})$ of Label Cover. The instance should be accepted if $OPT(\mathcal{L}) = 1$, and the instance should be rejected if $OPT(\mathcal{L})$ is at most c.*

Note that the reduction is from *Max3SAT(5)* (the problem of maximizing the number of satisfied clauses in a 3CNF formula where each variable occurs in exactly five clauses). The vertices in V correspond to the m clauses, and the vertices in W correspond to the n variables. Using Raz's parallel repetition theorem [35], we can amplify the gap, obtaining, for any positive integer k, an instance $\mathcal{L}' = (G' = (V', W', E'), [7^k], [2^k], \{\pi_{v'w'}\}_{(v',w')\in E'})$, where $|V'| = |V|^k$ and $|W'| = |W|^k$, such that $\mathrm{OPT}(\mathcal{L}) = 1$ implies $\mathrm{OPT}(\mathcal{L}') = 1$, and $\mathrm{OPT}(\mathcal{L}) \le c$ implies $\mathrm{OPT}(\mathcal{L}') \le 2^{-\gamma k}$, where $\gamma > 0$ is an absolute constant. Note that the sizes of both V' and W' are $n^{O(k)}$, where n is the number of variables in the *Max3SAT(5)* instance.

DEFINITION 4.2. *A partition system $\mathcal{P}(m, h, t)$ consists of t partitions $(A_1, \overline{A_1}), \ldots (A_t, \overline{A_t})$ of $[m]$, with the property that no collection of h sets, with at most one set from each partition, covers all of $[m]$.*

LEMMA 4.3 (see [29]). *For every h and t, there is an efficiently constructible partition system $\mathcal{P}(m, h, t)$, with $m = O(2^h h \log t)$.*

We now review the reduction from the *Label Cover* instance $\mathcal{L}'$ to a *Set Cover* instance $(\mathcal{S}, \mathcal{U})$. First, the universe $\mathcal{U}$ is as follows. Let $t = 2^k$, let h be a parameter to

be determined later, and let $m = m(t,h) = O(2^h h \log t)$ be the parameter specified by Lemma 4.3. For each edge $e \in E'$ we associate a subuniverse $\mathcal{U}_e = \{(e,i) \mid i \in [m]\}$. The entire universe $\mathcal{U}$ is the disjoint union of these $|E'|$ subuniverses. Associated with each edge e is a partition system $\mathcal{P}(m,h,t)$ over $\mathcal{U}_e$, with one partition associated with each of the possible labels in L_2. Thus each label $b \in [t]$ corresponds to a partition $(A_b^e, \overline{A_b^e})$ of $\mathcal{U}_e$. The size of the entire universe is $n^{O(k)}2^{O(h)}$. The set system $\mathcal{S}$ is the union of two collections of sets: $S(v,a)$, for each vertex $v \in V'$ and each label $a \in [7^k]$, and $S(w,b)$, for each $w \in W'$ and each label $b \in [2^k]$. In particular,

$$S(v,a) = \bigcup_{w:(v,w)\in E'} A^{(v,w)}_{\pi_{vw}(a)}, \qquad S(w,b) = \bigcup_{v:(v,w)\in E'} \overline{A_b^{(v,w)}}.$$

The following lemma is implicit in [29, 26].

LEMMA 4.4 (see [29, 26]). *If $OPT(\mathcal{L}') = 1$, then $(\mathcal{S},\mathcal{U})$ has a cover of size $|V'| + |W'|$. If $OPT(\mathcal{L}') \le 1/(2h^2)$, then every cover of $(\mathcal{S},\mathcal{U})$ has size at least $h(|V'| + |W'|)/16$.*

Choosing $h \le 2^{\gamma k/2 - 1/2}$, we obtain a gap of $h/16$ for the *Set Cover* instance from the $2^{\gamma k}$ gap of the *Label Cover* instance. For $k = O(\log\log n)$ sufficiently large, we have $|\mathcal{U}| = 2^{O(h)}$, and thus the gap is $\Omega(\log|\mathcal{U}|)$. The size of the *Set Cover* instance is quasi-polynomial in n. Thus a polynomial-time, $(h/16)$-approximation algorithm for *Set Cover* could distinguish between the cases $\mathrm{OPT}(\mathcal{L}') = 1$ and $\mathrm{OPT}(\mathcal{L}') \le 2^{-\gamma k}$ in time $2^{\mathrm{polylog}(n)}$, implying that NP is contained in $DTIME(2^{\mathrm{polylog}(n)})$.

We now show how to reduce instances of *Set Cover* of the above form to *Min-DNF(*)* instances. By the observations in section 3.1 it suffices to define three sets of vectors $\{u_{e,i} \mid (e,i) \in \mathcal{U}\}$, $\{t_{v,a} \mid v \in V',\ a \in L_1\}$, and $\{t_{w,b} \mid w \in W',\ b \in L_2\}$ such that the following conditions hold: (1) $u_{e,i} \le t_{v,a}$ iff $(e,i) \in S(v,a)$, for all $(e,i) \in \mathcal{U}$, $v \in V$, and $a \in L_1$, and (2) $u_{e,i} \le t_{w,b}$ iff $(e,i) \in S(w,b)$, for all $(e,i) \in \mathcal{U}$, $w \in W$, and $b \in L_2$. Let $r \in O(\log|V'|)$ be such that $\binom{r}{r/2} \ge \max(|V'|,|W'|)$. Our function will have variables $\{x_1,\ldots,x_r\} \cup \{x'_1,\ldots,x'_r\} \cup \{y_a \mid a \in L_1\} \cup \{y'_b \mid b \in L_2\}$. Thus the number of variables is $O(\log|V'| + |L_1| + |L_2|) = O(k\log n + 7^k)$.

We assign to each $v \in V'$ a unique set $S_v \subseteq \{1,\ldots,r\}$ of size $r/2$; similarly each $w \in W'$ is assigned a unique set $S_w \subseteq \{1,\ldots,r\}$ of size $r/2$. For each $v \in V'$ and $a \in L_1$, we define a Boolean vector $t_{v,a}$ as follows. The vector $t_{v,a}$ has zeros corresponding to those variables x_i such that $i \in S_v$, and it has a zero corresponding to y_a. The remaining bits of $t_{v,a}$ are ones. We similarly define, for each $w \in W'$ and $b \in L_2$, a Boolean vector $t_{w,b}$ having zeros corresponding to those variables x'_i such that $i \in S_w$, and a zero corresponding to y'_b and whose remaining bits are ones.

We now describe, for each $(e,i) \in \mathcal{U}$, a Boolean vector $u_{e,i}$. Suppose that $e = (v,w)$, and let $S(v,a_1),\ldots,S(v,a_k)$ and $S(w,b_1),\ldots,S(w,b_\ell)$ be all of the sets in $\mathcal{S}$ containing (e,i). Then $u_{e,i}$ has zeros in the positions corresponding to the following variables: (1) variables x_i, where $i \in S_v$, (2) variables x'_i, where $i \in S_w$, (3) variables y_{a_i}, where $1 \le i \le k$, and (4) variables y'_{b_i}, where $1 \le i \le \ell$. The remaining bits of $u_{e,i}$ are ones.

LEMMA 4.5. *For all $(e,i) \in \mathcal{U}$, $v \in V$, $w \in W$, $a \in L_1$, and $b \in L_2$, the following conditions hold: $u_{e,i} \le t_{v,a}$ iff $(e,i) \in S(v,a)$ and $u_{e,i} \le t_{w,b}$ iff $(e,i) \in S(w,b)$.*

Proof. Suppose first that $(e,i) \in S(v,a)$, where $v \in V'$ and $a \in L_1$. Then $e = (v,w)$ for some vertex $w \in W'$. The zeros of $t_{v,a}$ are in positions corresponding to variables x_i, where $i \in S_v$, and in the position corresponding to y_a. Since $e = (v,w)$, the vector $u_{e,i}$ has zeros in the positions corresponding to variables x_i, where $i \in S_v$, and since $(e,i) \in S(v,a)$ the vector $u_{e,i}$ also has a zero in the position corresponding

to y_a. Thus $u_{e,i} \leq t_{v,a}$. The case where $(e,i) \in S(w,b)$, where $w \in W'$ and $b \in L_2$, is symmetric.

Now suppose that $(e,i) \notin S(v,a)$, where $v \in V'$ and $a \in L_1$. Suppose that $e = (v', w')$. If $v' \neq v$, then there exists an index $j \in S_v \setminus S_{v'}$; $u_{e,i}$ has a one in the position corresponding to x_j, while $t_{v,a}$ has a zero in the same position, and thus $u_{e,i} \not\leq t_{v,a}$. So assume that $v' = v$. By the definition of $t_{v,a}$, we know that $t_{v,a}$ has a zero in the position corresponding to y_a. But since $e = (v, w')$, $u_{e,i}$ has a zero in this position iff $(e,i) \in S(v,a)$; as we have supposed that $(e,i) \notin S(v,a)$, it follows that the position in $u_{e,i}$ corresponding to y_a is set to one. Thus $u_{e,i} \not\leq t_{v,a}$. The case where $(e,i) \notin S(w,b)$, where $w \in W'$ and $b \in L_2$, is symmetric. □

By the results of section 3.1, the vectors $u_{e,i}$, $t_{v,a}$, and $t_{w,b}$ yield an instance of *Min-DNF(*)* on $O(k \log n + 7^k)$ variables whose optimum is equal to the optimum for the instance $(\mathcal{S}, \mathcal{U})$ of *Set Cover*.

THEOREM 4.6. *If NP $\not\subseteq$ DTIME($2^{\text{polylog}(n)}$), then there exists an absolute constant $\delta > 0$ such that no polynomial-time algorithm achieves an approximation ratio better than $(\log N)^{\delta}$ for Min-DNF(*), where N is the size of the input truth table.*

Proof. Let f be the partial function specified by our reduction. Claims 4.4 and 4.5, together with the results of section 3.1, imply that our *Min-DNF(*)* instance has the following properties: If $\text{OPT}(\mathcal{L}') = 1$, then *dnf-size*$(f) \leq |V'| + |W'|$; if $\text{OPT}(\mathcal{L}') \leq 2^{-\gamma k}$, then *dnf-size*$(f) \geq h(|V'| + |W'|)/16$, where $h = \Omega(2^{\gamma k/2})$. Let us take $k = \log\log n$, and thus $h = \Omega((\log n)^{\gamma/2})$. Let N be the size of the truth table for f. The number of variables of f is $\log N = O(k \log n + 7^k) = O((\log n)^{\log 7})$, and thus the gap is $h/16 = \Omega((\log N)^{\gamma/(2\log 7)})$. The truth table has size $2^{\text{polylog}\, n}$ and can be generated in time polynomial in its size. The theorem follows by taking $\delta = \gamma/(2 \log 7)$. □

4.2. Approximation-preserving reduction from *Min-DNF(*)* to *Min-DNF*. We modify the reduction from section 3.2 to make it approximation-preserving. Let f be a partial Boolean function over variables $x_1, \ldots, x_n$. Let $s = |f^{-1}(*)|$. We construct a new total function g' such that *dnf-size*$(g') = s \cdot$ *dnf-size*$(f) + s = s \cdot$ (*dnf-size*$(f) + 1$). Let $t = n + 1$, and let $S \subseteq \{0,1\}^t$ be a collection of s vectors, each containing an odd number of 1's. We add t new variables $z_1, \ldots, z_t$ and define

$$g'(\vec{x}\, y_1 y_2\, \vec{z}) = \begin{cases} 1 & \text{if } f(\vec{x}) = 1,\ y_1 = y_2 = 1, \text{ and } \vec{z} \in S, \\ 1 & \text{if } f(\vec{x}) = * \text{ and } y_1 = y_2 = 1, \\ 1 & \text{if } f(\vec{x}) = *,\ y_1 = p(\vec{x}), \text{ and } y_2 = \neg p(\vec{x}), \\ 0 & \text{otherwise.} \end{cases}$$

LEMMA 4.7. *dnf-size*$(g') = s \cdot$ *dnf-size*$(f) + s$.

Proof. For binary vector $\vec{w}$, we use $t(\vec{w})$ to denote the term $\left(\bigwedge_{i:w_i=1} w_i\right) \wedge \left(\bigwedge_{i:w_i=0} \neg w_i\right)$.

We first show that *dnf-size*$(g') \leq s \cdot$ *dnf-size*$(f) + s$. Suppose that φ is a smallest DNF consistent with f. Define a DNF ψ with terms of the following two types. First, for every input $\vec{x} \in f^{-1}(*)$, ψ contains the term $t(\vec{x}) \wedge y_{2-p(\vec{x})}$. These terms cover all inputs of the second and third types in the definition of g'. Second, for every term T of φ and every vector $\vec{z} \in S$, ψ contains the term $T \wedge y_1 \wedge y_2 \wedge t(\vec{z})$. These terms cover all inputs of the first type in the definition of g'. Finally, suppose that $\vec{x} y_1 y_2 \vec{z}$ satisfies ψ. Then one of the following conditions holds: (1) $\vec{x} \in f^{-1}(*)$, $y_1 = p(\vec{x})$, and $y_2 = \neg p(\vec{x})$; (2) $\vec{x} \in f^{-1}(*)$ and $y_1 = y_2 = 1$; (3) $\vec{x}$ satisfies φ (and thus $\vec{x} \in f^{-1}(1) \cup f^{-1}(*)$), $y_1 = y_2 = 1$, and $\vec{z} \in S$. In all three cases we have $g'(\vec{x} y_1 y_2 \vec{z}) = 1$, and thus ψ is consistent with g'. The number of terms in ψ is $|f^{-1}(*)| + |\varphi| \cdot |S| = s \cdot$ *dnf-size*$(f) + s$.

We next show that *dnf-size*$(g') \geq s \cdot$ *dnf-size*$(f) + s$. Suppose that ψ is a smallest DNF for g'. The same reasoning used in the proof of Lemma 3.3 shows that, for every $\vec{x} \in f^{-1}(*)$, ψ contains the term $t(\vec{x}) \wedge y_{2-p(\vec{x})}$. We now argue that, for each $\vec{z} \in S$, there exists a subformula $\psi_{\vec{z}}$ of ψ and a DNF $\psi'_{\vec{z}}$ over the $\vec{x}$ variables and consistent with f such that $\psi_{\vec{z}} = \bigvee_{T \in \psi'_{\vec{z}}} \big(T \wedge y_1 \wedge y_2 \wedge t(\vec{z})\big)$. Let $\vec{z} \in S$, and let $\psi_{\vec{z}}$ be the subformula of ψ consisting of those terms that are satisfied by $\vec{x}11\vec{z}$ for some $\vec{x} \in f^{-1}(1)$. Each term of $\psi_{\vec{z}}$ contains $y_1 \wedge y_2 \wedge t(\vec{z})$, since flipping either y_1 or y_2, or any bit of $\vec{z}$, produces an input that falsifies g'. It follows that $\psi_{\vec{z}} = \bigvee_{T \in \psi'_{\vec{z}}} (T \wedge y_1 \wedge y_2 \wedge t(\vec{z}))$, where $\psi'_{\vec{z}}$ is a DNF. It remains to show that $\psi'_{\vec{z}}$ is consistent with f. For every $\vec{x} \in f^{-1}(1)$, there is a term of ψ that is satisfied by $\vec{x}11\vec{z}$, and thus there is a corresponding term of $\psi'_{\vec{z}}$ that is satisfied by $\vec{x}$. On the other hand, every $\vec{x} \in f^{-1}(0)$ must falsify $\psi'_{\vec{z}}$, as otherwise $\vec{x}11\vec{z}$ would satisfy ψ.

It follows from the above that ψ consists of the terms $t(\vec{x}) \wedge y_{2-p(\vec{x})}$ for each $\vec{x} \in f^{-1}(*)$ and of the subformulas $\psi_{\vec{z}}$ for each $\vec{z} \in S$. These components are pairwise disjoint. Since $\psi'_{\vec{z}}$ is consistent with f, it follows that $\psi_{\vec{z}}$ contains at least *dnf-size*(f) terms, and thus the size of ψ is at least $s \cdot$ *dnf-size*$(f) + s$. □

The results of section 4.2, together with Theorem 4.6, yield the following hardness result for *Min-DNF*.

THEOREM 4.8. *If NP* $\not\subseteq$ *DTIME*$(2^{\text{polylog}(n)})$, *then there exists a constant* $\gamma > 0$ *such that no polynomial-time algorithm achieves an approximation ratio better than* $(\log N)^{\gamma}$ *for Min-DNF, where* N *is the size of the truth table.*

4.3. An improved hardness result under additional assumptions. In this section, we prove an $\Omega(\sqrt{\log N})$ hardness of approximation result for *Min-DNF* under the additional assumption that a restriction of *Set Cover* is $\Omega(\log n)$-hard to approximate.

DEFINITION 4.9. *The* f*-Frequency Bounded Set Cover problem is the restriction of Set Cover to instances where each element occurs in at most* $f(n)$ *sets, where* n *is the total size of the instance.*

It is well known [23] that a factor f approximation for *f-Frequency Bounded Set Cover* can be obtained in polynomial time. Thus for $f = o(\log n)$, *f-Frequency Bounded Set Cover* is not as hard to approximate as the general *Set Cover* problem. On the other hand, the reduction of Lund and Yannakakis showing an $\Omega(\log n)$ hardness of approximation for *Set Cover* produces an instance of $\Omega((\log n)^c)$-*Frequency-Bounded-Set-Cover*, for some constant c, which implies an $\Omega(\log n)$ hardness result for that problem. We conjecture that *f-Frequency-Bounded-Set-Cover* is NP-hard to approximate within a factor better than $c_2 \ln n$ for $f = c_1 \ln n$ and some constants c_1, c_2. Resolving this conjecture is an interesting question in its own right, since it postulates a frequency threshold (within a constant factor) for hardness. Assuming that the conjecture holds, we can prove an $\Omega(\sqrt{\log N})$ hardness of approximation result for *Min-DNF* using a simple, randomized reduction.

THEOREM 4.10. *If there exist constants* c_1 *and* c_2 *such that it is* ***NP****-hard to approximate* $(c_1 \ln n)$*-Frequency Bounded Set Cover to within* $c_2 \ln n$, *then there exists a constant* c_3 *such that no polynomial-time algorithm for Min-DNF achieves an approximation ratio better than* $c_3\sqrt{\log N}$ *unless* ***NP*** $\subseteq$ *DTIME*$(2^{\text{polylog}(n)})$.

Proof. Assume that there exist constants c_1 and c_2 as in the lemma. We prove an $\Omega(\sqrt{\log N})$ hardness of approximation for *Min-DNF(*)*; the reduction from section 4.2 extends the same result to *Min-DNF*. Let $(\mathcal{S}, \mathcal{U})$ be an instance of $(c_1 \ln n)$-*Frequency Bounded Set Cover* of size n. The idea of the reduction is as follows. First, we will map each set $S \in \mathcal{S}$ to a subset $f(S) \subseteq [b]$, for a suitably chosen parameter

b. Second, we define vectors $w^S \in \{0,1\}^b$ for each $S \in \mathcal{S}$, by letting w^S have zeros in those positions contained in $f(S)$ and ones elsewhere. Finally, we define vectors $u^x \in \{0,1\}^b$ for each $x \in \mathcal{U}$ having zeros in the positions contained in F_x, where

$$F_x = \bigcup_{S \in \mathcal{S}: x \in S} f(S),$$

and ones elsewhere. If the vectors satisfy the condition $u^x \leq w^S \iff x \in S$ for all $x \in \mathcal{U}$ and $S \in \mathcal{S}$, then by Lemma 3.1 we can construct an instance of *Min-DNF(*)* over b variables whose optimum is equal to the optimum for $(\mathcal{S},\mathcal{U})$. Notice that the definition of u^x implies that the "if" part of the condition is always satisfied. For the "only if" part to hold, it is necessary and sufficient that, for every S such that $x \notin S$, u_x has a one in a position where w^S has a zero; that is, $f(S) \not\subseteq F_x$. For $S \in \mathcal{S}$, let $f(S)$ be defined by choosing each $i \in [b]$ independently with probability p. Fix an element $x \in \mathcal{U}$ and a set $S \in \mathcal{S}$ such that $x \notin S$. We will show that the probability that $f(S) \subseteq F_x$ is small. For any choice of p, the probability that $f(S) \subseteq F_x$ is maximized when x occurs in exactly $c_1 \ln n$ sets (since it cannot occur in more than $c_1 \ln n$ sets). As we wish to find an upper bound for the probability that $f(S) \subseteq F_x$, we may therefore assume that x occurs in exactly $c_1 \ln n$ sets. For $1 \leq i \leq b$, let X_i be the indicator variable for the event $i \in F_x$. Then $\mathbf{E}[X_i] = 1 - (1-p)^{c_1 \ln n}$, and by letting $X = \sum_{1 \leq i \leq b} X_i$ be the size of F_x, linearity of expectation implies

$$\begin{aligned}\mathbf{E}[X] &= b(1 - (1-p)^{c_1 \ln n}) \\ &\approx b(1 - e^{-pc_1 \ln n}),\end{aligned}$$

which can be made smaller than $b/4$ by choosing $p \approx \ln(4/3)/(c_1 \ln n)$. The X_i's are independent, and we apply the simplified Chernoff bound $\Pr[X > (1+\delta)\mathbf{E}[X]] < 2^{-\delta \mathbf{E}[X]}$ to obtain

$$\Pr[X > b/2] < 2^{-b/4}.$$

Let us consider the case $|F_x| \leq b/2$. Then the probability that $f(S_j) \subseteq F_x$ is

$$\begin{aligned}\Pr\Big[f(S_j) \subseteq F_x \Big| |F_x| \leq b/2\Big] &\leq (1-p)^{b/2} \\ &\approx \left(1 - \frac{\ln(4/3)}{c_1 \ln n}\right)^{b/2} \\ &< e^{-b \ln(4/3)/(2c_1 \ln n)}.\end{aligned}$$

By choosing $b = 8c_1 \ln^2 n$, we have

$$\begin{aligned}\Pr[f(S_j) \subseteq F_x] &\leq 2^{-b/4} + e^{-4 \ln(4/3) \ln n} \\ &< e^{-3 \ln n} \\ &= 1/n^3.\end{aligned}$$

Applying the union bound, the probability that there exists an element $x \in \mathcal{U}$ and a set $S \in \mathcal{S}$, with $x \notin S$, such that $f(S) \subseteq F_x$, is at most $1/n$. Thus with probability at least $1 - 1/n$, we can apply the construction of Lemma 3.1 to the vectors u^x and w^S, to obtain an instance of *Min-DNF(*)* over $b = O(\log^2 n)$ variables whose

minimum DNF has the same size as the minimum set cover for $(\mathcal{S},\mathcal{U})$. The *Min-DNF(*)* instance has size $N = 2^b = 2^{O(\log^2 n)}$, and the probabilistic construction can be derandomized in quasi-polynomial time using the method of conditional probabilities (see, e.g., [7]). It follows that there is no polynomial-time algorithm for *Min-DNF(*)* which achieves an approximation ratio better than $c_2 \ln n = \Omega(\sqrt{\log N})$ unless $\mathbf{NP} \subseteq \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$. □

Let Hypothesis 1 be the hypothesis that there exists constants c_1, c_2 such that it is **NP**-hard to approximate $(c_1 \ln n)$-*Frequency Bounded Set Cover* to within $(c_2 \ln n)$. We gave a simple argument showing that, under Hypothesis 1, *Min-DNF* cannot be approximated to within a ratio better than some constant times $\sqrt{\log N}$ unless **NP** is in quasi-polynomial time. How believable is Hypothesis 1? A recent paper [16] proves the following related result which gives some evidence that Hypothesis 1 is true.

THEOREM 4.11 (see [16]). *For some constant ϵ, $(\ln n)^\epsilon$-Frequency Bounded Set Cover cannot be approximated to within a factor of 2 unless **NP** is in quasi-polynomial time.*

Hypothesis 1 is stronger than the above theorem in two ways. First, Hypothesis 1 requires that *Frequency Bounded Set Cover* be **NP**-hard instead of quasi-polynomial-hard. The second difference is more substantial. Hypothesis 1 requires that the problem is hard for frequency up to $(\ln n)$ instead of $(\ln n)^\epsilon$. It is possible that the above theorem can be improved to prove Hypothesis 1. In any case, the above theorem gives an alternative proof that *Min-DNF* cannot be approximated to within a ratio better than $(\log N)^\gamma$ for some $\gamma < 1$, unless **NP** is in quasi-polynomial time, where γ is basically $(\epsilon/2)$.

5. A tight example for the greedy algorithm. We show that there exist instances of *Min-DNF* for which the greedy *Set Cover* algorithm achieves an $\Omega(\log N)$ approximation ratio. Our approach is to take a standard worst-case *Set Cover* instance and to apply a version of the reductions of sections 3.1 and 4.2 to obtain first a *Min-DNF(*)* instance and then a *Min-DNF* instance. We then show that the greedy algorithm operates on the resulting *Min-DNF(*)* instance, and on the resulting *Min-DNF* instance, much as it does on the original *Set Cover* instance.

5.1. Tight example for greedy on *Min-DNF*(*). The starting point is the following *Set Cover* instance, on which the greedy *Set Cover* algorithm has worst-case behavior. The instance consists of $m-1$ pairwise disjoint sets $S_1, \ldots, S_{m-1}$ such that $|S_i| = 2^i$ and of two additional sets T_0 and T_1. For each set S_i, the set T_0 contains half of the elements in S_i, while T_1 contains the other half. On this set collection, the greedy algorithm chooses the cover consisting of all of the sets S_i, while the optimal solution consists only of T_0 and T_1.

Let $\mathcal{U}$ be the underlying universe. We define three sets of vectors $\{v^e \mid e \in \mathcal{U}\}$, $\{s^i \mid 1 \le i \le m-1\}$, and $\{t^0, t^1\}$, over $\{0,1\}^{2(m+1)}$, such that the following three conditions hold for all $e \in \mathcal{U}$ and all $1 \le i \le m-1$: (1) $v^e \le s^i$ iff $e \in S_i$; (2) $v^e \le t^0$ iff $e \in T_0$; (3) $v^e \le t^1$ iff $e \in T_1$. The vectors $\{v^e \mid e \in \mathcal{U}\}$ are defined according to the set in which they occur, as follows: Each element $e \in S_i$ is assigned a unique vector v^e from the set $\{x10^{m-i}\overline{x}01^{m-i} \mid x \in \{0,1\}^i\}$. The vectors $\{s^i \mid 1 \le i \le m-1\}$ and $\{t^0, t^1\}$ are defined as follows: $s^i = 1^i10^{m-i}1^i01^{m-i}$, $t^0 = 01^m1^{m+1}$, and $t^1 = 1^{m+1}01^m$. The set T_0 is defined as $\{e \in \mathcal{U} \mid v^e \le t^0\}$, and the set T_1 is defined as $\{e \in \mathcal{U} \mid v^e \le t^1\}$. It is easily verified that the sets $S_1, \ldots, S_{m-1}, T_0, T_1$ have the required structure: Namely, $S_1, \ldots, S_{m-1}$ are pairwise disjoint, S_i has size 2^i, and T_0 and T_1 are disjoint, each consisting of half of the elements from each of the sets

$S_1, \ldots, S_{m-1}$. Conditions (2) and (3) hold by definition, as does the "if" direction of condition (1). For the "only if" direction of (1), note that if $e \in S_j$ for $j \neq i$, then either bit $j+1$ or bit $(m+1)+(i+1)$ witnesses the fact that $v^e \not\leq s^i$.

The partial Boolean function f is defined as in the reduction from section 3.1: The ones of f are the vectors v^e for each $e \in \mathcal{U}$; the stars of f are those remaining inputs $\vec{x}$ such that $\vec{x} \leq s^i$ for some $1 \leq i \leq m-1$, or $\vec{x} \leq t^0$, or $\vec{x} \leq t^1$; and the remaining inputs are zeros. The following general lemma shows that the prime implicants of f, viewed as sets and considering only the ones of f that they cover, have exactly the same structure as the original set system.

LEMMA 5.1. *Let $\mathcal{S}$ be a set system over universe $\mathcal{U}$ such that no set in $\mathcal{S}$ contains another set in $\mathcal{S}$. Let $\{v^e \mid e \in \mathcal{U}\}$ and $\{w^S \mid S \in \mathcal{S}\}$ be sets of Boolean vectors such that the following condition holds for all $e \in \mathcal{U}$ and all $S \in \mathcal{S}$: $v^e \leq w^S$ iff $e \in S$. Let f be the function obtained from $\mathcal{S}$ as in Lemma* 3.1 *using the given vectors. Then the set of prime implicants of f is exactly $\{\tau(w^S) \mid S \in \mathcal{S}\}$.*

Proof. We first show that each term $\tau(w^S)$ is, indeed, a prime implicant of f. Suppose, on the contrary, that there is an implicant τ of f that subsumes $\tau(w^S)$ for some $S \in \mathcal{S}$. Note that all variables in τ are negated. Let $\vec{u}$ be a maximal truth assignment satisfying τ. Since $f(\vec{u}) = 1$, there is a set $S' \in \mathcal{S}$ such that $\vec{u} \leq w^{S'}$; that is, for each index i, if $w_i^{S'} = 0$, then $u_i = 0$. By our choice of $\vec{u}$ we have that $u_i = 0$ iff the literal $\neg x_i$ occurs in τ, and by definition $w_i^{S'} = 0$ iff the literal $\neg x_i$ occurs in $\tau(w_i^{S'})$. Thus for each index i, if the literal $\neg x_i$ occurs in $\tau(w^{S'})$, then it also occurs in τ. As both τ and $\tau(w^{S'})$ consist exclusively of negated literals, we have $\tau \implies \tau(w^{S'})$; since τ subsumes $\tau(w^S)$, we have

$$\tau(w^S) \implies \tau \implies \tau(w^{S'}).$$

For each $e \in U$, $e \in S \iff v^e \leq w^S$, and $v^e \leq w^S$ iff v^e satisfies $\tau(w^S)$. Thus,

$$e \in S \implies v^e \leq w^S \implies v^e \leq w^{S'} \implies e \in S'.$$

That is, $S \subseteq S'$, contradicting the assumption about $\mathcal{S}$.

We now show that every prime implicant of f is equal to $\tau(w^S)$ for some $S \in \mathcal{S}$. Let τ be a prime implicant of f, and let $\vec{u}$ be a maximal truth assignment satisfying τ. Then there exists $S \in \mathcal{S}$ such that $\vec{u} \leq w^S$, and thus $w_i^S = 0$ implies $u_i = 0$, and by the same argument as before we have that each literal $\neg x_i$ occurring in $\tau(w^S)$ also occurs in τ. It follows that $\tau = \tau(w^S)$. □

Lemma 5.1 implies that the prime implicants of f, viewed as sets, have exactly the same structure with respect to the ones of f as the original set collection has with respect to $\mathcal{U}$. It follows that the greedy algorithm finds a solution of size $m-1$, consisting of the terms $\tau(s^1) \wedge \cdots \wedge \tau(s^{m-1})$, while the optimal solution $\tau(t^0) \wedge \tau(t^1)$ has size two. As the instance has $n = 2m+2$ variables, the approximation ratio is $(m-1)/2 = (n-4)/4$.

5.2. Tight example for greedy on *Min-DNF*. We now extend the construction of the previous section to give an instance of *Min-DNF* for which the greedy algorithm achieves an approximation ratio of $\Omega(n) = \Omega(\log N)$. The instance is obtained by applying the reduction of section 4.2 to the function f from section 5.1. As in the proof of Lemma 4.7, we use $t(\vec{w})$ to denote the term $\left(\bigwedge_{i:w_i=1} w_i\right) \wedge \left(\bigwedge_{i:w_i=0} \neg w_i\right)$.

LEMMA 5.2. *Let $\mathcal{S}$ be a set of subsets of $[n]$, and let f be a partial Boolean function, such that $\mathcal{S}$ and f satisfy the conditions of Lemma* 3.1. *Let g be the total Boolean function obtained by applying the reduction from section* 4.2 *to f. Then the*

prime implicants of g consist of exactly the following: $\tau \wedge y_1 \wedge y_2 \wedge t(\vec{z})$, where τ is a prime implicant of f and $\vec{z} \in S$, and $t(\vec{x}) \wedge y_{2-p(\vec{x})}$, where $f(\vec{x}) = *$.

Proof. It is easy to verify that each term in the statement of the lemma is, indeed, a prime implicant of g, noting that every prime implicant τ of f must cover at least one vector in $f^{-1}(1)$, since each set in the original set system is nonempty.

We now argue that all prime implicants of g are of the above types. Let τ be an implicant of g: We will show that τ is subsumed by an implicant of one of the above two types. Let $\vec{x}y_1y_2\vec{z}$ be an assignment that satisfies τ. We first consider the case where $f(\vec{x}) = 1$. From the definition of g, it is clear that τ contains $t(\vec{z})$, y_1, and y_2, as flipping the corresponding bits of the assignment falsifies g. Moreover, the portion τ_x of τ containing x-variables is an implicant of f and is therefore subsumed by a prime implicant τ' of f. Thus $\tau' \wedge y_1 \wedge y_2 \wedge t(\vec{z})$ is an implicant of g and subsumes τ. Now consider the case where $f(\vec{x}) = *$, and assume without loss of generality that $\vec{x}$ has even parity. Then τ must contain y_2 and $t(\vec{x})$, as flipping any of these bits falsifies g, and thus τ is subsumed by $t(\vec{x}) \wedge y_2$. □

The inputs to g are of the form $\vec{x}y_1y_2\vec{z}$, where the length of $\vec{x}$ is n and the length of $\vec{z}$ is $t = n+1$. Each prime implicant of the second type in the statement of Lemma 5.2 covers 2^{t+1} ones of g, and the ones covered by these prime implicants are pairwise disjoint. The prime implicants of the first type each cover at most $2^{n-1} < 2^{t+1}$ ones of g. Thus the greedy algorithm begins by choosing all prime implicants of the second type. At this point, the prime implicants of the first type corresponding to different values of $\vec{z}$ cover disjoint subsets of the ones of g, so let us consider only a particular value of $\vec{z}$: The choices made by the greedy algorithm for other vectors $\vec{z}$ are independent of its behavior on this vector. Now the uncovered ones of g that are covered by a term $\tau \wedge y_1 \wedge y_2 \wedge t(\vec{z})$ are precisely those whose $\vec{x}$-component is a one of f, as the others are already covered by prime implicants of the second type. Thus the prime implicants of this type chosen by the greedy algorithm are exactly the set of prime implicants of the form $\tau \wedge y_1 \wedge y_2 \wedge t(\vec{z})$, where τ is a prime implicant that would be chosen by the greedy algorithm on input f. It follows that the greedy solution has size $s(m-1)+s = sm$, while the optimal solution has size $2s+s = 3s$. As the instance has $n = 2m+4+t$ variables, the approximation ratio is $m/3 = (n-t-4)/6 = \Omega(n)$.

6. Fixed parameter complexity. It is known that the decision problem "Given a truth table of a Boolean function f, and a number k, does f have a DNF with at most k terms?" can be solved in time $p(N, 2^{k^2})$, for some polynomial p, where N is the size of the truth table [22]. (This follows easily from the fact that if f is a Boolean formula that can be represented by a k-term DNF formula, then there exist at most 2^k prime implicants of f [12].) Thus, *Min-DNF* is *fixed parameter tractable* [17]. Moreover, because the size of the input truth table is $N = 2^n$, where n is the number of variables of f, it follows that *Min-DNF* is solvable in polynomial time for any $k = O(\sqrt{n})$.

It is an open question whether *Min-DNF* can be solved in polynomial time for $k = n$. But by applying a simple padding argument, we obtain the following corollary to the NP-completeness result for *Min-DNF*.

COROLLARY 6.1. *If there exists some constant $\epsilon > 0$ such that NP is not contained in DTIME$(2^{O(n^\epsilon)})$, then for some constant $c > 1$, Min-DNF for $k = n^c$ is not solvable in polynomial time (where n is the number of input variables of the Boolean function defined by the Min-DNF instance).*

Proof. Because *Min-DNF* is NP-complete, there exists a polynomial-time reduction from problems Π in NP to *Min-DNF*. If the input to Π is of size n, then the

input to the resulting *Min-DNF* problem will be a truth table of size $s = O(n^b)$ for some constant $b > 1$, defining a Boolean function on $\log s$ variables. The parameter k in the derived *Min-DNF* instance is no more than s, since for any truth table there is always a consistent DNF of size at most the size of the truth table. Let $c > 1$. Let $m = s^{\frac{1}{c}}$. Take the *Min-DNF* instance, and form a new *Min-DNF* instance by padding the function in the truth table with $m - \log s$ new dummy variables. Suppose *Min-DNF* is solvable in polynomial time when $k = n^c$, where n is the number of input variables of the Boolean function defined by the *Min-DNF* instance. Then the padded instance of *Min-DNF* can be solved in time polynomial in 2^m, and Π can be solved in time $2^{O(n^{\frac{b}{c}})}$, where n is the size of input to Π. For $c > \frac{b}{\epsilon}$, this is less than $2^{O(n^\epsilon)}$, a contradiction. □

7. *Min-DNF* and learning. One of the major problems in learning theory is to determine whether DNF formulas can be learned in polynomial time. There are connections between the complexity of *Min-DNF* and its fixed parameter versions, and the complexity of learning DNF formulas. This connection is strongest for "proper" learning models. In such models, any hypotheses used in the learning algorithm must be of the same type as the formulas being learned by the algorithm. Thus if the task is to learn DNF formulas, hypotheses must be DNF formulas. If the task is to learn k-term DNF formulas, then hypotheses must be k-term DNF formulas.

There has been a significant amount of research on learning k-term DNF formulas for small values of k in both proper and improper models (see, e.g., [8, 10, 27, 22, 34]). Pitt and Valiant showed that in the probably approximately correct (PAC) model, unless RP=NP one cannot learn k-term DNF formulas in polynomial time using hypotheses that are k-term DNF formulas (for constant k) [34]. Their proof actually shows that the consistency problem for k-term DNF is hard. This problem takes as input a partial Boolean function, specified by its 1's and 0's, and asks whether there is a k-term DNF formula consistent with those entries. The work of Pitt and Valiant was subsequently extended to obtain significantly stronger results on learning arbitrary length DNF formulas in the PAC learning model [2, 20]. We note that our reduction to *Min-DNF(*)* in fact implies that the consistency problem for k-term DNF is NP-hard for $k = n$, even when the underlying function depends on only $\log n$ of the n input variables (a $\log n$ "junta"); this in turn implies that proper PAC learning of n-term DNF formulas depending on $\log n$ of the n input variables is hard unless RP=NP.

Pillaipakkamnatt and Raghavan [33] showed that, for some $\epsilon < 1$ and some $c > 1$, $\log^c n$-term DNF cannot be learned in the membership and proper equivalence query model unless NP $\subseteq DTIME(2^{O(n^\epsilon)})$. Subsequently, Hellerstein and Raghavan proved that $\Omega(\log^{3+\epsilon} n)$-term DNF formulas cannot be learned in the same model; their proof involves a structural property of DNF formulas, and the result is without any assumptions [22]. (It can be improved to $\Omega(\log^{2+\epsilon})$.) It is open, however, whether $\log n$-term DNF formulas can be learned in polynomial time in this model; $\sqrt{\log n}$-term DNF can be so learned [10].

A polynomial-time algorithm for learning $\log n$-term DNF formulas in the membership and proper equivalence query model (i.e., with hypotheses that are $\log n$-term DNF formulas) would imply a polynomial-time algorithm for *Min-DNF* for $k = n$ [22]. The same proof shows that, for constant c, a polynomial-time algorithm for learning $\log^c n$-term DNF formulas would imply a polynomial-time algorithm for *Min-DNF* for $k = n^c$. It follows that the result of [33] mentioned above can also be derived from Corollary 6.1.

The relation between truth table minimization and learning with membership and equivalence queries relies on the following observation: Given a truth table representing a function f, one can simulate a membership and equivalence query algorithm for learning (a hypothesis representing) f by using the truth table to answer the queries. Feldman observed that one can also use the truth table of f to generate uniformly distributed examples of f. Combining this observation with the hardness of approximating *Min-DNF*, he showed hardness of proper PAC learning of min-DNF under the uniform distribution, with membership queries. More specifically, he showed that for some $\gamma > 0$, unless P=NP, there is no polynomial-time algorithm that PAC learns DNF formulas under the uniform distribution using hypotheses that are DNF formulas of size at most n^γ larger than the function being learned, even if membership queries are allowed [20].

8. Hardness of *Min-*AC^0_d. In this section we consider the problem of estimating *Min-*$AC^0_d(f)$, the size of the smallest AC^0_d circuit for f, given as input its $N(= 2^n)$-bit truth table. In [4] it was shown that neither *Min-Circuit*(f) (the size of the smallest circuit) nor *Min-*$NC^1(f)$ (the size of the smallest NC^1 circuit) can be approximated to within a factor of $N^{1-\epsilon}$ in polynomial time unless Blum integers can be factored in probabilistic polynomial time. Here we prove an analogous result for *Min-*AC^0.

We consider algorithms that produce an estimate that is at least *Min-*$AC^0_d(f)$ (e.g., algorithms that actually produce a circuit C for f and output the size of C). We say that such an algorithm is a $\lambda(N)$-approximation algorithm if the output is no more than $\lambda(N)$ *Min-*$AC^0_d(f)$. Since for each $d \geq 2$, every n-variate Boolean function f satisfies $1 \leq$ *Min-*$AC^0_d(f) \leq nN$, there is a trivial nN-approximation algorithm. In this section, we show that for any $\epsilon > 0$ there is a $d(\epsilon)$ such that there is no polynomial time $O(N^{1-\delta})$-approximation for *Min-*$AC^0_d(f)$ for any $\delta > 0$, unless m-bit Blum integers can be factored in probabilistic time 2^{m^ϵ}. We have not computed the relationship between d and ϵ, but we anticipate that this yields a meaningful inapproximability result for d as small as 10.

The proof of this inapproximability result follows along similar lines as the related results in [4]. In those proofs, the pseudorandom function generator of Naor and Reingold is used, which has the nice property that it can be computed in TC^0_d for some d. Any test that can distinguish random functions from functions generated by this generator can be used to factor Blum integers. A good approximation of TC^0_d circuit size can be used to distinguish pseudorandom and random functions, since random functions require circuits of exponential size, whereas the pseudorandom functions produced by the generator have small TC^0_d circuits. One more ingredient is still needed, in order to obtain inapproximability results for *Min-*AC^0_d. This ingredient is provided by Lemma 8.1, which shows that any function with small TC^0 circuits has "relatively small" AC^0 circuits. (This property extends even to complexity classes seemingly much larger than TC^0.) Taken together, this means that a good approximation to AC^0_d circuit size yields a good enough approximation to $TC^0_{d'}$ circuit size to yield subexponential upper bounds on the complexity of factoring Blum integers.

Because of the way the various parameters involved interact, we see no simple way to present the approximability result by merely appealing to results in [4]. Thus, we present the entire proof below. First, we show that everything in nondeterministic logspace (NL) has AC^0 circuits of subexponential size.

LEMMA 8.1. *For every language $\mathcal{L}$ in NL, and for every ϵ, there exists a d such that there are AC^0_d circuits of size 2^{n^ϵ} that recognize $\mathcal{L}$.*

Proof. Consider an NL machine M running in time $m = n^c$. On input x, we want to find out if M has an accepting path on x. To find an accepting path, it is sufficient to see if there is a sequence of $\sqrt{m}$ configurations of M $C_1, \ldots, C_{\sqrt{m}}$, where C_1 is the initial configuration of M on input x and $C_{\sqrt{m}}$ is the accepting configuration, with the property that for $1 \leq i < \sqrt{m}$ there is a computation path of length $\sqrt{m}$ from C_i to C_{i+1}. We view the configurations C_i as "checkpoints" along the computation path. This approach to finding an accepting path is straightforward to implement using a depth-three AC^0 circuit of size $2^{O(\sqrt{m}\log m)}$.

If the number of checkpoints chosen is $m^{1/3}$ instead of $\sqrt{m}$, then a similar strategy leads to a depth-five circuit of size $2^{O(m^{1/3}\log m)}$. That is, the top level of the depth-five circuit is an OR (over all of the $2^{O(m^{1/3}\log m)}$ sequences S of checkpoints) of the AND that each of the $m^{1/3} - 1$ pairs of adjacent checkpoints in S is connected by a path of length $m^{2/3}$. This latter condition can be checked by an OR over another $2^{O(m^{1/3}\log m)}$ sequences S' of $m^{1/3}$ checkpoints, of an AND that each of the $m^{1/3} - 1$ pairs of adjacent checkpoints in S' is connected by a path of length $m^{1/3}$. Since the input head of M can move only a distance of $m^{1/3}$ in $m^{1/3}$ steps, and each checkpoint specifies the position of the input head, the condition that a given pair of checkpoints is connected by a path of length $m^{1/3}$ depends only on $m^{1/3}$ input variables, namely, those centered around the two input head positions specified by the checkpoints. Thus this condition can be expressed by a CNF formula of size exponential in $m^{1/3}$. (The depth can be optimized somewhat, using closure under complement and merging adjacent layers—but we ignore such issues for now.)

Notice that, by increasing the depth from three to five and decreasing the number of checkpoints from $\sqrt{m}$ to $m^{1/3}$, one is able to obtain smaller circuits. Iterating the above idea gives depth-d AC^0 circuits of size 2^{n^ϵ}. This is basically a strengthening of Nepomnjaščiĭ's theorem [5, 32]. (The same claim, with an identical proof, holds for any language accepted by a nondeterministic machine running in polynomial time and using space $n^{o(1)}$. In particular, it holds for the complexity class LogCFL.) □

DEFINITION 8.2. *An integer M is called a Blum integer if $M = PQ$, where P and Q are two primes such that $P \equiv Q \equiv 3 \mod 4$. The Blum integer factorization problem is as follows. Given a Blum integer M, find the primes P and Q such that $1 < P \leq Q$ and $M = PQ$.*

THEOREM 8.3. *For every $\delta > 0$ and $\epsilon > 0$ there is a depth d such that if $\mathcal{B}$ is an algorithm that approximates Min-AC^0_d to within a factor of $N^{1-\delta}$ (where $N = 2^n$ is the size of the input truth table), then Blum integer factorization is in BPTIME$(2^{m^\epsilon})^{\mathcal{B}}$ (where m is the number of bits of the integer to be factored).*

Proof. We follow the proof given in [4]. In [31] a pseudorandom function ensemble $\{f_{M,r}(x) : \{0,1\}^n \to \{0,1\}\}_{M,r}$ is constructed with the following two properties:

- There is a polynomial size TC^0 circuit computing $f_{M,r}(x)$, given an $m = 2n$ bit integer M, a $4n^2 + 2n$-bit string r, and an n-bit string x. This means that there is a constant d' such that for each n there is a depth-d' threshold circuit of size at most $n^{d'}$ that takes as input M, r, x and outputs $f_{M,r}(x)$.
- For every probabilistic Turing machine $\mathcal{M}$ running in time $t(n)$ with oracle access to $f_{M,r}$ of query length n, there exists a probabilistic Turing machine $\mathcal{A}$ running in time $t(n)n^{O(1)}$ such that for every $2n$-bit Blum integer $M = PQ$, if $|Pr[\mathcal{M}^{f_{M,r}}(M) = 1] - Pr[\mathcal{M}^{R_n}(M) = 1]| > 1/2$, where R_n is a uniformly distributed random function ensemble, and the probability is taken over random r and random bits of $\mathcal{M}$, then $Pr[\mathcal{A}(M) = (P, Q)] > 1/2$. In other words, if $\mathcal{M}$ can distinguish the pseudorandom function ensemble from

a truly random function "efficiently," then Blum integers can be factored "efficiently" on a probabilistic machine.

Let $\delta > 0$ and $\epsilon > 0$ be given as in the theorem. Let $q = n^\epsilon$ and $Q = 2^q$. By the first property of the ensemble together with Lemma 8.1, there is a d such that for each sufficiently large n there is a depth-d AC^0 circuit that takes as input M, r, x and outputs $f_{M,r}(x)$ and has size at most $2^{n^{\epsilon/2}}$; for n sufficiently large this quantity is at most $Q^{\delta/2}$. In particular, for each choice of $M \in \{0,1\}^{2n}$ and $r \in \{0,1\}^{4n^2+2n}$, we can hardwire the values of M and r to get a circuit of size at most $Q^{\delta/2}$ for the function $f_{M,r}$.

For $y \in \{0,1\}^q$ let $\tilde{y} \in \{0,1\}^n$ be the concatentation of y with 0^{n-q}. For any function h defined on $\{0,1\}^n$, let $\tilde{h}$ be the function defined on $\{0,1\}^q$ by $\tilde{h}(y) = h(\tilde{y})$.

Suppose that $\mathcal{B}$ is a function as in the hypothesis of the theorem. We now construct an oracle Turing machine $\mathcal{M}$ which takes as input a $2n$-bit integer M and has oracle access to $\mathcal{B}$ and to an n-variate Boolean function g and reliably distinguishes the case that $g = f_{M,r}$ from the case that g is truly random. By the second property of the pseudorandom function generator $f_{M,r}$, when M is a Blum integer, this will be enough to factor M.

On input N, $\mathcal{M}$ queries $g(\tilde{y})$ for all $y \in \{0,1\}^{n^q}$, thus constructing the truth table for $\tilde{g}$ (of size Q). $\mathcal{M}$ then submits the truth table of $\tilde{g}$ to the approximation algorithm $\mathcal{B}$ and accepts iff the approximate circuit size for $\tilde{g}$ returned by $\mathcal{B}$ is greater than $Q^{1-\delta/2}$.

Now assume that the answer returned by $\mathcal{B}$ is within a $Q^{1-\delta}$ factor of the true answer. If $g = f_{M,r}$, then g (and therefore $\tilde{g}$) has a circuit of size at most $Q^{\delta/2}$, and so $\mathcal{B}$ always returns a value less than $Q^{\delta/2}Q^{1-\delta} = Q^{1-\delta/2}$ and $\mathcal{M}$ rejects. On the other hand, if g is taken uniformly at random from $\mathcal{R}_n$, then the distribution of $\tilde{g}$ is uniformly random from $\mathcal{R}_q$ and thus with extremely high probability requires AC^0_d circuits of size $Q/q > Q^{1-\delta/2}$ (since most functions require circuits of this size), and this condition causes $\mathcal{M}$ to accept. Hence $|Pr[\mathcal{M}^{f_{M,r}(x)}(M) = 1] - Pr[\mathcal{M}^{R_n}(M) = 1]| > 1/2$ for sufficiently large n. Thus, $\mathcal{M}$ can distinguish the pseudorandom function ensemble from a truly random one with probability greater than $1/2$, and thus Blum integers can be efficiently factored probabilistically. □

COROLLARY 8.4. *For all $\delta > 0$ and all $\epsilon > 0$ there exists a d such that Min-AC^0_d cannot be approximated to within a factor $n^{1-\delta}$ in BPP unless Blum integer factorization is in BPTIME(2^{n^ϵ}).*

9. Discussion. There are close connections between the hardness of function minimization problems and related learnability results. In addition to the connections discussed above in section 6, we mention two others: The complexity of *Min-DNF*(DNF) and of approximating *Min-DNF*(DNF) has been shown to be related to the problem of learning DNF with proper membership and equivalence queries [11, 22, 1], and results on learning circuits [9] yield positive results for approximating circuit minimization (cf. [39]). At a basic level, learning a formula or circuit involves gathering information about it and then synthesizing or compressing that information to produce a compact hypothesis. The need for compactness provides the connection to minimization. In many learning problems one can distinguish between informational complexity (the number of queries or sample size needed) and computational complexity (the amount of computation needed to process the information). Information about a formula or circuit typically consists just of input/output pairs. Truth table minimization problems are relevant to the computational hardness of learning; even if you have all input/output pairs, the question is whether you can compact that information in polynomial time.

The NP-hardness of proper PAC learning DNF and of *Min-DNF* are known. On the other hand, very strong inapproximability results are known for both proper PAC learning and the function minimization problem for complexity classes starting at AC^0. However, these latter results rely on cryptographic assumptions and are not known to hold under NP-hardness assumptions. Thus an important open question is to resolve the NP-hardness of both learnability results as well as function minimization results above for classes that are stronger than DNF.

Another open problem is to close the approximability gap for *Min-DNF*.

Appendix A. Reduction from *r-Uniform Set Cover*. The following lemma describes a modified version of the reduction given in section 3.1. Whereas the reduction in that section is from 3-*Partite Set Cover*, the reduction here is from *r-Uniform Set Cover* (all sets in the input set cover instance are of size r). Because the reduction here is not from a partite version of *Set Cover*, it requires different techniques than the reduction in section 3.1.

LEMMA A.1. *There is an algorithm that takes as input an r-uniform collection of subsets $\mathcal{S}$ over $[n]$ and produces the truth table of a partial Boolean function f such that the minimum size of a cover of $[n]$ with $\mathcal{S}$ is equal to the minimum number of terms in a DNF consistent with f. The algorithm runs in time $(n|\mathcal{S}|)^{O(r)}$, and the number of variables of f is $O(r \log(n|\mathcal{S}|))$.*

Proof. Let the r-uniform collection $\mathcal{S}$ over $[n]$ be given.

As in the proof of Lemma 3.2, we produce two indexed sets of vectors $V = \{v^i : i \in [n]\}$ and $W = \{w^A : A \in \mathcal{S}\}$ of length t satisfying the property $(*)$ that, for all $A \in \mathcal{S}$ and $i \in [n]$, $i \in A$ iff $v^i \leq w^A$. Again, we specify V and then define W according to the rule that, for $A \in \mathcal{S}$, w^A is the bitwise OR of $\{v^i : i \in A\}$. The construction of partial function f, given V and W, is then the same as in the proof of Lemma 3.2, and again it follows that the size of the minimum DNF consistent with f is equal to the size of the minimum cover of $[n]$ by $\mathcal{S}$.

We now describe the construction of V. Let P be the set of pairs (j, A), with $A \in \mathcal{S}$ and $j \in [n] - A$. The desired conditions on V can be restated as specifying that for all $(j, A) \in P$:

> $C(j, A)$: There is a bit position $\alpha \in [t]$ such that $v^j_\alpha = 1$ and $v^i_\alpha = 0$ for all $i \in A$.

If we choose $v^1, \ldots, v^n$ of length t at random where each bit is 1 independently with probability $1/r$, then for each $(j, A) \in P$ the probability that $C(j, A)$ does not hold is $(1 - \frac{1}{r}(1 - \frac{1}{r})^r)^t \leq e^{-t/3r}$, so the probability that $v^1, \ldots, v^r$ fails to meet the requirements is at most $|P|e^{-t/3r} \leq |\mathcal{S}|ne^{-t/3r}$. Thus if $t \geq 3r(1 + \ln(|\mathcal{S}|n))$, this random choice succeeds with probability more than $1/2$. This is enough for a randomized reduction. To make it deterministic, we derandomize this construction using the method of conditional probabilities (see, e.g., [7]). This is routine but technical, so we provide only a sketch. Let $X(j, A)$ be the random variable that is 1 if $C(j, A)$ fails. We want to choose $v^1, \ldots, v^r$ so that $X = \sum_{(j,A)\in P} X(j, A) = 0$. The above argument says that under random choice $\mathbf{Exp}[X] \leq 1/2$. The key point for derandomizing is that if we fix any subset of the bits in $v^1, \ldots, v^r$, then it is straightforward to compute the conditional expectation of X given this fixing in time $(|\mathcal{S}|n)^{O(1)})$. We can then use the method of conditional probabilities to fix these bits one at a time by always choosing the value of the bit that does not increase the expectation. Once all bits are fixed, we must have a good choice for V.

Clearly V and W can be constructed in time $(n|\mathcal{S}|)^{O(1)}$, with $t = O(r \log (n|\mathcal{S}|))$. Since the truth table has size 2^t, outputting it takes time $(n|\mathcal{S}|)^{O(r)}$. □

Combining the above reduction with the modified reduction from *Min-DNF(*)* to *Min-DNF* in section 4.2 yields a reduction from *r-Uniform Set Cover* to *Min-DNF*. By setting $r = \log N$, where N is the truth table size, one can then apply inapproximability results for *r-Uniform Set Cover* [19, 38] to show that *Min-DNF* cannot be approximated to within a factor of $\Omega(\log \log N)$ in polynomial time unless NP is in randomized quasi-polynomial time.

Acknowledgments. The authors gratefully acknowledge Uri Feige and Subhash Khot for helpful conversations on the inapproximability of partite restrictions of set cover. We thank the referees for their insightful comments.

REFERENCES

[1] A. Aizenstein, T. Hegedus, L. Hellerstein, and L. Pitt, *Complexity theoretic hardness results for query learning*, Comput. Complexity, 7 (1998), pp. 19–53.

[2] M. Alekhnovich, M. Braverman, V. Feldman, A. Klivans, and T. Pitassi, *Automatizability and learnability*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, Rome, Italy, 2004.

[3] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. Saks, *Minimizing DNF formulas and AC^0 circuits given a truth table*, in Proceedings of the 21st Annual IEEE Conference on Computational Complexity, Prague, Czech Republic, 2006, pp. 237–251.

[4] E. Allender, M. Koucky, D. Ronneberger, and S. Roy, *Derandomization and distinguishing complexity*, in Proceedings of the 18th Annual IEEE Conference on Computational Complexity, Aarhus, Denmark, 2003, pp. 209–220.

[5] E. Allender, M. Koucky, D. Ronneburger, and S. Roy, *Time-space tradeoffs in the counting hierarchy*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, IL, 2001, pp. 295–302.

[6] N. Alon, D. Moshkovitz, and S. Safra, *Algorithmic construction of sets for k-restrictions*, ACM Trans. Algorithms, 2 (2006), pp. 153–177; also available online at http://www.informatik.uni-trier.de/%7Eley/db/journals/talg/talg2.html#AlonMS06> (2):153-177 (2006).

[7] N. Alon, J. Spencer, and P. Erdös, *The Probabilistic Method*, John Wiley & Sons, New York, 1992.

[8] U. Berggren, *Linear time deterministic learning of k-term DNF*, in Proceedings of the 6th Annual ACM Conference on Computational Learning Theory, Santa Cruz, CA, USA, 1993, pp. 37–40.

[9] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. System Sci., 52 (1996), pp. 421–433.

[10] N. H. Bshouty, *Simple learning algorithms using divide and conquer*, Comput. Complexity, 6 (1997), pp. 174–194.

[11] N. H. Bshouty and L. Burroughs, *On the proper learning of axis parallel concepts*, in Proceedings of the 15th Annual Conference on Computational Learning Theory, Sydney, Australia, Lecture Notes in Comput. Sci. 2375, Springer-Verlag, Berlin, 2002, pp. 287–302.

[12] A. Chandra and G. Markowsky, *On the number of prime implicants*, Discrete Math., 24 (1978), pp. 7–11.

[13] V. Chvátal, *A greedy heuristic for the set covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[14] O. Coudert, *Two-level logic minimization: An overview*, Integration, the VLSI Journal, 17 (1994), pp. 97–140.

[15] S. Czort, *The Complexity of Minimizing Disjunctive Normal Form Formulas*, Master's thesis, University of Aarhus, 1999.

[16] I. Dinur, V. Guruswami, S. Khot, and O. Regev, *A new multilayered PCP and the hardness of hypergraph vertex cover*, SIAM J. Comput., 34 (2005), pp. 1129–1146.

[17] R. Downey and M. Fellows, *Parameterized Complexity*, Springer-Verlag, Berlin, 1999.

[18] R. Duh and M. Fürer, *Approximation of k-set cover by semi-local optimization*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 256–264.

[19] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[20] V. Feldman, *Hardness of Approximate Two-Level Logic Minimization and PAC Learning with Membership Queries*, ECCC report TR05-127, 2005. Extended version to appear in Proceedings of the 38th Annual ACM Symposium on the Theory of Computing, 2006, pp. 363–372.
[21] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
[22] L. Hellerstein and V. Raghavan, *Exact learning of DNF formulas using DNF hypothesis*, J. Comput. System Sci., 70 (2005), pp. 435–470.
[23] D. S. Hochbaum, *Approximation algorithms for the set covering and vertex cover problems*, SIAM J. Comput., 11 (1982), pp. 555–556.
[24] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.
[25] V. Kabanets and J. L. Cai, *Circuit minimization problem*, in Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing, Portland, OR, 2000, pp. 73–79.
[26] S. Khot, online lecture notes for *Probabilistically Checkable Proofs and Hardness of Approximation*, Lecture 3 (scribed by Deeparnab Chakrabarti), available at http://www.cc.gatech.edu/~khot/pcp-course.html.
[27] E. Kushilevitz, *A simple algorithm for learning $o(\log n)$-term DNF*, Inform. Process. Lett., 61 (1997), pp. 289–292.
[28] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[29] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.
[30] W. J. Masek, *Some NP-Complete Set Covering Problems*, manuscript, 1979.
[31] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudorandom functions*, J. ACM, 51 (2004), pp. 231–262.
[32] V. A. Nepomnjaščiĭ, *Rudimentary predicates and Turing calculations*, Math. Dokl., 11 (1970), pp. 1462–1465.
[33] K. Pillaipakkamnatt and V. Raghavan, *On the limits of proper learnability of subclasses of DNF formulas*, Machine Learning, 25 (1996), pp. 237–263.
[34] L. Pitt and L. G. Valiant, *Computational limitations on learning from examples*, J. ACM, 35 (1988), pp. 965–984.
[35] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[36] A. A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.
[37]
[38] L. Trevisan, *Non-approximability results for optimization problems on bounded degree instances*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Heraklion, Crete, Greece, 2001, pp. 453–461.
[39] C. Umans, *Hardness of approximating Σ_2^p minimization problems*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, NY, 1999, pp. 465–474.
[40] C. Umans, T. Villa, and A. L. Sangiovanni-Vincentelli, *Complexity of two-level logic minimization*, IEEE Trans. CAD Integrated Circuits Syst., 25 (2006), pp. 1230–1246.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 85–96

UNIFORM HASHING IN CONSTANT TIME AND OPTIMAL SPACE*

ANNA PAGH† AND RASMUS PAGH†

Abstract. Many algorithms and data structures employing hashing have been analyzed under the *uniform hashing* assumption, i.e., the assumption that hash functions behave like truly random functions. Starting with the discovery of universal hash functions, many researchers have studied to what extent this theoretical ideal can be realized by hash functions that do not take up too much space and can be evaluated quickly. In this paper we present an almost ideal solution to this problem: a hash function $h : U \to V$ that, on any set of n inputs, behaves like a truly random function with high probability, can be evaluated in constant time on a RAM and can be stored in $(1+\epsilon)n \lg |V| + O(n + \lg \lg |U|)$ bits. Here ϵ can be chosen to be any positive constant, so this essentially matches the entropy lower bound. For many hashing schemes this is the first hash function that makes their uniform hashing analysis come true, with high probability, without incurring overhead in time or space.

Key words. hash function, uniform hashing, randomized algorithms

AMS subject classifications. 68P5, 68P10, 68P20

DOI. 10.1137/060658400

1. Introduction.

Hashing is an important tool in randomized algorithms and data structures, with applications in such diverse fields as information retrieval, complexity theory, data mining, cryptology, and parallel algorithms. Many algorithms using hashing have been analyzed under the assumption of *uniform hashing*, i.e., the idealized assumption that the hash function employed is a truly random function. In this paper we present a theoretical justification for such analyses, in the form of the construction of a hash function that makes the uniform hashing assumption "come true" with high probability. Our hash function can be evaluated in constant time on a RAM, and its description uses very close to the minimum possible space.

1.1. History.

According to Knuth [16], the idea of hashing was originated in 1953 by H. P. Luhn. The basic idea is to use a function $h : U \to V$, called a *hash function*, that "mimics" a random function. In this way a "random" value $h(x)$ can be associated with each element from the domain U. In this paper, as in most other hash function constructions, we consider a universe of the form $U = \{0, \ldots, u-1\}$ and a range of the form $V = \{0, \ldots, v-1\}$, where $1 < v \leq u$.

Representation of a random function requires $u \lg v$ bits, so it is usually not feasible to actually store a randomly chosen function. For many years hashing was largely a heuristic, and practitioners used fixed functions that were empirically found to work well in cases where uniform hashing could be shown to work well.

The gap between hashing algorithms and their analysis narrowed with the advent of *universal hashing* [6]. The key insight was that it is often possible to get provable performance guarantees by choosing hash functions at random from a small family of functions (rather than from the family of all functions). The importance of the family being small is, of course, that a function from the family can be stored succinctly.

*Received by the editors April 28, 2006; accepted for publication (in revised form) September 19, 2007; published electronically March 28, 2008. A preliminary version of this paper appeared at Symposium on Theory of Computation (STOC), 2003. This work was completed in part while the authors were at BRICS, Department of Computer Science, University of Aarhus, Denmark.
http://www.siam.org/journals/sicomp/38-1/65840.html
†IT University of Copenhagen, Rued Langgaards Vej 7, 2300 København S, Denmark (annao@itu.dk, pagh@itu.dk).

Hash functions are usually chosen uniformly at random from some *family* $\mathcal{H}$ of hash functions. For example, the family

$$\{x \mapsto ((ax+b) \bmod p) \bmod v \mid 0 < a < p,\, 0 \le b < p\},$$

first studied by Carter and Wegman [6], has many known applications. The family is described by the parameters p and v, while a particular function in the family is given by the values of parameters a and b. In our results, we will distinguish between the space needed to represent the family and the space needed to represent a function in the family.

One property of the choice of hash function that often suffices to give performance guarantees is that it maps each set of k elements in U to uniformly random and independent values, where k is some parameter that depends on the application. If this holds for a random function from a family $\mathcal{H}$, we say that $\mathcal{H}$ is *k-wise independent*. There exist such function families whose functions can be stored in $O(k \lg u)$ bits of space [25]. For many years, all known k-wise independent families with nontrivial space usage required time $\Omega(k)$ for evaluating a hash function. A breakthrough was made by Siegel [23], who showed that high independence is achievable with relatively small families of hash functions that can be evaluated in *constant* time on a RAM.

The *RAM model* used in Siegel's result, as well as in this paper, is a standard unit cost RAM with an instruction set that includes multiplication, and a word size of $\Theta(\lg u)$ bits. The RAM has access to a source of random bits, and in particular we assume that a random value in V and a random word can be generated in constant time.

The two main performance parameters of a hash function family is the space needed to represent a function and the time necessary to compute a given function value from a representation. A tight bound on the number of bits needed to achieve k-wise independence is $\Theta(k \lg u)$ bits [3, 7]. Sometimes there is a trade-off between the space used to represent a function and its evaluation time. For example, Siegel [23] shows that if $u = k^{1+\Omega(1)}$, it is necessary to use $k^{1+\Omega(1)} \lg v$ bits of space to achieve constant evaluation time.

Siegel's construction of a k-wise independent family comes close to this lower bound (see Theorem 3). If one applies this family with $k = n$ to a set S of n elements, it will map these to independent and uniformly random values. We say that it is *uniform on S*. However, the space usage is superlinear meaning that, in many possible applications, the hash function description itself becomes asymptotically larger than all other parts of the data structure.

1.2. Our result. In this paper we present a family of hash functions that has the same performance as Siegel's family on any *particular* set of n elements and space usage close to the lower bound of $n \lg v + \lg \lg_v u$ bits shown in section 5. The previously best construction using $O(n \lg v + \lg \lg u)$ space is based on evaluation of a degree $n-1$ polynomial over a finite field and has $\Omega(n)$ evaluation time.

THEOREM 1. *Let $S \subseteq U = \{0, \ldots, u-1\}$ be a set of $n > 1$ elements. For any constants $c > 0$ and $\epsilon > 0$, and for $1 < v < u$, there is a RAM algorithm that, using time $\lg n (\lg v)^{O(1)}$ and $O(\lg n + \lg \lg u)$ bits of space, selects a family $\mathcal{H}$ of functions from U to $V = \{0, \ldots, v-1\}$ (independent of S) such that*

- *$\mathcal{H}$ is k-wise independent when restricted to S, with probability $1 - O(\frac{1}{n^c})$.*
- *A function in $\mathcal{H}$ can be represented by a RAM data structure using space $(1+\epsilon) n \lg v + O(n)$ bits such that function values can be computed in constant time. The data structure of a random function in $\mathcal{H}$ can be constructed in time $O(n)$.*

As our hash functions are optimal with respect to evaluation time and essentially optimal with respect to space usage, the only possible significant improvement would be an error probability that decreases more rapidly with n. Such a result could possibly be achieved for $u = n^{O(1)}$ by explicit constructions of certain expanders in Siegel's hash function construction.

Techniques. Our main technical ingredient is to use the two-choice paradigm [4] that has recently found a number of applications in load balancing and data structures. The central fact we make use of is that if we associate, using hashing, two memory locations in a linear size array with every element of the set S, then with high probability (whp.) there exists a way of associating keys with unique memory locations [19]. Essentially, each key gets the independence of its hash value from random bits at the memory location with which it is associated. A complication is that one needs to take care of cyclic dependencies that may arise, but this involves only a logarithmic number of elements, whp. The solution is to add a hash function that is independent (whp.) on the set of problematic elements.

Perspective. It should be noted that a data structure with slightly different functionality is very easy to construct: use a high performance dictionary such as the one in [9] to store elements from U and associated "hash values" from V. When a new function value is needed, it is randomly generated "on the fly" and stored with the element in the dictionary. The space needed for this is $O(n(\lg u + \lg v))$ bits, which is a constant factor from the space usage achieved by our data structure if $u = v^{O(1)}$. The main difference between this and the data structure of Theorem 1 is that our hash function can be generated once and for all, after which it may be used without any need for random bits. This also means that our hash function may be distributed and used by many parties without any need for additional communication. Another distinguishing feature is that our hash function will be uniform with high probability on *each* of $n^{O(1)}$ sets of size n. Thus it may be shared among many independently running algorithms or data structures.

1.3. Implications. The fact that the space usage of our hash function is linear in n means that a large class of hashing schemes can be implemented to perform, with high probability, *exactly as if uniform hashing was used*, increasing space by at most a constant factor. This means that our family makes a large number of analyses performed under the uniform hashing assumption come true with high probability.

Two comprehensive surveys of early data structures analyzed under the uniform hashing assumption can be found in the monographs of Gonnet [13] and Knuth [16]. Gonnet provides more than 100 references to books, surveys, and papers dealing with the analysis of classic hashing algorithms. This large body of work has made the characteristics of these schemes very well understood, under the uniform hashing assumption. As the classic hashing algorithms are often very simple to implement, and efficient in practice, they seem to be more commonly used in practice than newer schemes with provably good behavior. While our family may not be of practical importance for these hashing schemes, it does provide a theoretical bridge justifying the uniform hashing assumption for a large class of them. Previously, such justifications have been made for much more narrow classes of hashing schemes and have dealt only with certain performance parameters (see, e.g., [20, 21]). More details on applications of our scheme in hashing data structures can be found in the conference version of this paper [17].

In addition to the classic hashing schemes, our result provides the first provably efficient hashing based implementation of load balancing schemes of Azar et al. [4] and

Vöcking [24]. The fact that hash functions can be used to perform the random choices in these algorithms means that it is possible to retrace any previous load balancing decision by checking a small number of possible choices.

Finally, our construction has an application in cryptography, where it "derandomizes" a construction of Bellare, Goldreich, and Krawczyk [5] for the most important parameters. (See the discussion in section 3.1.1.)

1.4. Overview of the paper. The organization of the paper is as follows. In section 2 we provide the background information necessary to understand our construction. Specifically, we survey Siegel's construction, which will play an important role. Section 3 presents our initial construction, which achieves space that is within a constant factor of optimal. Section 4 shows, by a general reduction, how to reduce the space to (essentially) $1+\epsilon$ times the space lower bound shown in section 5.

2. Background. Theorem 1 can be seen as an improvement of Siegel's family of high performance hash functions [23]. The motivation for Siegel's work was that many algorithms employing hashing can be shown to work well if the hash functions are chosen at random from a *k-wise independent* family of functions, for suitably large k.

DEFINITION 1. *A family $\mathcal{H}$ of functions from U to V is* k-wise independent *if, for any distinct elements $x_1, \ldots, x_k \in U$, and any $y_1, \ldots, y_k \in V$, when $h \in \mathcal{H}$ is chosen uniformly at random,*

$$\Pr[h(x_1) = y_1 \wedge \cdots \wedge h(x_k) = y_k] = |V|^{-k}.$$

In other words, a random function from a k-wise independent family acts like a truly random function on any set of k elements of U. We note that several relaxed notions of k-wise independence exist, e.g., the notion of (c,k)-universality [8]. However, we don't know of any data structures in the literature that allow evaluation of a function from a (c,k)-universal (or similar) family in time $o(k)$, except for those achieving k-wise independence.

Siegel's construction. Siegel showed that for arbitrary constants $c > 0$ and $\epsilon > 0$ it is possible to construct, using $O(u^{\sqrt{\lg k/\lg u}+\epsilon} \lg v)$ bits of space, a family of functions from U to V with the following properties:

- It is k-wise independent with probability at least $1 - u^{-c}$.
- There is a RAM data structure of $O(u^{\sqrt{\lg k/\lg u}+\epsilon} \lg v)$ bits representing its functions such that function values can be computed in constant time.

Siegel mainly considered the case $k = u^{o(1)}$, e.g., $k = O(\lg u)$, where the space usage is dominated by the u^ϵ term. The positive probability that the family is not k-wise independent is due to the fact that Siegel's construction relies on a certain type of expander graph that, in lack of an explicit construction, is found by selecting a graph at random (and storing it). However, there is a small chance that the randomly chosen graph is not the desired expander, in which case there is no guarantee on the performance of the family. Also, there seems to be no known efficient way of generating a graph at random and verifying that it is the desired expander. (However, a slightly different class of expanders can be efficiently generated in this way [2].)

Space lower bound. It is inevitable that the space usage grows with u when constant evaluation time is required. Siegel shows the following trade-off between evaluation time and the size of the data structure.

THEOREM 2 (see Siegel [23]). *Consider any k-wise independent family $\mathcal{H}$ of functions from U to V and any RAM data structure using m words of $O(\lg v)$ bits to*

represent a function from $\mathcal{H}$. Then there exists $h \in \mathcal{H}$ and $x \in U$ such that the data structure for h requires time $\Omega(\min(\lg_{m/k}(u/k), k))$ on input x.

Note that when using optimal space, i.e., $m = O(k)$, the time needed to evaluate a function is $\Omega(\min(\lg(u/k), k))$.

Applying domain reduction. Theorem 2 establishes that high independence requires either high evaluation time or high space usage when u is large. A standard way of getting around problems with hashing from a large domain is to first perform a *domain reduction*, where elements of U are mapped to elements of a smaller domain U' using universal hashing. As this mapping cannot be 1-1, the domain reduction forces some hash function values to be identical. However, for any particular set S of n elements, the probability of two elements in S mapping to the same element of U' is $O(n^{-c})$ if $|U'| \geq n^{c+2}$. One universal family, suggested implicitly in [12] and explicitly in [18], uses primes in a certain range. A function from the family can be generated in expected time $(\lg|U'| + \lg\lg u)^{O(1)}$ and stored in $O(\lg|U'| + \lg\lg u)$ bits. Another universal family [8] has functions that can be generated in constant time and stored in $O(\lg u)$ bits. Both families support constant time evaluation of functions. In the following we will state all results using the former universal family, obtaining the smallest possible space at the cost of a modest amount of precomputation.

Using domain reduction with Siegel's family described above, one gets the following result. For $k = n$ it is similar to our main theorem, the main difference being that the space usage is superlinear.

THEOREM 3 (see Siegel [23]). *Let $S \subseteq U = \{0, \ldots, u-1\}$ be a set of n elements. For any constants $\epsilon > 0$ and $c > 0$ there is a RAM algorithm constructing a random family $\mathcal{SI}(U, V, k, n, c, \epsilon)$ of functions from U to $V = \{0, \ldots, v-1\}$ in expected time $O(s) + (\lg\lg u)^{O(1)}$ and $O(s \lg v + \lg\lg u)$ bits of space, where $s = n^{\sqrt{(c+2)\lg k / \lg n} + \epsilon}$, such that.*

- *With probability $1 - O(\frac{1}{n^c})$ the family is k-wise independent when restricted to S.*
- *There is a RAM data structure of $O(s \lg v + \lg\lg u)$ bits representing its functions such that function values can be computed in constant time. The data structure can be initialized to a random function in time $O(s)$.*

Notice that the space usage is $\omega(k^{\sqrt{2}})$ bits for all parameters, so it is truly superlinear in k. With currently known ways of constructing expanders, Siegel's hash function family exhibits high constant factors.

Other constructions. Other proposals for high-performance hash functions, due to Dietzfelbinger and Meyer auf der Heide [9, 10], appear more practical than Siegel's. However, these families only exhibit $O(1)$-wise independence and are difficult to analyze in general.

Dietzfelbinger and Woelfel [11] have analyzed a family similar to the abovementioned high-performance hash functions and showed that it may be used for hashing schemes whose analysis rests on a bipartite graph defined by a pair of hash functions. In particular, they are able to give an alternative to our initial uniform hashing construction, described in section 3, that is likely to be more practical. However, their construction restricts the size v of the range to be a prime number.

3. Initial hash function construction. In this section we describe a hash function family with properties as stated in Theorem 1 for *some* constant $\epsilon > 0$. In section 4 we will extend this to show Theorem 1 (where ϵ can be any positive constant). We use the notation $T[i]$ to denote the ith entry in an array T. By $[m]$ we denote the set $\{0, \ldots, m-1\}$.

3.1. The hash function family. We start with a definition.

DEFINITION 2. *Let $\mathcal{G}$ be a family of functions from U to V, and consider functions $i_1, i_2 : U \to [m]$. We define the family of functions*

$$\mathcal{H}(i_1, i_2, \mathcal{G}) = \{x \mapsto (T_1[i_1(x)] + T_2[i_2(x)] + g(x)) \bmod v \mid T_1, T_2 \in V^m \text{ and } g \in \mathcal{G}\}.$$

The hash function family considered in this section uses Siegel's construction of function families to get the functions i_1 and i_2, as well as the family $\mathcal{G}$ in the above definition.

DEFINITION 3. *For $n \leq u$ and any constant $c > 0$ we define the random family $\mathcal{H}_{n,c} = \mathcal{H}(i_1, i_2, \mathcal{G})$ of functions as follows. Let $k = \lceil n^{1/(2c+4)} \rceil$ and construct the random families*

$$\mathcal{G} = \mathcal{SI}(U, V, k, n, c, 1/4) \text{ and}$$
$$\mathcal{F} = \mathcal{SI}(U, [4n], k, n, c, 1/4)$$

according to Theorem 3. Pick i_1 and i_2 independently at random from $\mathcal{F}$.

3.1.1. Related constructions. A similar way of constructing a function family was presented in [9]. The essential change in the above definition compared to [9] is that we look up *two* values in tables, rather than just one.

The technique of using multiple lookups in a random (or pseudorandom) table to produce a new random value has previously been used in cryptography by Bellare, Goldreich, and Krawczyk [5] in connection with stateless evaluation of pseudorandom functions. The construction given in this paper is strictly more general that the one in [5] as we get a random *function* rather than just a way of generating random values. Also, function evaluation is deterministic, whereas the generation procedure in [5] is randomized. Our analysis is completely different from the one in [5].

On the other hand, our analysis shares some features with the analysis of cuckoo hashing in [19], as it rests on the analysis of the same random bipartite graph (generated by Siegel's hash functions). In fact, Dietzfelbinger and Woelfel show in [11] how to base uniform hashing (with range of size that is a prime number) on any hash function that works with cuckoo hashing.

3.2. Properties of the family. For two functions $i_1, i_2 : U \to [m]$ and a set $S \subseteq U$, let $G(i_1, i_2, S) = (A, B, E)$ be the bipartite graph that has left vertex set $A = \{a_1, \ldots, a_m\}$, right vertex set $B = \{b_1, \ldots, b_m\}$, and edge set

$$E = \{e_x \mid x \in S\}, \text{ where } e_x = (a_{i_1(x)}, b_{i_2(x)}).$$

We consider the edge e_x to be labeled by x. Note that there may be parallel edges with different labels.

We define a *leafless subgraph* $E' \subseteq E$ of a graph as a subset of the edges such that there is no vertex incident to exactly one edge in E'. A graph's *leafless part* $C \subseteq E$ consists of the edges that are on a cycle and the edges that are on a path connecting two cycles. (This is also known as the 2-core.)

LEMMA 1. *Let $S \subseteq U$ be a set of n elements and let $\mathcal{G}$ be a family of functions from U to V that is k-wise independent on S. If the total number of edges in the leafless part of $G(i_1, i_2, S) = (A, B, E)$ is at most k, then $\mathcal{H}(i_1, i_2, \mathcal{G})$ is uniform when restricted to S.*

Proof. Let S' be the set of all elements $x \in S$, where the edge e_x with label x is in the leafless part C of $G(i_1, i_2, S)$.

The proof is by induction on $|E \setminus C|$. In the base case we assume that $|E \setminus C| = 0$, i.e., that $S' = S$. Since $|S| \leq k$ it holds for any function h that the function family $x \mapsto (h(x) + g(x)) \bmod v$, where g is chosen from a k-wise independent family, is uniform on S. In particular, $\mathcal{H}(i_1, i_2, \mathcal{G})$ has this property.

For the inductive step, note that among the edges in $E \setminus C$ there has to be at least one edge with one unique endpoint. Let $e_{x^*} = (a_{i_1(x^*)}, b_{i_2(x^*)})$ be such an edge, $x^* \in S \setminus S'$. By symmetry we may assume that $a_{i_1(x^*)}$ is the unique endpoint. The induction hypothesis says that $\mathcal{H}(i_1, i_2, \mathcal{G})$ is uniform on $S \setminus \{x^*\}$, and for $h \in \mathcal{H}(i_1, i_2, \mathcal{G})$ chosen at random, all values $h(x)$ for $x \in S \setminus \{x^*\}$ are independent of the value $T_1[i_1(x^*)]$. These facts mean that given $g \in \mathcal{G}$ and all entries in vectors T_1 and T_2 except $T_1[i_1(x^*)]$, $h(x^*)$ is uniformly distributed when choosing $T_1[i_1(x^*)]$ at random. Hence $\mathcal{H}(i_1, i_2, \mathcal{G})$ is uniform on S. □

LEMMA 2. *For each set S of size n, and for $i_1, i_2 : U \to [4n]$ chosen at random from a family that is k-wise independent on S, $k \geq 32$, the probability that the leafless part C of the graph $G(i_1, i_2, S)$ has size at least k is $n/2^{\Omega(k)}$.*

Proof. Assume that $|C| \geq k$ and that $k \leq n$ is even (this may be assumed without loss of generality). Either there is a *connected* leafless subgraph in $G(i_1, i_2, S)$ of size at least $k/2$ or there is a leafless subgraph of size k', where $k/2 < k' \leq k$. In the first case there is a connected subgraph in $G(i_1, i_2, S)$ with exactly $k/2$ edges and at most $k/2 + 1$ vertices. In the second case there is a subgraph with k' edges and at most k' vertices in $G(i_1, i_2, S)$.

In the following we will count the number of different edge-labeled subgraphs with k' edges and at most $k' + 1$ vertices for $k/2 \leq k' \leq k$ to bound the probability of such a subgraph to appear in $G(i_1, i_2, S)$. Hence, we also get an upper bound on the probability that $|C|$ is at least k. Note that since i_1 and i_2 are chosen from a k-wise independent family, each subset of at most k elements of S will map to to random and independent edges. We will only consider subgraphs corresponding to at most k elements of S.

To count the number of different subgraphs with k' edges and at most $k' + 1$ vertices, for $k/2 \leq k' \leq k$, in a bipartite graph $G = (A, B, E)$ with $|A| = |B| = 4n$ and $|E| = n$, we count the number of ways to choose the edge labels, the vertices, and the endpoints of the edges such that they are among the chosen vertices. The k' edge labels can be chosen in $\binom{n}{k'} \leq (en/k')^{k'}$ ways. Since the number of vertices in the subgraph is at most $k' + 1$, and they are chosen from $8n$ vertices in G, the total number of ways in which they can be chosen is bounded by

$$\sum_{i=1}^{k'+1} \tbinom{8n}{i} \leq (8en/(k'+1))^{k'+1}.$$

Let k_a and k_b be the number of vertices chosen from A and B, respectively. The number of ways to choose an edge such that it has both its endpoints among the chosen vertices is $k_a k_b \leq ((k'+1)/2)^{2k'}$. In total, the number of different subgraphs with k' edges and up to $k' + 1$ vertices is at most

$$\begin{aligned}
&(en/k')^{k'} \cdot (8en/(k'+1))^{k'+1} \cdot ((k'+1)/2)^{2k'} \\
&= \tfrac{8en}{k'+1} \cdot (2e^2 \cdot n^2 \cdot \tfrac{k'+1}{k'})^{k'} \\
&\leq \tfrac{8en}{k'+1} \cdot (\tfrac{63}{4} \cdot n^2)^{k'},
\end{aligned}$$

using $k' \geq k/2 \geq 16$.

There are in total $(4n)^{2k'}$ graphs with k' specific edges. In particular, the probability that k' specific edges form a particular graph is $(4n)^{-2k'}$, using k'-wise independence. To get an upper bound on the probability that there is some subgraph with k' edges and at most $k'+1$ vertices, where $k/2 \leq k' \leq k$, we sum over all possible values of k':

$$\begin{aligned}\sum_{k/2\leq k'\leq k} & \tfrac{8en}{k'+1}\cdot(\tfrac{63}{4}\cdot n^2)^{k'}\cdot(4n)^{-2k'} \\ &= \sum_{k/2\leq k'\leq k} \tfrac{8en}{k'+1}\cdot(\tfrac{63}{64})^{k'} \\ &\leq (k/2+1)\cdot\tfrac{8en}{k/2+1}\cdot(\tfrac{63}{64})^{k/2} \\ &= n/2^{\Omega(k)}. \qquad \square\end{aligned}$$

We now proceed to show Theorem 1 for *some* constant $\epsilon > 0$. More precisely, we will show that the random family $\mathcal{H}_{n,c}$ of Definition 3 fulfills the requirements in the theorem. The families $\mathcal{G} = \mathcal{SI}(U, V, k, n, c, 1/4)$ and $\mathcal{F} = \mathcal{SI}(U, [4n], k, n, c, 1/4)$ are both k-wise independent with probability $1 - n^{-c}$ for sets of size up to n according to Theorem 3. If $\mathcal{F}$ is k-wise independent, then by Lemma 2 the probability that the leafless part of graph $G(i_1, i_2, S)$ has size at most k is at least $1 - n^{-\Omega(k)}$ if $k \geq 32$. We can assume without loss of generality that $k \geq 32$, since otherwise the theorem follows directly from Theorem 3. When the leafless part of graph $G(i_1, i_2, S)$ has size at most k, then, by Lemma 1, $\mathcal{H}_{n,c}$ is uniform on S if $\mathcal{G}$ is k-wise independent. The probability that $\mathcal{G}$ is k-wise independent, that $\mathcal{F}$ is k-wise independent, and that the leafless part of the graph $G(i_1, i_2, S)$ has size at most k is altogether $(1 - n^{-c})^2(1 - 2^{-\Omega(k)}) = 1 - O(n^{-c})$.

The construction of $\mathcal{H}_{n,c}$, i.e., constructing $\mathcal{F}$ and $\mathcal{G}$ and choosing i_1 and i_2, can according to Theorem 3 be done in expected $O(s)+(\lg\lg u)^{O(1)}$ time and $O(s \lg v)$ bits of space, where $s = n^{\sqrt{(c+2)\lg k/\lg n}+1/4} = O(n^{0.96})$. The space usage of a data structure representing a function from $\mathcal{H}_{n,c}$ is $O(n \lg v)$ bits for T_1 and T_2 and $O(s \lg v + \lg\lg u)$ bits for storing i_1, i_2, and g. The initialization time is dominated by the time used for initializing T_1 and T_2 to random arrays and the $(\lg\lg u)^{O(1)}$ term from Siegel's construction. Function values can clearly be computed in constant time.

4. Achieving near-optimal space. In this section we present a general reduction for decreasing the space used by a uniform hashing construction. Together with section 3 this will show Theorem 1. The idea of the reduction is the following. We construct a data structure of $(1+\epsilon/2)n \lg v + O(n)$ bits representing a function f such that for any set $S \subseteq U$ of n elements there exists, with probability $1 - O(n^{-c})$, a set $S' \subseteq S$, where $|S'| \geq (1 - \epsilon/16)\, n$, such that the following independence property holds:

- The values $(f(x))_{x\in S'}$ are independent and uniform in $[v]$, even when conditioned on particular f-values for elements in $S \backslash S'$.

To be precise, we associate with each possible choice of f a unique set S'. The uniformity property above holds when restricting f to the subset of functions associated with any particular set S'. (The probability space may be empty for some choices of S'.)

Now choose a function h according to a uniform hashing construction for sets of size $(\epsilon/16)\, n$, and error probability $O(n^{-c})$. We consider the function

$$x \mapsto (f(x) + h(x)) \bmod v.$$

For a given set $S \subseteq U$ of size n, there exists with probability $1 - O(n^{-c})$ a set S' with properties as stated above. With probability $1 - O(n^{-c})$ the function h then maps the elements of $S \backslash S'$ to uniformly distributed and independent values. Hence, using the independence property of f, the function $x \mapsto (f(x) + h(x)) \bmod v$ must be uniform on the set S with probability $1 - O(n^{-c})$.

Using the uniform hashing construction of section 3, the space to represent h is $(8\epsilon/16)\, n\lceil \lg v \rceil + o(n) + O(\lg \lg u) = (\epsilon/2) n \lg v + O(n + \lg \lg u)$ bits, so the total space for storing f and h is as required in Theorem 1.

4.1. Details of the reduction. It remains to see how to construct f in the desired space bound, with the desired preprocessing time, and such that function values can be computed in constant time. Below we will several times refer to a constant ℓ, which is assumed to be "sufficiently large." That is, the arguments are valid if ℓ is set to a sufficiently large constant. We first notice that for the case where $v \leq \ell$, Theorem 1 was already shown in section 3. Thus, in the following we may assume that $v > \ell$. Let $p \geq v$ be a prime in the range v to $v + O(v^{2/3})$. We know from [14] that there are $\Omega(v^{2/3}/\lg v)$ such primes, so p can be found in time $\lg n (\lg v)^{O(1)}$ with high probability by sampling [1].

Let $d = \lceil \ell/\epsilon^2 \rceil$, $k = \lceil n^{1/(2c+4)} \rceil$, $r_1 = \lceil (1 + \epsilon/2) n/d \rceil$, and let $r_2 > 9d^3/\epsilon$ be a constant. Since v can be assumed to be sufficiently large, we have that $p - v < \epsilon v/(9d)$. Now pick the following functions uniformly at random:

$$\rho_1 : U \to \{0, \ldots, r_1 - 1\} \text{ from } \mathcal{SI}(U, [r_1], k, n, c, 1/4),$$
$$\rho_2 : U \to \{0, \ldots, r_2 - 1\} \text{ from } \mathcal{SI}(U, [r_2], k, n, c, 1/4).$$

Also, pick functions $f_1, \ldots, f_{r_1}$ independently at random from the family of degree $d-1$ polynomials in the field of size p, which is known to be d-wise independent. Note that such a polynomial can be stored in $d\lceil \lg p \rceil = d \lg v + O(d)$ bits and evaluated in $O(d)$ time using arithmetic modulo p. Without loss of generality we may assume that $\epsilon < 1$ and $p \geq r_2$. Thus we may interpret a value of ρ_2 as an input to $f_1, \ldots, f_{r_1}$, and the following is well -defined:

$$f(x) = f_{\rho_1(x)}(\rho_2(x)).$$

Observe that f may have function values up to $p - 1$, i.e., not in $[v]$. However, we will define S' such that the function values of elements in S' are uniform in $[v]$, and this suffices for the reduction to work.

4.2. Analysis.

Time and space usage. We first argue that f has the desired properties with respect to storage, preprocessing time, and evaluation time. The storage required for ρ_1 and ρ_2 is bounded by the storage used in the construction of section 3, so it can be ignored in this context. The functions $f_1, \ldots, f_{r_1}$ require space $r_1(d \lg v + O(d)) = (1 + \epsilon/2) n \lg v + O(n)$ bits, and a function can be evaluated in $O(1)$ time, since d is a constant. Selection of ρ_1 and ρ_2 can be done in $o(n)$ time, while construction of $f_1, \ldots, f_{r_1}$ requires expected time $w^{O(1)}$ to find the prime p, and $O(n)$ time to choose the random polynomials.

Independence. To argue that f has the desired independence property, we let $S_i = \{x \in S \mid \rho_1(x) = i\}$ and define S' to be the union of the sets S_i for which

- $|S_i| \leq d$,
- $|\rho_2(S_i)| = |S_i|$, i.e., ρ_2 is 1-1 on S_i, and
- $f_i(\rho_2(S_i)) \subseteq [v]$.

In other words, if we think of ρ_1 as hashing into buckets of capacity d, the set S' consists of those elements that are hashed to a bucket i that does not overflow, whose elements have no collisions under ρ_2, and where f_i produces values in the right range for all elements in the bucket.

Consider a nonempty part of the probability space where f is associated with a particular set S' and has a specific, fixed value on each of the elements in $S\backslash S'$. We will argue that if f is chosen in this part of the probability space, we get a uniform function on S'. First, function values of elements hashing to different buckets are completely independent as $f_1, \ldots, f_{r_1}$ are chosen independently. If S_i is part of S', then $|S_i| \leq d$ by definition of S'. Since f_i is d-wise independent and there is no collision among elements in S_i under ρ_2, the f-values of elements in S_i are independent and uniformly random in $[v]$ (because of the requirement $f_i(\rho_2(S_i)) \subseteq [v]$). This concludes the argument for uniformity on S'.

Size of S'. Finally, we need to show that $|S'| \geq (1 - \epsilon/16)\, n$ with probability $1 - O(n^{-c})$. For this we split $[r_1]$ into blocks of at most 2^z consecutive integers $I_j = \{2^z j, \ldots, 2^z(j+1) - 1\}$, $j = 0, 1, 2, \ldots$, where $z = \Omega(\lg n)$ is the highest integer for which $2^z d < k$. If 2^z does not divide r_1, there will be one block of size less than 2^z. If we conservatively assume that all elements of S having a ρ_1-value in this final block will be part of S', it will follow from the arguments below that this will contribute only negligibly to S'. Thus we simply assume that 2^z divides r_1.

First we observe that for all j, $|\cup_{i\in I_j} S_i| < (1-\epsilon/4)2^z d$ with probability $1-n^{-\omega(1)}$. This follows from Chernoff bounds for random variables with limited dependence [22, Theorem 5.I.b]. On the condition that $|\cup_{i\in I_j} S_i| < (1 - \epsilon/4)2^z d$ (which is assumed in the following) the z least significant bits of the ρ_1-values of elements in $\cup_{i\in I_j} S_i$ will be random and independent for any particular j. We conclude the argument by proving that for any j, $|S' \cap (\cup_{i\in I_j} S_i)| \geq (1 - \epsilon/16)|\cup_{i\in I_j} S_i|$ with probability $1 - O(n^{-\omega(1)})$. By the union bound this will imply that S' has the desired size with probability $1 - O(n^{-\omega(1)})$.

Consider the indicator random variables Y_x, $x \in \cup_{i\in I_j} S_i$, where $Y_x = 1$ if and only if x has the same value under ρ_1 as at least d other elements in $\cup_{i\in I_j} S_i$. Observe that if $Y_x = 1$, then x is not included in S' due to the first requirement in the definition of S. By uniformity of ρ_1 in the z least-significant bits, and since the expected number of elements colliding with x is bounded by $(1 - \epsilon/4)d$, it follows from classical Chernoff bounds that $\Pr(Y_x = 1) \leq \epsilon/6$. The random variables Y_x are not independent; however, they are *negatively related* [15], which means that we can apply a Chernoff bound on the sum of the Y_xs to show that it is bounded by $(\epsilon/4)|\cup_{i\in I_j} S_i|$ with probability $1 - n^{\omega(1)}$ [15].

Finally, consider the indicator random variables X_i, $i \in I_j$, where $X_i = 1$ if and only if $|S_i| \leq d$ and either $|\rho_2(S_i)| < |S_i|$ or $f_i(\rho_2(S_i)) \not\subseteq [v]$. That is, X_i indicates whether the set S_i fails to be included in S' because of at least one of the two last requirements in the definition of S'. For each variable X_i equal to 1 we have at most d elements (those in S_i) that are not part of S'. We next show that with probability $1 - n^{\omega(1)}$ the sum $\sum_{i\in I_j} X_i$ is bounded by $2^z\epsilon/(4d)$, which means that the number of elements not included in S' due to requirements two and three is at most $(\epsilon/4)|\cup_{i\in I_j} S_i|$. Since ρ_2 is independent on all elements in $\cup_{i\in I_j} S_i$, the X_i are independent. By the choice of p and r_2 we have that for all i, $\Pr(X_i = 1) \leq \frac{p-v}{v} + \binom{d}{2}/r_2 < 2\epsilon/(9d)$. Hence by Chernoff bounds $\sum_{i\in I_j} X_i < 2^z\epsilon/(4d)$ with probability $1-n^{\omega(1)}$. Together with the similar bound above for the first requirement, this shows that S' has the desired size with high probability.

The combined construction. For ease of reference we state the full uniform hashing construction used to show Theorem 1:

$$x \mapsto (f_{\rho_1(x)}(\rho_2(x)) + T_1[i_1(x)] + T_2[i_2(x)] + g(x)) \bmod v.$$

5. Space lower bound. We now show that our space usage in bits is close to the best possible. To this end, note that any data structure achieving n-wise independence on a set S of n elements, with nonzero probability, must be able to represent every function from S to V.

THEOREM 4. *For integers $u \geq n \geq 2$ and $v \geq 2$, let $U = \{0, \ldots, u-1\}$ and $V = \{0, \ldots, v-1\}$. Any data structure representing functions $h : U \to V$ such that the restriction $h|_S$ to any set $S \subseteq U$ of n elements can be an arbitrary function from S to V must use space $\max(n \lg v, \lg \lg_v u)$ bits.*

Proof. Even for fixed S, $n \lg v$ bits are necessary to be able to represent all functions from S to V. Second, if the data structure can represent fewer than $\lg_v u$ different functions, there will be elements $x_1, x_2 \in U$ such that all functions map x_1 and x_2 to the same value, contradicting the assumptions of the theorem. Thus the data structure must have at least $\lg \lg_v u$ bits. □

Note that when $\lg v < \lg \lg u$, the second term in the lower bound is $\Omega(\lg \lg u)$, so the lower bound is $\Omega(n \lg v + \lg \lg u)$ bits.

Acknowledgments. We would like to thank Martin Dietzfelbinger for enlightening discussions and in particular for pointing out an error in the quotation of Siegel's result in the conference version of this paper. Also, we thank an anonymous reviewer for help on improving the presentation.

REFERENCES

[1] M. AGRAWAL, N. KAYAL, AND N. SAXENA, *PRIMES is in P*, Ann. Math., 160 (2004), pp. 781–793.

[2] N. ALON, *Eigenvalues and expanders*, Combinatorica, 6 (1986), pp. 83–96.

[3] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[4] Y. AZAR, A. Z. BRODER, A. R. KARLIN, AND E. UPFAL, *Balanced allocations*, SIAM J. Comput., 29 (1999), pp. 180–200.

[5] M. BELLARE, O. GOLDREICH, AND H. KRAWCZYK, *Stateless Evaluation of Pseudorandom Functions: Security Beyond the Birthday Barrier*, in Proceedings of 19th Annual International Cryptology Conference (CRYPTO'99), Lecture Notes in Comput. Sci. 1666, Springer Verlag, 1999, pp. 270–287.

[6] J. L. CARTER AND M. N. WEGMAN, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[7] B. CHOR, O. GOLDREICH, J. HASTAD, J. FRIEDMAN, S. RUDICH, AND R. SMOLENSKY, *The bit extraction problem of t-resilient functions (preliminary version)*, in Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS '85), IEEE Computer Society Press, 1985, pp. 396–407.

[8] M. DIETZFELBINGER, *Universal Hashing and k-Wise Independent Random Variables via Integer Arithmetic Without Primes*, in Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS '96), Lecture Notes in Comput. Sci. 1046, Springer Verlag, 1996, pp. 569–580.

[9] M. DIETZFELBINGER AND F. MEYER AUF DER HEIDE, *A New Universal Class of Hash Functions and Dynamic Hashing in Real Time*, in Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP '90), Lecture Notes in Comput. Sci. 443, Springer Verlag, 1990, pp. 6–19.

[10] M. DIETZFELBINGER AND F. MEYER AUF DER HEIDE, *High Performance Universal Hashing, with Applications to Shared Memory Simulations*, in Data Structures and Efficient Algorithms, Lecture Notes in Comput. Sci. 594, Springer, 1992, pp. 250–269.

[11] M. Dietzfelbinger and P. Woelfel, *Almost Random Graphs with Simple Hash Functions*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC '03), 2003, pp. 629–638.

[12] M. L. Fredman, J. Komlós, and E. Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, J. Assoc. Comput. Mach., 31 (1984), pp. 538–544.

[13] G. Gonnet, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1984.

[14] D. R. Heath-Brown and H. Iwaniec, *On the difference between consecutive primes*, Invent. Math., 55 (1979), pp. 49–69.

[15] S. Janson, *Large Deviation Inequalities for Sums of Indicator Variables*, Tech. report 34, Department of Mathematics, Uppsala University, 1993.

[16] D. E. Knuth, *Sorting and Searching*, 2nd ed., Art of Computer Programming 3, Addison-Wesley, Reading, MA, 1998.

[17] A. Östlin and R. Pagh, *Uniform hashing in constant time and linear space*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC '03), ACM Press, 2003, pp. 622–628.

[18] R. Pagh, *Dispersing hash functions*, in Proceedings of the 4th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '00), Proc. Informatics 8, Carleton Scientific, 2000, pp. 53–67.

[19] R. Pagh and F. F. Rodler, *Cuckoo hashing*, J. Algorithms, 51 (2004), pp. 122–144.

[20] J. P. Schmidt and A. Siegel, *On aspects of universality and performance for closed hashing (extended abstract)*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89), ACM Press, 1989, pp. 355–366.

[21] J. P. Schmidt and A. Siegel, *The analysis of closed hashing under limited randomness (extended abstract)*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90), ACM Press, 1990, pp. 224–234.

[22] J. P. Schmidt, A. Siegel, and A. Srinivasan, *Chernoff-Hoeffding bounds for applications with limited independence*, SIAM J. Discrete Math., 8 (1995), pp. 223–250.

[23] A. Siegel, *On universal classes of extremely random constant-time hash functions*, SIAM J. Comput., 33 (2004), pp. 505–543.

[24] B. Vöcking, *How asymmetry helps load balancing*, J. Assoc. Comput. Mach., 50 (2003), pp. 568–589.

[25] M. N. Wegman and J. L. Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 97–139

FUZZY EXTRACTORS: HOW TO GENERATE STRONG KEYS FROM BIOMETRICS AND OTHER NOISY DATA*

YEVGENIY DODIS†, RAFAIL OSTROVSKY‡, LEONID REYZIN§, AND ADAM SMITH¶

Abstract. We provide formal definitions and efficient secure techniques for turning noisy information into keys usable for *any* cryptographic application, and, in particular, reliably and securely authenticating biometric data. Our techniques apply not just to biometric information, but to any keying material that, unlike traditional cryptographic keys, is (1) not reproducible precisely and (2) not distributed uniformly. We propose two primitives: a *fuzzy extractor* reliably extracts nearly uniform randomness R from its input; the extraction is error-tolerant in the sense that R will be the same even if the input changes, as long as it remains reasonably close to the original. Thus, R can be used as a key in a cryptographic application. A *secure sketch* produces public information about its input w that does not reveal w and yet allows exact recovery of w given another value that is close to w. Thus, it can be used to reliably reproduce error-prone biometric inputs without incurring the security risk inherent in storing them. We define the primitives to be both formally secure and versatile, generalizing much prior work. In addition, we provide nearly optimal constructions of both primitives for various measures of "closeness" of input data, such as Hamming distance, edit distance, and set difference.

Key words. fuzzy extractors, fuzzy fingerprints, randomness extractors, error-correcting codes, biometric authentication, error-tolerance, nonuniformity, password-based systems, metric embeddings

AMS subject classifications. 68P25, 68P30, 68Q99, 94A17, 94A60, 94B35, 94B99

DOI. 10.1137/060651380

1. Introduction.

Cryptography traditionally relies on uniformly distributed and precisely reproducible random strings for its secrets. Reality, however, makes it difficult to create, store, and reliably retrieve such strings. Strings that are neither uniformly random nor reliably reproducible seem to be more plentiful. For example, a random person's fingerprint or iris scan is clearly not a uniform random string, nor does it get reproduced precisely each time it is measured. Similarly, a long pass-phrase (or answers to 15 questions [31] or a list of favorite movies [38]) is not uniformly random and is difficult to remember for a human user. This work is about using such

*Received by the editors February 2, 2006; accepted for publication (in revised form) September 21, 2007; published electronically March 28, 2008. A preliminary version of this work appeared in Eurocrypt 2004 [25].

http://www.siam.org/journals/sicomp/38-1/65138.html

†Department of Computer Science, New York University, 251 Mercer St., New York, NY 10012 (dodis@cs.nyu.edu). The work of this author was partly funded by the National Science Foundation under CAREER Award CCR-0133806 and Trusted Computing grant CCR-0311095, and by the New York University Research Challenge Fund 25-74100-N5237.

‡Department of Computer Science, University of California, Los Angeles, Box 951596, 3732D BH, Los Angeles, CA 90095 (rafail@cs.ucla.edu).

§Department of Computer Science, Boston University, 111 Cummington St., Boston, MA 02215 (reyzin@cs.bu.edu). The work of this author was partly funded by the National Science Foundation under grants CCR-0311485, CCF-0515100, and CNS-0202067.

¶Department of Computer Science and Engineering, Pennsylvania State University, 342 IST, University Park, PA 16803 (asmith@cse.psu.edu). The research reported here was done while this author was a student at the Computer Science and Artificial Intelligence Laboratory at MIT and a postdoctoral fellow at the Weizmann Institute of Science. The work of this author at MIT was partly funded by US A.R.O. grant DAAD19-00-1-0177 and by a Microsoft Fellowship. While at the Weizmann Institute, this author was supported by the Louis L. and Anita M. Perlman Postdoctoral Fellowship.

nonuniform and unreliable secrets in cryptographic applications. Our approach is rigorous and general, and our results have both theoretical and practical value.

To illustrate the use of random strings on a simple example, let us consider the task of password authentication. A user Alice has a password w and wants to gain access to her account. A trusted server stores some information $y = f(w)$ about the password. When Alice enters w, the server lets Alice in only if $f(w) = y$. In this simple application, we assume that it is safe for Alice to enter the password for the verification. However, the server's long-term storage is not assumed to be secure (e.g., y is stored in a publicly readable `/etc/passwd` file in UNIX [55]). The goal, then, is to design an efficient f that is hard to invert (i.e., given y it is hard to find w' such that $f(w') = y$) so that no one can figure out Alice's password from y. Recall that such functions f are called *one-way functions*.

Unfortunately, the solution above has several problems when used with passwords w available in real life. First, the definition of a one-way function assumes that w is *truly uniform* and guarantees nothing if this is not the case. However, human-generated and biometric passwords are far from uniform, although they do have some unpredictability in them. Second, Alice has to reproduce her password *exactly* each time she authenticates herself. This restriction severely limits the kinds of passwords that can be used. Indeed, a human can precisely memorize and reliably type in only relatively short passwords, which do not provide an adequate level of security. Greater levels of security are achieved by longer human-generated and biometric passwords, such as pass-phrases, answers to questionnaires, handwritten signatures, fingerprints, retina scans, voice commands, and other values selected by humans or provided by nature, possibly in combination (see [30] for a survey). These measurements seem to contain much more entropy than human-memorizable passwords. However, two biometric readings are rarely identical, even though they are likely to be close; similarly, humans are unlikely to precisely remember their answers to multiple questions from time to time, though such answers will likely be similar. In other words, the ability to tolerate a (limited) number of errors in the password while retaining security is crucial if we are to obtain greater security than provided by typical user-chosen short passwords.

The password authentication described above is just one example of a cryptographic application where the issues of nonuniformity and error-tolerance naturally come up. Other examples include any cryptographic application, such as encryption, signatures, or identification, where the secret key comes in the form of noisy nonuniform data.

Our definitions. As discussed above, an important general problem is to convert noisy nonuniform inputs into reliably reproducible, uniformly random strings. To this end, we propose a new primitive, termed *fuzzy extractor*. It extracts a uniformly random string R from its input w in a noise-tolerant way. Noise-tolerance means that if the input changes to some w' but remains close, the string R can be reproduced exactly. To assist in reproducing R from w', the fuzzy extractor outputs a non-secret string P. It is important to note that R remains uniformly random even given P. (Strictly speaking, R will be ϵ-close to uniform rather than uniform; ϵ can be made exponentially small, which makes R as good as uniform for the usual applications.)

Our approach is general: R extracted from w can be used as a key in a cryptographic application but, unlike traditional keys, need not be stored (because it can be recovered from any w' that is close to w). We define fuzzy extractors to be *information-theoretically* secure, thus allowing them to be used in cryptographic systems without introducing additional assumptions (of course, the cryptographic ap-

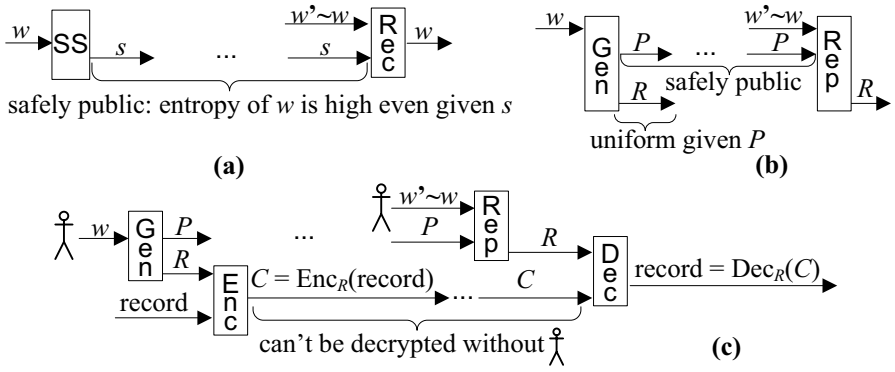


FIG. 1. (a) *Secure sketch.* (b) *Fuzzy extractor.* (c) *A sample application: User who encrypts a sensitive record using a cryptographically strong, uniform key R extracted from biometric w via a fuzzy extractor; both P and the encrypted record need not be kept secret, because no one can decrypt the record without a w' that is close.*

plication itself will typically have computational, rather than information-theoretic, security).

For a concrete example of how to use fuzzy extractors, in the password authentication case, the server can store $(P, f(R))$. When the user inputs w' close to w, the server reproduces the actual R using P and checks if $f(R)$ matches what it stores. The presence of P will help the adversary invert $f(R)$ only by the additive amount of ϵ, because R is ϵ-close to uniform even given P.[1] Similarly, R can be used for symmetric encryption, for generating a public-secret key pair, or for other applications that utilize uniformly random secrets.[2]

As a step in constructing fuzzy extractors, and as an interesting object in its own right, we propose another primitive, termed *secure sketch*. It allows precise reconstruction of a noisy input as follows: on input w, a procedure outputs a sketch s. Then, given s and a value w' close to w, it is possible to recover w. The sketch is secure in the sense that it does not reveal much about w: w retains much of its entropy even if s is known. Thus, instead of storing w for fear that later readings will be noisy, it is possible to store s instead, without compromising the privacy of w. A secure sketch, unlike a fuzzy extractor, allows for the precise reproduction of the original input but does not address nonuniformity.

Secure sketches, fuzzy extractors, and a sample encryption application are illustrated in Figure 1.

Secure sketches and extractors can be viewed as providing fuzzy key storage: they allow recovery of the secret key (w or R) from a faulty reading w' of the password w by using some public information (s or P). In particular, fuzzy extractors can be viewed as error- and nonuniformity-tolerant secret key *key-encapsulation mechanisms* [65].

Because different biometric information has different error patterns, we do not

[1] To be precise, we should note that because we do not require w, or hence P, to be efficiently samplable, we need f to be a one-way function even in the presence of samples from w; this is implied by security against circuit families.

[2] Naturally, the security of the resulting system should be properly defined and proven and will depend on the possible adversarial attacks. In particular, in this work we do not consider active attacks on P or scenarios in which the adversary can force multiple invocations of the extractor with related w and get to observe the different P values. See [8, 9, 23] for follow-up work that considers attacks on the fuzzy extractor itself.

assume any particular notion of closeness between w' and w. Rather, in defining our primitives, we simply assume that w comes from some metric space and that w' is no more than a certain distance from w in that space. We consider particular metrics only when building concrete constructions.

General results. Before proceeding to construct our primitives for concrete metrics, we make some observations about our definitions. We demonstrate that fuzzy extractors can be built out of secure sketches by utilizing strong *randomness extractors* [56], such as, for example, universal hash functions [12, 75] (randomness extractors, defined more precisely below, are families of hash which "convert" a high entropy input into a shorter, uniformly distributed output). We also provide a general technique for constructing secure sketches from transitive families of isometries, which is instantiated in concrete constructions later in the paper. Finally, we define a notion of a *biometric embedding* of one metric space into another and show that the existence of a fuzzy extractor in the target space, combined with a biometric embedding of the source into the target, implies the existence of a fuzzy extractor in the source space.

These general results help us in building and analyzing our constructions.

Our constructions. We provide constructions of secure sketches and fuzzy extractors in three metrics: Hamming distance, set difference, and edit distance. Unless stated otherwise, all the constructions are new.

Hamming distance (i.e., the number of symbol positions that differ between w and w') is perhaps the most natural metric to consider. We observe that the "fuzzy-commitment" construction of Juels and Wattenberg [39] based on error-correcting codes can be viewed as a (nearly optimal) secure sketch. We then apply our general result to convert it into a nearly optimal fuzzy extractor. While our results on the Hamming distance essentially use previously known constructions, they serve as an important stepping stone for the rest of the work.

The set difference metric (i.e., the size of the symmetric difference of two input sets w and w') is appropriate whenever the noisy input is represented as a subset of features from a universe of possible features.[3] We demonstrate the existence of optimal (with respect to entropy loss) secure sketches and fuzzy extractors for this metric. However, this result is mainly of theoretical interest, because (1) it relies on optimal constant-weight codes, which we do not know how to construct, and (2) it produces sketches of length proportional to the universe size. We then turn our attention to more efficient constructions for this metric in order to handle exponentially large universes. We provide two such constructions.

First, we observe that the "fuzzy vault" construction of Juels and Sudan [38] can be viewed as a secure sketch in this metric (and then converted to a fuzzy extractor using our general result). We provide a new, simpler analysis for this construction, which bounds the entropy lost from w given s. This bound is quite high unless one makes the size of the output s very large. We then improve the Juels–Sudan construction to reduce the entropy loss and the length of s to near optimal. Our improvement in the running time and in the length of s is exponential for large universe sizes. However, this improved Juels–Sudan construction retains a drawback of the original: it is able to handle only sets of the same fixed size (in particular, $|w'|$ must equal $|w|$).

[3] A perhaps unexpected application of the set difference metric was explored in [38]: A user would like to encrypt a file (e.g., her phone number) using a small subset of values from a large universe (e.g., her favorite movies) in such a way that those and only those with a similar subset (e.g., similar taste in movies) can decrypt it.

Second, we provide an entirely different construction, called PinSketch, that maintains the exponential improvements in sketch size and running time and also handles variable set size. To obtain it, we note that in the case of a small universe, a set can be simply encoded as its characteristic vector (1 if an element is in the set, 0 if it is not), and set difference becomes Hamming distance. Even though the length of such a vector becomes unmanageable as the universe size grows, we demonstrate that this approach can be made to work quite efficiently even for exponentially large universes (in particular, because it is not necessary to ever actually write down the vector). This involves a result that may be of independent interest: we show that BCH codes can be decoded in time polynomial in the *weight* of the received corrupted word (i.e., in *sublinear* time if the weight is small).

Finally, edit distance (i.e., the number of insertions and deletions needed to convert one string into the other) comes up, for example, when the password is entered as a string, due to typing errors or mistakes made in handwriting recognition. We discuss two approaches for secure sketches and fuzzy extractors for this metric. First, we observe that a recent low-distortion embedding of Ostrovsky and Rabani [57] immediately gives a construction for edit distance. The construction performs well when the number of errors to be corrected is very small (say, n^α for $\alpha < 1$) but cannot tolerate a large number of errors. Second, we give a biometric embedding (which is less demanding than a low-distortion embedding but suffices for obtaining fuzzy extractors) from the edit distance metric into the set difference metric. Composing it with a fuzzy extractor for set difference gives a different construction for edit distance, which does better when t is large; it can handle as many as $O(n/\log^2 n)$ errors with meaningful entropy loss.

Most of the above constructions are quite practical; some implementations are available [36].

Extending results for probabilistic notions of correctness. The definitions and constructions just described use a very strong error model: we require that secure sketches and fuzzy extractors accept *every* secret w' which is sufficiently close to the original secret w, with probability 1. Such a stringent model is useful, as it makes no assumptions on the stochastic and computational properties of the error process. However, slightly relaxing the error conditions allows constructions which tolerate a (provably) much larger number of errors, at the price of restricting the settings in which the constructions can be applied. In section 8, we extend the definitions and constructions of earlier sections to several relaxed error models.

It is well known that in the standard setting of error-correction for a binary communication channel, one can tolerate many more errors when the errors are random and independent than when the errors are determined adversarially. In contrast, we present fuzzy extractors that meet Shannon's bounds for correcting random errors and, moreover, can correct the same number of errors even when errors are adversarial. In our setting, therefore, under a proper relaxation of the correctness condition, adversarial errors are no stronger than random ones. The constructions are quite simple and draw on existing techniques from the coding literature [4, 22, 33, 43, 48].

Relation to previous work. Since our work combines elements of error-correction, randomness extraction, and password authentication, there has been a lot of related work.

The need to deal with nonuniform and low-entropy passwords has long been realized in the security community, and many approaches have been proposed. For example, Kelsey et al. [42] suggested using $f(w, r)$ in place of w for the password authentication scenario, where r is a public random "salt," to make a brute-force at-

tacker's life harder. While practically useful, this approach does not add any entropy to the password and does not formally address the needed properties of f. Another approach, more closely related to ours, is to add biometric features to the password. For example, Ellison et al. [28] proposed asking the user a series of n personalized questions and using these answers to encrypt the "actual" truly random secret R. A similar approach using the user's keyboard dynamics (and, subsequently, voice [52, 53]) was proposed by Monrose, Reiter, and Wetzel [54]. These approaches require the design of a secure "fuzzy encryption." The above works proposed heuristic designs (using various forms of Shamir's secret sharing) but gave no formal analysis. Additionally, error tolerance was addressed only by brute-force search.

A formal approach to error tolerance in biometrics was taken by Juels and Wattenberg [39] (for less formal solutions, see [20, 54, 28]), who provided a simple way to tolerate errors in *uniformly distributed* passwords. Frykholm and Juels [31] extended this solution and provided entropy analysis to which ours is similar. Similar approaches have been explored earlier in seemingly unrelated literature on cryptographic information reconciliation, often in the context of quantum cryptography (where Alice and Bob wish to derive a secret key from secrets that have small Hamming distance), particularly [4, 6]. Our construction for the Hamming distance is essentially the same as a component of the quantum oblivious transfer protocol of [6].

Juels and Sudan [38] provided the first construction for a metric other than Hamming: they constructed a "fuzzy vault" scheme for the set difference metric. The main difference is that [38] lacks a cryptographically strong definition of the object constructed. In particular, their construction leaks a significant amount of information about their analogue of R, even though it leaves the adversary with provably "many valid choices" for R. In retrospect, their informal notion is closely related to our secure sketches. Our constructions in section 6 improve exponentially over the construction of [38] for storage and computation costs, in the setting when the set elements come from a large universe.

Linnartz and Tuyls [45] defined and constructed a primitive very similar to a fuzzy extractor (that line of work was continued in [73].) The definition of [45] focuses on the continuous space $\mathbb{R}^n$ and assumes a particular input distribution (typically a known, multivariate Gaussian). Thus, our definition of a fuzzy extractor can be viewed as a generalization of the notion of a "shielding function" from [45]. However, our constructions focus on discrete metric spaces.

Other approaches have also been taken for guaranteeing the privacy of noisy data. Csirmaz and Katona [19] considered quantization for correcting errors in "physical random functions." (This corresponds roughly to secure sketches with no public storage.) Barral, Coron, and Naccache [3] proposed a system for offline, private comparison of fingerprints. Although seemingly similar, the problem they study is complementary to ours, and the two solutions can be combined to yield systems which enjoy the benefits of both.

Work on privacy amplification, e.g., [4, 5], as well as work on derandomization and hardness amplification, e.g., [37, 56], also addressed the need to extract uniform randomness from a random variable about which some information has been leaked. A major focus of follow-up research has been the development of (ordinary, not fuzzy) extractors with short seeds (see [63] for a survey). We use extractors in this work (though for our purposes, universal hashing is sufficient). Conversely, our work has been applied recently to privacy amplification: Ding [21] used fuzzy extractors for noise tolerance in Maurer's bounded storage model [47].

Independently of our work, similar techniques appeared in the literature on non-

cryptographic information reconciliation [51, 15] (where the goal is communication efficiency rather than secrecy). The relationship between secure sketches and efficient information reconciliation is explored further in section 9, which discusses, in particular, how our secure sketches for set differences provide more efficient solutions to the set and string reconciliation problems.

Follow-up work. Since the original presentation of this paper [25], several follow-up works have appeared (e.g., [8, 9, 27, 24, 67, 14, 44, 13]). We refer the reader to a recent survey about fuzzy extractors [26] for more information.

2. Preliminaries. Unless explicitly stated otherwise, all logarithms below are base 2. The *Hamming weight* (or just *weight*) of a string is the number of nonzero characters in it. We use U_ℓ to denote the uniform distribution on ℓ-bit binary strings. If an algorithm (or a function) f is randomized, we use the semicolon when we wish to make the randomness explicit; i.e., we denote by $f(x;r)$ the result of computing f on input x with randomness r. If X is a probability distribution, then $f(X)$ is the distribution induced on the image of f by applying the (possibly probabilistic) function f. If X is a random variable, we will (slightly) abuse notation and also denote by X the probability distribution on the range of the variable.

2.1. Metric spaces. A metric space is a set $\mathcal{M}$ with a distance function $\mathsf{dis} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^+ = [0, \infty)$. For the purposes of this work, $\mathcal{M}$ will always be a finite set, and the distance function will take on only integer values (with $\mathsf{dis}(x, y) = 0$ if and only if $x = y$) and will obey symmetry $\mathsf{dis}(x, y) = \mathsf{dis}(y, x)$ and the triangle inequality $\mathsf{dis}(x, z) \le \mathsf{dis}(x, y) + \mathsf{dis}(y, z)$ (we adopt these requirements for simplicity of exposition, even though the definitions and most of the results below can be generalized to remove these restrictions).

We will concentrate on the following metrics.

1. *Hamming metric.* Here $\mathcal{M} = \mathcal{F}^n$ for some alphabet $\mathcal{F}$, and $\mathsf{dis}(w, w')$ is the number of positions in which the strings w and w' differ.

2. *Set difference metric.* Here $\mathcal{M}$ consists of all subsets of a universe $\mathcal{U}$. For two sets w, w', their symmetric difference $w \triangle w' \stackrel{\text{def}}{=} \{x \in w \cup w' \mid x \notin w \cap w'\}$. The distance between two sets w, w' is $|w \triangle w'|$.[4] We will sometimes restrict $\mathcal{M}$ to contain only s-element subsets for some s.

3. *Edit metric.* Here $\mathcal{M} = \mathcal{F}^*$, and the distance between w and w' is defined to be the smallest number of character insertions and deletions needed to transform w into w'.[5] (This is different from the Hamming metric because insertions and deletions shift the characters that are to the right of the insertion/deletion point.)

As already mentioned, all three metrics seem natural for biometric data.

2.2. Codes and syndromes. Since we want to achieve error-tolerance in various metric spaces, we will use *error-correcting codes* for a particular metric. A code C is a subset $\{w_0, \ldots, w_{K-1}\}$ of K elements of $\mathcal{M}$. The map from i to w_i, which we will also sometimes denote by C, is called *encoding*. The *minimum distance* of C is the smallest $d > 0$ such that for all $i \neq j$ we have $\mathsf{dis}(w_i, w_j) \ge d$. In our case of integer metrics, this means that one can detect up to $(d - 1)$ "errors" in an element

[4] In the preliminary version of this work [25], we worked with this metric scaled by $\frac{1}{2}$; that is, the distance was $\frac{1}{2}|w \triangle w'|$. Not scaling makes more sense, particularly when w and w' are of potentially different sizes since $|w \triangle w'|$ may be odd. It also agrees with the Hamming distance of characteristic vectors; see section 6.

[5] Again, in [25], we worked with this metric scaled by $\frac{1}{2}$. Likewise, this makes little sense when strings can be of different lengths, and we avoid it here.

of $\mathcal{M}$. The *error-correcting distance* of C is the largest number $t > 0$ such that for every $w \in \mathcal{M}$ there exists at most one codeword c in the ball of radius t around w: $\mathsf{dis}(w, c) \leq t$ for at most one $c \in C$. This means that one can correct up to t errors in an element w of $\mathcal{M}$; we will use the term *decoding* for the map that finds, given w, the $c \in C$ such that $\mathsf{dis}(w, c) \leq t$ (note that for some w, such c may not exist, but if it exists, it will be unique; note also that decoding is not the inverse of encoding in our terminology). For integer metrics by triangle inequality we are guaranteed that $t \geq \lfloor (d-1)/2 \rfloor$. Since error correction will be more important than error detection in our applications, we denote the corresponding codes as $(\mathcal{M}, K, t)$-codes. For efficiency purposes, we will often want encoding and decoding to be polynomial-time.

For the Hamming metric over $\mathcal{F}^n$, we will sometimes call $k = \log_{|\mathcal{F}|} K$ the *dimension* of the code and denote the code itself as an $[n, k, d = 2t + 1]_{\mathcal{F}}$-code, following the standard notation in the literature. We will denote by $A_{|\mathcal{F}|}(n, d)$ the maximum K possible in such a code (omitting the subscript when $|\mathcal{F}| = 2$), and by $A(n, d, s)$ the maximum K for such a code over $\{0,1\}^n$ with the additional restriction that all codewords have exactly s ones.

If the code is linear (i.e., $\mathcal{F}$ is a field, $\mathcal{F}^n$ is a vector space over $\mathcal{F}$, and C is a linear subspace), then one can fix a parity-check matrix H as any matrix whose rows generate the orthogonal space $C^{\perp}$. Then for any $v \in \mathcal{F}^n$, the syndrome $\mathsf{syn}(v) \stackrel{\text{def}}{=} Hv$. The syndrome of a vector is its projection onto subspace that is orthogonal to the code and can thus be intuitively viewed as the vector modulo the code. Note that $v \in C \Leftrightarrow \mathsf{syn}(v) = 0$. Note also that H is an $(n-k) \times n$ matrix and that $\mathsf{syn}(v)$ is $n - k$ bits long.

The syndrome captures all the information necessary for decoding. That is, suppose a codeword c is sent through a channel and the word $w = c + e$ is received. First, the syndrome of w is the syndrome of e: $\mathsf{syn}(w) = \mathsf{syn}(c) + \mathsf{syn}(e) = 0 + \mathsf{syn}(e) = \mathsf{syn}(e)$. Moreover, for any value u, there is at most one word e of weight less than $d/2$ such that $\mathsf{syn}(e) = u$ (because the existence of a pair of distinct words e_1, e_2 would mean that $e_1 - e_2$ is a codeword of weight less than d, but since 0^n is also a codeword and the minimum distance of the code is d, this is impossible). Thus, knowing syndrome $\mathsf{syn}(w)$ is enough to determine the error pattern e if not too many errors occurred.

2.3. Min-entropy, statistical distance, universal hashing, and strong extractors. When discussing security, one is often interested in the probability that the adversary predicts a random value (e.g., guesses a secret key). The adversary's best strategy, of course, is to guess the most likely value. Thus, the *predictability* of a random variable A is $\max_a \Pr[A = a]$, and, correspondingly, the *min-entropy* $\mathbf{H}_\infty(A)$ is $-\log(\max_a \Pr[A = a])$ (min-entropy can thus be viewed as the "worst-case" entropy [16]; see also section 2.4).

The min-entropy of a distribution tells us how many nearly uniform random bits can be extracted from it. The notion of "nearly" is defined as follows. The *statistical distance between* two probability distributions A and B is $\mathbf{SD}(A, B) = \frac{1}{2} \sum_v |\Pr(A = v) - \Pr(B = v)|$.

Recall the definition of *strong randomness extractors* [56].

DEFINITION 1. *Let* $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^\ell$ *be a polynomial time probabilistic function which uses* r *bits of randomness. We say that* Ext *is an efficient* (n, m, ℓ, ϵ)-strong extractor *if for all min-entropy* m *distributions* W *on* $\{0,1\}^n$

$$\mathbf{SD}\left((\mathsf{Ext}(W; X), X), (U_\ell, X)\right) \leq \epsilon\,,$$

where X *is uniform on* $\{0,1\}^r$.

Strong extractors can extract at most $\ell = m - 2\log\left(\frac{1}{\epsilon}\right) + O(1)$ nearly random bits [59]. Many constructions match this bound (see Shaltiel's survey [63] for references). Extractor constructions are often complex since they seek to minimize the length of the seed X. For our purposes, the length of X will be less important, so universal hash functions [12, 75] (defined in the lemma below) will already give us the optimal $\ell = m - 2\log\left(\frac{1}{\epsilon}\right) + 2$, as given by the *leftover hash lemma* below (see [37, Lemma 4.8] as well as references therein for earlier versions).

LEMMA 2.1 (universal hash functions and the leftover hash/privacy-amplification lemma). *Assume a family of functions* $\{H_x : \{0,1\}^n \to \{0,1\}^\ell\}_{x \in X}$ *is* universal*: for all* $a \neq b \in \{0,1\}^n$, $\Pr_{x \in X}[H_x(a) = H_x(b)] = 2^{-\ell}$. *Then, for any random variable* W,[6]

$$\mathbf{SD}\left((H_X(W), X)\ ,\ (U_\ell, X)\right) \leq \frac{1}{2}\sqrt{2^{-\mathbf{H}_\infty(W)} 2^\ell}. \tag{2.1}$$

In particular, universal hash functions are (n, m, ℓ, ϵ)*-strong extractors whenever* $\ell \leq m - 2\log\left(\frac{1}{\epsilon}\right) + 2$.

2.4. Average min-entropy. Recall that *predictability* of a random variable A is $\max_a \Pr[A = a]$, and its *min-entropy* $\mathbf{H}_\infty(A)$ is $-\log(\max_a \Pr[A = a])$. Consider now a pair of (possibly correlated) random variables A, B. If the adversary finds out the value b of B, then predictability of A becomes $\max_a \Pr[A = a \mid B = b]$. On average, the adversary's chance of success in predicting A is then $\mathbb{E}_{b \leftarrow B}\left[\max_a \Pr[A = a \mid B = b]\right]$. Note that we are taking the *average* over B (which is not under adversarial control) but the *worst case* over A (because prediction of A is adversarial once b is known). Again, it is convenient to talk about security in log-scale, which is why we define the *average min-entropy* of A given B as simply the logarithm of the above:

$$\tilde{\mathbf{H}}_\infty(A \mid B) \stackrel{\text{def}}{=} -\log\left(\mathop{\mathbb{E}}_{b \leftarrow B}\left[\max_a \Pr[A = a \mid B = b]\right]\right) = -\log\left(\mathop{\mathbb{E}}_{b \leftarrow B}\left[2^{-\mathbf{H}_\infty(A|B=b)}\right]\right).$$

Because other notions of entropy have been studied in cryptographic literature, a few words are in order to explain why this definition is useful. Note the importance of taking the logarithm *after* taking the average (in contrast, for instance, to conditional Shannon entropy). One may think it more natural to define average min-entropy as $\mathbb{E}_{b \leftarrow B}\left[\mathbf{H}_\infty(A \mid B = b)\right]$, thus reversing the order of log and $\mathbb{E}$. However, this notion is unlikely to be useful in a security application. For a simple example, consider the case when A and B are 1000-bit strings distributed as follows: $B = U_{1000}$ and A is equal to the value b of B if the first bit of b is 0, and U_{1000} (independent of B) otherwise. Then for half of the values of b, $\mathbf{H}_\infty(A \mid B = b) = 0$, while for the other half, $\mathbf{H}_\infty(A \mid B = b) = 1000$, so $\mathbb{E}_{b \leftarrow B}\left[\mathbf{H}_\infty(A \mid B = b)\right] = 500$. However, it would be obviously incorrect to say that A has 500 bits of security. In fact, an adversary who knows the value b of B has a slightly greater than 50% chance of predicting the value of A by outputting b. Our definition correctly captures this 50% chance of prediction, because $\tilde{\mathbf{H}}_\infty(A \mid B)$ is slightly less than 1. In fact, our definition of average min-entropy is simply the logarithm of predictability.

The following useful properties of average min-entropy are proven in Appendix A. We also refer the reader to Appendix B for a generalization of average min-entropy and a discussion of the relationship between this notion and other notions of entropy.

[6] In [37], this inequality is formulated in terms of Rényi entropy of order two of W; the change to $\mathbf{H}_\infty(C)$ is allowed because the latter is no greater than the former.

LEMMA 2.2. *Let A, B, C be random variables. Then the following hold.*

(a) *For any $\delta > 0$, the conditional entropy $\mathbf{H}_\infty(A|B = b)$ is at least $\tilde{\mathbf{H}}_\infty(A|B) - \log(1/\delta)$ with probability at least $1 - \delta$ over the choice of b.*

(b) *If B has at most 2^λ possible values, then $\tilde{\mathbf{H}}_\infty(A \mid (B, C)) \geq \tilde{\mathbf{H}}_\infty((A, B) \mid C) - \lambda \geq \tilde{\mathbf{H}}_\infty(A \mid C) - \lambda$. In particular, $\tilde{\mathbf{H}}_\infty(A \mid B) \geq \mathbf{H}_\infty((A, B)) - \lambda \geq \mathbf{H}_\infty(A) - \lambda$.*

2.5. Average-case extractors. Recall from Definition 1 that a strong extractor allows one to extract almost all the min-entropy from some nonuniform random variable W. In many situations, W represents the adversary's uncertainty about some secret w conditioned on some side information i. Since this side information i is often probabilistic, we shall find the following generalization of a strong extractor useful (see Lemma 4.1).

DEFINITION 2. *Let $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^\ell$ be a polynomial time probabilistic function which uses r bits of randomness. We say that Ext is an efficient* average-case *(n, m, ℓ, ϵ)-strong extractor if, for all pairs of random variables (W, I) such that W is an n-bit string satisfying $\tilde{\mathbf{H}}_\infty(W \mid I) \geq m$, we have $\mathbf{SD}\left((\mathsf{Ext}(W; X), X, I), (U_\ell, X, I)\right) \leq \epsilon$, where X is uniform on $\{0,1\}^r$.*

To distinguish the strong extractors of Definition 1 from average-case strong extractors, we will sometimes call the former *worst-case* strong extractors. The two notions are closely related, as can be seen from the following simple application of Lemma 2.2(a).

LEMMA 2.3. *For any $\delta > 0$, if Ext is a (worst-case) $(n, m - \log\left(\frac{1}{\delta}\right), \ell, \epsilon)$-strong extractor, then Ext is also an average-case $(n, m, \ell, \epsilon + \delta)$-strong extractor.*

Proof. Assume (W, I) are such that $\tilde{\mathbf{H}}_\infty(W \mid I) \geq m$. Let $W_i = (W \mid I = i)$ and let us call the value i "bad" if $\mathbf{H}_\infty(W_i) < m - \log\left(\frac{1}{\delta}\right)$. Otherwise, we say that i is "good." By Lemma 2.2(a), $\Pr(i \text{ is bad}) \leq \delta$. Also, for any good i, we have that Ext extracts ℓ bits that are ϵ-close to uniform from W_i. Thus, by conditioning on the "goodness" of I, we get

$$\begin{aligned}
\mathbf{SD}\left((\mathsf{Ext}(W; X), X, I), (U_\ell, X, I)\right) &= \sum_i \Pr(i) \cdot \mathbf{SD}\left((\mathsf{Ext}(W_i; X), X), (U_\ell, X)\right) \\
&\leq \Pr(i \text{ is bad}) \cdot 1 \\
&\quad + \sum_{\text{good } i} \Pr(i) \cdot \mathbf{SD}\left((\mathsf{Ext}(W_i; X), X), (U_\ell, X)\right) \\
&\leq \delta + \epsilon. \qquad \square
\end{aligned}$$

However, for many strong extractors we do not have to suffer this additional dependence on δ, because the strong extractor may already be average-case. In particular, this holds for extractors obtained via universal hashing.

LEMMA 2.4 (generalized leftover hash lemma). *Assume $\{H_x : \{0,1\}^n \to \{0,1\}^\ell\}_{x \in X}$ is a family of universal hash functions. Then, for any random variables W and I,*

$$\mathbf{SD}\left((H_X(W), X, I)\ ,\ (U_\ell, X, I)\right) \leq \frac{1}{2}\sqrt{2^{-\tilde{\mathbf{H}}_\infty(W|I)} 2^\ell}. \tag{2.2}$$

In particular, universal hash functions are average-case *(n, m, ℓ, ϵ)-strong extractors whenever $\ell \leq m - 2\log\left(\frac{1}{\epsilon}\right) + 2$.*

Proof. Let $W_i = (W \mid I = i)$. Then

$$\begin{aligned}\mathbf{SD}\left((H_X(W), X, I)\ ,\ (U_\ell, X, I)\right) &= \mathop{\mathbb{E}}_i\left[\mathbf{SD}\left((H_X(W_i), X)\ ,\ (U_\ell, X)\right)\right] \\ &\le \frac{1}{2}\mathop{\mathbb{E}}_i\left[\sqrt{2^{-\mathbf{H}_\infty(W_i)}2^\ell}\right] \\ &\le \frac{1}{2}\sqrt{\mathop{\mathbb{E}}_i\left[2^{-\mathbf{H}_\infty(W_i)}2^\ell\right]} \\ &= \frac{1}{2}\sqrt{2^{-\tilde{\mathbf{H}}_\infty(W|I)}2^\ell}\,.\end{aligned}$$

In the above derivation, the first inequality follows from the standard leftover hash lemma (Lemma 2.1), and the second inequality follows from Jensen's inequality (namely, $\mathbb{E}[\sqrt{Z}] \le \sqrt{\mathbb{E}\,[Z]}$). □

3. New definitions.

3.1. Secure sketches. Let $\mathcal{M}$ be a metric space with distance function dis.

DEFINITION 3. *An* $(\mathcal{M}, m, \tilde{m}, t)$-secure sketch *is a pair of randomized procedures, "sketch" (*SS*) and "recover" (*Rec*), with the following properties:*

1. *The sketching procedure* SS *on input* $w \in \mathcal{M}$ *returns a bit string* $s \in \{0,1\}^*$.
2. *The recovery procedure* Rec *takes an element* $w' \in M$ *and a bit string* $s \in \{0,1\}^*$. *The* correctness *property of secure sketches guarantees that if* $\mathsf{dis}(w, w') \le t$, *then* $\mathsf{Rec}(w', \mathsf{SS}(w)) = w$. *If* $\mathsf{dis}(w, w') > t$, *then no guarantee is provided about the output of* Rec.
3. *The* security *property guarantees that for any distribution* W *over* $\mathcal{M}$ *with min-entropy* m, *the value of* W *can be recovered by the adversary who observes* s *with probability no greater than* $2^{-\tilde{m}}$. *That is,* $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{SS}(W)) \ge \tilde{m}$.

A secure sketch is efficient *if* SS *and* Rec *run in expected polynomial time.*

Average-case secure sketches. In many situations, it may well be that the adversary's information i about the password w is probabilistic, so that sometimes i reveals a lot about w, but most of the time w stays hard to predict even given i. In this case, the previous definition of a secure sketch is hard to apply: it provides no guarantee if $\mathbf{H}_\infty(W|i)$ is not fixed to at least m for some bad (but infrequent) values of i. A more robust definition would provide the same guarantee for all pairs of variables (W, I) such that predicting the value of W given the value of I is hard. We therefore define an *average-case* secure sketch as follows.

DEFINITION 4. *An* average-case $(\mathcal{M}, m, \tilde{m}, t)$-secure sketch *is a secure sketch (as defined in Definition* 3*) whose security property is strengthened as follows: for any random variables* W *over* $\mathcal{M}$ *and* I *over* $\{0,1\}^*$ *such that* $\tilde{\mathbf{H}}_\infty(W \mid I) \ge m$, *we have* $\tilde{\mathbf{H}}_\infty(W \mid (\mathsf{SS}(W), I)) \ge \tilde{m}$. *Note that an average-case secure sketch is also a secure sketch (take* I *to be empty).*

This definition has the advantage that it composes naturally, as Lemma 4.7 shows. All of our constructions will in fact be average-case secure sketches. However, we will often omit the term "average-case" for simplicity of exposition.

Entropy loss. The quantity $\tilde{m}$ is called the *residual (min-)entropy* of the secure sketch, and the quantity $\lambda = m - \tilde{m}$ is called the *entropy loss* of a secure sketch. In analyzing the security of our secure sketch constructions below, we will typically bound the entropy loss regardless of m, thus obtaining families of secure sketches that work for all m (in general, [62] shows that the entropy loss of a secure sketch is upperbounded by its entropy loss on the uniform distribution of inputs). Specifically, for a given construction of SS, Rec, and a given value t, we will get a value λ for the

entropy loss, such that, for *any* m, $(\mathsf{SS}, \mathsf{Rec})$ is an $(\mathcal{M}, m, m-\lambda, t)$-secure sketch. In fact, the most common way to obtain such secure sketches would be to bound the entropy loss by the length of the secure sketch $\mathsf{SS}(w)$, as given in the following simple lemma.

LEMMA 3.1. *Assume some algorithms* SS *and* Rec *satisfy the correctness property of a secure sketch for some value of* t *and that the output range of* SS *has size at most* 2^λ *(this holds, in particular, if the length of the sketch is bounded by* λ*). Then, for any min-entropy threshold* m, $(\mathsf{SS}, \mathsf{Rec})$ *form an average-case* $(\mathcal{M}, m, m-\lambda, t)$*-secure sketch for* $\mathcal{M}$*. In particular, for any* m*, the entropy loss of this construction is at most* λ*.*

Proof. The result follows immediately from Lemma 2.2(b), since $\mathsf{SS}(W)$ has at most 2^λ values: for any (W, I), $\tilde{\mathbf{H}}_\infty(W \mid (\mathsf{SS}(W), I)) \geq \tilde{\mathbf{H}}_\infty(W \mid I) - \lambda$. □

The above observation formalizes the intuition that a good secure sketch should be as short as possible. In particular, a short secure sketch will likely result in a better entropy loss. More discussion about this relation can be found in section 9.

3.2. Fuzzy extractors.

DEFINITION 5. *An* $(\mathcal{M}, m, \ell, t, \epsilon)$-fuzzy extractor *is a pair of randomized procedures, "generate" (*Gen*) and "reproduce" (*Rep*), with the following properties:*

1. *The generation procedure* Gen *on input* $w \in \mathcal{M}$ *outputs an extracted string* $R \in \{0,1\}^\ell$ *and a helper string* $P \in \{0,1\}^*$.
2. *The reproduction procedure* Rep *takes an element* $w' \in M$ *and a bit string* $P \in \{0,1\}^*$ *as inputs. The* correctness *property of fuzzy extractors guarantees that if* $\mathsf{dis}(w, w') \leq t$ *and* R, P *were generated by* $(R, P) \leftarrow \mathsf{Gen}(w)$*, then* $\mathsf{Rep}(w', P) = R$*. If* $\mathsf{dis}(w, w') > t$*, then no guarantee is provided about the output of* Rep.
3. *The* security *property guarantees that for any distribution* W *on* $\mathcal{M}$ *of min-entropy* m*, the string* R *is nearly uniform even for those who observe* P*: if* $(R, P) \leftarrow \mathsf{Gen}(W)$*, then* $\mathbf{SD}((R, P), (U_\ell, P)) \leq \epsilon$.

A fuzzy extractor is efficient *if* Gen *and* Rep *run in expected polynomial time.*

In other words, fuzzy extractors allow one to extract some randomness R from w and then successfully reproduce R from any string w' that is close to w. The reproduction uses the helper string P produced during the initial extraction; yet P need not remain secret, because R looks truly random even given P. To justify our terminology, notice that strong extractors (as defined in section 2) can indeed be seen as "nonfuzzy" analogues of fuzzy extractors, corresponding to $t = 0$, $P = X$, and $\mathcal{M} = \{0,1\}^n$.

We reiterate that the nearly uniform random bits output by a fuzzy extractor can be used in any cryptographic context that requires uniform random bits (e.g., for secret keys). The slight nonuniformity of the bits may decrease security, but by no more than their distance ϵ from uniform. By choosing ϵ negligibly small (e.g., 2^{-80} should be enough in practice), one can make the decrease in security irrelevant.

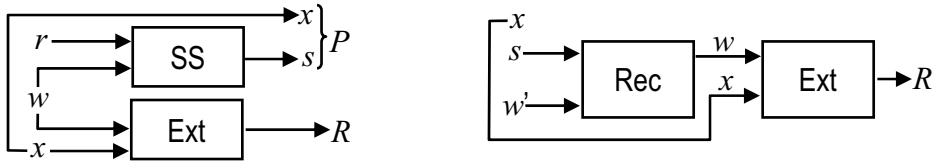
Similarly to secure sketches, the quantity $m - \ell$ is called the *entropy loss* of a fuzzy extractor. Also similarly, a more robust definition is that of an *average-case* fuzzy extractor, which requires that if $\tilde{\mathbf{H}}_\infty(W \mid I) \geq m$, then $\mathbf{SD}((R, P, I), (U_\ell, P, I)) \leq \epsilon$ for any auxiliary random variable I.

4. Metric-independent results.

In this section we demonstrate some general results that do not depend on specific metric spaces. They will be helpful in obtaining specific results for particular metric spaces below. In addition to the results in this section, some generic combinatorial lower bounds on secure sketches and fuzzy

extractors are contained in Appendix C. We will later use these bounds to show the near-optimality of some of our constructions for the case of uniform inputs.[7]

4.1. Construction of fuzzy extractors from secure sketches. Not surprisingly, secure sketches are quite useful in constructing fuzzy extractors. Specifically, we construct fuzzy extractors from secure sketches and strong extractors as follows: apply SS to w to obtain s, and a strong extractor Ext with randomness x to w to obtain R. Store (s, x) as the helper string P. To reproduce R from w' and $P = (s, x)$, first use $\mathsf{Rec}(w', s)$ to recover w and then $\mathsf{Ext}(w, x)$ to get R.



A few details need to be filled in. First, in order to apply Ext to w, we will assume that one can represent elements of $\mathcal{M}$ using n bits. Second, since after leaking the secure sketch value s, the password w has only *conditional* min-entropy, technically we need to use the *average-case* strong extractor, as defined in Definition 2. The formal statement is given below.

LEMMA 4.1 (fuzzy extractors from sketches). *Assume* $(\mathsf{SS}, \mathsf{Rec})$ *is an* $(\mathcal{M}, m, \tilde{m}, t)$*-secure sketch, and let* Ext *be an* average-case $(n, \tilde{m}, \ell, \epsilon)$*-strong extractor. Then the following* $(\mathsf{Gen}, \mathsf{Rep})$ *is an* $(\mathcal{M}, m, \ell, t, \epsilon)$*-fuzzy extractor:*

- $\mathsf{Gen}(w; r, x)$*: set* $P = (\mathsf{SS}(w; r), x)$ *and* $R = \mathsf{Ext}(w; x)$*, and output* (R, P)*.*
- $\mathsf{Rep}(w', (s, x))$*: recover* $w = \mathsf{Rec}(w', s)$*, and output* $R = \mathsf{Ext}(w; x)$*.*

Proof. From the definition of a secure sketch (Definition 3), we know that $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{SS}(W)) \geq \tilde{m}$. And since Ext is an average-case $(n, \tilde{m}, \ell, \epsilon)$-strong extractor,

$$\mathbf{SD}\left((\mathsf{Ext}(W; X), \mathsf{SS}(W), X), (U_\ell, \mathsf{SS}(W), X)\right) = \mathbf{SD}\left((R, P), (U_\ell, P)\right) \leq \epsilon\,. \qquad \square$$

On the other hand, if one would like to use a worst-case strong extractor, we can apply Lemma 2.3 to get the following corollary.

COROLLARY 4.2. *If* $(\mathsf{SS}, \mathsf{Rec})$ *is an* $(\mathcal{M}, m, \tilde{m}, t)$*-secure sketch and* Ext *is an* $(n, \tilde{m} - \log\left(\frac{1}{\delta}\right), \ell, \epsilon)$*-strong extractor, then the above construction* $(\mathsf{Gen}, \mathsf{Rep})$ *is a* $(\mathcal{M}, m, \ell, t, \epsilon + \delta)$*-fuzzy extractor.*

Both Lemma 4.1 and Corollary 4.2 hold (with the same proofs) for building *average-case* fuzzy extractors from *average-case* secure sketches.

While the above statements work for general extractors, for our purposes we can simply use universal hashing, since it is an average-case strong extractor that achieves the optimal [59] entropy loss of $2\log\left(\frac{1}{\epsilon}\right)$. In particular, the lemma below is an immediate corollary of Lemmas 2.4 and 4.1.

LEMMA 4.3. *If* $(\mathsf{SS}, \mathsf{Rec})$ *is an* $(\mathcal{M}, m, \tilde{m}, t)$*-secure sketch and* Ext *is an* $(n, \tilde{m}, \ell, \epsilon)$*-strong extractor given by universal hashing (in particular, any* $\ell \leq \tilde{m} - 2\log\left(\frac{1}{\epsilon}\right) + 2$ *can be achieved), then the above construction* $(\mathsf{Gen}, \mathsf{Rep})$ *is an* $(\mathcal{M}, m, \ell, t, \epsilon)$*-fuzzy extractor. In particular, one can extract up to* $(\tilde{m} - 2\log\left(\frac{1}{\epsilon}\right) + 2)$ *nearly uniform bits from a secure sketch with residual min-entropy* $\tilde{m}$*.*

[7] Although we believe our constructions to be near optimal for nonuniform inputs as well, and our combinatorial bounds in Appendix C are also meaningful for such inputs, at this time we can use these bounds effectively only for uniform inputs.

Again, if the above secure sketch is average-case secure, then so is the resulting fuzzy extractor. In fact, combining the above result with Lemma 3.1, we get the following general construction of average-case fuzzy extractors.

LEMMA 4.4. *Assume some algorithms* SS *and* Rec *satisfy the correctness property of a secure sketch for some value of* t*, and that the output range of* SS *has size at most* 2^λ *(this holds, in particular, if the length of the sketch is bounded by* λ*). Then, for any min-entropy threshold* m*, there exists an average-case* $(\mathcal{M}, m, m-\lambda-2\log\left(\frac{1}{\epsilon}\right)+2, t, \epsilon)$*-fuzzy extractor for* $\mathcal{M}$*. In particular, for any* m*, the entropy loss of the fuzzy extractor is at most* $\lambda + 2\log\left(\frac{1}{\epsilon}\right) - 2$*.*

4.2. Secure sketches for transitive metric spaces. We give a general technique for building secure sketches in *transitive* metric spaces, which we now define. A permutation π on a metric space $\mathcal{M}$ is an *isometry* if it preserves distances, i.e., $\mathsf{dis}(a,b) = \mathsf{dis}(\pi(a), \pi(b))$. A family of permutations $\Pi = \{\pi_i\}_{i\in\mathcal{I}}$ acts *transitively* on $\mathcal{M}$ if, for any two elements $a, b \in \mathcal{M}$, there exists $\pi_i \in \Pi$ such that $\pi_i(a) = b$. Suppose we have a family Π of transitive isometries for $\mathcal{M}$ (we will call such $\mathcal{M}$ *transitive*). For example, in the Hamming space, the set of all shifts $\pi_x(w) = w \oplus x$ is such a family (see section 5 for more details on this example).

Construction 1 (secure sketch for transitive metric spaces). Let C be an $(\mathcal{M}, K, t)$-code. Then the general sketching scheme SS is the following: given an input $w \in \mathcal{M}$, pick uniformly at random a codeword $b \in C$, pick uniformly at random a permutation $\pi \in \Pi$ such that $\pi(w) = b$, and output $\mathsf{SS}(w) = \pi$ (it is crucial that each $\pi \in \Pi$ should have a canonical description that is independent of how π was chosen and, in particular, independent of b and w; the number of possible outputs of SS should thus be $|\Pi|$). The recovery procedure Rec to find w given w' and the sketch π is as follows: find the closest codeword b' to $\pi(w')$, and output $\pi^{-1}(b')$.

Let Γ be the number of elements $\pi \in \Pi$ such that $\min_{w,b} |\{\pi | \pi(w) = b\}| \geq \Gamma$. That is, for each w and b, there are at least Γ choices for π. Then we obtain the following lemma.

LEMMA 4.5. $(\mathsf{SS}, \mathsf{Rec})$ *is an average-case* $(\mathcal{M}, m, m - \log|\Pi| + \log\Gamma + \log K, t)$*-secure sketch. It is efficient if operations on the code, as well as* π *and* π^{-1}*, can be implemented efficiently.*

Proof. Correctness is clear: when $\mathsf{dis}(w, w') \leq t$, then $\mathsf{dis}(b, \pi(w')) \leq t$, so decoding $\pi(w')$ will result in $b' = b$, which in turn means that $\pi^{-1}(b') = w$. The intuitive argument for security is as follows: we add $\log K + \log\Gamma$ bits of entropy by choosing b and π, and we subtract $\log|\Pi|$ by publishing π. Since, given π, w and b determine each other, the total entropy loss is $\log|\Pi| - \log K - \log\Gamma$. More formally, $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{SS}(W), I) = \tilde{\mathbf{H}}_\infty((W, \mathsf{SS}(W)) \mid I) - \log|\Pi|$ by Lemma 2.2(b). Given a particular value of w, there are K equiprobable choices for b and, further, at least Γ equiprobable choices for π once b is picked, and hence any given permutation π is chosen with probability at most $1/(K\Gamma)$ (because different choices for b result in different choices for π). Therefore, for all i, w, and π, $\Pr[W = w \wedge \mathsf{SS}(w) = \pi \mid I = i] \leq \Pr[W = w \mid I = i]/(K\Gamma)$; hence $\tilde{\mathbf{H}}_\infty((W, \mathsf{SS}(W)) \mid I) \geq \tilde{\mathbf{H}}_\infty(W \mid I) + \log K + \log\Gamma$. □

Naturally, security loss will be smaller if the code C is denser.

We will discuss concrete instantiations of this approach in sections 5 and 6.1.

4.3. Changing metric spaces via biometric embeddings. We now introduce a general technique that allows one to build fuzzy extractors and secure sketches in some metric space $\mathcal{M}_1$ from fuzzy extractors and secure sketches in some other metric space $\mathcal{M}_2$. Below, we let $\mathsf{dis}(\cdot,\cdot)_i$ denote the distance function in $\mathcal{M}_i$. The

technique is to *embed* $\mathcal{M}_1$ into $\mathcal{M}_2$ so as to "preserve" relevant parameters for fuzzy extraction.

DEFINITION 6. *A function $f : \mathcal{M}_1 \to \mathcal{M}_2$ is called a (t_1, t_2, m_1, m_2)-biometric embedding if the following two conditions hold:*

- *For any $w_1, w_1' \in \mathcal{M}_1$ such that $\mathsf{dis}(w_1, w_1')_1 \le t_1$, we have $\mathsf{dis}(f(w_1), f(w_2))_2 \le t_2$.*
- *For any distribution W_1 on $\mathcal{M}_1$ of min-entropy at least m_1, $f(W_1)$ has min-entropy at least m_2.*

The following lemma is immediate (correctness of the resulting fuzzy extractor follows from the first condition, and security follows from the second).

LEMMA 4.6. *If f is a (t_1, t_2, m_1, m_2)-biometric embedding of $\mathcal{M}_1$ into $\mathcal{M}_2$ and $(\mathsf{Gen}(\cdot), \mathsf{Rep}(\cdot,\cdot))$ is an $(\mathcal{M}_2, m_2, \ell, t_2, \epsilon)$-fuzzy extractor, then $(\mathsf{Gen}(f(\cdot)), \mathsf{Rep}(f(\cdot), \cdot))$ is an $(\mathcal{M}_1, m_1, \ell, t_1, \epsilon)$-fuzzy extractor.*

It is easy to define *average-case* biometric embeddings (in which $\tilde{\mathbf{H}}_\infty(W_1 \mid I) \ge m_1 \Rightarrow \tilde{\mathbf{H}}_\infty(f(W_1) \mid I) \ge m_2$) which would result in an analogous lemma for average-case fuzzy extractors.

For a similar result to hold for secure sketches, we need biometric embeddings with an additional property.

DEFINITION 7. *A function $f : \mathcal{M}_1 \to \mathcal{M}_2$ is called a* (t_1, t_2, λ)-biometric embedding with recovery information g *if the following hold:*

- *For any $w_1, w_1' \in \mathcal{M}_1$ such that $\mathsf{dis}(w_1, w_1')_1 \le t_1$, we have $\mathsf{dis}(f(w_1), f(w_2))_2 \le t_2$.*
- *$g : M_1 \to \{0,1\}^*$ is a function with range size at most 2^λ, and $w_1 \in M_1$ is uniquely determined by $(f(w_1), g(w_1))$.*

With this definition, we get the following analogue of Lemma 4.6.

LEMMA 4.7. *Let f be a (t_1, t_2, λ)-biometric embedding with recovery information g. Let $(\mathsf{SS}, \mathsf{Rec})$ be an $(\mathcal{M}_2, m_1 - \lambda, \tilde{m}_2, t_2)$ average-case secure sketch. Let $\mathsf{SS}'(w) = (\mathsf{SS}(f(w)), g(w))$. Let $\mathsf{Rec}'(w', (s, r))$ be the function obtained by computing $\mathsf{Rec}(w', s)$ to get $f(w)$ and then inverting $(f(w), r)$ to get w. Then $(\mathsf{SS}', \mathsf{Rec}')$ is an $(\mathcal{M}_1, m_1, \tilde{m}_2, t_1)$ average-case secure sketch.*

Proof. The correctness of this construction follows immediately from the two properties given in Definition 7. As for security, using Lemma 2.2(b) and the fact that the range of g has size at most 2^λ, we get that $\tilde{\mathbf{H}}_\infty(W \mid g(W)) \ge m_1 - \lambda$ whenever $\mathbf{H}_\infty(W) \ge m_1$. Moreover, since W is uniquely recoverable from $f(W)$ and $g(W)$, it follows that $\tilde{\mathbf{H}}_\infty(f(W) \mid g(W)) \ge m_1 - \lambda$ as well whenever $\mathbf{H}_\infty(W) \ge m_1$. Using the fact that $(\mathsf{SS}, \mathsf{Rec})$ is an *average-case* $(\mathcal{M}_2, m_1 - \lambda, \tilde{m}_2, t_2)$-secure sketch, we get that $\tilde{\mathbf{H}}_\infty(f(W) \mid (\mathsf{SS}(W), g(W))) = \tilde{\mathbf{H}}_\infty(f(W) \mid \mathsf{SS}'(W)) \ge \tilde{m}_2$. Finally, since the application of f can only reduce min-entropy, $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{SS}'(W)) \ge \tilde{m}_2$ whenever $\mathbf{H}_\infty(W) \ge m_1$. □

As we saw, the proof above critically used the notion of average-case secure sketches. Luckily, all our constructions (for example, those obtained via Lemma 3.1) are average-case, so this subtlety will not matter too much.

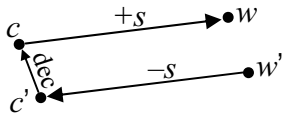
We will see the utility of this novel type of embedding in section 7.

5. Constructions for Hamming distance. In this section we consider constructions for the space $\mathcal{M} = \mathcal{F}^n$ under the Hamming distance metric. Let $F = |\mathcal{F}|$ and $f = \log_2 F$.

Secure sketches: The code-offset construction. For the case of $\mathcal{F} = \{0,1\}$, Juels and Wattenberg [39] considered a notion of "fuzzy commitment."[8] Given an $[n, k, 2t+1]_2$ error-correcting code C (not necessarily linear), they fuzzy-commit to x by publishing $w \oplus C(x)$. Their construction can be rephrased in our language to give a very simple construction of secure sketches for general $\mathcal{F}$.

We start with an $[n, k, 2t+1]_{\mathcal{F}}$ error-correcting code C (not necessarily linear). The idea is to use C to correct errors in w even though w may not be in C. This is accomplished by shifting the code so that a codeword matches up with w and storing the shift as the sketch. To do so, we need to view $\mathcal{F}$ as an additive cyclic group of order $\mathcal{F}$ (in the case of most common error-correcting codes, F will already be a field).

Construction 2 (code-offset construction). On input w, select a random codeword c (this is equivalent to choosing a random $x \in \mathcal{F}^k$ and computing $C(x)$), and set $\mathsf{SS}(w)$ to be the shift needed to get from c to w: $\mathsf{SS}(w) = w - c$. Then $\mathsf{Rec}(w', s)$ is computed by subtracting the shift s from w' to get $c' = w' - s$, decoding c' to get c (note that because $\mathsf{dis}(w', w) \le t$, so is $\mathsf{dis}(c', c)$), and computing w by shifting back to get $w = c + s$.



In the case of $\mathcal{F} = \{0,1\}$, addition and subtraction are the same, and we get that computation of the sketch is the same as the Juels–Wattenberg commitment: $\mathsf{SS}(w) = w \oplus C(x)$. In this case, to recover w given w' and $s = \mathsf{SS}(w)$, compute $c' = w' \oplus s$, decode c' to get c, and compute $w = c \oplus s$.

When the code C is linear, this scheme can be simplified as follows.

Construction 3 (syndrome construction). Set $\mathsf{SS}(w) = \mathsf{syn}(w)$. To compute $\mathsf{Rec}(w', s)$, find the unique vector $e \in \mathcal{F}^n$ of Hamming weight $\le t$ such that $\mathsf{syn}(e) = \mathsf{syn}(w') - s$, and output $w = w' - e$.

As explained in section 2, finding the short error-vector e from its syndrome is the same as decoding the code. It is easy to see that two constructions above are equivalent: given $\mathsf{syn}(w)$ one can sample from $w - c$ by choosing a random string v with $\mathsf{syn}(v) = \mathsf{syn}(w)$; conversely, $\mathsf{syn}(w - c) = \mathsf{syn}(w)$. To show that Rec finds the correct w, observe that $\mathsf{dis}(w' - e, w') \le t$ by the constraint on the weight of e, and $\mathsf{syn}(w' - e) = \mathsf{syn}(w') - \mathsf{syn}(e) = \mathsf{syn}(w') - (\mathsf{syn}(w') - s) = s$. There can be only one value within distance t of w' whose syndrome is s (else by subtracting two such values we get a codeword that is closer than $2t+1$ to 0, but 0 is also a codeword), so $w' - e$ must be equal to w.

As mentioned in the introduction, the syndrome construction has appeared before as a component of some cryptographic protocols over quantum and other noisy channels [6, 18], though it has not been analyzed the same way.

Both schemes are $(\mathcal{F}^n, m, m - (n-k)f, t)$-secure sketches. For the randomized scheme, the intuition for understanding the entropy loss is as follows: we add k random elements of $\mathcal{F}$ and publish n elements of $\mathcal{F}$. The formal proof is simply Lemma 4.5, because addition in $\mathcal{F}^n$ is a family of transitive isometries. For the syndrome scheme, this follows from Lemma 3.1, because the syndrome is $(n-k)$ elements of $\mathcal{F}$.

We thus obtain the following theorem.

[8] In their interpretation, one commits to x by picking a random w and publishing $\mathsf{SS}(w; x)$.

THEOREM 5.1. *Given an $[n, k, 2t+1]_{\mathcal{F}}$ error-correcting code, one can construct an average-case $(\mathcal{F}^n, m, m-(n-k)f, t)$-secure sketch, which is efficient if encoding and decoding are efficient. Furthermore, if the code is linear, then the sketch is deterministic, and its output is $(n-k)$ symbols long.*

In Appendix C we present some generic lower bounds on secure sketches and fuzzy extractors. Recall that $A_F(n,d)$ denotes the maximum number K of codewords possible in a code of distance d over n-character words from an alphabet of size F. Then by Lemma C.1, we obtain that the entropy loss of a secure sketch for the Hamming metric is at least $nf - \log_2 A_F(n, 2t+1)$ when the input is uniform (that is, when $m = nf$), because $K(\mathcal{M}, t)$ from Lemma C.1 is in this case equal to $A_F(n, 2t+1)$ (since a code that corrects t Hamming errors must have minimum distance at least $2t+1$). This means that if the underlying code is optimal (i.e., $K = A_F(n, 2t+1)$), then the code-offset construction above is optimal for the case of uniform inputs, because its entropy loss is $nf - \log_F K \log_2 F = nf - \log_2 K$. Of course, we do not know the exact value of $A_F(n,d)$, let alone any efficiently decodable codes which meet the bound, for many settings of F, n, and d. Nonetheless, the code-offset scheme gets as close to optimality as possible given the coding constraints. If better efficient codes are invented, then better (i.e., lower loss or higher error-tolerance) secure sketches will result.

Fuzzy extractors. As a warm-up, consider the case when W is uniform ($m = n$), and look at the code-offset sketch construction: $v = w - C(x)$. For $\mathsf{Gen}(w)$, output $R = x$, $P = v$. For $\mathsf{Rep}(w', P)$, decode $w' - P$ to obtain $C(x)$, and apply C^{-1} to obtain x. The result, quite clearly, is an $(\mathcal{F}^n, nf, kf, t, 0)$-fuzzy extractor, since v is truly random and independent of x when w is random. In fact, this is exactly the usage proposed by Juels and Wattenberg [39], except they viewed the above fuzzy extractor as a way to use w to "fuzzy commit" to x, without revealing information about x.

Unfortunately, the above construction setting $R = x$ works only for uniform W, since otherwise v would leak information about x.

In general, we use the construction in Lemma 4.3 combined with Theorem 5.1 to obtain the following theorem.

THEOREM 5.2. *Given any $[n, k, 2t+1]_{\mathcal{F}}$ code C and any m, ϵ, there exists an average-case $(\mathcal{M}, m, \ell, t, \epsilon)$-fuzzy extractor, where $\ell = m + kf - nf - 2\log\left(\frac{1}{\epsilon}\right) + 2$. The generation Gen and recovery Rep are efficient if C has efficient encoding and decoding.*

6. Constructions for set difference. We now turn to inputs that are subsets of a universe $\mathcal{U}$; let $n = |\mathcal{U}|$. This corresponds to representing an object by a list of its features. Examples include "minutiae" (ridge meetings and endings) in a fingerprint, short strings which occur in a long document, or lists of favorite movies.

Recall that the distance between two sets w, w' is the size of their symmetric difference: $\mathsf{dis}(w, w') = |w \triangle w'|$. We will denote this metric space by $\mathsf{SDif}(\mathcal{U})$. A set w can be viewed as its *characteristic vector* in $\{0,1\}^n$, with 1 at position $x \in \mathcal{U}$ if $x \in w$, and 0 otherwise. Such representation of sets makes set difference the same as the Hamming metric. However, we will mostly focus on settings where n is much larger than the size of w, so that representing a set w by n bits is much less efficient than, say, writing down a list of elements in w, which requires only $|w| \log n$ bits.

Large versus small universes. More specifically, we will distinguish two broad categories of settings. Let s denote the size of the sets that are given as inputs to the secure sketch (or fuzzy extractor) algorithms. Most of this section studies situations

where the universe size n is superpolynomial in the set size s. We call this the "large universe" setting. In contrast, the "small universe" setting refers to situations in which $n = poly(s)$. We want our various constructions to run in polynomial time and use polynomial storage space. In the large universe setting, the n-bit string representation of a set becomes too large to be usable—we will strive for solutions that are polynomial in s and $\log n$.

In fact, in many applications—for example, when the input is a list of book titles—it is possible that the actual universe is not only large, but also difficult to enumerate, making it difficult to even find the position in the characteristic vector corresponding to $x \in w$. In that case, it is natural to enlarge the universe to a well-understood class—for example, to include all possible strings of a certain length, whether or not they are actual book titles. This has the advantage that the position of x in the characteristic vector is simply x itself; however, because the universe is now even larger, the dependence of running time on n becomes even more important.

Fixed versus flexible set size. In some situations, all objects are represented by feature sets of exactly the same size s, while in others the sets may be of arbitrary size. In particular, the original set w and the corrupted set w' from which we would like to recover the original need not be of the same size. We refer to these two settings as *fixed* and *flexible* set size, respectively. When the set size is fixed, the distance $\mathsf{dis}(w, w')$ is always even: $\mathsf{dis}(w, w') = t$ if and only if w and w' agree on exactly $s - \frac{t}{2}$ points. We will denote the restriction of $\mathsf{SDif}(\mathcal{U})$ to s-element subsets by $\mathsf{SDif}_s(\mathcal{U})$.

Summary. As a point of reference, we will see below that $\log \binom{n}{s} - \log A(n, 2t+1, s)$ is a lower bound on the entropy loss of any secure sketch for set difference (whether or not the set size is fixed). Recall that $A(n, 2t + 1, s)$ represents the size of the largest code for Hamming space with minimum distance $2t + 1$, in which every word has weight exactly s. In the large universe setting, where $t \ll n$, the lower bound is approximately $t \log n$. The relevant lower bounds are discussed at the end of sections 6.1 and 6.2.

In the following sections we will present several schemes which meet this lower bound. The setting of small universes is discussed in section 6.1. We discuss the code-offset construction (from section 5), as well as a permutation-based scheme which is tailored to a fixed set size. The latter scheme is optimal for this metric but impractical.

In the remainder of the section, we discuss schemes for the large universe setting. In section 6.2 we give an improved version of the scheme of Juels and Sudan [38]. Our version achieves optimal entropy loss and storage $t \log n$ for fixed set size (notice the entropy loss does not depend on the set size s, although the running time does). The new scheme provides an exponential improvement over the original parameters (which are analyzed in Appendix D). Finally, in section 6.3 we describe how to adapt syndrome decoding algorithms for BCH codes to our application. The resulting scheme, called PinSketch, has optimal storage and entropy loss $t \log(n + 1)$, handles flexible set sizes, and is probably the most practical of the schemes presented here. Another scheme achieving similar parameters (but less efficiently) can be adapted from information reconciliation literature [51]; see section 9 for more details.

We do not discuss fuzzy extractors beyond mentioning here that each secure sketch presented in this section can be converted to a fuzzy extractor using Lemma 4.3. We have already seen an example of such a conversion in section 5.

Table 1 summarizes the constructions discussed in this section.

6.1. Small universes. When the universe size is polynomial in s, there are a number of natural constructions. The most direct one, given previous work, is the

TABLE 1

Summary of secure sketches for set difference. Notes: In the Juels–Sudan (JS) scheme, r is a parameter, $s \le r \le n$; generic syndrome and permutation-based schemes achieve entropy loss similar to that of the improved Juels–Sudan scheme and PinSketch, $\approx t \log n$, when $t \ll n$; see section 6.3 for running time of PinSketch.

	Entropy loss	Storage	Time	Set size
Juels–Sudan [38]	$t \log n + \log \left(\binom{n}{r} / \binom{n-s}{r-s} \right) + 2$	$r \log n$	$poly(r \log(n))$	Fixed
Generic syndrome	$n - \log A(n, 2t+1)$	$n - \log A(n, 2t+1)$ (for linear codes)	$poly(n)$	Flexible
Permutation-based	$\log \binom{n}{s} - \log A(n, 2t+1, s)$	$O(n \log n)$	$poly(n)$	Fixed
Improved JS	$t \log n$	$t \log n$	$poly(s \log n)$	Fixed
PinSketch	$t \log(n+1)$	$t \log(n+1)$	$poly(s \log n)$	Flexible

construction of Juels and Sudan [38]. Unfortunately, that scheme requires a fixed set size and achieves relatively poor parameters (see Appendix D).

We suggest two possible constructions. The first involves representing sets as n-bit strings and using the constructions of section 5. The second construction, presented below, requires a fixed set size but achieves slightly improved parameters by going through "constant-weight" codes.

Permutation-based sketch. Recall the general construction of section 4.2 for transitive metric spaces. Let Π be a set of all permutations on $\mathcal{U}$. Given $\pi \in \Pi$, make it a permutation on $\mathsf{SDif}_s(\mathcal{U})$ naturally: $\pi(w) = \{\pi(x) | x \in w\}$. This makes Π into a family of transitive isometries on $\mathsf{SDif}_s(\mathcal{U})$, and thus the results of section 4.2 apply.

Let $C \subseteq \{0,1\}^n$ be any $[n, k, 2t+1]$ binary code in which all words have weight exactly s. Such codes have been studied extensively (see, e.g., [1, 11] for a summary of known upper and lower bounds). View elements of the code as sets of size s. We obtain the following scheme, which produces a sketch of length $O(n \log n)$.

Construction 4 (permutation-based sketch). On input $w \subseteq \mathcal{U}$ of size s, choose $b \subseteq \mathcal{U}$ at random from the code C, and choose a random permutation $\pi : \mathcal{U} \to \mathcal{U}$ such that $\pi(w) = b$ (that is, choose a random matching between w and b and a random matching between $\mathcal{U} - w$ and $\mathcal{U} - b$). Output $\mathsf{SS}(w) = \pi$ (say, by listing $\pi(1), \ldots, \pi(n)$). To recover w from w' such that $\mathsf{dis}(w, w') \le t$ and π, compute $b' = \pi^{-1}(w')$, decode the characteristic vector of b' to obtain b, and output $w = \pi(b)$.

This construction is efficient as long as decoding is efficient (everything else takes time $O(n \log n)$). By Lemma 4.5, its entropy loss is $\log \binom{n}{s} - k$: here $|\Pi| = n!$ and $\Gamma = s!(n-s)!$, so $\log |\Pi| - \log \Gamma = \log n!/(s!(n-s)!)$.

Comparing the Hamming scheme with the permutation scheme. The code-offset construction was shown to have entropy loss $n - \log A(n, 2t+1)$ if an optimal code is used; the random permutation scheme has entropy loss $\log \binom{n}{s} - \log A(n, 2t+1, s)$ for an optimal code. The Bassalygo–Elias inequality (see [72]) shows that the bound on the random permutation scheme is always at least as good as the bound on the code offset scheme: $A(n, d) \cdot 2^{-n} \le A(n, d, s) \cdot \binom{n}{s}^{-1}$. This implies that $n - \log A(n, d) \ge \log \binom{n}{s} - \log A(n, d, s)$. Moreover, standard packing arguments give better constructions of constant-weight codes than they do of ordinary codes.[9] In fact, the random

[9] This comes from the fact that the intersection of a ball of radius d with the set of all words of weight s is much smaller than the ball of radius d itself.

permutations scheme is optimal for this metric, just as the code-offset scheme is optimal for the Hamming metric.

We show this as follows. Restrict t to be even, because $\mathsf{dis}(w, w')$ is always even if $|w| = |w'|$. Then the minimum distance of a code over $\mathsf{SDif}_s(\mathcal{U})$ that corrects up to t errors must be at least $2t + 1$. Indeed, suppose not. Then take two codewords, c_1 and c_2, such that $\mathsf{dis}(c_1, c_2) \le 2t$. There are k elements in c_1 that are not in c_2 (call their set $c_1 - c_2$), and k elements in c_2 that are not in c_1 (call their set $c_2 - c_1$) with $k \le t$. Starting with c_1, remove $t/2$ elements of $c_1 - c_2$ and add $t/2$ elements of $c_2 - c_1$ to obtain a set w (note that here we are using that t is even; if $k < t/2$, then use k elements). Then $\mathsf{dis}(c_1, w) \le t$ and $\mathsf{dis}(c_2, w) \le t$, and so if the received word is w, the receiver cannot be certain whether the sent word was c_1 or c_2 and hence cannot correct t errors.

Therefore, by Lemma C.1, we get that the entropy loss of a secure sketch must be at least $\log \binom{n}{s} - \log A(n, 2t + 1, s)$ in the case of a uniform input w. Thus in principle, it is better to use the random permutation scheme. Nonetheless, there are caveats. First, we do not know of *explicitly* constructed constant-weight codes that beat the Elias–Bassalygo inequality and would thus lead to better entropy loss for the random permutation scheme than for the Hamming scheme (see [11] for more on constructions of constant-weight codes and [1] for upper bounds). Second, much more is known about efficient implementation of decoding for ordinary codes than for constant-weight codes; for example, one can find off-the-shelf hardware and software for decoding many binary codes. In practice, the Hamming-based scheme is likely to be more useful.

6.2. Improving the construction of Juels and Sudan. We now turn to the large universe setting, where n is superpolynomial in the set size s, and we would like operations to be polynomial in s and $\log n$.

Juels and Sudan [38] proposed a secure sketch for the set difference metric with fixed set size (called a "fuzzy vault" in that paper). We present their original scheme here with an analysis of the entropy loss in Appendix D. In particular, our analysis shows that the original scheme has good entropy loss only when the storage space is very large.

We suggest an improved version of the Juels–Sudan scheme which is simpler and achieves much better parameters. The entropy loss and storage space of the new scheme are both $t \log n$, which is optimal. (The same parameters are also achieved by the BCH-based construction PinSketch in section 6.3.) Our scheme has the advantage of being even simpler to analyze, and the computations are simpler. As with the original Juels–Sudan scheme, we assume $n = |\mathcal{U}|$ is a prime power and work over $\mathcal{F} = GF(n)$.

An intuition for the scheme is that the numbers $y_{s+1}, \ldots, y_r$ from the Juels–Sudan scheme need not be chosen at random. One can instead evaluate them as $y_i = p'(x_i)$ for some polynomial p'. One can then represent the entire list of pairs (x_i, y_i) implicitly, using only a few of the coefficients of p'. The new sketch is deterministic (this was not the case for our preliminary version in [25]). Its implementation is available [36].

Construction 5 (improved Juels–Sudan secure sketch for sets of size s). To compute $\mathsf{SS}(w)$, do the following:

1. Let $p'()$ be the unique monic polynomial of degree exactly s such that $p'(x) = 0$ for all $x \in w$. (That is, let $p'(z) \stackrel{\text{def}}{=} \prod_{x \in w}(z - x)$.)
2. Output the coefficients of $p'()$ of degree $s - 1$ down to $s - t$.

This is equivalent to computing and outputting the first t symmetric polynomials of the values in A; i.e., if $w = \{x_1, \ldots, x_s\}$, then output

$$\sum_i x_i,\ \sum_{i \neq j} x_i x_j, \ldots,\ \sum_{S \subseteq [s], |S| = t} \left(\prod_{i \in S} x_i \right).$$

To compute $\mathsf{Rec}(w', p')$, where $w' = \{a_1, a_2, \ldots, a_s\}$, do the following:

1. Create a new polynomial p_{high}, of degree s which shares the top $t+1$ coefficients of p'; that is, let $p_{\text{high}}(z) \stackrel{\text{def}}{=} z^s + \sum_{i=s-t}^{s-1} a_i z^i$.
2. Evaluate p_{high} on all points in w' to obtain s pairs (a_i, b_i).
3. Use $[s, s-t, t+1]_{\mathcal{U}}$ Reed–Solomon decoding (see, e.g., [7, 72]) to search for a polynomial p_{low} of degree $s-t-1$ such that $p_{\text{low}}(a_i) = b_i$ for at least $s - t/2$ of the a_i values. If no such polynomial exists, then stop and output "fail."
4. Output the list of zeros (roots) of the polynomial $p_{\text{high}} - p_{\text{low}}$ (see, e.g., [66] for root-finding algorithms; they can be sped up by first factoring out the known roots—namely, $(z - a_i)$ for the $s - t/2$ values of a_i that were not deemed erroneous in the previous step).

To see that this secure sketch can tolerate t set difference errors, suppose that $\mathsf{dis}(w, w') \leq t$. Let p' be as in the sketch algorithm; that is, $p'(z) = \prod_{x \in w}(z - x)$. The polynomial p' is monic; that is, its leading term is z^s. We can divide the remaining coefficients into two groups: the high coefficients, denoted $a_{s-t}, \ldots, a_{s-1}$, and the low coefficients, denoted $b_1, \ldots, b_{s-t-1}$:

$$p'(z) = \underbrace{z^s + \sum_{i=s-t}^{s-1} a_i z^i}_{p_{\text{high}}(z)} + \underbrace{\sum_{i=0}^{s-t-1} b_i z^i}_{q(z)}.$$

We can write p' as $p_{\text{high}} + q$, where q has degree $s - t - 1$. The recovery algorithm gets the coefficients of p_{high} as input. For any point x in w, we have $0 = p'(x) = p_{\text{high}}(x) + q(x)$. Thus, p_{high} and $-q$ agree at all points in w. Since the set w intersects w' in at least $s - t/2$ points, the polynomial $-q$ satisfies the conditions of step 3 in Rec. That polynomial is unique, since no two distinct polynomials of degree $s - t - 1$ can get the correct b_i on more than $s - t/2$ a_is (else, they agree on at least $s - t$ points, which is impossible). Therefore, the recovered polynomial p_{low} must be $-q$; hence $p_{\text{high}}(x) - p_{\text{low}}(x) = p'(x)$. Thus, Rec computes the correct p' and therefore correctly finds the set w, which consists of the roots of p'.

Since the output of SS is t field elements, the entropy loss of the scheme is at most $t \log n$ by Lemma 3.1. (We will see below that this bound is tight, since any sketch must lose at least $t \log n$ in some situations.) We have proven the following theorem.

THEOREM 6.1 (analysis of improved Juels–Sudan). *Construction* 5 *is an average-case* $(\mathsf{SDif}_s(\mathcal{U}), m, m - t \log n, t)$ *secure sketch. The entropy loss and storage of the scheme are at most* $t \log n$*, and both the sketch generation* $\mathsf{SS}()$ *and the recovery procedure* $\mathsf{Rec}()$ *run in time polynomial in* s*,* t*, and* $\log n$*.*

Lower bounds for fixed set size in a large universe. The short length of the sketch makes this scheme feasible for essentially any ratio of set size to universe size (we only need $\log n$ to be polynomial in s). Moreover, for large universes the entropy loss $t \log n$ is essentially optimal for uniform inputs (i.e., when $m = \log \binom{n}{s}$). We show this as follows. As already mentioned in section 6.1, Lemma C.1 shows that for a uniformly distributed input, the best possible entropy loss is $m - m' \geq \log \binom{n}{s} - \log A(n, 2t+1, s)$.

By Theorem 12 of Agrell, Vardy, and Zeger [1],

$$A(n, 2t+2, s) \le \frac{\binom{n}{s-t}}{\binom{s}{s-t}}.$$

Noting that $A(n, 2t+1, s) = A(n, 2t+2, s)$ because distances in $\mathsf{SDif}_s(\mathcal{U})$ are even, the entropy loss is at least

$$\begin{aligned} m - m' &\ge \log \binom{n}{s} - \log A(n, 2t+1, s) \\ &\ge \log \binom{n}{s} - \log \left(\binom{n}{s-t} \Big/ \binom{s}{s-t} \right) \\ &= \log \binom{n-s+t}{t}. \end{aligned}$$

When $n \gg s$, this last quantity is roughly $t \log n$, as desired.

6.3. Large universes via the Hamming metric: Sublinear-time decoding. In this section, we show that the syndrome construction of section 5 can in fact be adapted for small sets in a large universe, using specific properties of algebraic codes. We will show that BCH codes, which contain Hamming and Reed–Solomon codes as special cases, have these properties. As opposed to the constructions of the previous section, the construction of this section is flexible and can accept input sets of any size.

Thus we obtain a sketch for sets of flexible size, with entropy loss and storage $t \log(n+1)$. We will assume that n is one less than a power of 2: $n = 2^m - 1$ for some integer m, and will identify $\mathcal{U}$ with the nonzero elements of the binary finite field of degree m: $\mathcal{U} = GF(2^m)^*$.

Syndrome manipulation for small-weight words. Suppose now that we have a small set $w \subseteq \mathcal{U}$ of size s, where $n \gg s$. Let x_w denote the characteristic vector of w (see the beginning of section 6). Then the syndrome construction says that $\mathsf{SS}(w) = \mathsf{syn}(x_w)$. This is an $(n-k)$-bit quantity. Note that the syndrome construction gives us no special advantage over the code-offset construction when the universe is small: storing the n-bit $x_w + C(r)$ for a random k-bit r is not a problem. However, it is a substantial improvement when $n \gg n - k$.

If we want to use $\mathsf{syn}(x_w)$ as the sketch of w, then we must choose a code with $n - k$ very small. In particular, the entropy of w is at most $\log \binom{n}{s} \approx s \log n$, and so the entropy loss $n - k$ had better be at most $s \log n$. Binary BCH codes are suitable for our purposes: they are a family of $[n, k, \delta]_2$ linear codes with $\delta = 2t + 1$ and $k = n - tm$ (assuming $n = 2^m - 1$) (see, e.g., [72]). These codes are optimal for $t \ll n$ by the Hamming bound, which implies that $k \le n - \log \binom{n}{t}$ [72].[10] Using the syndrome sketch with a BCH code C, we get entropy loss $n - k = t \log(n+1)$, which is essentially the same as the $t \log n$ of the improved Juels–Sudan scheme (recall that $\delta \ge 2t + 1$ allows us to correct t set difference errors).

The only problem is that the scheme appears to require computation time $\Omega(n)$, since we must compute $\mathsf{syn}(x_w) = Hx_w$ and, later, run a decoding algorithm to recover

[10] The Hamming bound is based on the observation that for any code of distance δ, the balls of radius $\lfloor (\delta - 1)/2 \rfloor$ centered at various codewords must be disjoint. Each such ball contains $\binom{n}{\lfloor (\delta-1)/2 \rfloor}$ points, and so $2^k \binom{n}{\lfloor (\delta-1)/2 \rfloor} \le 2^n$. In our case $\delta = 2t + 1$, and so the bound yields $k \le n - \log \binom{n}{t}$.

x_w. For BCH codes, this difficulty can be overcome. A word of small weight w can be described by listing the positions on which it is nonzero. We call this description the *support* of x_w and write $\mathrm{supp}(x_w)$ (note that $\mathrm{supp}(x_w) = w$; see the discussion of enlarging the universe appropriately at the beginning of section 6).

The following lemma holds for general BCH codes (which include binary BCH codes and Reed–Solomon codes as special cases). We state it for binary codes since that is most relevant to the application.

LEMMA 6.2. *For a $[n, k, \delta]$ binary BCH code C one can compute*

- $\mathsf{syn}(x)$*, given* $\mathrm{supp}(x)$*, in time polynomial in* δ*,* $\log n$*, and* $|\mathrm{supp}(x)|$*,*
- $\mathrm{supp}(x)$*, given* $\mathsf{syn}(x)$ *(when x has weight at most $(\delta - 1)/2$), in time polynomial in δ and* $\log n$*.*

The proof of Lemma 6.2 requires a careful reworking of the standard BCH decoding algorithm. The details are presented in Appendix E. For now, we present the resulting secure sketch for set difference.

Construction 6 (PinSketch). To compute $\mathsf{SS}(w) = \mathsf{syn}(x_w)$, do the following:

1. Let $s_i = \sum_{x \in w} x^i$ (computations in $GF(2^m)$).
2. Output $\mathsf{SS}(w) = (s_1, s_3, s_5, \ldots, s_{2t-1})$.

To recover $\mathsf{Rec}(w', (s_1, s_3, \ldots, s_{2t-1}))$, do the following:

1. Compute $(s'_1, s'_3, \ldots, s'_{2t-1}) = \mathsf{SS}(w') = \mathsf{syn}(x_{w'})$.
2. Let $\sigma_i = s'_i - s_i$ (in $GF(2^m)$, so "$-$" is the same as "$+$").
3. Compute $\mathrm{supp}(v)$ such that $\mathsf{syn}(v) = (\sigma_1, \sigma_3, \ldots, \sigma_{2t-1})$ and $|\mathrm{supp}(v)| \le t$ by Lemma 6.2.
4. If $\mathsf{dis}(w, w') \le t$, then $\mathrm{supp}(v) = w \triangle w'$. Thus, output $w = w' \triangle \mathrm{supp}(v)$.

An implementation of this construction, including the reworked BCH decoding algorithm, is available [36].

The bound on entropy loss is easy to see: the output is $t \log(n+1)$ bits long, and hence the entropy loss is at most $t \log(n+1)$ by Lemma 3.1. We obtain the following theorem.

THEOREM 6.3. *PinSketch is an average-case* $(\mathsf{SDif}(\mathcal{U}), m, m - t\log(n+1), t)$*-secure sketch for set difference with storage* $t \log(n+1)$*. The algorithms* SS *and* Rec *both run in time polynomial in t and* $\log n$*.*

7. Constructions for edit distance. The space of interest in this section is the space $\mathcal{F}^*$ for some alphabet $\mathcal{F}$, with distance between two strings defined as the number of character insertions and deletions needed to get from one string to the other. Denote this space by $\mathsf{Edit}_{\mathcal{F}}(n)$. Let $F = |\mathcal{F}|$.

First, note that applying the generic approach for transitive metric spaces (as with the Hamming space and the set difference space for small universe sizes) does not work here, because the edit metric is not known to be transitive. Instead, we consider embeddings of the edit metric on $\{0,1\}^n$ into the Hamming or set difference metric of much larger dimension. We look at two types: standard low-distortion embeddings and "biometric" embeddings as defined in section 4.3.

For the binary edit distance space of dimension n, we obtain secure sketches and fuzzy extractors correcting t errors with entropy loss roughly $tn^{o(1)}$, using a standard embedding, and $2.38\sqrt[3]{tn \log n}$, using a relaxed embedding. The first technique works better when t is small, say, $n^{1-\gamma}$ for a constant $\gamma > 0$. The second technique is better when t is large; it is meaningful roughly as long as $t < \frac{n}{15 \log^2 n}$.

7.1. Low-distortion embeddings. A (standard) embedding with distortion D is an injection $\psi : \mathcal{M}_1 \hookrightarrow \mathcal{M}_2$ such that for any two points $x, y \in \mathcal{M}_1$, the ratio

$\frac{\mathsf{dis}(\psi(x),\psi(y))}{\mathsf{dis}(x,y)}$ is at least 1 and at most D.

When the preliminary version of this paper appeared [25], no nontrivial embeddings were known mapping edit distance into ℓ_1 or the Hamming metric (i.e., known embeddings had distortion $O(n)$). Recently, Ostrovsky and Rabani [57] gave an embedding of the edit metric over $\mathcal{F} = \{0,1\}$ into ℓ_1 with subpolynomial distortion. It is an injective, polynomial time computable embedding which can be interpreted as mapping to the Hamming space $\{0,1\}^d$, where $d = \mathrm{poly}(n)$.[11]

FACT 7.1 (see [57]). *There is a polynomial time computable embedding denoted* $\psi_{\mathrm{ed}} : \mathsf{Edit}_{\{0,1\}}(n) \hookrightarrow \{0,1\}^{\mathrm{poly}(n)}$ *with distortion* $D_{\mathrm{ed}}(n) \stackrel{\mathrm{def}}{=} 2^{O(\sqrt{\log n \log\log n})}$.

We can compose this embedding with the fuzzy extractor constructions for the Hamming distance to obtain a fuzzy-extractor for edit distance which will be good when t, the number of errors to be corrected, is quite small. Recall that instantiating the syndrome fuzzy-extractor construction (Theorem 5.2) with a BCH code allows one to correct t' errors out of d at the cost of $t' \log d + 2\log\left(\frac{1}{\epsilon}\right) - 2$ bits of entropy.

Construction 7. For any length n and error threshold t, let ψ_{ed} be the embedding given by Fact 7.1 from $\mathsf{Edit}_{\{0,1\}}(n)$ into $\{0,1\}^d$ (where $d = \mathrm{poly}(n)$), and let syn be the syndrome of a BCH code correcting $t' = tD_{\mathrm{ed}}(n)$ errors in $\{0,1\}^d$. Let $\{H_x\}_{x\in X}$ be a family of universal hash functions from $\{0,1\}^d$ to $\{0,1\}^\ell$ for some ℓ. To compute Gen on input $w \in \mathsf{Edit}_{\{0,1\}}(n)$, pick a random x and output

$$R = H_x(\psi_{\mathrm{ed}}(w)) \ , P = (\mathsf{syn}(\psi_{\mathrm{ed}}(w)), x)\,.$$

To compute Rep on inputs w' and $P = (s, x)$, compute $y = \mathsf{Rec}(\psi_{\mathrm{ed}}(w'), s)$, where Rec is from Construction 3, and output $R = H_x(y)$.

Because ψ_{ed} is injective, a secure sketch can be constructed similarly: $\mathsf{SS}(w) = \mathsf{syn}(\psi(w))$. To recover w from w' and s, compute $\psi_{\mathrm{ed}}^{-1}(\mathsf{Rec}(\psi_{\mathrm{ed}}(w')))$. However, this secure sketch is not known to be efficient, because it is not known how to compute ψ_{ed}^{-1} efficiently.

PROPOSITION 7.2. *For any* n, t, m, *there is an average-case* $(\mathsf{Edit}_{\{0,1\}}(n), m, m', t)$*-secure sketch and an efficient average-case* $(\mathsf{Edit}_{\{0,1\}}(n), m, \ell, t, \epsilon)$*-fuzzy extractor where* $m' = m - t2^{O(\sqrt{\log n \log\log n})}$ *and* $\ell = m' - 2\log\left(\frac{1}{\epsilon}\right) + 2$. *In particular, for any* $\alpha < 1$, *there exists an efficient fuzzy extractor tolerating* n^α *errors with entropy loss* $n^{\alpha+o(1)} + 2\log\left(\frac{1}{\epsilon}\right)$.

Proof. Construction 7 is the same as the construction of Theorem 5.2 (instantiated with a BCH-code-based syndrome construction) acting on $\psi_{\mathrm{ed}}(w)$. Because ψ_{ed} is injective, the min-entropy of $\psi_{\mathrm{ed}}(w)$ is the same as the min-entropy m of w. The entropy loss in Construction 3 instantiated with BCH codes is $t' \log d = t2^{O(\sqrt{\log n \log\log n})} \log \mathrm{poly}(n)$. Because $2^{O(\sqrt{\log n \log\log n})}$ grows faster than $\log n$, this is the same as $t2^{O(\sqrt{\log n \log\log n})}$. □

Note that the peculiar-looking distortion function from Fact 7.1 increases more slowly than any polynomial in n, but still faster than any polynomial in $\log n$. In sharp contrast, the best lower bound states that any embedding of $\mathsf{Edit}_{\{0,1\}}(n)$ into ℓ_1 (and hence Hamming) must have distortion at least $\Omega(\log n / \log\log n)$ [2]. Closing the gap between the two bounds remains an open problem.

General alphabets. To extend the above construction to general $\mathcal{F}$, we represent each character of $\mathcal{F}$ as a string of $\log F$ bits. This is an embedding $\mathcal{F}^n$ into $\{0,1\}^{n\log F}$,

[11] The embedding of [57] produces strings of integers in the space $\{1,\ldots,O(\log n)\}^{\mathrm{poly}(n)}$, equipped with ℓ_1 distance. One can convert this into the Hamming metric with only a logarithmic blowup in length by representing each integer in unary.

which increases edit distance by a factor of at most $\log F$. Then $t' = t(\log F)D_{\mathrm{ed}}(n)$ and $d = \mathrm{poly}(n, \log F)$. Using these quantities, we get the generalization of Proposition 7.2 for larger alphabets (again, by the same embedding) by changing the formula for m' to $m' = m - t(\log F)2^{O(\sqrt{\log(n \log F) \log\log(n \log F)})}$.

7.2. Relaxed embeddings for the edit metric. In this section, we show that a relaxed notion of embedding, called a *biometric embedding* in section 4.3, can produce fuzzy extractors and secure sketches that are better than what one can get from the embedding of [57] when t is large (they are also much simpler algorithmically, which makes them more practical). We first discuss fuzzy extractors and later extend the technique to secure sketches.

Fuzzy extractors. Recall that unlike low-distortion embeddings, biometric embeddings do not care about relative distances, as long as points that were "close" (closer than t_1) do not become "distant" (farther apart than t_2). The only additional requirement of a biometric embedding is that it preserve some min-entropy: we do not want too many points to collide together. We now describe such an embedding from the edit distance to the set difference.

A *c-shingle* is a length-c consecutive substring of a given string w. A *c-shingling* [10] of a string w of length n is the set (ignoring order or repetition) of all $(n-c+1)$ c-shingles of w. (For instance, a 3-shingling of "abcdecdeah" is {abc, bcd, cde, dec, ecd, dea, eah}.) Thus, the range of the c-shingling operation consists of all nonempty subsets of size at most $n-c+1$ of $\mathcal{F}^c$. Let $\mathsf{SDif}(\mathcal{F}^c)$ stand for the set difference metric over subsets of $\mathcal{F}^c$ and SH_c stand for the c-shingling map from $\mathsf{Edit}_{\mathcal{F}}(n)$ to $\mathsf{SDif}(\mathcal{F}^c)$. We now show that SH_c is a good biometric embedding.

LEMMA 7.3. *For any c, SH_c is an average-case $(t_1, t_2 = (2c-1)t_1, m_1, m_2 = m_1 - \lceil \frac{n}{c} \rceil \log_2(n-c+1))$-biometric embedding of $\mathsf{Edit}_{\mathcal{F}}(n)$ into $\mathsf{SDif}(\mathcal{F}^c)$.*

Proof. Let $w, w' \in \mathsf{Edit}_{\mathcal{F}}(n)$ be such that $\mathsf{dis}(w, w') \le t_1$ and I be the sequence of at most t_1 insertions and deletions that transforms w into w'. It is easy to see that each character deletion or insertion adds at most $(2c-1)$ to the symmetric difference between $\mathsf{SH}_c(w)$ and $\mathsf{SH}_c(w')$, which implies that $\mathsf{dis}(\mathsf{SH}_c(w), \mathsf{SH}_c(w')) \le (2c-1)t_1$, as needed.

For $w \in \mathcal{F}^n$, define $g_c(w)$ as follows. Compute $\mathsf{SH}_c(w)$ and store the resulting shingles in lexicographic order $h_1 \dots h_k$ $(k \le n-c+1)$. Next, naturally partition w into $\lceil n/c \rceil$ c-shingles $s_1 \dots s_{\lceil n/c \rceil}$, all disjoint except for (possibly) the last two, which overlap by $c\lceil n/c \rceil - n$ characters. Next, for $1 \le j \le \lceil n/c \rceil$, set p_j to be the index $i \in \{0 \dots k\}$ such that $s_j = h_i$. In other words, p_j tells the index of the jth disjoint shingle of w in the alphabetically ordered k-set $\mathsf{SH}_c(w)$. Set $g_c(w) = (p_1, \dots, p_{\lceil n/c \rceil})$. (For instance, g_3("abcdecdeah") $= (1, 5, 4, 6)$, representing the alphabetical order of "abc", "dec", "dea", and "eah" in SH_3("abcdecdeah").) The number of possible values for $g_c(w)$ is at most $(n-c+1)^{\lceil \frac{n}{c} \rceil}$, and w can be completely recovered from $\mathsf{SH}_c(w)$ and $g_c(w)$.

Now, assume W is any distribution of min-entropy at least m_1 on $\mathsf{Edit}_{\mathcal{F}}(n)$. Applying Lemma 2.2(b), we get $\tilde{\mathbf{H}}_\infty(W \mid g_c(W)) \ge m_1 - \lceil \frac{n}{c} \rceil \log_2(n-c+1)$. Since $\Pr(W = w \mid g_c(W) = g) = \Pr(\mathsf{SH}_c(W) = \mathsf{SH}_c(w) \mid g_c(W) = g)$ (because given $g_c(w)$, $\mathsf{SH}_c(w)$ uniquely determines w and vice versa), by applying the definition of $\tilde{\mathbf{H}}_\infty$, we obtain $\mathbf{H}_\infty(\mathsf{SH}_c(W)) \ge \tilde{\mathbf{H}}_\infty(\mathsf{SH}_c(W) \mid g_c(W)) = \tilde{\mathbf{H}}_\infty(W \mid g_c(W))$. The same proof holds for average min-entropy, conditioned on some auxiliary information I. ∎

By Theorem 6.3, for universe $\mathcal{F}^c$ of size F^c and distance threshold $t_2 = (2c-1)t_1$, we can construct a secure sketch for the set difference metric with entropy loss $t_2\lceil \log(F^c + 1) \rceil$ ($\lceil \cdot \rceil$ because Theorem 6.3 requires the universe size to be one less

than a power of 2). By Lemma 4.3, we can obtain a fuzzy extractor from such a sketch, with additional entropy loss $2\log\left(\frac{1}{\epsilon}\right) - 2$. Applying Lemma 4.6 to the above embedding and this fuzzy extractor, we obtain a fuzzy extractor for $\mathsf{Edit}_{\mathcal{F}}(n)$, any input entropy m, any distance t, and any security parameter ϵ, with the following entropy loss:

$$\left\lceil \frac{n}{c} \right\rceil \cdot \log_2(n-c+1) + (2c-1)t\lceil \log(F^c+1)\rceil + 2\log\left(\frac{1}{\epsilon}\right) - 2$$

(the first component of the entropy loss comes from the embedding, the second from the secure sketch for set difference, and the third from the extractor). The above sequence of lemmas results in the following construction, parameterized by shingle length c and a family of universal hash functions $\mathcal{H} = \{\mathsf{SDif}(\mathcal{F}^c) \to \{0,1\}^l\}_{x\in X}$, where l is equal to the input entropy m minus the entropy loss above.

Construction 8 (fuzzy extractor for edit distance). To compute $\mathsf{Gen}(w)$ for $|w| = n$:

1. Compute $\mathsf{SH}_c(w)$ by computing $n-c+1$ shingles $(v_1, v_2, \ldots, v_{n-c+1})$ and removing duplicates to form the shingle set v from w.
2. Compute $s = \mathsf{syn}(x_v)$ as in Construction 6.
3. Select a hash function $H_x \in \mathcal{H}$ and output $(R = H_x(v), P = (s, x))$.

To compute $\mathsf{Rep}(w', (s, x))$:

1. Compute $\mathsf{SH}_c(w')$ as above to get v'.
2. Use $\mathsf{Rec}(v', s)$ from in Construction 6 to recover v.
3. Output $R = H_x(v)$.

We thus obtain the following theorem.

THEOREM 7.4. *For any n, m, c, and $0 < \epsilon \le 1$, there is an efficient average-case $(\mathsf{Edit}_{\mathcal{F}}(n), m, m - \lceil \frac{n}{c} \rceil \log_2(n-c+1) - (2c-1)t\lceil \log(F^c+1)\rceil - 2\log\left(\frac{1}{\epsilon}\right) + 2, t, \epsilon)$-fuzzy extractor.*

Note that the choice of c is a parameter; by ignoring $\lceil\cdot\rceil$ and replacing $n-c+1$ with n, $2c-1$ with $2c$, and F^c+1 with F^c, we get that the minimum entropy loss occurs near

$$c = \left(\frac{n\log n}{4t\log F}\right)^{1/3}$$

and is about $2.38\,(t\log F)^{1/3}\,(n\log n)^{2/3}$ (2.38 is really $\sqrt[3]{4} + 1/\sqrt[3]{2}$). In particular, if the original string has a linear amount of entropy $\theta(n\log F)$, then we can tolerate $t = \Omega(n\log^2 F/\log^2 n)$ insertions and deletions while extracting $\theta(n\log F) - 2\log\left(\frac{1}{\epsilon}\right)$ bits. The number of bits extracted is linear; if the string length n is polynomial in the alphabet size F, then the number of errors tolerated is linear also.

Secure sketches. Observe that the proof of Lemma 7.3 actually demonstrates that our biometric embedding based on shingling is an embedding with recovery information g_c. Observe also that it is easy to reconstruct w from $\mathsf{SH}_c(w)$ and $g_c(w)$. Finally, note that PinSketch (Construction 6) is an average-case secure sketch (as are all secure sketches in this work). Thus, combining Theorem 6.3 with Lemma 4.7, we obtain the following theorem.

Construction 9 (secure sketch for edit distance). For $\mathsf{SS}(w)$, compute $v = \mathsf{SH}_c(w)$ and $s_1 = \mathsf{syn}(x_v)$ as in Construction 8. Compute $s_2 = g_c(w)$, writing each p_j as a string of $\lceil \log n \rceil$ bits. Output $s = (s_1, s_2)$. For $\mathsf{Rec}(w', (s_1, s_2))$, recover v as in Construction 8, sort it in alphabetical order, and recover w by stringing along elements of v according to indices in s_2.

THEOREM 7.5. *For any n, m, c, and $0 < \epsilon \leq 1$, there is an efficient average-case $(\mathsf{Edit}_{\mathcal{F}}(n), m, m - \lceil \frac{n}{c} \rceil \log_2(n - c + 1) - (2c - 1)t\lceil \log(F^c + 1) \rceil, t)$-secure sketch.*

The discussion about optimal values of c from above applies equally here.

Remark. In our definitions of secure sketches and fuzzy extractors, we required the original w and the (potentially) modified w' to come from the same space $\mathcal{M}$. This requirement was for simplicity of exposition. We can allow w' to come from a larger set, as long as distance from w is well defined. In the case of edit distance, for instance, w' can be shorter or longer than w; all the above results will apply as long as it is still within t insertions and deletions.

8. Probabilistic notions of correctness. The error model considered so far in this work is very strong: we required that secure sketches and fuzzy extractors accept *every* secret w' within distance t of the original input w, with no probability of error.

Such a stringent model is useful as it makes no assumptions on either the exact stochastic properties of the error process or the adversary's computational limits. However, Lemma C.1 shows that secure sketches (and fuzzy extractors) correcting t errors can only be as "good" as error-correcting codes with minimum distance $2t + 1$. By slightly relaxing the correctness condition, we will see that one can tolerate many more errors. For example, there is no good code which can correct $n/4$ errors in the binary Hamming metric: by the Plotkin bound (see, e.g., [69, Lecture 8]) a code with minimum distance greater than $n/2$ has at most $2n$ codewords. Thus, there is no secure sketch with residual entropy $m' \geq \log n$ which can correct $n/4$ errors with probability 1. However, with the relaxed notions of correctness below, one can tolerate arbitrarily close to $n/2$ errors, i.e., correct $n(\frac{1}{2} - \gamma)$ errors for any constant $\gamma > 0$, and still have residual entropy $\Omega(n)$.

In this section, we discuss three relaxed error models and show how the constructions of the previous sections can be modified to gain greater error-correction in these models. We will focus on secure sketches for the binary Hamming metric. The same constructions yield fuzzy extractors (by Lemma 4.1). Many of the observations here also apply to metrics other than Hamming.

A common point is that we will require only that the a corrupted input w' be recovered with probability at least $1 - \alpha < 1$ (the probability space varies). We describe each model in terms of the additional assumptions made on the error process. We describe constructions for each model in the subsequent sections.

Random errors (section 8.1). Assume there is a *known* distribution on the errors which occur in the data. For the Hamming metric, the most common distribution is the binary symmetric channel BSC_p: each bit of the input is flipped with probability p and left untouched with probability $1 - p$. We require that for any input w, $\mathsf{Rec}(W', \mathsf{SS}(w)) = w$ with probability at least $1 - \alpha$ over the coins of SS and over W' drawn applying the noise distribution to w.

In that case, one can correct an error rate up to Shannon's bound on noisy channel coding. This bound is tight. Unfortunately, the assumption of a known noise process is too strong for most applications: there is no reason to believe we understand the exact distribution on errors which occur in complex data such as biometrics.[12] However, it provides a useful baseline by which to measure results for other models.

[12]Since the assumption here plays a role only in correctness, it is still more reasonable than assuming that we know exact distributions on the data in proofs of *secrecy*. However, in both cases, we would like to enlarge the class of distributions for which we can provably satisfy the definition of security.

Input-dependent errors (section 8.2). The errors are adversarial, subject only to the conditions that (a) the error magnitude $\mathsf{dis}(w, w')$ is bounded to a maximum of t, and (b) the corrupted word *depends only on the input* w, and not on the secure sketch $\mathsf{SS}(w)$. Here we require that for any pair w, w' at distance at most t, we have $\mathsf{Rec}(w', \mathsf{SS}(w)) = w$ with probability at least $1 - \alpha$ over the coins of SS.

This model encompasses any complex noise process which has been observed to never introduce more than t errors. Unlike the assumption of a particular distribution on the noise, the bound on magnitude can be checked experimentally. Perhaps surprisingly, in this model we can tolerate just as large an error rate as in the model of random errors. That is, we can tolerate an error rate up to Shannon's coding bound and no more.

Computationally bounded errors (section 8.3). The errors are adversarial and may depend on both w and the publicly stored information $\mathsf{SS}(w)$. However, we assume that the errors are introduced by a process of bounded computational power. That is, there is a probabilistic circuit of polynomial size (in the length n) which computes w' from w. The adversary cannot, for example, forge a digital signature and base the error pattern on the signature.

It is not clear whether this model allows correcting errors up to the Shannon bound, as in the two models above. The question is related to open questions on the construction of efficiently list-decodable codes. However, when the error rate is either very high or very low, then the appropriate list-decodable codes exist, and we can indeed match the Shannon bound.

Analogues for noisy channels and the Hamming metric. Models analogous to those above have been studied in the literature on codes for noisy binary channels (with the Hamming metric). Random errors and computationally bounded errors both make obvious sense in the coding context [64, 48]. The second model—input-dependent errors—does not immediately make sense in a coding situation, since there is no data other than the transmitted codeword on which errors could depend. Nonetheless, there is a natural, analogous model for noisy channels: one can allow the sender and receiver to share either (1) common, secret random coins (see [22, 43] and references therein) or (2) a side channel with which they can communicate a small number of noise-free, secret bits [33].

Existing results on these three models for the Hamming metric can be transported to our context using the code-offset construction:

$$\mathsf{SS}(w; x) = w \oplus C(x)\,.$$

Roughly, any code which corrects errors in the models above will lead to a secure sketch (respectively, fuzzy extractor) which corrects errors in the model. We explore the consequences for each of the three models in the next sections.

8.1. Random errors. The random error model was famously considered by Shannon [64]. He showed that for any discrete, memoryless channel, the rate at which information can be reliably transmitted is characterized by the maximum mutual information between the inputs and outputs of the channel. For the binary symmetric channel with crossover probability p, this means that there exist codes encoding k bits into n bits, tolerating error probability p in each bit if and only if

$$\frac{k}{n} < 1 - h(p) - \delta(n),$$

where $h(p) = -p \log p - (1-p)\log(1-p)$ and $\delta(n) = o(1)$. Computationally efficient codes achieving this bound were found later, most notably by Forney [29]. We can use

the code-offset construction $\mathsf{SS}(w; x) = w \oplus C(x)$ with an appropriate concatenated code [29] or, equivalently, $\mathsf{SS}(w) = \mathsf{syn}_C(w)$ since the codes can be linear. We obtain the following proposition.

PROPOSITION 8.1. *For any error rate $0 < p < 1/2$ and constant $\delta > 0$, for large enough n there exist secure sketches with entropy loss $(h(p) + \delta)n$ which correct the error rate of p in the data with high probability (roughly $2^{-c_\delta n}$ for a constant $c_\delta > 0$).*

The probability here is taken over the errors *only (the distribution on input strings w can be arbitrary).*

The quantity $h(p)$ is less than 1 for any p in the range $(0, 1/2)$. In particular, one can get nontrivial secure sketches even for a very high error rate p as long as it is less than $1/2$; in contrast, no secure sketch which corrects errors with probability 1 can tolerate $t \geq n/4$. Note that several other works on biometric cryptosystems consider the model of randomized errors and obtain similar results, though the analyses assume that the distribution on inputs is uniform [71, 17].

A matching impossibility result. The bound above is tight. The matching impossibility result also applies to input-dependent and computationally bounded errors, since random errors are a special case of both more complex models.

We start with an intuitive argument: If a secure sketch allows recovering from random errors with high probability, then it must contain enough information about w to describe the error pattern (since given w' and $\mathsf{SS}(w)$, one can recover the error pattern with high probability). Describing the outcome of n independent coin flips with probability p of heads requires $nh(p)$ bits, and so the sketch must reveal $nh(p)$ bits about w.

In fact, that argument simply shows that $nh(p)$ bits of Shannon information are leaked about w, whereas we are concerned with min-entropy loss as defined in section 3. To make the argument more formal, let W be uniform over $\{0,1\}^n$ and observe that with high probability over the output of the sketching algorithm, $v = \mathsf{SS}(w)$, the conditional distribution $W_v = W|_{\mathsf{SS}(W)=v}$ forms a good code for the binary symmetric channel. That is, for most values v, if we sample a random string w from $W|_{\mathsf{SS}(W)=v}$ and send it through a binary symmetric channel, we will be able to recover the correct value w. That means there exists some v such that (a) W_v is a good code and (b) $\mathbf{H}_\infty(W_v)$ is close to $\tilde{\mathbf{H}}_\infty(W|\mathsf{SS}(W))$. Shannon's noisy coding theorem says that such a code can have entropy at most $n(1 - h(p) + o(1))$. Thus the construction above is optimal.

PROPOSITION 8.2. *For any error rate $0 < p < 1/2$, any secure sketch SS which corrects random errors (with rate p) with probability at least $2/3$ has entropy loss at least $n(h(p) - o(1))$; that is, $\tilde{\mathbf{H}}_\infty(W|\mathsf{SS}(W)) \leq n(1 - h(p) - o(1))$ when W is drawn uniformly from $\{0,1\}^n$.*

8.2. Randomizing input-dependent errors. Assuming errors distributed randomly according to a known distribution seems very limiting. In the Hamming metric, one can construct a secure sketch which achieves the same result as with random errors for every error process where the magnitude of the error is bounded, as long as the errors are independent of the output of $\mathsf{SS}(W)$. The same technique was used previously by Bennett et al. [4, p. 216] and, in a slightly different context, Lipton [46, 22].

The idea is to choose a random permutation $\pi : [n] \to [n]$, permute the bits of w before applying the sketch, and store the permutation π along with $\mathsf{SS}(\pi(w))$. Specifically, let C be a linear code tolerating a p fraction of random errors with

redundancy $n - k \approx nh(p)$. Let

$$\mathsf{SS}(w;\pi) = (\pi,\ \mathsf{syn}_C(\pi(w)))\,,$$

where $\pi : [n] \to [n]$ is a random permutation and, for $w = w_1 \cdots w_n \in \{0,1\}^n$, $\pi(w)$ denotes the permuted string $w_{\pi(1)}w_{\pi(2)}\cdots w_{\pi(n)}$. The recovery algorithm operates in the obvious way: it first permutes the input w' according to π and then runs the usual syndrome recovery algorithm to recover $\pi(w)$.

For any particular pair w, w', the difference $w \oplus w'$ will be mapped to a random vector of the same weight by π, and any code for the binary symmetric channel (with rate $p \approx t/n$) will correct such an error with high probability.

Thus we can construct a sketch with entropy loss $n(h(t/n) - o(1))$ which corrects any t flipped bits with high probability. This is optimal by the lower bound for random errors (Proposition 8.2), since a sketch for data-dependent errors will also correct random errors. It is also possible to reduce the amount of randomness, so that the *size* of the sketch meets the same optimal bound [67].

An alternative approach to input-dependent errors is discussed in the last paragraph of section 8.3.

8.3. Handling computationally bounded errors via list decoding. As mentioned above, many results on noisy coding for other error models in Hamming space extend to secure sketches. The previous sections discussed random, and randomized, errors. In this section, we discuss constructions [33, 43, 48] which transform a *list-decodable* code, defined below, into uniquely decodable codes for a particular error model. These transformations can also be used in the setting of secure sketches, leading to better tolerance of computationally bounded errors. For some ranges of parameters, this yields optimal sketches, that is, sketches which meet the Shannon bound on the fraction of tolerated errors.

List-decodable codes. A code C in a metric space $\mathcal{M}$ is called *list-decodable* with list size L and distance t if for every point $x \in \mathcal{M}$, there are at most L codewords within distance t of $\mathcal{M}$. A list-decoding algorithm takes as input a word x and returns the corresponding list $c_1, c_2, \ldots$ of codewords. The most interesting setting is when L is a small polynomial (in the description size $\log |\mathcal{M}|$), and there exists an efficient list-decoding algorithm. It is then feasible for an algorithm to go over each word in the list and accept if it has some desirable property. There are many examples of such codes for the Hamming space; for a survey see Guruswami's thesis [32]. Recently there has been significant progress in constructing list-decodable codes for large alphabets, e.g., [58, 34].

Similarly, we can define a *list-decodable secure sketch* with size L and distance t as follows: for any pair of words $w, w' \in \mathcal{M}$ at distance at most t, the algorithm $\mathsf{Rec}(w', \mathsf{SS}(w))$ returns a list of at most L points in $\mathcal{M}$; if $\mathsf{dis}(w, w') \le t$, then one of the words in the list must be w itself. The simplest way to obtain a list-decodable secure sketch is to use the code-offset construction of section 5 with a list-decodable code for the Hamming space. One obtains a different example by running the improved Juels–Sudan scheme for set difference (Construction 5), replacing ordinary decoding of Reed–Solomon codes with list decoding. This yields a significant improvement in the number of errors tolerated at the price of returning a list of possible candidates for the original secret.

Sieving the list. Given a list-decodable secure sketch SS, all that is needed is to store some additional information which allows the receiver to disambiguate w from the list. Let us suggestively name the additional information $Tag(w;R)$, where R

is some additional randomness (perhaps a key). Given a list-decodable code C, the sketch will typically look like

$$\mathsf{SS}(w; x) = (\ w \oplus C(x),\ Tag(w)\ ).$$

On inputs w' and (Δ, tag), the recovery algorithm consists of running the list-decoding algorithm on $w' \oplus \Delta$ to obtain a list of possible codewords $C(x_1), \ldots, C(x_L)$. There is a corresponding list of candidate inputs $w_1, \ldots, w_L$, where $w_i = C(x_i) \oplus \Delta$, and the algorithm outputs the first w_i in the list such that $Tag(w_i) = tag$. We will choose the function $Tag()$ so that the adversary cannot arrange to have two values in the list with valid tags.

We consider two $Tag()$ functions, inspired by [33, 43, 48].

1. Recall that for computationally bounded errors, the corrupted string w' depends on *both* w and $\mathsf{SS}(w)$, but w' is computed by a probabilistic circuit of size polynomial in n.

Consider $Tag(w) = \mathsf{hash}(w)$, where hash is drawn from a collision-resistant function family. More specifically, we will use some extra randomness r to choose a key *key* for a collision-resistant hash family. The output of the sketch is then

$$\mathsf{SS}(w; x, r) = (\ w \oplus C(x),\ key(r),\ \mathsf{hash}_{key(r)}(w)\ ).$$

If the list-decoding algorithm for the code C runs in polynomial time, then the adversary succeeds only if he can find a value $w_i \neq w$ such that $\mathsf{hash}_{key}(w_i) = \mathsf{hash}_{key}(w)$, that is, only by finding a collision for the hash function. By assumption, a polynomially bounded adversary succeeds only with negligible probability.

The additional entropy loss, beyond that of the code-offset part of the sketch, is bounded above by the output length of the hash function. If α is the desired bound on the adversary's success probability, then for standard assumptions on hash functions this loss will be polynomial in $\log(1/\alpha)$.

In principle this transformation can yield sketches which achieve the optimal entropy loss $n(h(t/n) - o(1))$, since codes with polynomial list size L are known to exist for error rates approaching the Shannon bound. However, in order to use the construction the code must also be equipped with a reasonably efficient algorithm for finding such a list. This is necessary both so that recovery will be efficient and, more subtly, for the proof of security to go through (that way we can assume that the polynomial time adversary knows the list of words generated during the recovery procedure). We do not know of *efficient* (i.e., polynomial time constructible and decodable) binary list-decodable codes which meet the Shannon bound for all choices of parameters. However, when the error rate is near $\frac{1}{2}$ such codes are known [35]. Thus, this type of construction yields essentially optimal sketches when the error rate is near 1/2. This is quite similar to analogous results on channel coding [48]. Relatively little is known about the performance of efficiently list-decodable codes in other parameter ranges for binary alphabets [32].

2. A similar, even simpler, transformation can be used in the setting of input-dependent errors (i.e., when the errors depend only on the input and not on the sketch, but the adversary is not assumed to be computationally bounded). One can store $Tag(w) = (I, h_I(w))$, where $\{h_i\}_{i \in \mathcal{I}}$ comes from a universal hash family mapping from $\mathcal{M}$ to $\{0,1\}^\ell$, where $\ell = \log\left(\frac{1}{\alpha}\right) + \log L$, and α is the probability of an incorrect decoding.

The proof is simple: the values $w_1, \ldots, w_L$ do not depend on I, and so for any value $w_i \neq w$, the probability that $h_I(w_i) = h_I(w)$ is $2^{-\ell}$. There are at most L

possible candidates, and so the probability that any one of the elements in the list is accepted is at most $L \cdot 2^{-\ell} = \alpha$. The additional entropy loss incurred is at most $\ell = \log\left(\frac{1}{\alpha}\right) + \log(L)$.

In principle, this transformation can do as well as the randomization approach of the previous section. However, we do not know of efficient binary list-decodable codes meeting the Shannon bound for most parameter ranges. Thus, in general, randomizing the errors (as in the previous section) works better in the input-dependent setting.

9. Secure sketches and efficient information reconciliation. Suppose Alice holds a set w and Bob holds a set w' that are close to each other. They wish to reconcile the sets, i.e., to discover the symmetric difference $w \triangle w'$ so that they can take whatever appropriate (application-dependent) action to make their two sets agree. Moreover, they wish to do this communication-efficiently, without having to transmit entire sets to each other. This problem is known as set reconciliation and naturally arises in various settings.

Let $(\mathsf{SS}, \mathsf{Rec})$ be a secure sketch for set difference that can handle distance up to t; furthermore, suppose that $|w \triangle w'| \leq t$. Then if Bob receives $s = \mathsf{SS}(w)$ from Alice, he will be able to recover w, and therefore $w \triangle w'$, from s and w'. Similarly, Alice will be able find $w \triangle w'$ upon receiving $s' = \mathsf{SS}(w')$ from Bob. This will be communication-efficient if $|s|$ is small. Note that our secure sketches for set difference of sections 6.2 and 6.3 are indeed short—in fact, they are secure precisely because they are short. Thus, they also make good set reconciliation schemes.

Conversely, a good (single-message) set reconciliation scheme makes a good secure sketch: simply make the message the sketch. The entropy loss will be at most the length of the message, which is short in a communication-efficient scheme. Thus, the set reconciliation scheme CPISync of [51] makes a good secure sketch. In fact, it is quite similar to the secure sketch of section 6.2, except instead of the top t coefficients of the characteristic polynomial it uses the values of the polynomial at t points.

PinSketch of section 6.3, when used for set reconciliation, achieves the same parameters as CPISync of [51], except decoding is faster, because instead of spending t^3 time to solve a system of linear equations, it spends t^2 time for Euclid's algorithm. Thus, it can be substituted wherever CPISync is used, such as PDA synchronization [68] and PGP key server updates [49]. Furthermore, optimizations that improve computational complexity of CPISync through the use of interaction [50] can also be applied to PinSketch.

Of course, secure sketches for other metrics are similarly related to information reconciliation for those metrics. In particular, ideas for edit distance very similar to ours were independently considered in the context of information reconciliation by [15].

Appendix A. Proof of Lemma 2.2. Recall that Lemma 2.2 considered random variables A, B, C and consisted of two parts, which we prove one after the other.

Part (a) stated that for any $\delta > 0$, the conditional entropy $\mathbf{H}_\infty(A|B = b)$ is at least $\tilde{\mathbf{H}}_\infty(A|B) - \log(1/\delta)$ with probability at least $1 - \delta$ (the probability here is taken over the choice of b). Let $p = 2^{-\tilde{\mathbf{H}}_\infty(A|B)} = \mathbb{E}_b\left[2^{-\mathbf{H}_\infty(A|B=b)}\right]$. By the Markov inequality, $2^{-\mathbf{H}_\infty(A|B=b)} \leq p/\delta$ with probability at least $1 - \delta$. Taking logarithms, part (a) follows.

Part (b) stated that if B has at most 2^λ possible values, then $\tilde{\mathbf{H}}_\infty(A \mid (B, C)) \geq \tilde{\mathbf{H}}_\infty((A, B) \mid C) - \lambda \geq \tilde{\mathbf{H}}_\infty(A \mid C) - \lambda$. In particular, $\tilde{\mathbf{H}}_\infty(A \mid B) \geq \mathbf{H}_\infty((A, B)) - \lambda \geq \mathbf{H}_\infty(A) - \lambda$. Clearly, it suffices to prove the first assertion (the second follows from

taking C to be constant). Moreover, the second inequality of the first assertion follows from the fact that $\Pr[A = a \wedge B = b \mid C = c] \le \Pr[A = a \mid C = c]$ for any c. Thus, we prove only that $\tilde{\mathbf{H}}_\infty(A \mid (B, C)) \ge \tilde{\mathbf{H}}_\infty((A, B) \mid C) - \lambda$:

$$\begin{aligned}
\tilde{\mathbf{H}}_\infty(A \mid (B,C)) &= -\log \mathop{\mathbb{E}}_{(b,c)\leftarrow(B,C)} \left[\max_a \Pr[A = a \mid B = b \wedge C = c]\right] \\
&= -\log \sum_{(b,c)} \max_a \Pr[A = a \mid B = b \wedge C = c] \Pr[B = b \wedge C = c] \\
&= -\log \sum_{(b,c)} \max_a \Pr[A = a \wedge B = b \mid C = c] \Pr[C = c] \\
&= -\log \sum_b \mathop{\mathbb{E}}_{c\leftarrow C} \left[\max_a \Pr[A = a \wedge B = b \mid C = c]\right] \\
&\ge -\log \sum_b \mathop{\mathbb{E}}_{c\leftarrow C} \left[\max_{a,b'} \Pr[A = a \wedge B = b' \mid C = c]\right] \\
&= -\log \sum_b 2^{-\tilde{\mathbf{H}}_\infty((A,B)|C)} \ge -\log 2^{\lambda} 2^{-\tilde{\mathbf{H}}_\infty((A,B)|C)} \\
&= \tilde{\mathbf{H}}_\infty((A,B) \mid C) - \lambda\,.
\end{aligned}$$

The first inequality in the above derivation holds since taking the maximum over all pairs (a, b') (instead of over pairs (a, b) where b is fixed) increases the terms of the sum and hence decreases the negative log of the sum.

Appendix B. On smooth variants of average min-entropy and the relationship to smooth Rényi entropy. Min-entropy is a rather fragile measure; a single high-probability element can ruin the min-entropy of an otherwise good distribution. This is often circumvented within proofs by considering a distribution which is close to the distribution of interest, but which has higher entropy. Renner and Wolf [60] systematized this approach with the notion of ϵ-*smooth* min-entropy (they use the term "Rényi entropy of order ∞" instead of "min-entropy"), which considers all distributions that are ϵ-close:

$$\mathbf{H}^\epsilon_\infty(A) = \max_{B:\ \mathbf{SD}(A,B)\le\epsilon} \mathbf{H}_\infty(B)\,.$$

Smooth min-entropy very closely relates to the amount of extractable nearly uniform randomness: if one can map A to a distribution that is ϵ-close to U_m, then $\mathbf{H}^\epsilon_\infty(A) \ge m$; conversely, from any A such that $\mathbf{H}^\epsilon_\infty(A) \ge m$, and for any ϵ_2, one can extract $m - 2\log(\frac{1}{\epsilon_2})$ bits that are $\epsilon+\epsilon_2$-close to uniform (see [60] for a more precise statement; the proof of the first statement follows by considering the inverse map, and the proof of the second follows from the leftover hash lemma, which is discussed in more detail in Lemma 2.4). For some distributions, considering the smooth min-entropy will improve the number and quality of extractable random bits.

A smooth version of average min-entropy can also be considered, defined as

$$\tilde{\mathbf{H}}^\epsilon_\infty(A \mid B) = \max_{(C,D):\ \mathbf{SD}((A,B),(C,D))\le\epsilon} \tilde{\mathbf{H}}_\infty(C \mid D)\,.$$

It similarly relates very closely to the number of extractable bits that look nearly uniform to the adversary who knows the value of B and is therefore perhaps a better measure for the quality of a secure sketch that is used to obtain a fuzzy extractor.

All our results can be cast in terms of smooth entropies throughout, with appropriate modifications (if input entropy is ϵ-smooth, then output entropy will also be ϵ-smooth, and extracted random strings will be ϵ further away from uniform). We avoid doing so for simplicity of exposition. However, for some input distributions, particularly ones with few elements of relatively high probability, this will improve the result by giving more secure sketches or longer-output fuzzy extractors.

Finally, a word is in order on the relation of average min-entropy to conditional min-entropy, introduced by Renner and Wolf in [61], and defined as $\mathbf{H}_\infty(A \mid B) = -\log \max_{a,b} \Pr(A = a \mid B = b) = \min_b \mathbf{H}_\infty(A \mid B = b)$ (an ϵ-smooth version is defined analogously by considering all distributions (C, D) that are within ϵ of (A, B) and taking the maximum among them). This definition is too strict: it takes the worst-case b, while for randomness extraction (and many other settings, such as predictability by an adversary), average-case b suffices. Average min-entropy leads to more extractable bits. Nevertheless, after smoothing, the two notions are equivalent up to an additive $\log(\frac{1}{\epsilon})$ term: $\tilde{\mathbf{H}}^\epsilon_\infty(A \mid B) \geq \mathbf{H}^\epsilon_\infty(A \mid B)$ and $\mathbf{H}_\infty{}^{\epsilon+\epsilon_2}(A \mid B) \geq \tilde{\mathbf{H}}^\epsilon_\infty(A \mid B) - \log(\frac{1}{\epsilon_2})$ (for the case of $\epsilon = 0$, this follows by constructing a new distribution that eliminates all b for which $\mathbf{H}_\infty(A \mid B = b) < \tilde{\mathbf{H}}_\infty(A \mid B) - \log(\frac{1}{\epsilon_2})$, which will be within ϵ_2 of the (A, B) by Markov's inequality; for $\epsilon > 0$, an analogous proof works). Note that by Lemma 2.2(b), this implies a simple chain rule for $\mathbf{H}^\epsilon_\infty$ (a more general one is given in [61, section 2.4]): $\mathbf{H}_\infty{}^{\epsilon+\epsilon_2}(A \mid B) \geq \tilde{\mathbf{H}}^\epsilon_\infty((A, B)) - H_0(B) - \log(\frac{1}{\epsilon_2})$, where $H_0(B)$ is the logarithm of the number of possible values of B.

Appendix C. Lower bounds from coding. Recall that an $(\mathcal{M}, K, t)$-code is a subset of the metric space $\mathcal{M}$ which can *correct* t errors (this is slightly different from the usual notation of coding theory literature).

Let $K(\mathcal{M}, t)$ be the largest K for which there exists an $(\mathcal{M}, K, t)$-code. Given any set S of 2^m points in $\mathcal{M}$, we let $K(\mathcal{M}, t, S)$ be the largest K such that there exists an $(\mathcal{M}, K, t)$-code all of whose K points belong to S. Finally, we let $L(\mathcal{M}, t, m) = \log(\min_{|S|=2^m} K(n, t, S))$. Of course, when $m = \log|\mathcal{M}|$, we get that $L(\mathcal{M}, t, n) = \log K(\mathcal{M}, t)$. The exact determination of quantities $K(\mathcal{M}, t)$ and $K(\mathcal{M}, t, S)$ is a central problem of coding theory and is typically very hard. To the best of our knowledge, the quantity $L(\mathcal{M}, t, m)$ was not explicitly studied in any of three metrics that we study, and its exact determination seems hard as well.

We give two simple lower bounds on the entropy loss (one for secure sketches and the other for fuzzy extractors) which show that our constructions for the Hamming and set difference metrics output as much entropy m' as possible when the original input distribution is uniform. In particular, because the constructions have the same entropy loss regardless of m, they are optimal in terms of the entropy loss $m - m'$. We conjecture that the constructions also have the highest possible value m' for all values of m, but we do not have a good enough understanding of $L(\mathcal{M}, t, m)$ (where $\mathcal{M}$ is the Hamming metric) to substantiate the conjecture.

LEMMA C.1. *The existence of an $(\mathcal{M}, m, m', t)$-secure sketch implies that $m' \leq L(\mathcal{M}, t, m)$. In particular, when $m = \log|\mathcal{M}|$ (i.e., when the password is truly uniform), $m' \leq \log K(\mathcal{M}, t)$.*

Proof. Assume that SS is such a secure sketch. Let S be any set of size 2^m in $\mathcal{M}$, and let W be uniform over S. Then we must have $\tilde{\mathbf{H}}_\infty(W \mid \mathsf{SS}(W)) \geq m'$. In particular, there must be some value v such that $\mathbf{H}_\infty(W \mid \mathsf{SS}(W) = v) \geq m'$. But this means that conditioned on $\mathsf{SS}(W) = v$, there are at least $2^{m'}$ points w in S (call this set T) which could produce $\mathsf{SS}(W) = v$. We claim that these $2^{m'}$ values of w form a code of error-correcting distance t. Indeed, otherwise there would be a point $w' \in \mathcal{M}$

such that $\mathsf{dis}(w_0, w') \le t$ and $\mathsf{dis}(w_1, w') \le t$ for some $w_0, w_1 \in T$. But then we must have that $\mathsf{Rec}(w', v)$ is equal to both w_0 and w_1, which is impossible. Thus, the set T above must form an $(\mathcal{M}, 2^{m'}, t)$-code inside S, which means that $m' \le \log K(\mathcal{M}, t, S)$. Since S was arbitrary, the bound follows. □

LEMMA C.2. *The existence of $(\mathcal{M}, m, \ell, t, \epsilon)$-fuzzy extractors implies that $\ell \le L(\mathcal{M}, t, m) - \log(1-\epsilon)$. In particular, when $m = \log|\mathcal{M}|$ (i.e., when the password is truly uniform), $\ell \le \log K(\mathcal{M}, t) - \log(1-\epsilon)$.*

Proof. Assume that $(\mathsf{Gen}, \mathsf{Rep})$ is such a fuzzy extractor. Let S be any set of size 2^m in $\mathcal{M}$, let W be uniform over S, and let $(R, P) \leftarrow \mathsf{Gen}(W)$. Then we must have that $\mathbf{SD}((R, P), (U_\ell, P)) \le \epsilon$. In particular, there must be some value p of P such that R is ϵ-close to U_ℓ conditioned on $P = p$. In particular, this means that conditioned on $P = p$, there are at least $(1-\epsilon)2^\ell$ points $r \in \{0,1\}^\ell$ (call this set T) which could be extracted with $P = p$. Now, map every $r \in T$ to some arbitrary $w \in S$ which could have produced r with nonzero probability given $P = p$, and call this map C. C must define a code with error-correcting distance t by the same reasoning as in Lemma C.1. □

Observe that, as long as $\epsilon < 1/2$, we have $0 < -\log(1-\epsilon) < 1$, so the lower bounds on secure sketches and fuzzy extractors differ by less than a bit.

Appendix D. Analysis of the original Juels–Sudan construction. In this section we present a new analysis for the Juels–Sudan secure sketch for set difference. We will assume that $n = |\mathcal{U}|$ is a prime power and work over the field $\mathcal{F} = GF(n)$. On input set w, the original Juels–Sudan sketch is a list of r pairs of points (x_i, y_i) in $\mathcal{F}$, for some parameter r, $s < r \le n$. It is computed as follows.

Construction 10 (original Juels–Sudan secure sketch [38]). Input: a set $w \subseteq \mathcal{F}$ of size s and parameters $r \in \{s+1, \dots, n\}$, $t \in \{1, \dots, s\}$.

1. Choose $p()$ at random from the set of polynomials of degree at most $k = s - t - 1$ over $\mathcal{F}$.

Write $w = \{x_1, \dots, x_s\}$, and let $y_i = p(x_i)$ for $i = 1, \dots, s$.

2. Choose $r - s$ distinct points $x_{s+1}, \dots, x_r$ at random from $\mathcal{F} - w$.
3. For $i = s+1, \dots, r$, choose $y_i \in \mathcal{F}$ at random such that $y_i \ne p(x_i)$.
4. Output $\mathsf{SS}(w) = \{(x_1, y_1), \dots, (x_r, y_r)\}$ (in lexicographic order of x_i).

The parameter t measures the error-tolerance of the scheme: given $\mathsf{SS}(w)$ and a set w' such that $w \triangle w' \le t$, one can recover w by considering the pairs (x_i, y_i) for $x_i \in w'$ and running Reed–Solomon decoding to recover the low-degree polynomial $p(\cdot)$. When the parameter r is very small, the scheme corrects approximately twice as many errors with good probability (in the "input-dependent" sense from section 8). When r is low, however, we show here that the bound on the entropy loss becomes very weak.

The parameter r dictates the amount of storage necessary, one on hand, and also the security of the scheme (that is, for $r = s$ the scheme leaks all information and for larger and larger r there is less information about w). Juels and Sudan actually propose two analyses for the scheme. First, they analyze the case where the secret w is distributed uniformly over all subsets of size s. Second, they provide an analysis of a nonuniform password distribution, but only for the case $r = n$ (that is, their analysis applies only in the small universe setting, where $\Omega(n)$ storage is acceptable). Here we give a simpler analysis which handles nonuniformity and any $r \le n$. We get the same results for a broader set of parameters.

LEMMA D.1. *The entropy loss of the Juels–Sudan scheme is at most $t \log n + \log\binom{n}{r} - \log\binom{n-s}{r-s} + 2$.*

Proof. This is a simple application of Lemma 2.2(b). $\mathbf{H}_\infty((W, \mathsf{SS}(W)))$ can be computed as follows. Choosing the polynomial p (which can be uniquely recovered from w and $\mathsf{SS}(w)$) requires $s - t$ random choices from $\mathcal{F}$. The choice of the remaining x_i's requires $\log \binom{n-s}{r-s}$ bits, and choosing the $y_i's$ requires $r - s$ random choices from $\mathcal{F} - \{p(x_i)\}$. Thus, $\mathbf{H}_\infty((W, \mathsf{SS}(W))) = \mathbf{H}_\infty(W) + (s-t)\log n + \log \binom{n-s}{r-s} + (r-s)\log(n-1)$. The output can be described in $\log \left(\binom{n}{r} n^r\right)$ bits. The result follows by Lemma 2.2(b) after observing that $(r-s)\log \frac{n}{n-1} < n \log \frac{n}{n-1} \le 2$. □

In the large universe setting, we will have $r \ll n$ (since we wish to have storage polynomial in s). In that setting, the bound on the entropy loss of the Juels–Sudan scheme is in fact very large. We can rewrite the entropy loss as $t \log n - \log \binom{r}{s} + \log \binom{n}{s} + 2$, using the identity $\binom{n}{r}\binom{r}{s} = \binom{n}{s}\binom{n-s}{r-s}$. Now the entropy of W is at most $\binom{n}{s}$, and so our lower bound on the remaining entropy is $(\log \binom{r}{s} - t \log n - 2)$. To make this quantity large requires making r very large.

Appendix E. BCH syndrome decoding in sublinear time. We show that the standard decoding algorithm for BCH codes can be modified to run in time polynomial in the length of the syndrome. This works for BCH codes over any field $GF(q)$, which include Hamming codes in the binary case and Reed–Solomon for the case $n = q - 1$. BCH codes are handled in detail in many textbooks (e.g., [72]); our presentation here is quite concise. For simplicity, we discuss only primitive, narrow-sense BCH codes here; the discussion extends easily to the general case.

The algorithm discussed here has been revised due to an error pointed out by Ari Trachtenberg. Its implementation is available [36].

We will use a slightly nonstandard formulation of BCH codes. Let $n = q^m - 1$ (in the binary case of interest in section 6.3, $q = 2$). We will work in two finite fields: $GF(q)$ and a larger extension field $\mathcal{F} = GF(q^m)$. BCH codewords, formally defined below, are then vectors in $GF(q)^n$. In most common presentations, one indexes the n positions of these vectors by discrete logarithms of the elements of $\mathcal{F}^*$: position i, for $1 \le i \le n$, corresponds to α^i, where α generates the multiplicative group $\mathcal{F}^*$. However, there is no inherent reason to do so: they can be indexed by elements of $\mathcal{F}$ directly rather than by their discrete logarithms. Thus, we say that a word has value p_x at position x, where $x \in \mathcal{F}^*$. If one ever needs to write down the entire n-character word in an ordered fashion, one can arbitrarily choose a convenient ordering of the elements of $\mathcal{F}$ (e.g., by using some standard binary representation of field elements); for our purposes this is not necessary, as we do not store entire n-bit words explicitly, but rather represent them by their supports: $\mathrm{supp}(v) = \{(x, p_x) \mid p_x \neq 0\}$. Note that for the binary case of interest in section 6.3, we can define $\mathrm{supp}(v) = \{x \mid p_x \neq 0\}$, because p_x can take only two values: 0 or 1.

Our choice of representation will be crucial for efficient decoding: in the more common representation, the last step of the decoding algorithm requires one to find the position i of the error from the field element α^i. However, no efficient algorithms for computing the discrete logarithm are known if q^m is large (indeed, a lot of cryptography is based on the assumption that such an efficient algorithm does not exist). In our representation, the field element α^i will in fact be the position of the error.

Definition 8. *The (narrow-sense, primitive) BCH code of designed distance δ over $GF(q)$ (of length $n \ge \delta$) is given by the set of vectors of the form $(c_x)_{x \in \mathcal{F}^*}$ such that each c_x is in the smaller field $GF(q)$, and the vector satisfies the constraints $\sum_{x \in \mathcal{F}^*} c_x x^i = 0$ for $i = 1, \ldots, \delta - 1$, with arithmetic done in the larger field $\mathcal{F}$.*

To explain this definition, let us fix a generator α of the multiplicative group of

the large field $\mathcal{F}^*$. For any vector of coefficients $\left(c_x\right)_{x\in\mathcal{F}^*}$, we can define a polynomial

$$c(z) = \sum_{x\in GF(q^m)^*} c_x z^{\mathsf{dlog}(x)},$$

where $\mathsf{dlog}(x)$ is the discrete logarithm of x with respect to α. The conditions of the definition are then equivalent to the requirement (more commonly seen in presentations of BCH codes) that $c(\alpha^i) = 0$ for $i = 1, \ldots, \delta - 1$, because $(\alpha^i)^{\mathsf{dlog}(x)} = (\alpha^{\mathsf{dlog}(x)})^i = x^i$.

We can simplify this somewhat. Because the coefficients c_x are in $GF(q)$, they satisfy $c_x^q = c_x$. Using the identity $(x + y)^q = x^q + y^q$, which holds even in the large field $\mathcal{F}$, we have $c(\alpha^i)^q = \sum_{x\neq 0} c_x^q x^{iq} = c(\alpha^{iq})$. Thus, roughly a $1/q$ fraction of the conditions in the definition are redundant: we need only to check that they hold for $i \in \{1, \ldots, \delta - 1\}$ such that $q \not| i$.

The syndrome of a word (not necessarily a codeword) $(p_x)_{x\in\mathcal{F}^*} \in GF(q)^n$ with respect to the BCH code above is the vector

$$\mathsf{syn}(p) = p(\alpha^1), \ldots, p(\alpha^{\delta-1}), \quad \text{where} \quad p(\alpha^i) = \sum_{x\in\mathcal{F}^*} p_x x^i.$$

As mentioned above, we do not in fact have to include the values $p(\alpha^i)$ such that $q|i$.

Computing with low-weight words. A low-weight word $p \in GF(q)^n$ can be represented either as a long string or, more compactly, as a list of positions where it is nonzero and its values at those points. We call this representation the support list of p and denote it $\mathrm{supp}(p) = \{(x, p_x)\}_{x:p_x\neq 0}$.

LEMMA E.1. *For a q-ary BCH code C of designed distance δ, one can compute*

(1) $\mathsf{syn}(p)$ *from* $\mathrm{supp}(p)$ *in time polynomial in* δ, $\log n$, *and* $|\mathrm{supp}(p)|$, *and*

(2) $\mathrm{supp}(p)$ *from* $\mathsf{syn}(p)$ *(when p has weight at most $(\delta-1)/2$) in time polynomial in δ and* $\log n$.

Proof. Recall that $\mathsf{syn}(p) = (p(\alpha), \ldots, p(\alpha^{\delta-1}))$, where $p(\alpha^i) = \sum_{x\neq 0} p_x x^i$. Part (1) is easy, since to compute the syndrome we need only to compute the powers of x. This requires about $\delta \cdot \mathsf{weight}(p)$ multiplications in $\mathcal{F}$. For part (2), we adapt Berlekamp's BCH decoding algorithm, based on its presentation in [72]. Let $M = \{x \in \mathcal{F}^* | p_x \neq 0\}$, and define

$$\sigma(z) \stackrel{\text{def}}{=} \prod_{x\in M} (1 - xz) \quad \text{and} \quad \omega(z) \stackrel{\text{def}}{=} \sigma(z) \sum_{x\in M} \frac{p_x xz}{(1 - xz)}.$$

Since $(1 - xz)$ divides $\sigma(z)$ for $x \in M$, we see that $\omega(z)$ is in fact a polynomial of degree at most $|M| = \mathsf{weight}(p) \leq (\delta - 1)/2$. The polynomials $\sigma(z)$ and $\omega(z)$ are known as the error locator polynomial and evaluator polynomial, respectively; observe that $\gcd(\sigma(z), \omega(z)) = 1$.

We will in fact work with our polynomials modulo z^δ. In this arithmetic the inverse of $(1 - xz)$ is $\sum_{\ell=1}^{\delta} (xz)^{\ell-1}$; that is,

$$(1 - xz) \sum_{\ell=1}^{\delta} (xz)^{\ell-1} \equiv 1 \mod z^\delta.$$

We are given $p(\alpha^\ell)$ for $\ell = 1, \ldots, \delta$. Let $S(z) = \sum_{\ell=1}^{\delta-1} p(\alpha^\ell) z^\ell$. Note that $S(z) \equiv \sum_{x\in M} p_x \frac{xz}{(1-xz)} \mod z^\delta$. This implies that

$$S(z)\sigma(z) \equiv \omega(z) \mod z^\delta.$$

The polynomials $\sigma(z)$ and $\omega(z)$ satisfy the following four conditions: they are of degree at most $(\delta-1)/2$ each, they are relatively prime, the constant coefficient of σ is 1, and they satisfy this congruence. In fact, let $w'(z), \sigma'(z)$ be any nonzero solution to this congruence, where degrees of $w'(z)$ and $\sigma'(z)$ are at most $(\delta-1)/2$. Then $w'(z)/\sigma'(z) = \omega(z)/\sigma(z)$. (To see why this is so, multiply the initial congruence by $\sigma'()$ to get $\omega(z)\sigma'(z) \equiv \sigma(z)\omega'(z) \mod z^{\delta}$. Since both sides of the congruence have degree at most $\delta-1$, they are in fact equal as polynomials.) Thus, there is at most one solution $\sigma(z), \omega(z)$ satisfying all four conditions, which can be obtained from any $\sigma'(z), \omega'(z)$ by reducing the resulting fraction $\omega'(z)/\sigma'(z)$ to obtain the solution of minimal degree with the constant term of σ equal to 1.

Finally, the roots of $\sigma(z)$ are the points x^{-1} for $x \in M$, and the exact value of p_x can be recovered from $\omega(x^{-1}) = p_x \prod_{y \in M, y \neq x}(1 - yx^{-1})$ (this is needed only for $q > 2$, because for $q = 2$, $p_x = 1$). Note that it is possible that a solution to the congruence will be found even if the input syndrome is not a syndrome of any p with $\mathsf{weight}(p) > (\delta-1)/2$ (it is also possible that a solution to the congruence will not be found at all, or that the resulting $\sigma(z)$ will not split into distinct nonzero roots). Such a solution will not give the correct p. Thus, if there is no guarantee that $\mathsf{weight}(p)$ is actually at most $(\delta-1)/2$, it is necessary to recompute $\mathsf{syn}(p)$ after finding the solution, in order to verify that p is indeed correct.

Representing coefficients of $\sigma'(z)$ and $\omega'(z)$ as unknowns, we see that solving the congruence requires only solving a system of δ linear equations (one for each degree of z, from 0 to $\delta-1$) involving $\delta+1$ variables over $\mathcal{F}$, which can be done in $O(\delta^3)$ operations in $\mathcal{F}$ using, e.g., Gaussian elimination. The reduction of the fraction $\omega'(z)/\sigma'(z)$ requires simply running Euclid's algorithm for finding the gcd of two polynomials of degree less than δ, which takes $O(\delta^2)$ operations in $\mathcal{F}$. Suppose the resulting σ has degree e. Then one can find the roots of σ as follows. First test that σ indeed has e distinct roots by testing that $\sigma(z) | z^{q^m} - z$ (this is a necessary and sufficient condition, because every element of $\mathcal{F}$ is a root of $z^{q^m} - z$ exactly once). This can be done by computing $(z^{q^m} \mod \sigma(z))$ and testing if it equals $z \bmod \sigma$; it takes m exponentiations of a polynomial to the power q, i.e., $O((m \log q)e^2)$ operations in $\mathcal{F}$. Then apply an equal-degree-factorization algorithm (e.g., as described in [66]), which also takes $O((m \log q)e^2)$ operations in $\mathcal{F}$. Finally, after taking inverses of the roots of $\mathcal{F}$ and finding p_x (which takes $O(e^2)$ operations in $\mathcal{F}$), recompute $\mathsf{syn}(p)$ to verify that it is equal to the input value.

Because $m \log q = \log(n+1)$ and $e \leq (\delta-1)/2$, the total running time is $O(\delta^3 + \delta^2 \log n)$ operations in $\mathcal{F}$; each operation in $\mathcal{F}$ can done in time $O(\log^2 n)$, or faster using advanced techniques.

One can improve this running time substantially. The error locator polynomial $\sigma()$ can be found in $O(\log \delta)$ convolutions (multiplications) of polynomials over $\mathcal{F}$ of degree $(\delta-1)/2$ each [7, section 11.7] by exploiting the special structure of the system of linear equations being solved. Each convolution can be performed asymptotically in time $O(\delta \log \delta \log\log \delta)$ (see, e.g., [74]), and the total time required to find σ gets reduced to $O(\delta \log^2 \delta \log\log \delta)$ operation in $\mathcal{F}$. This replaces the δ^3 term in the above running time.

While this is asymptotically very good, Euclidean-algorithm-based decoding [70], which runs in $O(\delta^2)$ operations in $\mathcal{F}$, will find $\sigma(z)$ faster for reasonable values of δ (certainly for $\delta < 1000$). The algorithm finds σ as follows:

```
set R_old(z) ← z^{δ-1}, R_cur(z) ← S(z)/z, V_old(z) ← 0, V_cur(z) ← 1.
while deg(R_cur(z)) ≥ (δ-1)/2:
     divide R_old(z) by R_cur(z) to get quotient q(z) and remainder
```

$R_{\text{new}}(z)$;
set $V_{\text{new}}(z) \leftarrow V_{\text{old}}(z) - q(z)V_{\text{cur}}(z)$;
set $R_{\text{old}}(z) \leftarrow R_{\text{cur}}(z)$, $R_{\text{cur}}(z) \leftarrow R_{\text{new}}(z)$, $V_{\text{old}}(z) \leftarrow V_{\text{cur}}(z)$, $V_{\text{cur}}(z) \leftarrow V_{\text{new}}(z)$.
set $c \leftarrow V_{\text{cur}}(0)$; set $\sigma(z) \leftarrow V_{\text{cur}}(z)/c$ and $\omega(z) \leftarrow z \cdot R_{\text{cur}}(z)/c$

In the above algorithm, if $c = 0$, then the correct $\sigma(z)$ does not exist, i.e., $\mathsf{weight}(p) > (\delta-1)/2$. The correctness of this algorithm can be seen by observing that the congruence $S(z)\sigma(z) \equiv \omega(z) \pmod{z^\delta}$ can have z factored out of it (because $S(z)$, $\omega(z)$, and z^δ are all divisible by z) and rewritten as $(S(z)/z)\sigma(z) + u(z)z^{\delta-1} = \omega(z)/z$ for some $u(z)$. The obtained σ is easily shown to be the correct one (if one exists at all) by applying [66, Theorem 18.7] (to use the notation of that theorem, set $n = z^{\delta-1}$, $y = S(z)/z$, $t^* = r^* = (\delta-1)/2$, $r' = \omega(z)/z$, $s' = u(z)$, $t' = \sigma(z)$).

The root finding of σ can also be sped up. Asymptotically, detecting if a polynomial over $\mathcal{F} = GF(q^m) = GF(n+1)$ of degree e has e distinct roots and finding these roots can be performed in time $O(e^{1.815}(\log n)^{0.407})$ operations in $\mathcal{F}$ using the algorithm of Kaltofen and Shoup [40], or in time $O(e^2 + (\log n)e \log e \log\log e)$ operations in $\mathcal{F}$ using the EDF algorithm of Cantor and Zassenhaus.[13] For reasonable values of e, the Cantor–Zassenhaus EDF algorithm with Karatsuba's multiplication algorithm [41] for polynomials will be faster, giving root-finding running time of $O(e^2 + e^{\log_2 3} \log n)$ operations in $\mathcal{F}$. Note that if the actual weight e of p is close to the maximum tolerated $(\delta-1)/2$, then finding the roots of σ will actually take longer than finding σ. □

A dual view of the algorithm. Readers may be more familiar with a different, evaluation-based formulation of BCH codes, in which codewords are generated as follows. Let $\mathcal{F}$ again be an extension of $GF(q)$, and let n be the length of the code (note that $|\mathcal{F}^*|$ is not necessarily equal to n in this formulation). Fix distinct $x_1, x_2, \ldots, x_n \in \mathcal{F}$. For every polynomial c over the large field $\mathcal{F}$ of degree at most $n - \delta$, the vector $(c(x_1), c(x_2), \ldots c(x_n))$ is a codeword if and only if every coordinate of the vector happens to be in the smaller field: $c(x_i) \in GF(q)$ for all i. In particular, when $\mathcal{F} = GF(q)$, then every polynomial leads to a codeword, thus giving Reed–Solomon codes.

The syndrome in this formulation can be computed as follows: given a vector $y = (y_1, y_2, \ldots, y_n)$ find the interpolating polynomial $P = p_{n-1}x^{n-1} + p_{n-2}x^{n-2} + \cdots + p_0$ over $\mathcal{F}$ of degree at most $n-1$ such that $P(x_i) = y_i$ for all i. The syndrome is then the negative top $\delta - 1$ coefficients of P: $\mathsf{syn}(y) = (-p_{n-1}, -p_{n-2}, \ldots, -p_{n-(\delta-1)})$. (It is easy to see that this is a syndrome: it is a linear function that is zero exactly on the codewords.)

When $n = |\mathcal{F}| - 1$, we can index the n-component vectors by elements of $\mathcal{F}^*$, writing codewords as $(c(x))_{x\in\mathcal{F}^*}$. In this case, the syndrome of $(y_x)_{x\in\mathcal{F}^*}$ defined as the negative top $\delta - 1$ coefficients of P such that for all $x \in F^*$, $P(x) = y_x$ is equal to the syndrome defined following Definition 8 as $\sum_{x\in\mathcal{F}} y_x x^i$ for $i = 1, 2, \ldots, \delta - 1$.[14] Thus, when $n = |\mathcal{F}| - 1$, the codewords obtained via the evaluation-based definition are *identical* to the codewords obtain via Definition 8, because codewords are simply

[13] See [66, section 21.3], and substitute the most efficient known polynomial arithmetic. For example, the procedures described in [74] take time $O(e \log e \log\log e)$ instead of time $O(e^2)$ to perform modular arithmetic operations with degree-e polynomials.

[14] This statement can be shown as follows: because both maps are linear, it is sufficient to prove that they agree on a vector $(y_x)_{x\in\mathcal{F}^*}$ such that $y_a = 1$ for some $a \in \mathcal{F}^*$ and $y_x = 0$ for $x \neq a$. For such a vector, $\sum_{x\in\mathcal{F}} y_x x^i = a^i$. On the other hand, the interpolating polynomial $P(x)$ such that $P(x) = y_x$ is $-ax^{n-1} - a^2x^{n-2} - \cdots - a^{n-1}x - 1$ (indeed, $P(a) = -n = 1$; furthermore, multiplying $P(x)$ by $x - a$ gives $a(x^n - 1)$, which is zero on all of $\mathcal{F}^*$; hence $P(x)$ is zero for every $x \neq a$).

elements with the zero syndrome, and the syndrome maps agree.

This is an example of a remarkable duality between evaluations of polynomials and their coefficients: the syndrome can be viewed either as the evaluation of a polynomial whose coefficients are given by the vector, or as the coefficients of the polynomial whose evaluations are given by a vector.

The syndrome decoding algorithm above has a natural interpretation in the evaluation-based view. Our presentation is an adaptation of Welch–Berlekamp decoding as presented in, e.g., [69, Chapter 10].

Suppose $n = |F| - 1$ and $x_1, \ldots, x_n$ are the nonzero elements of the field. Let $y = (y_1, y_2, \ldots, y_n)$ be a vector. We are given its syndrome $\mathsf{syn}(y) = (-p_{n-1}, -p_{n-2}, \ldots, -p_{n-(\delta-1)})$, where $p_{n-1}, \ldots, p_{n-(\delta-1)}$ are the top coefficients of the interpolating polynomial P. Knowing only $\mathsf{syn}(y)$, we need to find at most $(\delta-1)/2$ locations x_i such that correcting all the corresponding y_i will result in a codeword. Suppose that codeword is given by a degree-$(n-\delta)$ polynomial c. Note that c agrees with P on all but the error locations. Let $\rho(z)$ be the polynomial of degree at most $(\delta-1)/2$ whose roots are exactly the error locations. (Note that $\sigma(z)$ from the decoding algorithm above is the same $\rho(z)$ but with coefficients in reverse order, because the roots of σ are the inverses of the roots of ρ.) Then $\rho(z) \cdot P(z) = \rho(z) \cdot c(z)$ for $z = x_1, x_2, \ldots, x_n$. Since $x_1, \ldots, x_n$ are all the nonzero field elements, $\prod_{i=1}^{n}(z - x_i) = z^n - 1$. Thus,

$$\rho(z) \cdot c(z) \;=\; \rho(z) \cdot P(z) \bmod \prod_{i=1}^{n}(z - x_i) \;=\; \rho(z) \cdot P(z) \bmod (z^n - 1)\,.$$

If we write the left-hand side as $\alpha_{n-1}x^{n-1} + \alpha_{n-2}x^{n-2} + \cdots + \alpha_0$, then the above equation implies that $\alpha_{n-1} = \cdots = \alpha_{n-(\delta-1)/2} = 0$ (because the degree if $\rho(z) \cdot c(z)$ is at most $n - (\delta+1)/2$). Because $\alpha_{n-1}, \ldots, \alpha_{n-(\delta-1)/2}$ depend on the coefficients of ρ as well as on $p_{n-1}, \ldots, p_{n-(\delta-1)}$, but not on lower coefficients of P, we obtain a system of $(\delta-1)/2$ equations for $(\delta-1)/2$ unknown coefficients of ρ. A careful examination shows that it is essentially the same system we had for $\sigma(z)$ in the algorithm above. The lowest-degree solution to this system is indeed the correct ρ, by the same argument which was used to prove the correctness of σ in Lemma E.1. The roots of ρ are the error locations. For $q > 2$, the actual corrections that are needed at the error locations (in other words, the light vector corresponding to the given syndrome) can then be recovered by solving the linear system of equations implied by the value of the syndrome.

Acknowledgments. This work evolved over several years, and discussions with many people enriched our understanding of the material at hand. In roughly chronological order, we thank Piotr Indyk for discussions about embeddings and for his help in the proof of Lemma 7.3; Madhu Sudan for helpful discussions about the construction of [38] and the uses of error-correcting codes; Venkat Guruswami for enlightenment about list decoding; Pim Tuyls for pointing out relevant previous work; Chris Peikert for pointing out the model of computationally bounded adversaries from [48]; Ari Trachtenberg for finding an error in the preliminary version of Appendix E; Ronny Roth for discussions about efficient BCH decoding; Kevin Harmon and Soren Johnson for their implementation work; and Silvio Micali and anonymous referees for suggestions on presenting our results.

REFERENCES

[1] E. Agrell, A. Vardy, and K. Zeger, *Upper bounds for constant-weight codes*, IEEE Trans. Inform. Theory, 46 (2000), pp. 2373–2395.

[2] A. Andoni and R. Krauthgamer, *The computational hardness of estimating edit distance*, in IEEE Symposium on Foundations of Computer Science (FOCS), Providence, RI, 2007, pp. 724–734.

[3] C. Barral, J.-S. Coron, and D. Naccache, *Externalized Fingerprint Matching*, Tech. report 2004/021, Cryptology e-print archive, http://eprint.iacr.org, 2004.

[4] C. H. Bennett, G. Brassard, and J.-M. Robert, *Privacy amplification by public discussion*, SIAM J. Comput., 17 (1988), pp. 210–229.

[5] C. H. Bennett, G. Brassard, C. Crépeau, and U. M. Maurer, *Generalized privacy amplification*, IEEE Trans. Inform. Theory, 41 (1995), pp. 1915–1923.

[6] C. H. Bennett, G. Brassard, C. Crépeau, and M.-H. Skubiszewska, *Practical quantum oblivious transfer*, in Advances in Cryptology—CRYPTO '91, 1991, Lecture Notes in Comput. Sci. 576, J. Feigenbaum, ed., Springer-Verlag, New York, 1992, pp. 351–366.

[7] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison Wesley Longman, Reading, MA, 1983.

[8] X. Boyen, *Reusable cryptographic fuzzy extractors*, in Proceedings of the 11th ACM Conference on Computer and Communication Security, ACM, New York, 2004, pp. 82–91.

[9] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, *Secure remote authentication using biometric data*, in Advances in Cryptology—EUROCRYPT 2005, Lecture Notes in Comput. Sci. 3494, R. Cramer, ed., Springer-Verlag, New York, 2005, pp. 147–163.

[10] A. Broder, *On the resemblence and containment of documents*, in Proceedings of Compression and Complexity of Sequences, IEEE Computer Society, Washington, DC, 1997, pp. 21–29.

[11] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W. D. Smith, *A new table of constant weight codes*, IEEE Trans. Inform. Theory, 36 (1990), pp. 1334–1380.

[12] J. L. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[13] E.-C. Chang, V. Fedyukovych, and Q. Li, *Secure Sketch for Multi-Sets*, Tech. report 2006/090, Cryptology e-print archive, http://eprint.iacr.org, 2006.

[14] E.-C. Chang and Q. Li, *Hiding secret points amidst chaff*, in Advances in Cryptology—EUROCRYPT 2006, Lecture Notes in Comput. Sci. 4004, S. Vaudenay, ed., Springer-Verlag, New York, 2006, pp. 59–72.

[15] V. Chauhan and A. Trachtenberg, *Reconciliation puzzles*, in Proceedings of IEEE Globecom, Dallas, TX, 2004. pp. 600–604.

[16] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[17] G. Cohen and G. Zémor, *Generalized coset schemes for the wire-tap channel: Application to biometrics*, in Proceedings of the IEEE International Symposium on Information Theory, Chicago, IL, 2004, p. 46.

[18] C. Crépeau, *Efficient cryptographic protocols based on noisy channels*, in Advances in Cryptology—EUROCRYPT 97, 1997, Lecture Notes in Comput. Sci. 1233, W. Fumy, ed., Springer-Verlag, New York, pp. 306–317.

[19] L. Csirmaz and G. O. H. Katona, *Geometrical cryptography*, in Proceedings of the International Workshop on Coding and Cryptography, Versailles, France, 2003, pp. 578–599.

[20] G. I. Davida, Y. Frankel, B. J. Matt, and R. Peralta, *On the relation of error correction and cryptography to an off line biometric based identification scheme*, in Proceedings of the International Workshop on Coding and Cryptography, Paris, France, 1999; also available online from http://citeseer.ist.psu.edu/389295.html and http://www.cs.yale.edu/homes/peralta/papers/iris.ps.

[21] Y. Z. Ding, *Error correction in the bounded storage model*, in Theory of Cryptology, Lecture Notes in Comput. Sci. 3378, Springer-Verlag, Berlin, 2005, pp. 578–599.

[22] Y. Z. Ding, P. Gopalan, and R. J. Lipton, *Error Correction against Computationally Bounded Adversaries*, manuscript, 2004.

[23] Y. Dodis, J. Katz, L. Reyzin, and A. Smith, *Robust fuzzy extractors and authenticated key agreement from close secrets*, in Advances in Cryptology—CRYPTO 2006, Lecture Notes in Comput. Sci. 4117, C. Dwork, ed., Springer-Verlag, Berlin, 2006, pp. 232–250.

[24] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, *Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data*, Tech. report 2003/235, Cryptology ePrint archive, http://eprint.iacr.org, 2006.

[25] Y. Dodis, L. Reyzin, and A. Smith, *Fuzzy extractors: How to generate strong keys from biometrics and other noisy data*, in Advances in Cryptology—EUROCRYPT 2004, Lecture Notes in Comput. Sci. 3027, C. Cachin and J. Camenisch, eds., Springer-Verlag, Berlin, 2004, pp. 523–540.

[26] Y. Dodis, L. Reyzin, and A. Smith, *Fuzzy extractors*, in Security with Noisy Data, Springer-Verlag, Berlin, 2007, pp. 79–100.
[27] Y. Dodis and A. Smith, *Correcting errors without leaking partial information*, in Proceedings of the ACM Symposium on Theory of Computing, H. N. Gabow and R. Fagin, eds., ACM, New York, 2005, pp. 654–663.
[28] C. Ellison, C. Hall, R. Milbert, and B. Schneier, *Protecting keys with personal entropy*, Future Generation Computer Systems, 16 (2000), pp. 311–318.
[29] G. D. Forney, *Concatenated Codes*, Ph.D. thesis, MIT, Cambridge, MA, 1966.
[30] N. Frykholm, *Passwords: Beyond the Terminal Interaction Model*, Master's thesis, Umeå University, Umeå, Sweden, 2000.
[31] N. Frykholm and A. Juels, *Error-tolerant password recovery*, in Proceedings of the 8th ACM Conference on Computer and Communication Security, ACM, New York, 2001, pp. 1–8.
[32] V. Guruswami, *List Decoding of Error-Correcting Codes*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 2001.
[33] V. Guruswami, *List decoding with side information*, in Proceedings of the IEEE Conference on Computational Complexity, IEEE Computer Society, Washington, DC, 2003, p. 300.
[34] V. Guruswami and A. Rudra, *Explicit capacity-achieving list-decodable codes*, in Proceedings of the ACM Symposium on Theory of Computing, J. M. Kleinberg, ed., ACM, New York, 2006, pp. 1–10.
[35] V. Guruswami and M. Sudan, *List decoding algorithms for certain concatenated codes*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, pp. 181–190.
[36] K. Harmon, S. Johnson, and L. Reyzin, *An Implementation of Syndrome Encoding and Decoding for Binary BCH Codes, Secure Sketches and Fuzzy Extractors*, available at http://www.cs.bu.edu/~reyzin/code/fuzzy.html (2006).
[37] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
[38] A. Juels and M. Sudan, *A fuzzy vault scheme*, Des. Codes Cryptogr., 38 (2006), pp. 237–257.
[39] A. Juels and M. Wattenberg, *A fuzzy commitment scheme*, in Proceedings of the 6th ACM Conference on Computer and Communication Security, ACM, New York, 1999, pp. 28–36.
[40] E. Kaltofen and V. Shoup, *Subquadratic-time factoring of polynomials over finite fields*, in Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1995, pp. 398–406.
[41] A. A. Karatsuba and Y. Ofman, *Multiplication of multidigit numbers on automata*, Soviet Physics Doklady, 7 (1963), pp. 595–596.
[42] J. Kelsey, B. Schneier, C. Hall, and D. Wagner, *Secure applications of low-entropy keys*, in ISW, Lecture Notes in Comput. Sci. 1396, E. Okamoto, G. I. Davida, and M. Mambo, eds., Springer-Verlag, Berlin, 1997, pp. 121–134.
[43] M. Langberg, *Private codes or succinct random codes that are (almost) perfect*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), IEEE Computer Society, Washington, DC, 2004, pp. 325–334.
[44] Q. Li, Y. Sutcu, and N. Memon, *Secure sketch for biometric templates*, in Advances in Cryptology—ASIACRYPT 2006, Lecture Notes in Comput. Sci. 4284, Springer-Verlag, Berlin, 2006, pp. 99–113.
[45] J.-P. M. G. Linnartz, and P. Tuyls, *New shielding functions to enhance privacy and prevent misuse of biometric templates*, in AVBPA 2003, Lecture Notes in Comput. Sci. 2688, Springer-Verlag, Berlin, 2003, pp. 393–402.
[46] R. J. Lipton, *A new approach to information theory*, in STACS, Lecture Notes in Comput. Sci. 775, P. Enjalbert, E. W. Mayr, and K. W. Wagner, eds., Springer-Verlag, Berlin, 1994, pp. 699–708.
[47] U. Maurer, *Secret key agreement by public discussion from common information*, IEEE Trans. Inform. Theory, 39 (1993), pp. 733–742.
[48] S. Micali, C. Peikert, M. Sudan, and D. Wilson, *Optimal error correction against computationally bounded noise*, in Theory of Cryptology, Lecture Notes in Comput. Sci. 3378, Springer-Verlag, Berlin, 2005, pp. 1–16.
[49] Y. Minsky, *The SKS OpenPGP Key Server, Version* 1.0.5, http://www.nongnu.org/sks (March, 2004).
[50] Y. Minsky and A. Trachtenberg, *Scalable set reconciliation*, in 40th Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, 2002, pp. 1607–1616.
[51] Y. Minsky, A. Trachtenberg, and R. Zippel, *Set reconciliation with nearly optimal communication complexity*, IEEE Trans. Inform. Theory, 49 (2003), pp. 2213–2218.

[52] F. Monrose, M. K. Reiter, Q. Li, and S. Wetzel, *Cryptographic key generation from voice*, in Proceedings of the IEEE Symposium on Security and Privacy, M. Abadi and R. Needham, eds., IEEE Computer Society, Washington, DC, 2001, pp. 202–213.

[53] F. Monrose, M. K. Reiter, Q. Li, and S. Wetzel, *Using voice to generate cryptographic keys*, in A Speaker Odyssey: The Speaker Recognition Workshop, Crete, Greece, 2001, pp. 237–242.

[54] F. Monrose, M. K. Reiter, and S. Wetzel, *Password hardening based on keystroke dynamics*, in Proceedings of the 6th ACM Conference on Computer and Communication Security, ACM, New York, 1999, pp. 73–82.

[55] R. Morris and K. Thomson, *Password security: A case history*, Communications of the ACM, 22 (1979), pp. 594–597.

[56] N. Nisan and D. Zuckerman, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–53.

[57] R. Ostrovsky and Y. Rabani, *Low distortion embeddings for edit distance*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, 2005, pp. 218–224.

[58] F. Parvaresh and A. Vardy, *Correcting errors beyond the Guruswami-Sudan radius in polynomial time*, in Proceedings of the IEEE International Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2005, pp. 285–294.

[59] J. Radhakrishnan and A. Ta-Shma, *Bounds for dispersers, extractors, and depth-two superconcentrators*, SIAM J. Discrete Math., 13 (2000), pp. 2–24.

[60] R. Renner and S. Wolf, *Smooth Rényi entropy and applications*, in Proceedings of the IEEE International Symposium on Information Theory, Chicago, IL, 2004, p. 233.

[61] R. Renner and S. Wolf, *Simple and tight bounds for information reconciliation and privacy amplification*, in Advances in Cryptology—ASIACRYPT 2005, Lecture Notes in Comput. Sci. 3788, B. Roy, ed., Springer-Verlag, Berlin, 2005, pp. 199–216.

[62] L. Reyzin, *Entropy Loss is Maximal for Uniform Inputs*, Tech. report BUCS-TR-2007-011, CS Department, Boston University, Boston, MA, 2007; available online from http://www.cs.bu.edu/techreports/.

[63] R. Shaltiel, *Recent developments in explicit constructions of extractors*, Bull. EATCS, 77 (2002), pp. 67–95.

[64] C. E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal, 27 (1948), pp. 379–423, 623–656.

[65] V. Shoup, *A Proposal for an ISO Standard for Public Key Encryption*, http://eprint.iacr.org/2001/112 (2001).

[66] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, Cambridge, UK, 2005; available online from http://shoup.net.

[67] A. Smith, *Scrambling adversarial errors using few random bits*, in ACM–SIAM Symposium on Discrete Algorithms (SODA), H. Gabow, ed., ACM, New York, SIAM, Philadelphia, 2007.

[68] D. Starobinski, A. Trachtenberg, and S. Agarwal, *Efficient PDA synchronization*, IEEE Trans. Mobile Computing, 2 (2003), pp. 40–51.

[69] M. Sudan, *Lecture Notes for an Algorithmic Introduction to Coding Theory*, course taught at MIT, Cambridge, MA, 2001.

[70] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, *A method for solving key equation for decoding Goppa codes*, Inform. and Control, 27 (1975), pp. 87–99.

[71] P. Tuyls and J. Goseling, *Capacity and examples of template-protecting biometric authentication systems*, in ECCV Workshop BioAW, Lecture Notes in Comput. Sci. 3087, D. Maltoni and A. K. Jain, eds., Springer-Verlag, Berlin, 2004, pp. 158–170.

[72] J. H. van Lint, *Introduction to Coding Theory*, Springer-Verlag, Berlin, 1992.

[73] E. Verbitskiy, P. Tuyls, D. Denteneer, and J.-P. Linnartz, *Reliable biometric authentication with privacy protection*, in Proceedings of the 24th Benelux Symposium on Information Theory, Society for Information Theory, The Benelux, 2003.

[74] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 2003.

[75] M. N. Wegman and J. L. Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 140–180

SUB-CONSTANT ERROR LOW DEGREE TEST OF ALMOST-LINEAR SIZE*

DANA MOSHKOVITZ† AND RAN RAZ†

Abstract. Given (the table of) a function $f : \mathbb{F}^m \to \mathbb{F}$ over a finite field $\mathbb{F}$, a *low degree tester* tests its agreement with an m-variate polynomial of total degree at most d over $\mathbb{F}$. The tester is usually given access to an oracle $\mathcal{A}$ providing the *supposed* restrictions of f to affine subspaces of constant dimension (e.g., lines, planes, etc.). The tester makes very few (probabilistic) queries to f and to $\mathcal{A}$ (say, one query to f and one query to $\mathcal{A}$) and decides whether to accept or reject based on the replies. We wish to minimize two parameters of the tester: its *error* and its *size*. The error bounds the probability that the tester accepts although the function is far from a low degree polynomial. The size is the number of bits required to write the oracle replies on all possible tester queries. Low degree testing is a central ingredient in most constructions of probabilistically checkable proofs (PCPs). The error of the low degree tester is related to the error of the PCP, and its size is related to the size of the PCP. We design and analyze new low degree testers that have both *subconstant error* $o(1)$ and *almost-linear size* $n^{1+o(1)}$ (where $n = |\mathbb{F}|^m$). Previous constructions of subconstant error testers had *polynomial size*. These testers enabled the construction of PCPs with subconstant error, but polynomial size. Previous constructions of almost-linear size testers obtained only *constant error*. These testers were used to construct almost-linear size PCPs with constant error. The testers we present in this work enabled the construction of PCPs with both subconstant error and almost-linear size.

Key words. low degree testing, plane vs. point test, sampling, probabilistically checkable proofs

AMS subject classifications. 68Q10, 68Q15, 68Q17

DOI. 10.1137/060656838

1. Introduction.

1.1. Low degree testing. Let $\mathbb{F}$ be a finite field, and let m and d be two positive integers. [A particular setting of parameters to have in mind is the one used in constructions of probabilistically checkable proofs: a large field $\mathbb{F}$, a smaller m, and a fairly large d that satisfy $m^{O(1)}d \leq o(|\mathbb{F}|)$.]

Define $\mathcal{P}$ to be the set of all m-variate polynomials of total degree at most d over $\mathbb{F}$. The *agreement* of a function $f : \mathbb{F}^m \to \mathbb{F}$ with a low degree polynomial is

$$\mathsf{agr}(f, \mathcal{P}) \stackrel{def}{=} \max_{Q \in \mathcal{P}} \left\{ \Pr_{\vec{x} \in \mathbb{F}^m} [f(\vec{x}) = Q(\vec{x})] \right\}.$$

Note that $\mathsf{agr}(f, \mathcal{P})$ is simply $1 - \Delta(f, \mathcal{P})$, where Δ denotes the (normalized) Hamming distance between functions that are given by their tables.

A *low degree tester* is a probabilistic procedure M that is meant to check the agreement of a function f with a low degree polynomial by making as few *queries* to f as possible. If $f \in \mathcal{P}$, M should always accept, while if f is far from $\mathcal{P}$ (i.e., $\mathsf{agr}(f, \mathcal{P})$ is small), M should reject with significant probability.

It is easy to see that, when having oracle access only to f, any low degree tester must make more than d queries. To break this degree barrier, the low degree tester

*Received by the editors April 10, 2006; accepted for publication (in revised form) October 1, 2007; published electronically March 28, 2008.

http://www.siam.org/journals/sicomp/38-1/65683.html

†Department of Computer Science and Applied Mathematics, The Weizmann Institute, 76100 Rehovot, Israel (dana.moshkovitz@weizmann.ac.il, ran.raz@weizmann.ac.il). The first author's research was supported by an Adams Fellowship of the Israel Academy of Sciences and Humanities and by an ISF grant. The second author's research was supported by ISF and BSF grants.

is usually given access to an additional oracle $\mathcal{A}$ providing the *supposed* restrictions of f to affine subspaces of constant dimension (e.g., lines, planes, etc.). We assume, without loss of generality, that these restrictions in themselves are polynomials of total degree at most d over the subspaces.

The tester is required to satisfy:

- *Completeness*: If $f \in \mathcal{P}$, then there is an oracle $\mathcal{A}$ that makes the tester accept with probability 1.
- *Soundness*: If $\mathsf{agr}(f, \mathcal{P})$ is small, then for any oracle $\mathcal{A}$ the tester may accept only with a small probability.

Rubinfeld and Sudan [18] designed the line vs. point tester that makes only two probabilistic queries. This tester picks independently at random a line l in $\mathbb{F}^m$ and a point $\vec{x} \in l$, queries the oracle $\mathcal{A}$ for the (supposed) restriction of f to l (which is simply a univariate polynomial of degree at most d over $\mathbb{F}$), queries f at $\vec{x}$, and checks whether the two restrictions are consistent on $\vec{x}$, i.e., $\mathcal{A}(l)(\vec{x}) = f(\vec{x})$.

The importance of low degree testers comes from the key role they play in the construction of *probabilistically checkable proofs* (PCPs), which are proofs for NP statements that can be probabilistically verified by making only a constant number of queries to the proof [4, 10, 2, 1]. This motivated further improvements to low degree testing.

Specifically, the following parameters were of interest:

1. *Queries.* How many queries does the tester make?
2. *Error.* How sound is the tester?
3. *Size.* How many bits are needed to write the oracle replies on all possible queries?

Henceforth, the number of queries will always be 2. The two other parameters are discussed next.

1.1.1. Error. To prove that a low degree tester is sound, most results address contrapositive arguments of the following type: assume that the tester accepts with probability $\gamma \geq \gamma_0$, and show the existence of a low degree polynomial that agrees with f on at least $\approx \gamma$ of the points. In this case, we say that γ_0 bounds the *error* of the tester, since the probability that the tester accepts although the function is very far from a low degree polynomial is at most γ_0.

The first analyses of the line vs. point tester [18, 2, 12] showed only that the error of the tester is bounded away from 1. The error can be amplified to any *constant*, by a constant number of repetitions. Nevertheless, to keep the total number of queries constant, one cannot perform more than a constant number of repetitions.

Only a later, more careful, inspection [3, 17] revealed that there are low degree testers with a *subconstant error*. Specifically, [3, 17] proved claims of the following type for various low degree testers: there exist (large enough) constants $C \geq 1$, $a, b \geq 0$, and a (small enough) constant $0 < c \leq 1$ such that the error is at most $Cm^a d^b / |\mathbb{F}|^c$. In other words, the error can be made arbitrarily small by taking m and d to be small enough with respect to $|\mathbb{F}|$. The number of queries remains 2.

Arora and Sudan [3] proved that the error of the line vs. point tester is in fact subconstant. Their proof was algebraic in nature. Raz and Safra [17] proved a subconstant error for a slightly different tester, by considering planes that intersect on a line or a plane and a point within it. Their proof was more combinatorial in nature. The two proofs led to the construction of PCPs with subconstant error [3, 17, 9].

1.1.2. Size. Let us represent the set of honest oracles by a code. That is, for every polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ of degree at most d, we have a code word. The code

word has an entry for every affine subspace s that the tester may query. This entry contains the oracle's reply when it is queried regarding s, i.e., the restriction of Q to s. The *size* of a tester is the length (in bits) of a code word.

For instance, the size of Rubinfeld and Sudan's line vs. point tester [18] is roughly $|\mathbb{F}|^{2m}(d+1)\log|\mathbb{F}|$: for every line (defined by two points), the oracle should provide a univariate polynomial of degree at most d over $\mathbb{F}$. The size of a tester is measured with respect to $n = |\mathbb{F}^m|$. The size of the line vs. point tester [18] is quadratic $n^{2+o(1)}$.

Alternatively, we refer to the *randomness* of the tester, which is the amount of random bits that the tester requires. Note that the size of a tester that uses r random bits to make q queries to a proof over alphabet Σ is bounded by $2^r \cdot q \log|\Sigma|$. Thus, when the number of queries q is constant and the alphabet Σ is relatively small, 2^r is a good estimate on the size.

For instance, to pick a random line and a random point within it, we merely have to pick a random point $\vec{x} \in \mathbb{F}^m$ and a random direction $\vec{y} \in \mathbb{F}^m$. The line is $\vec{x} + t \cdot \vec{y}$ for $t \in \mathbb{F}$. Hence, the randomness of the line vs. point tester [18] is $2m \log|\mathbb{F}| = \log(|\mathbb{F}|^{2m})$.

The size of a tester is related to the size of probabilistically checkable proofs and locally testable codes constructed by using it. Hence, Goldreich and Sudan [13] suggested to improve the line vs. point tester by considering a relatively small subset of lines (instead of all lines). Goldreich and Sudan achieved *nonexplicit* constant error tester of *almost-linear* size $n^{1+o(1)}$ instead of quadratic size $n^{2+o(1)}$.

Shortly afterwards, Ben-Sasson et al. [7] gave an explicit construction of a constant error line vs. point tester of almost-linear size. Their idea was to choose a line by picking a uniformly distributed point over $\mathbb{F}^m$ (as before) and a direction that is uniformly distributed over a small ϵ-biased set $S \subseteq \mathbb{F}^m$. They showed that the error of this tester is bounded away from 1. Unfortunately, their analysis is inherently able to show only error larger than $\frac{1}{2}$. It is possible that their tester has smaller error, but proving it would require a substantially different analysis.

The work of [13, 7] gave rise to explicit constructions of almost-linear size PCPs with constant error [13, 7, 5]. The recent work of Dinur [8] also constructs almost-linear size PCPs with constant error, based on the PCP theorem of [2, 1] and the work of Ben-Sasson and Sudan [6]. Both use low degree testers with constant error. Dinur's work [8] also gives new constructions of PCPs without low degree testers. However, at this point, these constructions achieve neither subconstant error nor almost-linear size.

1.2. Our contribution: Randomness-efficient subconstant error testers. We design and analyze two low degree testers that have both subconstant error and almost-linear size. Subsequent to this work and by using it, we showed a construction of a PCP with both subconstant error and almost-linear size [15].

Before we present our testers, let us revisit the construction of Ben-Sasson et al. [7] for constant error and point out the most severe difficulty one encounters when trying to argue it has error smaller than $\frac{1}{2}$. The reader who is not familiar with the work of Ben-Sasson et al. may skip this and move directly to the text after Remark 1.1.

Assume a line vs. point tester that inspects only lines whose directions are taken from a small *random* set $S \subseteq \mathbb{F}^m$. Recall that Ben-Sasson et al. used a small ϵ-biased set because of its *pseudorandom* properties [7].

Consider two linearly independent directions $\vec{y}_1, \vec{y}_2 \in S$. With high probability, the set S does not contain any additional vector from the linear span of $\vec{y}_1$ and $\vec{y}_2$ (since the fractional size of the linear span is merely $|\mathbb{F}|^2 / |\mathbb{F}|^m$). Thus, the only lines inside this two-dimensional linear subspace that get inspected by the tester are

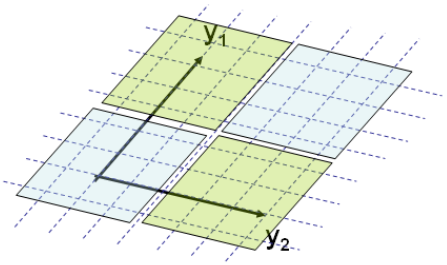


FIG. 1.1. *Acceptance probability* $\frac{1}{2}$ *inside a plane does not necessarily imply agreement with a low degree polynomial.*

those that are parallel to either $\vec{y}_1$ or $\vec{y}_2$. It is known by a lemma of Polishchuk and Spielman [16] that if the acceptance probability of the line vs. point test in this setting approaches 1, then there exists a low degree polynomial for the entire subspace that agrees with almost all lines. However, it may be the case that the acceptance probability is as large as $\frac{1}{2}$, although the agreement of those lines with any low degree polynomial is very small.

Let us demonstrate this in Figure 1.1.

Note that each of the $|\mathbb{F}|^2$ points on the plane spanned by $\vec{y}_1$ and $\vec{y}_2$ can be uniquely represented as $\alpha_1\vec{y}_1 + \alpha_2\vec{y}_2$, where $\vec{\alpha} = (\alpha_1, \alpha_2) \in \mathbb{F}^2$.

Let $d \ll d' \ll \frac{|\mathbb{F}|}{4}$. Define polynomials $C, R \in \mathbb{F}[\alpha_1, \alpha_2]$ (for *columns* and *rows*, respectively) as follows. The degree of C in α_1 is d, and the degree in α_2 is d'. The degree of R in α_1 is d', and the degree in α_2 is d. Let the $|\mathbb{F}|$ columns, i.e., lines parallel to $\vec{y}_1$, agree with C. Let the $|\mathbb{F}|$ rows, i.e., lines parallel to $\vec{y}_2$, agree with R. Note that all lines, columns and rows, are assigned (univariate) polynomials of degree at most d.

Let points in the dark region agree with columns and points in the bright region agree with rows. Both R and C are polynomials of degree *at least* $d' \gg d$ for the plane that agree with at least $\frac{1}{2}$ of the points. However, no polynomial of degree *at most* d for the plane agrees with a fraction of more than $\frac{4d'}{|\mathbb{F}|} = o(1)$ of the points. On the other hand, the acceptance probability of the line vs. point tester on this plane (where all lines are assigned polynomials of degree at most d) is at least $\frac{1}{2}$.

Remark 1.1. One may consider other low degree testers, such as the *line vs. line* tester, in order to solve the problem we described. However, it is not known whether or not the line vs. line tester (on the plane) with lines parallel to the axes gives a probability of error lower than $\frac{1}{2}$.

We manage to overcome this difficulty by considering sets that are *not pseudo-random*. Our key idea is to consider a subfield $\mathbb{H} \subseteq \mathbb{F}$ and generate subspaces by picking directions uniformly over $\mathbb{H}^m$ instead of over $\mathbb{F}^m$. Note that this eliminates the problem we described: for every two $\vec{y}_1, \vec{y}_2 \in \mathbb{H}^m$, for every two scalars $\alpha_1, \alpha_2 \in \mathbb{H}$, we have $\alpha_1\vec{y}_1 + \alpha_2\vec{y}_2 \in \mathbb{H}^m$.

Moreover, the field structure of $\mathbb{H}$ allows us to use the combinatorial approach of Raz and Safra [17], and, more importantly, it allows us to use induction: the structure of the problem when restricted to affine subspaces of dimension $k \leq m$ is the same as its structure in $\mathbb{F}^m$.

As in the analysis of Raz and Safra [17], we abandon the line vs. point test and address subspaces of dimension larger than 1 rather than lines. Specifically, given

1. Pick uniformly and independently at random $\vec{z} \in \mathbb{F}^m$, $\vec{y}_1, \vec{y}_2 \in \mathbb{H}^m$.
2. Accept if either $\vec{y}_1, \vec{y}_2$ are linearly dependent or if the plane p through $\vec{z}$ in directions $\vec{y}_1, \vec{y}_2$ satisfies $\mathcal{A}(p)(\vec{z}) = f(\vec{z})$.

FIG. 1.2. *Randomness-efficient plane vs. point tester.*

access to f and to an oracle $\mathcal{A}$, our *randomness-efficient plane vs. point* tester chooses a plane and a point within it and checks that they are consistent as shown in Figure 1.2.

Note that the same plane p goes through many points $\vec{z} \in \mathbb{F}^m$ and in many directions $\vec{y}_1, \vec{y}_2 \in \mathbb{H}^m$. However, the oracle's reply $\mathcal{A}(p)$ depends on the plane p and not on its representation given by $\vec{z}$ and $\vec{y}_1, \vec{y}_2$.

For $\mathbb{H} = \mathbb{F}$, the randomness-efficient plane vs. point tester is exactly the plane vs. point tester of Raz and Safra [17]. However, in our work the more interesting case is $|\mathbb{H}| \leq |\mathbb{F}|^{o(1)}$. In this case, the tester requires only $m \log |\mathbb{F}| + 2m \log |\mathbb{H}| = m \log |\mathbb{F}| (1 + o(1))$ bits of randomness. This corresponds to an almost-linear size $n^{1+o(1)}$ (recall that $n = |\mathbb{F}^m|$). The tester is randomness efficient in comparison to all known testers with subconstant error, such as the tester of Arora and Sudan [3] that requires $2m \log |\mathbb{F}|$ bits of randomness and the tester of Raz and Safra [17] that requires $3m \log |\mathbb{F}|$ bits of randomness. As to testers with constant error: that of Ben-Sasson et al. [7] requires $m \log |\mathbb{F}| + O(\log \log |\mathbb{F}^m|)$ bits of randomness, which is (usually) less than the randomness of our tester, but the difference is only in the dependence of the *low order term* in m.

The tester is clearly *complete*; namely, if there exists a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ of degree at most d such that for every $\vec{x} \in \mathbb{F}^m$, $f(\vec{x}) = Q(\vec{x})$ and for every affine subspace s the oracle $\mathcal{A}$ replies $\mathcal{A}(s) = Q_{|s}$, then the tester accepts with probability 1. We show that the tester is also *sound*: if the tester accepts with probability γ, then f agrees with a polynomial of total degree at most md on a fraction of at least $\gamma - \varepsilon$ of the points in $\mathbb{F}^m$, where $\varepsilon \leq \text{const} \cdot m(\sqrt[8]{\frac{1}{|\mathbb{H}|}} + \sqrt[4]{\frac{md}{|\mathbb{F}|}})$. Note that the analysis works for any acceptance probability γ. In particular, this means that, when γ is significantly larger than ε, say, $\gamma \geq 100\varepsilon$, f agrees with a polynomial of total degree at most md on at least $\approx \gamma$ of the points. [Even if $\mathbb{H} = \mathbb{F}$, the constants 4 and 8 in the error expression appear to improve on the results of [3, 17], where unspecified constants were given.]

The downside of the randomness-efficient plane vs. point tester is that it allows us only to argue something about the agreement of the oracle with a polynomial of degree md rather than d. Hence, we design another tester that has essentially the same parameters but ensures agreement with a polynomial of degree at most d.

The additional consideration that comes into play when designing the new tester is *degree preservation*. We want the total degree of a polynomial not to decrease when restricted to most of the subspaces queried by the tester. We achieve this by picking one of the *directions* for the subspace (rather than the base point) uniformly from $\mathbb{F}^m$. In order to keep the size almost linear, this tester considers linear subspaces (i.e., affine subspaces through the origin) rather than general affine subspaces. A related technique was previously used in [7].

Specifically, given access to f and to an oracle $\mathcal{A}$, the *randomness-efficient subspace vs. point* tester chooses a three-dimensional subspace and a point within it and checks that they are consistent as shown in Figure 1.3.

1. Pick uniformly and independently at random $\vec{z} \in \mathbb{F}^m$, $\vec{y}_1, \vec{y}_2 \in \mathbb{H}^m$.
2. Accept if either $\vec{z}, \vec{y}_1, \vec{y}_2$ are linearly dependent or if the linear subspace s spanned by $\vec{z}, \vec{y}_1, \vec{y}_2$ satisfies $\mathcal{A}(s)(\vec{z}) = f(\vec{z})$.

FIG. 1.3. *Randomness-efficient subspace vs. point tester.*

This tester uses the same number of random bits as the randomness-efficient plane vs. point tester $m \log |\mathbb{F}| + 2m \log |\mathbb{H}|$, and its size is only slightly larger (as the answer size is larger: the oracle should provide polynomials over three-dimensional subspaces rather than two-dimensional subspaces). For this small price, we manage to prove a stronger soundness claim: if the randomness-efficient subspace vs. point tester accepts with probability γ, then f agrees with a polynomial of total degree at most d (rather than md) on a fraction of at least $\gamma - \varepsilon$ of the points in $\mathbb{F}^m$, where $\varepsilon \leq \text{const} \cdot m(\sqrt[8]{\frac{1}{|\mathbb{H}|}} + \sqrt[4]{\frac{md}{|\mathbb{F}|}})$. This follows rather easily from the soundness of the randomness-efficient plane vs. point tester together with an argument showing that the degree of the recovered polynomials must in fact be at most d.

There is a trade-off between the size of the testers and their error. To make the size as small as possible, one wishes to minimize $|\mathbb{H}|$. In particular, to get an almost-linear size, one needs to take $|\mathbb{H}| \leq |\mathbb{F}|^{o(1)}$. On the other hand, to make the error as small as possible, one wishes to maximize $|\mathbb{H}|$. In particular, to get a subconstant error, one needs to take $|\mathbb{H}| \geq \omega(m^8)$.

All finite fields are isomorphic to $GF(p^k)$ for a prime p and a natural number k. All subfields of $GF(p^k)$ are isomorphic to $GF(p^r)$ for $r|k$. For a wide family of finite fields $GF(p^k)$ there are subfields of suitable sizes (see [14, 11] for analysis of the distribution of k's with suitable divisors). Though, indeed, not every finite field is such. We wish to emphasize that, in the settings that interest us (e.g., construction of PCPs), *we get to choose the field*. For instance, we can take $\mathbb{F} = GF(2^{r_1 \cdot r_2})$ for appropriate r_1, r_2.

1.3. Sampling. A basic step in our proof is the analysis of the sampling properties of affine subspaces with directions over a subfield. This analysis may be of independent interest.

By *sampling* we refer to assertions of the following nature: if one colors a large enough fraction of the points in $\mathbb{F}^m$ green, then a subspace (e.g., a line) picked at random is likely to hit the green points in almost their true fraction.

First, let us consider the non-randomness-efficient setting. For instance, consider choosing a line by picking a point and a direction independently at random from $\mathbb{F}^m$. The indicator variables "is the ith point on the line green?" for $i = 1, \ldots, |\mathbb{F}|$ are *pairwise independent*. Thus, one can easily bound the variance of the number of green points on a line. This yields a sampling property by Chebyshev's inequality (see, e.g., [3]).

In the randomness-efficient setting, more subtle arguments are needed. For instance, consider the work of Ben-Sasson et al. [7]. They use an ϵ-biased set $S \subseteq \mathbb{F}^m$ and choose a line by independently picking a uniformly distributed base point in $\mathbb{F}^m$ and a uniformly distributed direction in S. They show that *almost-pairwise independence* still holds, and this allows them to bound the variance, by bounding the covariances.

Our set of directions is $\mathbb{H}^m$, which does not have a small bias (when $\mathbb{H} \subsetneq \mathbb{F}$). Nevertheless, we are still able to prove a sampling property. We observe that we

can directly bound the variance of the number of green points on a line by analyzing the convolution of two relatively simple functions. We do this by means of Fourier analysis. The difference between the previous approaches and our approach is that, instead of giving one bound for the probability that two points $i \neq j$ on a line are green *for every* $i \neq j$, we directly bound the *average* probability over all pairs $i \neq j$.

The extension to higher-dimensional subspaces is a relatively simple consequence of the analysis for lines.

1.4. More randomness-efficient line samplers. Ariel Gabizon has noted that our analysis implies numerous randomness-efficient line samplers.

Recall that the set $\mathbb{H}^m$ for a subfield $\mathbb{H} \subseteq \mathbb{F}$—in addition to implying a sampling property—also has an algebraic structure that is essential for our analysis. However, if one is interested only in the sampling property, more randomness-efficient constructions may be obtained.

Jointly with Ariel we arrived at the following corollaries to our analysis.

Direct product construction. Our sampling lemma holds for any field $\mathbb{F} = GF(p^k)$ and a subset of it $H \subseteq \mathbb{F}$ (not necessarily a subfield). Formally we state the following.

COROLLARY 1.2. *For any subset* $A \subseteq \mathbb{F}^m$ *of density* $\mu = |A| / |\mathbb{F}^m|$, *for any* $\varepsilon > 0$,

$$\Pr_{\vec{x} \in \mathbb{F}^m, \vec{y} \in H^m} \left[\left| \frac{|l_{\vec{x},\vec{y}} \cap A|}{|l_{\vec{x},\vec{y}}|} - \mu \right| \geq \varepsilon \right] \leq \frac{1}{|H|} \cdot \frac{\mu}{\varepsilon^2},$$

where $l_{\vec{x},\vec{y}} \stackrel{def}{=} \{\vec{x} + t \cdot \vec{y} \mid t \in \mathbb{F}\}$.

Linear code construction. Assume a linear code of length k, dimension m, and relative distance $1 - \delta$ over alphabet $\mathbb{F} = GF(p)$, given by its generating matrix

$$\begin{pmatrix} - & \vec{y}_1 & - \\ & \vdots & \\ - & \vec{y}_k & - \end{pmatrix}.$$

Let $S = \{\vec{y}_1, \ldots, \vec{y}_k\} \subseteq \mathbb{F}^m$ be the set of rows of the generating matrix. Then our analysis actually implies the following.

COROLLARY 1.3. *For any subset* $A \subseteq \mathbb{F}^m$ *of density* $\mu = |A| / |\mathbb{F}^m|$, *for any* $\varepsilon > 0$,

$$\Pr_{\vec{x} \in \mathbb{F}^m, \vec{y} \in S} \left[\left| \frac{|l_{\vec{x},\vec{y}} \cap A|}{|l_{\vec{x},\vec{y}}|} - \mu \right| \geq \varepsilon \right] \leq \delta \cdot \frac{\mu}{\varepsilon^2},$$

where $l_{\vec{x},\vec{y}} \stackrel{def}{=} \{\vec{x} + t \cdot \vec{y} \mid t \in \mathbb{F}\}$.

Note that every $S \subseteq \mathbb{F}^m$ that is ϵ-biased forms a generating matrix of a linear code with distance $1 - (\frac{1}{|\mathbb{F}|} + \epsilon \cdot \frac{|\mathbb{F}|-1}{|\mathbb{F}|})$. Yet, the converse does not necessarily hold, and the corollary is a strengthening of the sampling lemma of [7] for the case $\mathbb{F} = GF(p)$.

A randomness-efficient line sampler can be constructed by using an efficient linear code. For instance, we can use the Reed–Solomon code that corresponds to $S = \{(1, t, t^2, \ldots, t^{m-1}) \mid t \in \mathbb{F}\}$. This code has relative distance $1 - \delta$ for $\delta = \frac{m-1}{|\mathbb{F}|}$. It gives a line sampler that has randomness complexity $(m + 1) \log |\mathbb{F}|$ and query complexity $|\mathbb{F}|$.

1.5. Proof outline. We first prove the soundness of the randomness-efficient plane vs. point tester and then deduce the soundness of the randomness-efficient subspace vs. point tester from it. For the purpose of this outline we consider only the first. Assume that the randomness-efficient plane vs. point tester, given access to an input function $f : \mathbb{F}^m \to \mathbb{F}$ and oracle $\mathcal{A}$, accepts with probability γ. Let us prove the existence of a polynomial over $\mathbb{F}^m$ of degree at most md that agrees with f on at least $\gamma - \varepsilon$ fraction of the points, for $\varepsilon \leq \text{const} \cdot m(\sqrt[8]{\frac{1}{|\mathbb{H}|}} + \sqrt[4]{\frac{md}{|\mathbb{F}|}})$.

1.5.1. Reformulating our goal. First, let us reformulate the problem in a more convenient manner. For dimensions k, m, where $k \leq m$, let $\mathcal{S}_k^m$ be the family of all affine subspaces of dimension k in $\mathbb{F}^m$ that are of the type in which we are interested. Namely, a k-dimensional affine subspace $s \subseteq \mathbb{F}^m$ is in $\mathcal{S}_k^m$ if it can be written as $s = \{\vec{z} + \sum_{i=1}^k \alpha_i \vec{y}_i \mid (\alpha_1, \ldots, \alpha_k) \in \mathbb{F}^k\}$ for some point $\vec{z} \in \mathbb{F}^m$ and some linearly independent directions $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$ (where the linear independence is over $\mathbb{F}$).

We can express (up to very small additive errors) the acceptance probability of the tester given access to $f : \mathbb{F}^m \to \mathbb{F}$ and $\mathcal{A}$ as follows:

$$\begin{aligned}\Pr\left[\text{tester accepts}\right] &\approx \Pr_{s \in \mathcal{S}_2^m, \vec{x} \in s}\left[\mathcal{A}(s)(\vec{x}) = f(\vec{x})\right] \\ &= \mathop{\mathbf{E}}_{s \in \mathcal{S}_2^m}\left[\Pr_{\vec{x} \in s}\left[\mathcal{A}(s)(\vec{x}) = f(\vec{x})\right]\right].\end{aligned}$$

For an affine subspace s and a degree d, let $\mathcal{Q}_{s,d}$ be the set of polynomials of degree at most d over s. It is evident from the last expression that an oracle $\mathcal{A}$ that optimizes the acceptance probability of the tester on input f assigns each subspace $s \in \mathcal{S}_2^m$ a polynomial $Q \in \mathcal{Q}_{s,d}$ that maximizes the agreement $Q(\vec{x}) = f(\vec{x})$ on points $\vec{x} \in s$. Hence, for every dimension m, function $f : \mathbb{F}^m \to \mathbb{F}$, dimension k, and degree d, consider the *average agreement* of f with polynomial of degree at most d over subspaces $s \in \mathcal{S}_k^m$:

$$\mathsf{agr}_d^{k,m}(f) \stackrel{def}{=} \mathop{\mathbf{E}}_{s \in \mathcal{S}_k^m}\left[\max_{Q \in \mathcal{Q}_{s,d}}\left\{\Pr_{\vec{x} \in s}\left[Q(\vec{x}) = f(\vec{x})\right]\right\}\right].$$

Then

$$\gamma = \Pr\left[\text{tester accepts}\right] \lesssim \mathsf{agr}_d^{2,m}(f).$$

For every m, the space $\mathbb{F}^m$ is the only affine subspace of dimension m in $\mathbb{F}^m$, and $\mathbb{H}^m$ contains a basis for $\mathbb{F}^m$, so $\mathcal{S}_m^m = \{\mathbb{F}^m\}$. Thus, for every dimension m, function $f : \mathbb{F}^m \to \mathbb{F}$, degree d, and fraction γ, $\mathsf{agr}_d^{m,m}(f) \geq \gamma$ means that there exists $Q : \mathbb{F}^m \to \mathbb{F}$ of degree at most d such that $\Pr_{\vec{x} \in \mathbb{F}^m}\left[Q(\vec{x}) = f(\vec{x})\right] \geq \gamma$.

We conclude that our goal can be reformulated as showing that a large average agreement over planes implies a large average agreement over $\mathbb{F}^m$. More accurately, for every function $f : \mathbb{F}^m \to \mathbb{F}$ and fraction $0 \leq \gamma \leq 1$,

$$\mathsf{agr}_d^{2,m}(f) \geq \gamma \quad \Rightarrow \quad \mathsf{agr}_{md}^{m,m}(f) \geq \gamma - \varepsilon.$$

1.5.2. Main idea. We fix a dimension m, and our proof is by induction on the dimension k of the affine subspaces within $\mathbb{F}^m$. We assume that $\mathsf{agr}_d^{2,m}(f) \geq \gamma$ and show that, for every dimension $2 \leq k \leq m$,

$$\mathsf{agr}_{kd}^{k,m}(f) \geq \gamma - \frac{k}{m} \cdot \varepsilon.$$

Fix a dimension k such that $\mathsf{agr}^{k-1,m}_{(k-1)d}(f) \geq \gamma - \frac{k-1}{m} \cdot \varepsilon$, and let us outline how the induction step is done.

Consider *any* affine subspace $s \in \mathcal{S}^m_k$. Assume that s contains the point $\vec{z} \in \mathbb{F}^m$ and is in directions $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$, where $\vec{y}_1, \ldots, \vec{y}_k$ are linearly independent over $\mathbb{F}$. The directions within s, $\{\vec{x}_1 - \vec{x}_2 \mid \vec{x}_1, \vec{x}_2 \in s\}$, are precisely $\sum_{i=1}^k \alpha_i \vec{y}_i$ for $\vec{\alpha} = (\alpha_1, \ldots, \alpha_k) \in \mathbb{F}^k$. Moreover, *since* $\mathbb{H}$ *is a subfield of* $\mathbb{F}$,

$$\vec{\alpha} \in \mathbb{H}^k \quad \Leftrightarrow \quad \sum_{i=1}^k \alpha_i \vec{y}_i \in \mathbb{H}^m.$$

Therefore (unlike the construction of [7] via ϵ-biased sets), the families of affine subspaces we consider preserve the following two properties enabling induction:

1. *Self-similarity.* Every affine subspace $s \in \mathcal{S}^m_k$ is mapped onto $\mathbb{F}^k$ (via the natural bijection $\vec{z} + \sum_{i=1}^k \alpha_i \vec{y}_i \in s \;\leftrightarrow\; \vec{\alpha} \in \mathbb{F}^k$) such that the directions the tester considers (namely, the vectors in $\mathbb{H}^m$) that are also in s are mapped onto $\mathbb{H}^k$.
2. *Uniformity.* For every dimension $k' \leq k$, each subspace $s \in \mathcal{S}^m_k$ contains exactly the same number of subspaces $s' \in \mathcal{S}^m_{k'}$, and each subspace $s' \in \mathcal{S}^m_{k'}$ is contained in exactly the same number of subspaces $s \in \mathcal{S}^m_k$.

Let $f_{|s} : \mathbb{F}^k \to \mathbb{F}$ denote the restriction of f to s; namely, for every $(\alpha_1, \ldots, \alpha_k) \in \mathbb{F}^k$, let $f_{|s}(\alpha_1, \ldots, \alpha_k) = f(\vec{z} + \sum_{i=1}^k \alpha_i \vec{y}_i)$.

Consider some degree d' and dimension $k' \leq k$. By *self-similarity* and *uniformity*,

$$\mathsf{agr}^{k',m}_{d'}(f) = \mathop{\mathbf{E}}_{s \in \mathcal{S}^m_k} \left[\mathsf{agr}^{k',k}_{d'}(f_{|s}) \right]. \tag{1.1}$$

Thus, it is sufficient (as we will see shortly) to show that, for every function $f : \mathbb{F}^k \to \mathbb{F}$ and every fraction $0 \leq \gamma \leq 1$,

$$\mathsf{agr}^{k-1,k}_{(k-1)d}(f) \geq \gamma \quad \Rightarrow \quad \mathsf{agr}^{k,k}_{kd}(f) \geq \gamma - \frac{\varepsilon}{m}. \tag{1.2}$$

The inductive step is then completed by applying the induction hypothesis as well as (1.1) and (1.2) above:

$$\begin{aligned}
\mathsf{agr}^{k,m}_{kd}(f) &= \mathop{\mathbf{E}}_{s \in \mathcal{S}^m_k} \left[\mathsf{agr}^{k,k}_{kd}(f_{|s}) \right] \\
&\geq \mathop{\mathbf{E}}_{s \in \mathcal{S}^m_k} \left[\mathsf{agr}^{k-1,k}_{(k-1)d}(f_{|s}) - \frac{\varepsilon}{m} \right] \\
&= \mathsf{agr}^{k-1,m}_{(k-1)d}(f) - \frac{\varepsilon}{m} \\
&\geq \gamma - \frac{k}{m} \cdot \varepsilon.
\end{aligned}$$

1.5.3. Proving (1.2). By an adaptation of an idea by Raz and Safra [17], we can prove that there exists a small error $\delta \ll \varepsilon/m$ such that, for every function $f : \mathbb{F}^k \to \mathbb{F}$ and every fraction $0 \leq \gamma \leq 1$,

$$\mathsf{agr}^{k-1,k}_{(k-1)d}(f) \geq \gamma \quad \Rightarrow \quad \mathsf{agr}^{k,k}_{2(k-1)d}(f) \geq \gamma^2 - \delta.$$

The idea of Raz and Safra [17] centers around a construction of a *consistency graph*. The vertices of the graph are the affine subspaces of dimension $(k-1)$ within $\mathbb{F}^k$

(namely, *hyperplanes*). The edges of the graph indicate whether there is an agreement between assignments of degree $(k-1)d$ polynomials to the hyperplanes. Due to its algebraic structure, the graph has a combinatorial property called *almost-transitivity*. It allows us to use a graph-theoretic lemma originally proven in [17] and go up from dimension $(k-1)$ to dimension k.

The reduction to the graph-theoretic setting introduces a certain deterioration of the degree and agreement parameters. The degree doubles (from $(k-1)d$ to $2(k-1)d$ rather than to kd), and the agreement is raised to the power of two (from γ to $\gamma^2-\delta$ rather than to $\gamma-\varepsilon/m$). We cannot tolerate either deterioration, since they ultimately cause an exponential decay in k. Hence, we apply steps of what we call *consolidation* to retain the desired parameters. Similar techniques were already used in previous work, and they rely on the sampling properties we discussed above.

1.6. Organization. We state the main theorems regarding the soundness of our testers in section 2. The rest of the paper is devoted to proving these theorems. We start with some preliminary definitions and propositions in section 3. We discuss basic properties of affine subspaces with directions over a subfield in section 4. We prove sampling properties in section 5. This allows us to prove consolidation claims in section 6. We present and analyze the consistency graph in section 7 and use it for going up one dimension in section 8. The soundness of the randomness-efficient plane vs. point tester is proven via induction in section 9. We show that the soundness of the randomness-efficient subspace vs. point tester follows in section 10. We give the proof of the combinatorial lemma of [17] in the appendix.

2. Our results.

2.1. Notation. In all that follows, we consider a finite field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$, a dimension m, and a degree d.

Given vectors $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{F}^m$, we define the *linear subspace* they span by

$$span\{\vec{y}_1, \ldots, \vec{y}_k\} \stackrel{def}{=} \{a_1\vec{y}_1 + \cdots + a_k\vec{y}_k \mid a_1, \ldots, a_k \in \mathbb{F}\}.$$

We say that $\vec{y}_1, \ldots, \vec{y}_k$ are *linearly independent* and denote $\mathsf{ind}(\vec{y}_1, \ldots, \vec{y}_k)$ if for every $a_1, \ldots, a_k \in \mathbb{F}$, if $\sum_{i=1}^k a_i\vec{y}_i = 0$, then $a_1 = \cdots = a_k = 0$. Throughout the paper we will refer to a span over $\mathbb{F}$ (and not over a subfield, even if the vectors are over a subfield). Note that vectors $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$ are linearly independent over $\mathbb{H}$ if and only if $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$ are linearly independent over $\mathbb{F}$.

Given two sets $A, B \subseteq \mathbb{F}^m$, we define $A + B \stackrel{def}{=} \{\vec{x} + \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}$. Given a point $\vec{x} \in \mathbb{F}^m$ and a set $A \subseteq \mathbb{F}^m$, define $\vec{x} + A \stackrel{def}{=} \{\vec{x}\} + A$. A k-dimensional *affine subspace* in the vector space $\mathbb{F}^m$ is defined by a base point $\vec{x} \in \mathbb{F}^m$ and k linearly independent directions $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{F}^m$ as

$$\text{affine}(\vec{x}; \vec{y}_1, \ldots, \vec{y}_k) \stackrel{def}{=} \vec{x} + span\{\vec{y}_1, \ldots, \vec{y}_k\}.$$

Points are 0-dimensional affine subspaces. *Lines* are 1-dimensional affine subspaces. *Planes* are 2-dimensional affine subspaces. Every affine subspace can be equivalently represented by many choices of vectors $\vec{x}; \vec{y}_1, \ldots, \vec{y}_k$, but, clearly, there is an affine transformation between every two representations of the same affine subspace.

An *m-variate polynomial* over a field $\mathbb{F}$ is a function $Q : \mathbb{F}^m \to \mathbb{F}$ of the form

$$Q(x_1, \ldots, x_m) = \sum_{i_1, \ldots, i_m} a_{i_1, \ldots, i_m} x_1^{i_1} \ldots x_m^{i_m},$$

where all of the *coefficients* $a_{i_1,\ldots,i_m}$ are in $\mathbb{F}$. The *degree* of Q is

$$\deg Q \stackrel{def}{=} \max\left\{\sum_{j=1}^{m} i_j \mid a_{i_1,\ldots,i_m} \neq 0\right\},$$

where the degree of the *identically zero* polynomial is defined to be 0.

The restriction of a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ to an affine subspace s represented as $s = \text{affine}(\vec{x}; \vec{y}_1, \ldots, \vec{y}_k)$ is a polynomial in k variables: $Q_{|s}(\alpha_1, \ldots, \alpha_k) \stackrel{def}{=} Q(\vec{x} + \alpha_1\vec{y}_1 + \ldots + \alpha_k\vec{y}_k)$. We will sometimes wish to refer to a polynomial Q defined over an affine subspace s without specifying the subspace's representation, in which case we will use the notation $Q(\vec{x})$ for a point $\vec{x} \in s$. Note that the degree of the polynomial does not depend on the representation of s.

2.2. Oracles. We assume an oracle $\mathcal{A}$ that, given any affine subspace s in $\mathbb{F}^m$, provides a polynomial $\mathcal{A}(s)$ of degree at most d defined over s. For the sake of simplicity, we do not refer to both an oracle $\mathcal{A}$ and a function $f : \mathbb{F}^m \to \mathbb{F}$ as in the introduction. Instead, we assume that f's values on points $\vec{x}$ are given by $\mathcal{A}(\vec{x})$. Our testers query $\mathcal{A}$ only on affine subspaces of constant dimension. However, for the analysis, it will be convenient to consider oracles queried regarding higher-dimensional affine subspaces as well. Hence, an oracle $\mathcal{A}$ is defined to provide a value for any affine subspace.

For a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, we will use the notation $(Q \equiv \mathcal{A})(s)$ to indicate that Q and $\mathcal{A}$ agree on a subspace s, i.e., for every $\vec{x} \in s$, $Q(\vec{x}) = \mathcal{A}(s)(\vec{x})$.

2.3. Low degree testers. Define two predicates for our two testers: for $\vec{z} \in \mathbb{F}^m$ and $\vec{y}_1, \vec{y}_2 \in \mathbb{H}^m$, let the following apply:

1. $PlanePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2)$: $\vec{y}_1, \vec{y}_2$ are linearly dependent or
$$\mathcal{A}(\text{affine}(\vec{z}; \vec{y}_1, \vec{y}_2))(\vec{z}) = \mathcal{A}(\vec{z});$$
2. $SpacePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2)$: $\vec{z}, \vec{y}_1, \vec{y}_2$ are linearly dependent or
$$\mathcal{A}(\text{affine}(\vec{0}; \vec{z}, \vec{y}_1, \vec{y}_2))(\vec{z}) = \mathcal{A}(\vec{z}).$$

2.4. Soundness. To prove that a tester is sound we assume that it accepts with probability γ when given access to an oracle $\mathcal{A}$ and show the agreement of $\mathcal{A}$ with a low degree polynomial. Specifically, for a subconstant ε, we prove two claims, which we argue to be essentially equivalent:

1. *(Decoding)* There exists a low degree polynomial that is consistent with the oracle $\mathcal{A}$ on at least a $\gamma - \varepsilon$ fraction of the points.
2. *(List decoding)* For every $0 < \delta < 1$, there exists a short list of $t = t(\delta)$ low degree polynomials that *explains* all of the tester's acceptance but a $\delta + \varepsilon$ fraction of the probability (explanation follows).

When saying that a list of polynomials explains almost all of the success, we mean that, with high probability over the random bits of the tester (i.e., over the choice of a subspace and a point within it), either the tester rejects or one of the polynomials agrees with the oracle on the subspace and on the point. There is a trade-off between the amount of success explained and the length of the list: the more one wishes to explain, the longer the list is.

We wish ε to be as small as possible. The parameter ε we achieve depends on $\frac{md}{|\mathbb{F}|}$. This comes from the use of the Schwartz–Zippel lemma. It also depends on $\frac{1}{|\mathbb{H}|}$, which is the price we pay for considering the subfield $\mathbb{H}$ instead of the entire field $\mathbb{F}$.

The statement for the randomness-efficient plane vs. point tester is as follows. Note that we make no effort to optimize the constants.

THEOREM 1 (plane vs. point soundness). *Fix a dimension $m \geq 2$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$, and a degree d. Denote $\varepsilon \stackrel{def}{=} 2^7 m \left(\sqrt[8]{\frac{1}{|\mathbb{H}|}} + \sqrt[4]{\frac{md}{|\mathbb{F}|}}\right)$. For every oracle $\mathcal{A}$:*

1. (Decoding) *There exists a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q \leq md$, such that*

$$\Pr_{\vec{x} \in \mathbb{F}^m} [Q(\vec{x}) = \mathcal{A}(\vec{x})] \geq \Pr_{\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m} \left[PlanePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2)\right] - \varepsilon;$$

2. (List decoding) *For every $\delta > 2\varepsilon$, there exist $t \leq 2/\delta$ polynomials $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q_i \leq md$, such that*

$$\Pr_{\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m} \left[\neg PlanePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2) \ \vee \ \exists i \ (Q_i \equiv \mathcal{A})(\text{affine}(\vec{z}; \vec{y}_1, \vec{y}_2))\right]$$
$$\geq \quad 1 - \delta - 2\varepsilon.$$

We prove a similar theorem for the randomness-efficient subspace vs. point tester. Note that for this tester we manage to show agreement with polynomials of degree at most d rather than md.

THEOREM 2 (subspace vs. point soundness). *Fix a dimension $m \geq 3$, a field $\mathbb{F}$, a subfield $\mathbb{H} \subseteq \mathbb{F}$, and a degree d. Denote $\varepsilon \stackrel{def}{=} 2^7 m \left(\sqrt[8]{\frac{1}{|\mathbb{H}|}} + \sqrt[4]{\frac{md}{|\mathbb{F}|}}\right)$. For every oracle $\mathcal{A}$:*

1. (Decoding) *There exists a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q \leq d$, such that*

$$\Pr_{\vec{x} \in \mathbb{F}^m} [Q(\vec{x}) = \mathcal{A}(\vec{x})] \geq \Pr_{\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m} \left[SpacePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2)\right] - 3\varepsilon;$$

2. (List decoding) *For every $\delta > 3\varepsilon$, there exist $t \leq 2/\delta$ polynomials $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q_i \leq d$, such that*

$$\Pr_{\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m} \left[\neg SpacePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2) \ \vee \ \exists i \ (Q_i \equiv \mathcal{A})(\text{affine}(\vec{0}; \vec{z}, \vec{y}_1, \vec{y}_2))\right]$$
$$\geq 1 - \delta - 3\varepsilon.$$

It is interesting to note that our sampling arguments also imply a converse to the above theorems: for any polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q \leq d$, there exists an oracle $\mathcal{A}'$ agreeing with $\mathcal{A}$ on the points and assigning affine subspaces polynomials of degree at most d such that both of our testers accept with probability at least $\Pr_{\vec{x} \in \mathbb{F}^m} [Q(\vec{x}) = \mathcal{A}(\vec{x})] - \varepsilon$ when given access to $\mathcal{A}'$.

3. Preliminaries.

3.1. Orthogonality and vector spaces. Given a vector $\vec{y} \in \mathbb{F}^m$, we write $\vec{y} = (y_1, \ldots, y_m)$. For a sequence of vectors $\vec{y}_1, \ldots, \vec{y}_k$, we write, for every $1 \leq i \leq k$, $\vec{y}_i = (y_{i,1}, \ldots, y_{i,m})$.

We define an *inner product* between two vectors $\vec{x}, \vec{y} \in \mathbb{F}^m$ as $(\vec{x}, \vec{y}) \stackrel{def}{=} \sum_{i=1}^m x_i \cdot y_i$. We say that $\vec{x}, \vec{y}$ are *orthogonal* if $(\vec{x}, \vec{y}) = 0$.

PROPOSITION 3.1. *For every $\vec{y} \neq \vec{0} \in \mathbb{F}^m$, for every $c \in \mathbb{F}$,*

$$\Pr_{\vec{z} \in \mathbb{H}^m} [(\vec{z}, \vec{y}) = c] \leq \frac{1}{|\mathbb{H}|}.$$

Proof. As $\vec{y} \neq \vec{0} \in \mathbb{F}^m$, there exists $1 \leq i \leq m$ such that $y_i \neq 0$. For every fixing of all $\vec{z}$'s coordinates but the ith, the condition $(\vec{z}, \vec{y}) = c$ uniquely determines z_i to

some scalar in $\mathbb{F}$. This scalar may or may not be in the subfield $\mathbb{H}$, but, in any case, there exists at most one possibility for $z_i \in \mathbb{H}$. ∎

PROPOSITION 3.2. *For every $\vec{y} \neq \vec{0} \in \mathbb{F}^m$, for every $k < m$,*

$$\Pr_{\vec{y}_1,\ldots,\vec{y}_k \in \mathbb{H}^m} [\vec{y} \in span\{\vec{y}_1,\ldots,\vec{y}_k\} \mid \mathit{ind}(\vec{y}_1,\ldots,\vec{y}_k)] \leq \frac{1}{|\mathbb{H}|}.$$

Proof. Consider uniformly distributed linearly independent $\vec{y}_1,\ldots,\vec{y}_k \in \mathbb{H}^m$. Pick uniformly and independently at random a vector $\vec{z} \neq \vec{0} \in \mathbb{H}^m$ that is orthogonal to $\vec{y}_1,\ldots,\vec{y}_k$ (there exist such vectors since $k < m$). Note that for every $\vec{y} \in span\{\vec{y}_1,\ldots,\vec{y}_k\}$ it holds that $(\vec{z},\vec{y}) = 0$. By Proposition 3.1, since $\vec{z}$ is uniformly distributed over $\mathbb{H}^m \setminus \{\vec{0}\}$, this happens with probability at most $\frac{1}{|\mathbb{H}|}$. ∎

PROPOSITION 3.3. *For every subset $A \subseteq \mathbb{F}^m$ with $|A| > |\mathbb{F}|^{m-1}$, there exist linearly independent $\vec{y}_1,\ldots,\vec{y}_m \in \mathbb{F}^m$ such that, for every $1 \leq i \leq m$, $\vec{y}_i \in A$.*

Proof. We have

$$|span(A)| \geq |A| > |\mathbb{F}|^{m-1}.$$

Since $span(A)$ is a linear subspace in $\mathbb{F}^m$, we must have $|span(A)| = |\mathbb{F}|^m$. Thus, $span(A) = \mathbb{F}^m$, and so A contains a basis for $\mathbb{F}^m$. ∎

3.2. Polynomials. The Schwartz–Zippel lemma shows that different low degree polynomials differ on most points,

PROPOSITION 3.4 (Schwartz–Zippel). *For two different polynomials $Q, P : \mathbb{F}^m \to \mathbb{F}$ with $\deg Q, \deg P \leq d$,*

$$\Pr_{\vec{x} \in \mathbb{F}^m} [Q(\vec{x}) = P(\vec{x})] \leq \frac{d}{|\mathbb{F}|}.$$

The Schwartz–Zippel lemma can be viewed as showing the unique-decoding property of the Reed–Muller code. This immediately implies a list-decoding property, namely, that only a few polynomials can agree with a function on many of the points.

We include a simple proof of this property.

PROPOSITION 3.5 (list decoding). *Fix a finite field $\mathbb{F}$ and a dimension m. Let $f : \mathbb{F}^m \to \mathbb{F}$ be some function, and consider some degree $d \leq |\mathbb{F}|$. Then, for any $\delta \geq 2\sqrt{\frac{d}{|\mathbb{F}|}}$, if $Q_1,\ldots,Q_l : \mathbb{F}^m \to \mathbb{F}$ are* different *polynomials of degree at most d, and for every $1 \leq i \leq l$, the polynomial Q_i agrees with f on at least a δ fraction of the points, i.e., $\Pr_{\vec{x} \in \mathbb{F}^m} [Q_i(\vec{x}) = f(\vec{x})] \geq \delta$, then $l \leq \frac{2}{\delta}$.*

Proof. Let $\delta \geq 2\sqrt{\frac{d}{|\mathbb{F}|}}$, and assume by way of contradiction that there exist $l = \lfloor \frac{2}{\delta} \rfloor + 1$ different polynomials $Q_1,\ldots,Q_l : \mathbb{F}^m \to \mathbb{F}$ as stated.

For every $1 \leq i \leq l$, let $A_i \stackrel{def}{=} \{\vec{x} \in \mathbb{F}^m \mid Q_i(\vec{x}) = f(\vec{x})\}$. By inclusion-exclusion,

$$|\mathbb{F}^m| \geq \left|\bigcup_{i=1}^{l} A_i\right| \geq \sum_{i=1}^{l} |A_i| - \sum_{i \neq j} |A_i \cap A_j|.$$

By Schwartz–Zippel, for every $1 \leq i \neq j \leq l$, $|A_i \cap A_j| \leq \frac{d}{|\mathbb{F}|} \cdot |\mathbb{F}^m|$. Therefore, by the premise,

$$|\mathbb{F}^m| \geq l\delta\,|\mathbb{F}^m| - \binom{l}{2} \frac{d}{|\mathbb{F}|}\,|\mathbb{F}^m|.$$

On one hand, since $l > \frac{2}{\delta}$, we get $l\delta > 2$. On the other hand, since $\frac{2}{\delta} \leq \sqrt{\frac{|\mathbb{F}|}{d}}$ and $d \leq |\mathbb{F}|$, we get $\binom{l}{2} \leq \frac{|\mathbb{F}|}{d}$. This results in a contradiction. ∎

4. Affine subspaces with directions over a subfield. In this section we prove basic facts regarding affine subspaces in $\mathbb{F}^m$ that are spanned by directions over a subfield $\mathbb{H} \subseteq \mathbb{F}$. All of the properties we prove for such subspaces are well known when $\mathbb{H} = \mathbb{F}$.

For $0 \leq k \leq m$, consider the set of representations of affine subspaces with directions over a subfield

$$\mathcal{R}_k^m \stackrel{def}{=} \{(\vec{z}; \vec{y}_1, \ldots, \vec{y}_k) \mid \vec{z} \in \mathbb{F}^m, \vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m, \mathsf{ind}(\vec{y}_1, \ldots, \vec{y}_k)\}.$$

The corresponding set of affine subspaces is

$$\mathcal{S}_k^m \stackrel{def}{=} \{\text{affine}(r) \mid r \in \mathcal{R}_k^m\}.$$

First we would like to assert that every subspace in $\mathcal{S}_k^m$ is associated with the same number of tuples in $\mathcal{R}_k^m$ and that every subspace in $\mathcal{S}_k^m$ contains the same number of subspaces in $\mathcal{S}_{k'}^m$ for $k' \leq k$.

PROPOSITION 4.1 (uniformity). *For every dimension k, there is a number $T = T(k)$ such that, for every $s \in \mathcal{S}_k^m$, $|\{r \in \mathcal{R}_k^m \mid s = \text{affine}(r)\}| = T$.*

PROPOSITION 4.2 (uniformity downwards). *For every dimension $k' \leq k$, there is a number $T = T(k, k')$ such that, for every $s \in \mathcal{S}_k^m$, $|\{s' \in \mathcal{S}_{k'}^m \mid s' \subseteq s\}| = T$.*

To prove both assertions we introduce an additional notation allowing us to refer to affine subspaces in $\mathcal{S}_k^m$ as isomorphic copies of $\mathbb{F}^k$. Fix an affine subspace together with a representation for it: $s = \text{affine}(\vec{z}; \vec{y}_1, \ldots, \vec{y}_k)$. For a representation $r = (\vec{\alpha}_0; \vec{\alpha}_1, \ldots, \vec{\alpha}_{k'})$ of a k'-dimensional affine subspace within $\mathbb{F}^k$, we define the representation r *relative to* (the representation of) the space s by

$$r_s \stackrel{def}{=} \left(\vec{z} + \sum_{i=1}^{k} \vec{\alpha}_{0,i} \vec{y}_i \;;\; \sum_{i=1}^{k} \vec{\alpha}_{1,i} \vec{y}_i, \ldots, \sum_{i=1}^{k} \vec{\alpha}_{k',i} \vec{y}_i \right).$$

Note that, since $\vec{y}_1, \ldots, \vec{y}_k$ are linearly independent, if two representations r, r' are the same relative to a subspace s, $r_s = r'_s$, then they are the same representation $r = r'$.

Denote the corresponding relative affine subspace:

$$\text{affine}_s(r) \stackrel{def}{=} \text{affine}(r_s).$$

Note that, for every r, $\text{affine}_s(r) \subseteq s$. Moreover, if $\text{affine}(r) = \text{affine}(r')$, then $\text{affine}_s(r) = \text{affine}_s(r')$. Now the above two propositions follow from the following proposition.

PROPOSITION 4.3. *For every subspace $s \in \mathcal{S}_k^m$, for every dimension $k' \leq k$,*

$$S_1 \stackrel{def}{=} |\{r \in \mathcal{R}_{k'}^m \mid \text{affine}(r) \subseteq s\}| = \left|\mathcal{R}_{k'}^k\right| \stackrel{def}{=} S_2.$$

Proof. Fix a subspace $s \in \mathcal{S}_k^m$, and fix a tuple $(\vec{z}; \vec{y}_1, \ldots, \vec{y}_k) \in \mathcal{R}_k^m$, with $s = \text{affine}(\vec{z}; \vec{y}_1, \ldots, \vec{y}_k)$.

1. $S_1 \geq S_2$: For every tuple $r = (\vec{\alpha}_0; \vec{\alpha}_1, \ldots, \vec{\alpha}_{k'}) \in \mathcal{R}_{k'}^k$, the tuple r_s satisfies $r_s \in \mathcal{R}_{k'}^m$ and $\text{affine}(r_s) \subseteq s$.
2. $S_1 \leq S_2$: For every tuple $r \in \mathcal{R}_{k'}^m$ satisfying $\text{affine}(r) \subseteq s$, there exists exactly one $\alpha = (\vec{\alpha}_0; \vec{\alpha}_1, \ldots, \vec{\alpha}_{k'})$, $\vec{\alpha}_0, \vec{\alpha}_1, \ldots, \vec{\alpha}_{k'} \in \mathbb{F}^k$, $\mathsf{ind}(\vec{\alpha}_1, \ldots, \vec{\alpha}_{k'})$, such that $r = \alpha_s$. Since $r \in \mathcal{R}_{k'}^m$ and $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$, also $\vec{\alpha}_1, \ldots, \vec{\alpha}_{k'} \in \mathbb{H}^k$. ∎

Every subspace in $\mathcal{S}_k^m$ is contained in the same number of subspaces in $\mathcal{S}_{k'}^m$ for $k' \geq k$.

PROPOSITION 4.4 (uniformity upwards). *For every dimension $k \leq k' \leq m$, there is a number $T = T(m,k,k')$ such that, for every subspace $s \in \mathcal{S}_k^m$,*

$$|\{s' \in \mathcal{S}_{k'}^m \mid s' \supseteq s\}| = T.$$

Proof. Let us introduce an additional piece of notation: $\mathcal{L}_{k'}^m$ is the set of all linear subspaces of dimension k' in $\mathbb{F}^m$ spanned by vectors from $\mathbb{H}^m$.

Fix $s = \text{affine}(\vec{z}; \vec{y}_1, \ldots, \vec{y}_k) \in \mathcal{S}_k^m$. Since $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$ are linearly independent, the proposition will clearly follow if we prove the following:

$$S_1 \stackrel{def}{=} |\{s' \in \mathcal{S}_{k'}^m \mid s' \supseteq s\}| = |\{Y' \in \mathcal{L}_{k'}^m \mid Y' \supseteq \{\vec{y}_1, \ldots, \vec{y}_k\}\}| \stackrel{def}{=} S_2.$$

1. $S_1 \leq S_2$: Let $s' = \text{affine}(\vec{z}'; \vec{y}'_1, \ldots, \vec{y}'_{k'}) \in \mathcal{S}_{k'}^m$, $(\vec{z}'; \vec{y}'_1, \ldots, \vec{y}'_{k'}) \in \mathcal{R}_{k'}^m$, $s' \supseteq s$. Let $Y' = span\{\vec{y}'_1, \ldots, \vec{y}'_{k'}\}$. Clearly, Y' is in $\mathcal{L}_{k'}^m$, and Y' is uniquely defined by s', $s' = \vec{z}' + Y'$. It holds that $\vec{z} \in s \subseteq s' = \vec{z}' + Y'$; thus $\vec{z}' \in \vec{z} + Y'$, and, hence, $s' = \vec{z} + Y'$. Let $1 \leq i \leq k$. It holds that $\vec{z} + \vec{y}_i \in s \subseteq s'$. This implies that $\vec{z} + \vec{y}_i \in \vec{z} + Y'$. Hence, $\vec{y}_i \in Y'$. Therefore, $\{\vec{y}_1, \ldots, \vec{y}_k\} \subseteq Y'$.
2. $S_1 \geq S_2$: Let $Y' \in \mathcal{L}_{k'}^m$, $Y' \supseteq \{\vec{y}_1, \ldots, \vec{y}_k\}$. Clearly, $\vec{z} + Y' \in \mathcal{S}_{k'}^m$ and $s \subseteq \vec{z} + Y'$. ∎

Uniformity is so important because it allows us to count in several ways. A simple argument of this nature is that the fraction of affine subspaces $s \in \mathcal{S}_k^m$ satisfying some condition is exactly the same as the fraction of $r \in \mathcal{R}_k^m$ such that affine(r) satisfies the condition. Let us demonstrate a more sophisticated argument of this nature. Fix $k' \leq k$. Suppose that we have a predicate R indicating whether an affine subspace $s \in \mathcal{S}_k^m$ and an affine subspace $s' \in \mathcal{S}_{k'}^m$ *contained in it*, $s' \subseteq s$, satisfy some relation. Then

$$\mathop{\mathbf{E}}_{s}\left[\Pr_{s' \subseteq s}[R(s,s')]\right] = \mathop{\mathbf{E}}_{s'}\left[\Pr_{s \supseteq s'}[R(s,s')]\right].$$

The intersection between two subspaces $s_1 \in \mathcal{S}_{k^{(1)}}^m$ and $s_2 \in \mathcal{S}_{k^{(2)}}^m$ is again (provided it is not empty) a subspace in $\mathcal{S}_{k^{(3)}}^m$ for some $k^{(3)}$.

PROPOSITION 4.5 (closure under intersection). *If $s_1 \in \mathcal{S}_{k^{(1)}}^m$ and $s_2 \in \mathcal{S}_{k^{(2)}}^m$, where $s_1 \cap s_2 \neq \phi$, then there exists $k^{(3)}$ such that $s_1 \cap s_2 \in \mathcal{S}_{k^{(3)}}^m$.*

Proof. Write $s_1 = \vec{z}_1 + V_1$ and $s_2 = \vec{z}_2 + V_2$, where $\vec{z}_1, \vec{z}_2 \in \mathbb{F}^m$ and $V_1, V_2 \subseteq \mathbb{F}^m$ are linear subspaces spanned by vectors in $\mathbb{H}^m$. Assume that $\vec{x} \in s_1 \cap s_2$. Then we can alternatively write $s_1 = \vec{x} + V_1$ and $s_2 = \vec{x} + V_2$. Thus, $s_1 \cap s_2 = \vec{x} + (V_1 \cap V_2)$. The proposition follows by noticing that $V_1 \cap V_2$ can be spanned by vectors in $\mathbb{H}^m$. ∎

A useful representation of affine subspaces is given in the following proposition.

PROPOSITION 4.6 (affine subspaces as solutions of linear equations). *Let $s = \text{affine}(\vec{z}; \vec{y}_1, \ldots, \vec{y}_k) \in \mathcal{S}_k^m$, and let $\vec{\alpha}_1, \ldots, \vec{\alpha}_{m-k} \in \mathbb{H}^m$ be $(m-k)$ linearly independent vectors orthogonal to $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$. Then*

$$s = \{\vec{x} \in \mathbb{F}^m \mid \forall 1 \leq j \leq m-k,\ (\vec{x}, \vec{\alpha}_j) = (\vec{z}, \vec{\alpha}_j)\}.$$

Proof. Fix $\vec{x} \in s$. Hence, there exists $\vec{c} \in \mathbb{F}^k$ such that $\vec{x} = \vec{z} + \sum_{i=1}^k c_i \vec{y}_i$. For every $1 \leq j \leq m-k$,

$$(\vec{x}, \vec{\alpha}_j) = \left(\vec{z} + \sum_{i=1}^k c_i \vec{y}_i, \vec{\alpha}_j\right) = (\vec{z}, \vec{\alpha}_j) + \sum_{i=1}^k c_i \cdot (\vec{y}_i, \vec{\alpha}_j) = (\vec{z}, \vec{\alpha}_j).$$

Thus, $s \subseteq \{\vec{x} \in \mathbb{F}^m \mid \forall 1 \leq j \leq m-k,\ (\vec{x}, \vec{\alpha}_j) = (\vec{z}, \vec{\alpha}_j)\}$. The proposition follows by noticing that, in addition, the two sets are of size $\left|\mathbb{F}^k\right|$. □

5. Affine subspaces with directions over a subfield sample well. We say that an affine subspace s in $\mathbb{F}^m$ samples a set $A \subseteq \mathbb{F}^m$ well if the fraction of points from A contained in it, i.e., $\frac{|s \cap A|}{|s|}$, is approximately $\frac{|A|}{|\mathbb{F}^m|}$. We say that a distribution $\mathcal{D}$ on affine subspaces in $\mathbb{F}^m$ samples well if, no matter how one fixes a large enough subset $A \subseteq \mathbb{F}^m$, a random subspace $s \sim \mathcal{D}$ samples A well with high probability. In this section we use Fourier analysis to show that the distributions induced by our testers sample well.

5.1. Fourier transform. Let $(G, +)$ be a finite Abelian group. Consider functions from the group to the complex numbers $f : G \to \mathbb{C}$. One example for such a function is the indicator function of a multiset $A \subseteq G$, i.e., the function $\mathcal{I}_A$ that assigns every $\vec{x} \in G$ its multiplicity in A.

We define an inner product between functions $f, g : G \to \mathbb{C}$ as

$$\langle f, g \rangle \stackrel{def}{=} \frac{1}{|G|} \sum_{x \in G} f(x)\overline{g(x)}.$$

A *character* of G is a homomorphism $\chi : G \to \mathbb{C}^*$, where $\mathbb{C}^*$ is the multiplicative group of the complex numbers. Namely, for every $x, y \in G$,

$$\chi(x + y) = \chi(x) \cdot \chi(y).$$

Every group G trivially has identically 1 function as a character.

It can be shown that the set of all characters of G forms an orthonormal basis for the space of all functions $f : G \to \mathbb{C}$ under the inner product defined above. Hence, every function $f : G \to \mathbb{C}$ can be equivalently represented as $f(x) = \sum_\chi \hat{f}(\chi) \cdot \chi(x)$, where $\hat{f}(\chi) \stackrel{def}{=} \langle f, \chi \rangle$ is called the *Fourier coefficient* of f corresponding to the character χ. The linear transformation from f to $\hat{f}$ is called the *Fourier transform* of f.

We will need two basic facts regarding the Fourier transform.

PROPOSITION 5.1 (Parseval's identity). *For two functions $f, g : G \to \mathbb{C}$, $\langle f, g \rangle = |G| \cdot \langle \hat{f}, \hat{g} \rangle = \sum_\chi \hat{f}(\chi)\overline{\hat{g}(\chi)}$.*

Define the *convolution* of two functions $f, g : G \to \mathbb{C}$, denoted $(f * g) : G \to \mathbb{C}$, as $(f * g)(x) \stackrel{def}{=} \frac{1}{|G|} \sum_{y \in G} f(y)g(x - y)$.

PROPOSITION 5.2 (convolution formula). *Fix two functions $f, g : G \to \mathbb{C}$. For every character χ of G, $\widehat{(f * g)}(\chi) = \hat{f}(\chi) \cdot \hat{g}(\chi)$.*

We focus on the additive group $G = \mathbb{F}^m$ for some finite field $\mathbb{F} = GF(p^k)$. The field $\mathbb{F}$ is also viewed as a vector space of dimension k over the field $GF(p)$.

Denote $\omega_p = e^{2\pi i/p}$ the pth *primitive root of unity* in $\mathbb{C}$. For every $\alpha \in \mathbb{F}^m$, there is a character $\chi_\alpha : \mathbb{F}^m \to \mathbb{C}$,

$$\chi_\alpha(x) \stackrel{def}{=} \omega_p^{\sum_{i=1}^m (\alpha_i, x_i)}.$$

Note that we view α_i, x_i as vectors in $GF(p)^k$. Their inner product is in $GF(p)$ and so is the sum in the above expression.

For a function $f : \mathbb{F}^m \to \mathbb{C}$, we denote its Fourier coefficient corresponding to the character χ_α by $\hat{f}(\alpha)$.

5.2. Sampling lemma. In this subsection we prove our basic lemma via Fourier analysis. Given $z, y \in \mathbb{F}^m$ and a subset $A \subseteq \mathbb{F}^m$, define $X_{z,y}$ to be the number of $c \in \mathbb{F}$ satisfying $z + c \cdot y \in A$. Clearly, the expectation of $X_{z,y}$ when picking independently at random $z \in \mathbb{F}^m$ and $y \in \mathbb{H}^m$ is $|\mathbb{F}| \cdot \frac{|A|}{|\mathbb{F}^m|}$. We bound the variance of $X_{z,y}$, implying that it is concentrated around its expectation.

LEMMA 5.3. *For any subset $A \subseteq \mathbb{F}^m$ of density $\mu = |A|/|\mathbb{F}^m|$,*

$$\mathop{\mathbf{Var}}_{z \in \mathbb{F}^m, y \in \mathbb{H}^m}[X_{z,y}] \leq |\mathbb{F}|^2 \frac{\mu}{|\mathbb{H}|}.$$

Proof. If we denote the indicator function of A by $\mathcal{I}_A$, and the indicator function of the multiset $\{c \cdot y \mid c \in \mathbb{F}\}$ by $\mathcal{I}_{\mathbb{F}y}$, we can express:

$$X_{z,y} = \sum_{x \in \mathbb{F}^m} \mathcal{I}_A(x)\mathcal{I}_{\mathbb{F}y}(z-x) = |\mathbb{F}^m| \cdot (\mathcal{I}_A * \mathcal{I}_{\mathbb{F}y})(z).$$

Hence, by Parseval's identity and the convolution formula,

$$\begin{aligned}
\mathop{\mathbf{E}}_{z \in \mathbb{F}^m, y \in \mathbb{H}^m}\left[X_{z,y}^2\right] &= \frac{1}{|\mathbb{F}^m|\,|\mathbb{H}^m|} \cdot \sum_{y \in \mathbb{H}^m} \sum_{z \in \mathbb{F}^m} \left(|\mathbb{F}^m|\,(\mathcal{I}_A * \mathcal{I}_{\mathbb{F}y})(z)\right)^2 \\
&= \frac{|\mathbb{F}^m|^2}{|\mathbb{H}^m|} \cdot \sum_{y \in \mathbb{H}^m} \sum_{\alpha \in \mathbb{F}^m} \left|\widehat{(\mathcal{I}_A * \mathcal{I}_{\mathbb{F}y})}(\alpha)\right|^2 \\
&= \frac{|\mathbb{F}^m|^2}{|\mathbb{H}^m|} \cdot \sum_{y \in \mathbb{H}^m} \sum_{\alpha \in \mathbb{F}^m} \left|\hat{\mathcal{I}}_A(\alpha)\right|^2 \cdot \left|\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)\right|^2.
\end{aligned}$$

By definition, for any multiset $S \subseteq \mathbb{F}^m$, $\hat{\mathcal{I}}_S(\vec{0}) = \frac{|S|}{|\mathbb{F}^m|}$ (where $|S| = \sum_{\vec{x} \in \mathbb{F}^m} \mathcal{I}_S(\vec{x})$), and hence

$$\begin{aligned}
\mathop{\mathbf{E}}_{z \in \mathbb{F}^m, y \in \mathbb{H}^m}\left[X_{z,y}^2\right] &= \frac{|\mathbb{F}^m|^2}{|\mathbb{H}^m|} \cdot \sum_{y \in \mathbb{H}^m} \left(\left|\hat{\mathcal{I}}_A(\vec{0})\right|^2 \cdot \left|\hat{\mathcal{I}}_{\mathbb{F}y}(\vec{0})\right|^2 + \sum_{\alpha \neq \vec{0} \in \mathbb{F}^m} \left|\hat{\mathcal{I}}_A(\alpha)\right|^2 \cdot \left|\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)\right|^2\right) \\
&= \left(\frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}\right)^2 + \sum_{\alpha \neq \vec{0} \in \mathbb{F}^m} \left(\left|\hat{\mathcal{I}}_A(\alpha)\right|^2 \cdot \frac{|\mathbb{F}^m|^2}{|\mathbb{H}^m|} \sum_{y \in \mathbb{H}^m} \left|\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)\right|^2\right).
\end{aligned}$$

We will show that $\frac{|\mathbb{F}^m|^2}{|\mathbb{H}^m|} \cdot \sum_{y \in \mathbb{H}^m} |\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)|^2 \leq \frac{|\mathbb{F}|^2}{|\mathbb{H}|}$. Let us see how the lemma follows. Bu using this bound and applying Parseval's identity again, we get

$$\begin{aligned}
\mathop{\mathbf{E}}_{z \in \mathbb{F}^m, y \in \mathbb{H}^m}\left[X_{z,y}^2\right] &\leq \left(\frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}\right)^2 + \frac{|\mathbb{F}|^2}{|\mathbb{H}|} \cdot \sum_{\alpha \neq \vec{0} \in \mathbb{F}^m} \left|\hat{\mathcal{I}}_A(\alpha)\right|^2 \\
&\leq \left(\frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}\right)^2 + \frac{|\mathbb{F}|^2}{|\mathbb{H}|} \cdot \frac{1}{|\mathbb{F}^m|} \cdot \sum_{z \in \mathbb{F}^m} |\mathcal{I}_A(z)|^2 \\
&= \left(\frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}\right)^2 + \frac{|\mathbb{F}|^2}{|\mathbb{H}|} \cdot \frac{|A|}{|\mathbb{F}^m|}.
\end{aligned}$$

By linearity of expectations,

$$\mathop{\mathbf{E}}_{z \in \mathbb{F}^m, y \in \mathbb{H}^m}[X_{z,y}] = \frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}.$$

Therefore,

$$\begin{aligned}\operatorname*{\mathbf{Var}}_{z\in\mathbb{F}^m,y\in\mathbb{H}^m}[X_{z,y}] &= \operatorname*{\mathbf{E}}_{z,y}\left[X_{z,y}^2\right] - \operatorname*{\mathbf{E}}_{z,y}[X_{z,y}]^2 \\ &\le \left(\frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}\right)^2 + \frac{|\mathbb{F}|^2}{|\mathbb{H}|}\cdot\frac{|A|}{|\mathbb{F}^m|} - \left(\frac{|\mathbb{F}|\,|A|}{|\mathbb{F}^m|}\right)^2 \\ &= |\mathbb{F}|^2\,\frac{\mu}{|\mathbb{H}|}.\end{aligned}$$

We conclude that proving the lemma boils down to proving the following.

CLAIM 5.3.1. *For every $\alpha \neq \vec{0} \in \mathbb{F}^m$,*

$$\frac{1}{|\mathbb{H}^m|}\cdot\sum_{y\in\mathbb{H}^m}\left|\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)\right|^2 \le \frac{|\mathbb{F}|^2}{|\mathbb{F}^m|^2}\cdot\frac{1}{|\mathbb{H}|}.$$

Proof. Assume that $\mathbb{F} = GF(p^k)$. Fix some $\alpha \neq \vec{0} \in \mathbb{F}^m$.

$$\left|\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)\right| = |\langle \mathcal{I}_{\mathbb{F}y}, \chi_\alpha\rangle| = \left|\frac{1}{|\mathbb{F}^m|}\cdot\sum_{z\in\mathbb{F}^m}\mathcal{I}_{\mathbb{F}y}(z)\omega_p^{-\sum_{i=1}^m(\alpha_i,z_i)}\right| = \left|\frac{1}{|\mathbb{F}^m|}\cdot\sum_{c\in\mathbb{F}}\omega_p^{-\sum_{i=1}^m(\alpha_i,c\cdot y_i)}\right|.$$

Multiplication by a field element $a \in \mathbb{F}$ in the field $\mathbb{F} = GF(p^k)$ corresponds to a linear transformation in the vector space $GF(p)^k$. That is, for every $a \in \mathbb{F}$, there exists a $k \times k$ matrix M_a over $GF(p)$ such that, for every $b \in \mathbb{F} = GF(p)^k$, $a \cdot b = M_a b$. Hence,

$$\begin{aligned}\sum_{i=1}^m(\alpha_i, c\cdot y_i) &= \sum_{i=1}^m(\alpha_i, M_{y_i}c) \\ &= \sum_{i=1}^m(M_{y_i}^T\alpha_i, c) \\ &= \sum_{i=1}^m(M_{y_i}^T\alpha_i, c).\end{aligned}$$

Thus, for every $y \in \mathbb{H}^m$,

$$\left|\hat{\mathcal{I}}_{\mathbb{F}y}(\alpha)\right| = \begin{cases} 0 & \sum_{i=1}^m M_{y_i}^T\alpha_i \neq \vec{0}, \\ \frac{|\mathbb{F}|}{|\mathbb{F}^m|} & \text{otherwise.}\end{cases}$$

Assume that $1 \le i \le m$ is such that $\alpha_i \neq \vec{0} \in GF(p)^k$. Note that, for every $a_1 \neq a_2 \in \mathbb{F}$, we know that $M_{a_1}^T\alpha_i \neq M_{a_2}^T\alpha_i$. (For every $b \neq 0 \in \mathbb{F}$, $a_1 \cdot b \neq a_2 \cdot b$. Thus, for every $b \neq \vec{0} \in GF(p)^k$, $(M_{a_1} - M_{a_2})b \neq \vec{0}$ and for every $b \neq \vec{0} \in GF(p)^k$, $M_{a_1}^T b - M_{a_2}^T b = (M_{a_1} - M_{a_2})^T b \neq \vec{0}$.) Hence, for every $v \in GF(p)^k$, there exists at most one $a \in \mathbb{H}$ for which $M_a^T\alpha_i = v$. In particular,

$$\Pr_{\vec{y}\in\mathbb{H}^m}\left[M_{y_i}^T\alpha_i = -\sum_{j\neq i}M_{y_j}^T\alpha_j\right] \le \frac{1}{|\mathbb{H}|}.$$

The claim follows. □

Remark 5.4. To get the corollaries stated in the introduction, note the following:

1. Only Claim 5.3.1 uses the nature/structure of $\mathbb{H}^m$.
2. Claim 5.3.1 does not require $\mathbb{H}$ to be a subfield of $\mathbb{F}$. Its proof holds for any subset $H \subseteq \mathbb{F}$.
3. If $\mathbb{F} = GF(p)$, then for any subset $S \subseteq \mathbb{F}^m$, for any $\vec{\alpha} \neq \vec{0} \in \mathbb{F}^m$,

$$\Pr_{\vec{y}\in S}\left[\sum_{i=1}^{m} M_{y_i}^T \alpha_i \neq \vec{0}\right] = \Pr_{\vec{y}\in S}\left[\sum_{i=1}^{m} y_i \cdot \alpha_i \neq 0\right] = \Pr_{\vec{y}\in S}\left[(\vec{y},\vec{\alpha}) \neq 0\right].$$

We have that $\min_{\vec{\alpha}\neq\vec{0}\in\mathbb{F}^m} \Pr_{\vec{y}\in S}\left[(\vec{y},\vec{\alpha}) \neq 0\right]$ is the relative distance of the linear code obtained when using the vectors of S as the rows of a generating matrix.

5.3. Affine subspaces sample well. By using the sampling lemma (Lemma 5.3), we can prove that the uniform distribution over lines in $\mathcal{S}_1^m$ samples well. Note that the sampling lemma does not show exactly this, as it considers y uniformly distributed over $\mathbb{H}^m$ instead of over $\mathbb{H}^m \setminus \{\vec{0}\}$.

LEMMA 5.5. *For any $A \subseteq \mathbb{F}^m$ of density $\mu = |A|/|\mathbb{F}^m|$,*

$$\operatorname*{\mathbf{Var}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right] \leq \frac{\mu}{|\mathbb{H}|}.$$

Proof. Note that the probability that a random point in $\mathbb{F}^m$ is in A is the same as the expected fraction of points in A on a random line in $\mathcal{S}_1^m$:

$$\operatorname*{\mathbf{E}}_{p\in\mathcal{S}_0^m}\left[\frac{|p\cap A|}{|p|}\right] = \operatorname*{\mathbf{E}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right] = \mu,$$

but the variance may only decrease when considering lines rather than points:

$$\operatorname*{\mathbf{Var}}_{p\in\mathcal{S}_0^m}\left[\frac{|p\cap A|}{|p|}\right] \geq \operatorname*{\mathbf{Var}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right].$$

Hence, since $\mathbf{Var}[X] = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right]$ and expectations satisfy that, for every random variable Y and set A, $\mathbf{E}[Y] = \Pr[Y \in A]\cdot\mathbf{E}[Y|Y\in A] + \Pr[Y\notin A]\cdot\mathbf{E}[Y|Y\notin A]$,

$$\begin{aligned}
\operatorname*{\mathbf{Var}}_{z\in\mathbb{F}^m, y\in\mathbb{H}^m}\left[\frac{1}{|\mathbb{F}|}\cdot X_{z,y}\right] &= \frac{1}{|\mathbb{H}|^m}\cdot\operatorname*{\mathbf{Var}}_{p\in\mathcal{S}_0^m}\left[\frac{|p\cap A|}{|p|}\right] + \left(1-\frac{1}{|\mathbb{H}|^m}\right)\cdot\operatorname*{\mathbf{Var}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right] \\
&\geq \frac{1}{|\mathbb{H}|^m}\cdot\operatorname*{\mathbf{Var}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right] + \left(1-\frac{1}{|\mathbb{H}|^m}\right)\cdot\operatorname*{\mathbf{Var}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right] \\
&= \operatorname*{\mathbf{Var}}_{l\in\mathcal{S}_1^m}\left[\frac{|l\cap A|}{|l|}\right].
\end{aligned}$$

The lemma follows from Lemma 5.3. □

By using the analysis for dimension 1, we can bound the variance of the hitting rate for any larger dimension,

LEMMA 5.6. *Fix dimensions k and m, $1 \leq k \leq m$. For any $A \subseteq \mathbb{F}^m$ of density $\mu = |A|/|\mathbb{F}^m|$,*

$$\operatorname*{\mathbf{Var}}_{s\in\mathcal{S}_k^m}\left[\frac{|s\cap A|}{|s|}\right] \leq \frac{\mu}{|\mathbb{H}|}.$$

Proof. Pick $s \in \mathcal{S}_k^m$ and additional $r \in \mathcal{R}_1^k$ independently at random. Denote by $l = \text{affine}_s(r)$ the line within s corresponding to r (the notation affine_s was introduced in section 4). By uniformity, l is uniformly distributed in $\mathcal{S}_1^m$. Hence, by Lemma 5.5 and uniformity,

$$\begin{aligned}\mathop{\mathbf{Var}}_{s}\left[\frac{|s\cap A|}{|s|}\right] &= \mathop{\mathbf{Var}}_{s}\left[\mathop{\mathbf{E}}_{r}\left[\frac{|l\cap A|}{|l|}\right]\right]\\ &\le \mathop{\mathbf{Var}}_{s,r}\left[\frac{|l\cap A|}{|l|}\right]\\ &\le \frac{\mu}{|\mathbb{H}|}. \qquad \square\end{aligned}$$

We can now bound the deviation of the hitting rate from its expected value.

COROLLARY 5.7 (sampling). *Fix dimensions k and m, $1 \le k \le m$. Fix $A \subseteq \mathbb{F}^m$ of density $\mu = |A| / |\mathbb{F}^m|$. Then, for any $\varepsilon > 0$,*

$$\Pr_{s\in\mathcal{S}_k^m}\left[\left|\frac{|s\cap A|}{|s|} - \mu\right| \ge \varepsilon\right] \le \frac{\mu}{\varepsilon^2\,|\mathbb{H}|}.$$

Proof. Apply Lemma 5.6 and then Chebyshev's inequality. $\square$

5.4. Linear subspaces sample well. We can similarly prove that linear subspaces with one direction chosen from $\mathbb{F}^m$ and all other directions chosen from $\mathbb{H}^m$ sample well. We will need this lemma to analyze the randomness-efficient subspace vs. point tester.

LEMMA 5.8. *Fix dimensions k and m, $1 \le k < m$. Fix a set $A \subseteq \mathbb{F}^m$ of density $\mu = |A|/|\mathbb{F}^m|$. Pick uniformly $\vec{z} \in \mathbb{F}^m$, $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$, such that $\vec{z}, \vec{y}_1, \ldots, \vec{y}_k$ are linearly independent. Denote $s = \text{affine}(\vec{0}; \vec{z}, \vec{y}_1, \ldots, \vec{y}_k)$. Then*

$$\mathop{\mathbf{E}}_{s}\left[\left(\frac{|s\cap A|}{|s|} - \mu\right)^2\right] \le \frac{\mu}{|\mathbb{H}|} + \frac{1}{|\mathbb{F}|}.$$

Proof. Pick an additional scalar $\alpha \in \mathbb{F}$ independently at random. Let $s_\alpha = \text{affine}(\alpha\vec{z}; \vec{y}_1, \ldots, \vec{y}_k)$. Note that s_α is distributed in $\mathcal{S}_k^m$ as follows: with probability $\frac{1}{|\mathbb{F}|}$, s_α is uniformly distributed in the set of affine subspaces in $\mathcal{S}_k^m$ through the origin; with probability $1 - \frac{1}{|\mathbb{F}|}$, s_α is uniformly distributed in the set of affine subspaces in $\mathcal{S}_k^m$ that do not contain the origin. Therefore,

$$\begin{aligned}\mathop{\mathbf{E}}_{s,\alpha}\left[\left(\frac{|s_\alpha\cap A|}{|s_\alpha|} - \mu\right)^2\right] &\le 1 \cdot \mathop{\mathbf{E}}_{s'\in\mathcal{S}_k^m}\left[\left(\frac{|s'\cap A|}{|s'|} - \mu\right)^2\right] + \frac{1}{|\mathbb{F}|}\cdot 1\\ &= \mathop{\mathbf{Var}}_{s'\in\mathcal{S}_k^m}\left[\frac{|s'\cap A|}{|s'|}\right] + \frac{1}{|\mathbb{F}|}.\end{aligned}$$

By Lemma 5.6,

$$\mathop{\mathbf{E}}_{s,\alpha}\left[\left(\frac{|s_\alpha\cap A|}{|s_\alpha|} - \mu\right)^2\right] \le \frac{\mu}{|\mathbb{H}|} + \frac{1}{|\mathbb{F}|}.$$

By Jensen inequality,

$$\mathop{\mathbf{E}}_{s}\left[\left(\frac{|s\cap A|}{|s|} - \mu\right)^2\right] \le \mathop{\mathbf{E}}_{s,\alpha}\left[\left(\frac{|s_\alpha\cap A|}{|s_\alpha|} - \mu\right)^2\right].$$

The lemma follows. $\square$

We can now bound the deviation of the hitting rate from its expected value.

COROLLARY 5.9 (sampling). *Fix dimensions k and m, $1 \le k < m$. Fix a set $A \subseteq \mathbb{F}^m$ of density $\mu = |A|/|\mathbb{F}^m|$. Pick uniformly $\vec{z} \in \mathbb{F}^m$, $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^m$, such that $\vec{z}, \vec{y}_1, \ldots, \vec{y}_k$ are linearly independent. Denote $s = \text{affine}(\vec{0}; \vec{z}, \vec{y}_1, \ldots, \vec{y}_k)$. Then, for any $\varepsilon > 0$,*

$$\Pr_s \left[\left| \frac{|s \cap A|}{|s|} - \mu \right| \ge \varepsilon \right] \le \frac{1}{\varepsilon^2} \cdot \left(\frac{\mu}{|\mathbb{H}|} + \frac{1}{|\mathbb{F}|} \right).$$

Proof. Apply Markov inequality on Lemma 5.8. □

6. Consolidation. In this section we show that *weak* low degree testing claims imply *strong* low degree testing claims. Specifically, we are interested in the following (for exact definitions, see the next subsections):

1. *Decoding/list decoding*: By decoding we refer to finding a single polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ agreeing with the oracle on many of the points. By list decoding we refer to finding a short list of polynomials $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$ explaining almost all of the acceptance probability of a tester.
2. *Consistency consolidation*: We are able to construct polynomials $Q : \mathbb{F}^m \to \mathbb{F}$ agreeing with the oracle on some fraction of the points and wish to find polynomials agreeing with the oracle on a larger fraction of the points.
3. *Degree consolidation*: We are able to construct polynomials $Q : \mathbb{F}^m \to \mathbb{F}$ of degree at most $d' \ge d$ and wish to find polynomials of degree at most d.

We call such arguments *consolidating arguments*. They are standard in the low degree testing literature (see, e.g., [3, 17, 9]); however, they require some adaptation to our new setting. In the following subsections we provide the statements and the proofs of the exact claims we need.

6.1. From decoding to list decoding. If we have a way to decode, then we can list decode by repeatedly applying decoding. In our setting, it is easy to force the decoding process to output a polynomial that differs from existing polynomials by modifying the oracle.

LEMMA 6.1 (from decoding to list decoding). *Assume that $|\mathbb{F}| \ge 4$. Fix a distribution $\mathcal{D}$ over affine subspaces of dimension $k > 0$ in $\mathbb{F}^m$. Fix a function $f : \mathbb{R} \to \mathbb{R}$ and a degree d' such that $d \le d' \le |\mathbb{F}| - 3$. If*
(decoding):
for every success probability $0 < \gamma \le 1$ *and oracle $\mathcal{A}$,*
(much consistency)

$$\mathop{\mathbf{E}}_{s \sim \mathcal{D}} \left[\Pr_{\vec{x} \in s} [\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})] \right] \ge \gamma$$

implies
(a relatively low degree polynomial that slightly agrees with the oracle),
there exists a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q \le d'$, such that

$$\Pr_{\vec{x} \in \mathbb{F}^m} [Q(\vec{x}) = \mathcal{A}(\vec{x})] \ge f(\gamma).$$

Then
(list-decoding):
for every oracle $\mathcal{A}$,

(almost all consistency is explained by a relatively short list),
fix $\epsilon_0 \stackrel{def}{=} \sqrt{\frac{d'}{|\mathbb{F}|}}$. *For every* $\epsilon_0 < \delta < 1$ *such that* $\delta' \stackrel{def}{=} f(\delta - \epsilon_0) - \epsilon_0 \geq 2\epsilon_0$, *there exists a list of* $t \leq 2/\delta'$ *polynomials* $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$, *with* $\deg Q_i \leq d'$, *such that*

$$\mathop{\mathbf{E}}_{s \sim \mathcal{D}} \left[\Pr_{\vec{x} \in s} \left[\mathcal{A}(s)(\vec{x}) \neq \mathcal{A}(\vec{x}) \ \vee \ \exists i \ (Q_i \equiv \mathcal{A})(s) \right] \right] \geq 1 - \delta.$$

Proof. Assume by way of contradiction that decoding holds and there exists an oracle $\mathcal{A}$ for which there exists $\epsilon_0 < \delta < 1$ satisfying $f(\delta - \epsilon_0) - \epsilon_0 \geq 2\epsilon_0$ such that there is no list decoding for δ.

Let $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$ be all polynomials of degree at most d' for which

$$\Pr_{\vec{x} \in \mathbb{F}^m} [Q_i(\vec{x}) = \mathcal{A}(\vec{x})] \geq \delta'.$$

By Proposition 3.5, $t \leq 2/\delta'$. By our assumption, $Q_1, \ldots, Q_t$ is not a list decoding for δ. Note that $t \leq 1/\epsilon_0$.

When picking a subspace $s \sim \mathcal{D}$ and a point $\vec{x}$ uniformly distributed in s, define the following events:

1. C: $\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})$ *(consistent).*
2. P: $\exists i \in [t]$, $\mathcal{A}(\vec{x}) = Q_i(\vec{x})$ *(point explained).*
3. S: $\exists i \in [t]$, $(Q_i \equiv \mathcal{A})(s)$ *(subspace explained).*

By using this notation, the contradicting assumption implies that there is much consistency within unexplained subspaces:

$$\Pr_{s,\vec{x}}[C \wedge \neg S] = 1 - \Pr_{s,\vec{x}}[\neg C \vee S] = 1 - \mathop{\mathbf{E}}_{s} \left[\Pr_{\vec{x}} [\neg C \vee S] \right] > \delta.$$

When C and P both happen, the polynomial $\mathcal{A}(s)$ agrees with a polynomial Q_i for some $i \in [t]$ on the point $\vec{x}$. Hence, by a union bound over the $i \in [t]$ and by the Schwartz–Zippel lemma, an unexplained subspace is rarely consistent with explained points:

$$\Pr_{s,\vec{x}}[C \wedge P | \neg S] \leq \frac{td'}{|\mathbb{F}|} \leq \frac{1}{\epsilon_0} \cdot \epsilon_0^2 = \epsilon_0.$$

Thus, there is much consistency on unexplained points:

$$\begin{aligned}
\Pr_{s,\vec{x}}[C \wedge \neg P] &\geq \Pr_{s,\vec{x}}[C \wedge \neg P \wedge \neg S] \\
&= \Pr_{s,\vec{x}}[C \wedge \neg S] - \Pr_{s,\vec{x}}[C \wedge P \wedge \neg S] \\
&\geq \Pr_{s,\vec{x}}[C \wedge \neg S] - \Pr_{s,\vec{x}}[C \wedge P | \neg S] \\
&> \delta - \epsilon_0.
\end{aligned}$$

Pick an arbitrary polynomial $Q' : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q' = d' + 1$. Define a new oracle $\mathcal{A}'$ as follows: $\mathcal{A}'$ assigns $Q'(\vec{x})$ to all explained points $\vec{x}$ and agrees with $\mathcal{A}$ on all other affine subspaces (recall that points are affine subspaces of dimension 0). Hence,

$$\begin{aligned}
\mathop{\mathbf{E}}_{s \sim \mathcal{D}} \left[\Pr_{\vec{x} \in s} [\mathcal{A}'(s)(\vec{x}) = \mathcal{A}'(\vec{x})] \right] &\geq \Pr_{s \sim \mathcal{D}, \vec{x} \in s} [\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x}) \wedge \mathcal{A}(\vec{x}) = \mathcal{A}'(\vec{x})] \\
&\geq \Pr_{s,\vec{x}}[C \wedge \neg P] \\
&> \delta - \epsilon_0.
\end{aligned}$$

Thus, by decoding, there exists a polynomial Q, $\deg Q \le d'$, agreeing with $\mathcal{A}'$ on many of the points

$$\Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}'(\vec{x}) = Q(\vec{x})] \ge f(\delta - \epsilon_0).$$

The polynomials Q and Q' are necessarily distinct (they do not have the same degree). Thus, by the Schwartz–Zippel lemma,

$$\Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}'(\vec{x}) = Q(\vec{x}) \wedge \mathcal{A}'(\vec{x}) \ne \mathcal{A}(\vec{x})] \le \Pr_{\vec{x}\in\mathbb{F}^m}[Q'(\vec{x}) = Q(\vec{x})] \le \frac{d'+1}{|\mathbb{F}|} \le \epsilon_0.$$

Hence,

$$\begin{aligned}\Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}(\vec{x}) = Q(\vec{x}) = \mathcal{A}'(\vec{x})] &= \Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}'(\vec{x}) = Q(\vec{x})] \\ &\quad - \Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}'(\vec{x}) = Q(\vec{x}) \wedge \mathcal{A}'(\vec{x}) \ne \mathcal{A}(\vec{x})] \\ &\ge f(\delta - \epsilon_0) - \epsilon_0 \\ &= \delta'.\end{aligned}$$

Therefore,

$$\Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}(\vec{x}) = Q(\vec{x})] \ge \Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}(\vec{x}) = Q(\vec{x}) = \mathcal{A}'(\vec{x})] \ge \delta'.$$

Hence, there exists $i \in [t]$ such that $Q = Q_i$. However, if this is the case, by the definition of $\mathcal{A}'$,

$$\delta' \le \Pr_{\vec{x}\in\mathbb{F}^m}[\mathcal{A}(\vec{x}) = Q_i(\vec{x}) = \mathcal{A}'(\vec{x})] \le \Pr_{\vec{x}\in\mathbb{F}^m}[Q'(\vec{x}) = Q(\vec{x})] \le \epsilon_0,$$

which is a contradiction. □

We can additionally demand that each member of the list decoding agrees with the oracle on many of the subspaces, i.e., there are no nonuseful members in the list.

LEMMA 6.2 (pruning the list). *Fix a distribution $\mathcal{D}$ over affine subspaces in $\mathbb{F}^m$. For every $0 < \epsilon < 1$ and oracle $\mathcal{A}$, if $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$ are $t > 0$ polynomials satisfying*

(almost all consistency is explained by the list)

$$\mathop{\mathbf{E}}_{s\sim\mathcal{D}}\left[\Pr_{\vec{x}\in s}[\mathcal{A}(s)(\vec{x}) \ne \mathcal{A}(\vec{x}) \ \vee \ \exists i \ (Q_i \equiv \mathcal{A})(s)]\right] \ge 1 - \delta,$$

then there exists a sublist $T \subseteq [t]$ such that

1. (each polynomial agrees with the oracle on many of the subspaces) *for every $i \in T$,*

$$\Pr_{s\sim\mathcal{D}}[(Q_i \equiv \mathcal{A})(s)] > \frac{\epsilon}{t};$$

2. (still almost all consistency is explained by the list)

$$\mathop{\mathbf{E}}_{s\sim\mathcal{D}}\left[\Pr_{\vec{x}\in s}[\mathcal{A}(s)(\vec{x}) \ne \mathcal{A}(\vec{x}) \ \vee \ \exists i \in T \ (Q_i \equiv \mathcal{A})(s)]\right] \ge 1 - \delta - \epsilon.$$

Proof. We prune the given list $Q_1, \ldots, Q_t$ by throwing away any polynomial Q_i, for which the first item does not hold. In other words,

$$T \stackrel{def}{=} \left\{ i \in [t] \;\middle|\; \Pr_{s \sim \mathcal{D}} \left[(Q_i \equiv \mathcal{A})(s) \right] > \frac{\epsilon}{t} \right\}.$$

By the union bound, $\Pr_{s \sim \mathcal{D}} \left[\exists i \in [t] \setminus T,\ (Q_i \equiv \mathcal{A})(s) \right] \leq t \cdot \frac{\epsilon}{t} = \epsilon$. For a subspace $s \sim \mathcal{D}$ and a point $\vec{x}$ uniformly distributed in s, define the following events:

1. C: $\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})$ *(consistent).*
2. B: $\exists i \in [t]$, $(Q_i \equiv \mathcal{A})(s)$ *(explained before).*
3. N: $\exists i \in T$, $(Q_i \equiv \mathcal{A})(s)$ *(explained now).*

By using this notation, we have (e.g., by observing the appropriate Venn diagram)

$$\begin{aligned} \mathop{\mathbf{E}}_{s} \left[\Pr_{\vec{x}} \left[\neg C \vee N \right] \right] &= \Pr_{s,\vec{x}} \left[\neg C \vee N \right] \\ &\geq \Pr_{s,\vec{x}} \left[\neg C \vee B \right] - \Pr_{s,\vec{x}} \left[B \wedge \neg N \right] \\ &\geq 1 - \delta - \epsilon. \qquad \square \end{aligned}$$

6.2. Consistency consolidation. In this subsection, we prove a lemma allowing us to deduce that a *significant consistency* γ together with a list decoding for it imply that at least one of the polynomials in the list agrees with the oracle on almost a γ fraction of the points. The lemma requires that the distribution over affine subspaces samples well (see section 5). Together with Lemma 6.1 that transforms decoding into list decoding, this lemma allows us to improve the consistency we manage to recover.

We phrase a rather general lemma addressing *distributional oracles* instead of oracles. We say that $\widetilde{\mathcal{A}}$ is a *distributional oracle* if it assigns each affine subspace s a *distribution* over functions $s \to \mathbb{F}$ (not necessarily a *single* polynomial of degree at most d over s). Our semantic even permits the distribution to produce a *null* function $\perp$ with some probability. The null function satisfies that, for every subspace s, point $\vec{x} \in s$, and scalar $a \in \mathbb{F}$, the probability (over $\widetilde{\mathcal{A}}$, namely, over the randomness in choosing $\widetilde{\mathcal{A}}(s)$) that $\widetilde{\mathcal{A}}(s)(\vec{x}) = a$, when $\widetilde{\mathcal{A}}(s)$ evaluates to $\perp$, is 0.

LEMMA 6.3 (from list decoding to decoding). *Fix a distribution $\mathcal{D}$ over affine subspaces that sample well; i.e., there exists $\Delta : [0,1] \to [0,1]$ such that, for every set $A \subseteq \mathbb{F}^m$, for every $0 < \varepsilon < 1$,*

$$\Pr_{s \sim \mathcal{D}} \left[\left| \frac{|s \cap A|}{|s|} - \frac{|A|}{|\mathbb{F}^m|} \right| \geq \varepsilon \right] \leq \Delta(\varepsilon).$$

Let $\mathcal{A}$ denote an oracle, and let $\widetilde{\mathcal{A}}$ denote a distributional oracle. Assume that

1. (the oracles are γ-consistent)

$$\mathop{\mathbf{E}}_{\widetilde{\mathcal{A}}} \left[\mathop{\mathbf{E}}_{s \sim \mathcal{D}} \left[\Pr_{\vec{x} \in s} \left[\widetilde{\mathcal{A}}(s)(\vec{x}) = \mathcal{A}(\vec{x}) \right] \right] \right] \geq \gamma.$$

2. (most consistency is explained by a relatively short list)
   There exist t functions $f_1, \ldots, f_t : \mathbb{F}^m \to \mathbb{F}$ such that

$$\mathop{\mathbf{E}}_{\widetilde{\mathcal{A}}} \left[\mathop{\mathbf{E}}_{s \sim \mathcal{D}} \left[\Pr_{\vec{x} \in s} \left[\widetilde{\mathcal{A}}(s)(\vec{x}) \neq \mathcal{A}(\vec{x}) \ \vee\ \exists i\ (f_i \equiv \widetilde{\mathcal{A}})(s) \right] \right] \right] \geq 1 - \delta.$$

Then, for any $0 < \varepsilon < 1$ *such that* $\varepsilon \geq t \cdot \Delta(\varepsilon)$*, there exists* $1 \leq i \leq t$ *such that*

$$\Pr_{\vec{x} \in \mathbb{F}^m} [f_i(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma - \delta - 2\varepsilon.$$

Proof. Assume, by way of contradiction, that for every $1 \leq i \leq t$, $\Pr_{\vec{x} \in \mathbb{F}^m}[f_i(\vec{x}) = \mathcal{A}(\vec{x})] < \gamma - \delta - 2\varepsilon$. Let us bound the consistency towards a contradiction to the first item of the premise. For every $1 \leq i \leq t$, define the set of points explained by f_i:

$$A_i \stackrel{def}{=} \{\vec{x} \in \mathbb{F}^m \mid f_i(\vec{x}) = \mathcal{A}(\vec{x})\}.$$

For every $1 \leq i \leq t$, note that $\mu_i \stackrel{def}{=} \frac{|A_i|}{|\mathbb{F}^m|} < \gamma - \delta - 2\varepsilon$.

As $\mathcal{D}$ samples well, for every $1 \leq i \leq t$, a random subspace $s \sim \mathcal{D}$ is not likely to hit A_i much more than it is expected:

$$\Pr_{s \sim \mathcal{D}} \left[\frac{|s \cap A_i|}{|s|} \geq \mu_i + \varepsilon \right] \leq \Delta(\varepsilon) \leq \frac{\varepsilon}{t}.$$

By the union bound,

$$\Pr_{s \sim \mathcal{D}} \left[\exists i \in [t],\ \frac{|s \cap A_i|}{|s|} \geq \gamma - \delta - \varepsilon \right] \leq \varepsilon.$$

For a random oracle assignment $\widetilde{\mathcal{A}}$, a subspace $s \sim \mathcal{D}$, and a uniformly distributed point $\vec{x} \in s$ chosen independently at random, define the following events:

1. B: $\exists i \in [t]$, $|s \cap A_i| \geq (\gamma - \delta - \varepsilon) \cdot |s|$ *(bad subspace).*
2. C: $\widetilde{\mathcal{A}}(s)(\vec{x}) = \mathcal{A}(\vec{x})$ *(consistent).*
3. E: $\exists i \in [t]$, $(f_i \equiv \widetilde{\mathcal{A}})(s)$ *(explained).*

By using this notation, we have established that

$$\begin{aligned}
\Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C \wedge E] &= \Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C \wedge E \wedge \neg B] + \Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C \wedge E \wedge B] \\
&\leq \Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C | E \wedge \neg B] + \Pr_{s} [B] \\
&< (\gamma - \delta - \varepsilon) + \varepsilon \\
&= \gamma - \delta.
\end{aligned}$$

The second item of the premise implies that

$$\begin{aligned}
\Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C] &= \Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C \wedge \neg E] + \Pr_{\widetilde{\mathcal{A}},s,\vec{x}} [C \wedge E] \\
&< \delta + (\gamma - \delta) \\
&= \gamma.
\end{aligned}$$

This contradicts the first item of the premise. ∎

6.3. Degree consolidation. *Degree consolidation* shows that, if one reconstructs a polynomial of not too large degree that agrees with the oracle on many of our subspaces, the polynomial's true degree is, in fact, low. The reason is that the polynomial's degree does not decrease much when restricted to almost all of our subspaces.

First we prove a lemma allowing us to deduce degree d if one of the *directions* of our subspaces is distributed over $\mathbb{F}^m$ (rather than $\mathbb{H}^m$). This is used only in the analysis of the randomness-efficient subspace vs. point tester.

LEMMA 6.4 (degree d consolidation). *Fix dimensions k and m, $0 \le k < m$. Fix an oracle $\mathcal{A}$ assigning polynomials of degree at most d to all affine subspaces. Suppose that a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ satisfies the following for some $0 \le \delta \le 1$:*

1. $\deg Q \le \delta\,|\mathbb{F}|$.
2. *Q and $\mathcal{A}$ agree on a linear subspace chosen at random:*

$$\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m}\Big[(Q\equiv\mathcal{A})(\mathrm{affine}(\vec{0};\vec{z},\vec{y}_1,\ldots,\vec{y}_k))\,|\,\mathit{ind}(\vec{z},\vec{y}_1,\ldots,\vec{y}_k)\Big] > \delta+\frac{1}{|\mathbb{F}|}.$$

Then $\deg Q \le d$.

Proof. Assume by way of contradiction that $\deg Q > d$. Consider linearly independent $\vec{z}\in\mathbb{F}^m$ and $\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m$. Denote $s = \mathrm{affine}(\vec{0};\vec{z},\vec{y}_1,\ldots,\vec{y}_k)$, and observe the polynomial

$$Q_{|s}(\alpha_0,\alpha_1,\ldots,\alpha_k) = Q(\alpha_0\vec{z}+\alpha_1\vec{y}_1+\cdots+\alpha_k\vec{y}_k).$$

Note that each of the coefficients of this polynomial can be viewed as a polynomial in $z_1,\ldots,z_m$ and $y_{1,1},\ldots,y_{1,m},\ldots,y_{k,1},\ldots,y_{k,m}$ of total degree at most $\deg Q$. In particular, consider the coefficient of the degree $\deg Q$ monomial $\alpha_0^{\deg Q}$ in $Q_{|s}$. Note that it depends solely on $z_1,\ldots,z_m$ (and not on $y_{1,1},\ldots,y_{1,m},\ldots,y_{k,1},\ldots,y_{k,m}$). Hence, let us denote it by $P(z_1,\ldots,z_m)$.

To analyze P we will need more notation. Denote

$$Q(x_1,\ldots,x_m) = \sum_{i_1\cdots i_m} a_{i_1\cdots i_m} x_1^{i_1}\cdots x_m^{i_m}.$$

Define $I \stackrel{def}{=} \{(i_1,\ldots,i_m) \mid \sum_j i_j = \deg Q\}$. Now,

$$P(z_1,\ldots,z_m) = \sum_{(i_1\ldots i_m)\in I} a_{i_1\cdots i_m} z_1^{i_1}\cdots z_m^{i_m}.$$

Thus, by definition, $\deg P = \deg Q$, and P is not identically zero.

Clearly,

$$\begin{aligned}\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m}&\Big[\deg Q_{|\mathrm{affine}(\vec{0};\vec{z},\vec{y}_1,\ldots,\vec{y}_k)} > d\,|\mathsf{ind}(\vec{z},\vec{y}_1,\ldots,\vec{y}_k)\Big]\\ &\ge \Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m}[P(\vec{z})\ne 0\,|\mathsf{ind}(\vec{z},\vec{y}_1,\ldots,\vec{y}_k)].\end{aligned}$$

By the Schwartz–Zippel lemma, we have

$$\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m}[P(\vec{z})\ne 0] \ge 1-\frac{\deg Q}{|\mathbb{F}|} \ge 1-\delta.$$

For any linearly independent $\vec{y}_1,\ldots,\vec{y}_k$, the probability that a uniformly distributed $\vec{z}\in\mathbb{F}^m$ satisfies $\neg\mathsf{ind}(\vec{z},\vec{y}_1,\ldots,\vec{y}_k)$ is at most $\frac{1}{|\mathbb{F}|}$. Therefore,

$$\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m}\Big[\deg Q_{|\mathrm{affine}(\vec{0};\vec{z},\vec{y}_1,\ldots,\vec{y}_k)} > d \mid \mathsf{ind}(\vec{z},\vec{y}_1,\ldots,\vec{y}_k)\Big] \ge 1-\delta-\frac{1}{|\mathbb{F}|}.$$

However, $\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\ldots,\vec{y}_k\in\mathbb{H}^m}[\deg Q_{|\mathrm{affine}(\vec{0};\vec{z},\vec{y}_1,\ldots,\vec{y}_k)} \le d \mid \mathsf{ind}(\vec{z},\vec{y}_1,\ldots,\vec{y}_k)] > \delta + \frac{1}{|\mathbb{F}|}$. ∎

Next we prove a lemma allowing us to deduce degree md (rather than d), even if we observe only affine subspaces in $\mathcal{S}_k^m$. This lemma will be used in the analysis of the randomness-efficient plane vs. point tester.

LEMMA 6.5 (degree md consolidation). *Fix dimensions k and m, $1 \le k \le m$. Fix an oracle $\mathcal{A}$ assigning polynomials of degree at most d to all affine subspaces. Suppose that a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$ satisfies the following for some $0 \le \delta \le 1$:*

1. $\deg Q \le \delta\,|\mathbb{F}|$.
2. $\Pr_{s \in \mathcal{S}_k^m}\left[(Q \equiv \mathcal{A})(s)\right] > \delta + \frac{1}{|\mathbb{H}|}$.

Then $\deg Q \le md$.

Proof. By the premise and uniformity,

$$\Pr_{\vec{y}_1,\ldots,\vec{y}_k \in \mathbb{H}^m}\left[\Pr_{\vec{z}\in\mathbb{F}^m}\left[(Q \equiv \mathcal{A})(\mathrm{affine}(\vec{z};\vec{y}_1,\ldots,\vec{y}_k))\right] > \delta \;\middle|\; \mathsf{ind}(\vec{y}_1,\ldots,\vec{y}_k)\right] > \frac{1}{|\mathbb{H}|}.$$

Thus,

$$\Pr_{\vec{y}\neq\vec{0}\in\mathbb{H}^m}\left[\Pr_{\vec{z}\in\mathbb{F}^m}\left[\deg Q_{|\mathrm{affine}(\vec{z};\vec{y})} \le d\right] > \delta\right] > \frac{1}{|\mathbb{H}|}.$$

By Proposition 3.3, there exist linearly independent $\vec{y}_1,\ldots,\vec{y}_m \in \mathbb{H}^m$ such that, for every $1 \le i \le m$,

$$\Pr_{\vec{z}\in\mathbb{F}^m}\left[\deg Q_{|\mathrm{affine}(\vec{z};\vec{y}_i)} \le d\right] > \delta.$$

$\vec{y}_1,\ldots,\vec{y}_m$ is a basis for $\mathbb{F}^m$. Thus, every point $\vec{x} \in \mathbb{F}^m$ can be represented as $\vec{x} = \sum_{i=1}^m \alpha_i \vec{y}_i$ for some $\alpha_1,\ldots,\alpha_m \in \mathbb{F}$. Hence, view Q as a polynomial in variables $\alpha_1,\ldots,\alpha_m$. Assume by way of contradiction that $\deg Q > md$. Hence, there exists $1 \le i \le m$ such that the degree of Q in the variable α_i, which we will denote by D, is larger than d. The coefficient of α_i^D in the polynomial $Q_{|\mathrm{affine}(\vec{z};\vec{y}_i)}$ is a nonzero polynomial $P(z_1,\ldots,z_m)$ of degree at most $\deg Q$. Hence, by the Schwartz–Zippel lemma,

$$\Pr_{\vec{z}\in\mathbb{F}^m}\left[P(z_1,\ldots,z_m) = 0\right] \le \frac{\deg Q}{|\mathbb{F}|} \le \delta.$$

Thus, $\Pr_{\vec{z}\in\mathbb{F}^m}\left[\deg Q_{|\mathrm{affine}(\vec{z};\vec{y}_i)} \le d\right] \le \delta$, which is a contradiction. ∎

7. Consistency graph. Fix a dimension $k \ge 3$. In this section we define and analyze a *graph* that captures the consistency among hyperplanes in $\mathbb{F}^k$, i.e., affine subspaces of dimension $(k-1)$. By using the graph we prove a list decoding lemma (Lemma 7.4). This lemma is used in the analysis of the randomness-efficient plane vs. point tester to go up one dimension (see section 8). Lemma 7.4 is also the only lemma in this section that is used outside it.

The idea is a variation of the analysis of Raz and Safra for the non-randomness-efficient setting [17]. Our crucial observation is that we can essentially still apply their analysis *when considering only directions with coordinates in a subfield* $\mathbb{H} \subseteq \mathbb{F}$ instead of the entire field $\mathbb{F}$.

7.1. Graph construction. Given an oracle $\mathcal{A}$ assigning affine subspaces polynomials of degree at most d, define a simple undirected graph $G_{\mathcal{A}} = (V, E_{\mathcal{A}})$ that captures the consistency among affine subspaces in $\mathcal{S}_{k-1}^k$ as follows. Let the vertices be the set of affine subspaces $V \stackrel{def}{=} \mathcal{S}_{k-1}^k$. Let the edges indicate whether two

affine subspaces are assigned polynomials that are consistent on the intersection of the subspaces:

$$E_{\mathcal{A}} \stackrel{def}{=} \{(s_1, s_2) \mid \forall \vec{x} \in s_1 \cap s_2,\ \mathcal{A}(s_1)(\vec{x}) = \mathcal{A}(s_2)(\vec{x})\}.$$

Note that every two subspaces in $\mathcal{S}^k_{k-1}$ are either parallel (i.e., identify or do not intersect) or intersect by an affine subspace from $\mathcal{S}^k_{k-2}$ (see closedness under intersection; Proposition 4.5).

7.2. Graph is almost transitive. A graph $G = (V, E)$ is said to be *transitive* if, for every three vertices $u, v, w \in V$, if $(u, v) \in E$ and $(v, w) \in E$, then $(u, w) \in E$. In other words, a graph is transitive if and only if for every two vertices $u, w \in V$, $u \neq w$, they are not neighbors, namely, $(u, w) \notin E$, no vertex $v \in V$ neighbors both u and w, i.e., for every $v \in V$, either $(u, v) \notin E$ or $(v, w) \notin E$.

We first wish to establish that the graph is *almost transitive* in the sense that every two vertices that are not neighbors do not have too many common neighbors (whereas, if the graph had been transitive, they would not have had common neighbors at all).

LEMMA 7.1 (almost-transitivity). *Fix an oracle $\mathcal{A}$ assigning affine subspaces polynomials of degree at most d. Let $G_{\mathcal{A}} = (V, E_{\mathcal{A}})$ be its corresponding consistency graph for affine subspaces of dimension $k \geq 3$. Then, for every two different affine subspaces $s_1, s_2 \in V$,*

$$(s_1, s_2) \notin E_{\mathcal{A}} \quad \Rightarrow \quad \Pr_{s_3 \in V}[(s_1, s_3) \in E_{\mathcal{A}} \ \wedge\ (s_3, s_2) \in E_{\mathcal{A}}] \leq \frac{1}{|\mathbb{H}|} + \frac{d}{|\mathbb{F}|}.$$

Proof. Assume that $(s_1, s_2) \notin E_{\mathcal{A}}$. By definition, there exists $\vec{x} \in s_1 \cap s_2$, for which $\mathcal{A}(s_1)(\vec{x}) \neq \mathcal{A}(s_2)(\vec{x})$. Hence, $a \stackrel{def}{=} s_1 \cap s_2 \in \mathcal{S}^k_{k-2}$, and $\mathcal{A}(s_1)$ and $\mathcal{A}(s_2)$ induce two different polynomials of degree at most d on a. Let us denote these polynomials by P_1 and P_2, respectively. Fix a representation in $\mathcal{R}^k_{k-2}$ for a. We say that a vertex $s_3 \in V$ *spots inconsistency* if there exists $\vec{x} \in s_3 \cap a$ such that $P_1(\vec{x}) \neq P_2(\vec{x})$. We wish to argue that a random vertex $s_3 \in V$ is likely to spot inconsistency.

Pick uniformly $r = (\vec{z}; \vec{y}_1, \ldots, \vec{y}_{k-1}) \in \mathcal{R}^k_{k-1}$. Let us say that $s_3 = \text{affine}(r)$ is *bad* if s_3 either contains a or does not intersect it. Since $(k-2)+(k-1) \geq k$, for s_3 to be bad, a's directions must be linearly dependent on $\vec{y}_1, \ldots, \vec{y}_{k-1}$. Hence, by uniformity and by Proposition 3.2,

$$\Pr_{s_3 \in V}[s_3 \text{ is bad}] \leq \frac{1}{|\mathbb{H}|}. \tag{7.1}$$

By the Schwartz–Zippel lemma, $\Pr_{\vec{x} \in a}[P_1(\vec{x}) \neq P_2(\vec{x})] \geq 1 - \frac{d}{|\mathbb{F}|}$. For all of the hyperplanes s that do not contain a but do intersect it, the dimension of their intersection with a is $(k-1)+(k-2)-k = k-3$. Let $I \stackrel{def}{=} \{s \cap a \mid s \in V; a \nsubseteq s \wedge s \cap a \neq \phi\}$. By closedness under intersection and uniformity, $\mathbf{E}_{a' \in I}[\Pr_{\vec{x} \in a'}[P_1(\vec{x}) \neq P_2(\vec{x})]] = \Pr_{\vec{x} \in a}[P_1(\vec{x}) \neq P_2(\vec{x})] \geq 1 - \frac{d}{|\mathbb{F}|}$. By uniformity,

$$\Pr_{s_3 \in V}[s_3 \text{ spots inconsistency} \,|\, s_3 \text{ is not bad}] \geq 1 - \frac{d}{|\mathbb{F}|}. \tag{7.2}$$

By combining inequalities (7.1) and (7.2), we get

$$\Pr_{s_3}[s_3 \text{ spots inconsistency}] \geq 1 - \frac{1}{|\mathbb{H}|} - \frac{d}{|\mathbb{F}|}.$$

If s_3 spots inconsistency, then either $(s_1, s_3) \notin E_{\mathcal{A}}$ or $(s_3, s_2) \notin E_{\mathcal{A}}$. Thus, $(s_1, s_3) \in E_{\mathcal{A}}$ and $(s_3, s_2) \in E_{\mathcal{A}}$ with probability at most $\frac{1}{|\mathbb{H}|} + \frac{d}{|\mathbb{F}|}$. □

7.3. Graph-based list decoding. The almost-transitivity of the graph $G_{\mathcal{A}}$ can be used to prove that, other than relatively few edges, the graph is truly transitive, i.e., composed of disjoint cliques. Moreover, these cliques are relatively large. This was shown by Raz and Safra [17].

LEMMA 7.2 (graph partition). *Fix $\epsilon = \frac{1}{|\mathbb{H}|} + \frac{d}{|\mathbb{F}|}$. Fix an oracle $\mathcal{A}$ assigning affine subspaces polynomials of degree at most d. Let $G_{\mathcal{A}} = (V, E_{\mathcal{A}})$ be its corresponding consistency graph for affine subspaces of dimension $k \geq 3$. Then there exists a partition of the vertices of $G_{\mathcal{A}}$ into cliques $V = \biguplus_{i=1}^{t} V_i$, such that the following apply:*

1. (All nontrivial cliques are large) *For every $1 \leq i \leq t$, either $|V_i| = 1$ or $|V_i| > 2\sqrt{\epsilon}\,|V|$.*
2. (Almost all edges are within cliques)

$$\Pr_{s_1, s_2 \in V}\left[(s_1, s_2) \notin E_{\mathcal{A}} \ \vee \ \exists i \ s_1, s_2 \in V_i\right] \geq 1 - 5\sqrt{\epsilon}.$$

Proof. The proof follows by Lemma 7.1 and the combinatorial lemma of Raz and Safra [17] (for completeness we include a proof for this lemma; see Lemma A.1 in the appendix). □

A large clique in $G_{\mathcal{A}}$ corresponds to a single relatively low degree polynomial agreeing with the oracle $\mathcal{A}$ on all affine subspaces associated with the vertices in the clique.

LEMMA 7.3 (from large clique to polynomial). *Fix an oracle $\mathcal{A}$ assigning affine subspaces polynomials of degree at most d. Let $G_{\mathcal{A}} = (V, E_{\mathcal{A}})$ be its corresponding consistency graph for affine subspaces of dimension $k \geq 3$. Then, for every large clique $U \subseteq V$, $|U| > (\frac{2d}{|\mathbb{F}|} + \frac{1}{|\mathbb{H}|}) \cdot |V|$, there exists a polynomial $Q : \mathbb{F}^k \to \mathbb{F}$, with $\deg Q \leq 2d$, such that, for every $s \in U$, $(Q \equiv \mathcal{A})(s)$.*

Proof. For linearly independent $\vec{y}_1, \ldots, \vec{y}_{k-1} \in \mathbb{F}^k$, there are exactly $|\mathbb{F}|$ different hyperplanes of the form $\vec{z} + span\{\vec{y}_1, \ldots, \vec{y}_{k-1}\}$ for some $\vec{z} \in \mathbb{F}^k$. Let us denote their set by $\mathcal{H}[\vec{y}_1, \ldots, \vec{y}_{k-1}]$.

Pick uniformly at random linearly independent $\vec{y}_1, \ldots, \vec{y}_{k-1} \in \mathbb{H}^k$, and consider the random variable X denoting the fraction of hyperplanes in $\mathcal{H}[\vec{y}_1, \ldots, \vec{y}_{k-1}]$ that land in U.

By linearity of expectations,

$$\mathop{\mathbf{E}}_{\vec{y}_1, \ldots, \vec{y}_{k-1} \in \mathbb{H}^k \ : \ \mathsf{ind}(\vec{y}_1, \ldots, \vec{y}_{k-1})}[X] = \frac{|U|}{|V|} > \frac{2d}{|\mathbb{F}|} + \frac{1}{|\mathbb{H}|}.$$

Hence, since $0 \leq X \leq 1$,

$$\Pr_{\vec{y}_1, \ldots, \vec{y}_{k-1} \in \mathbb{H}^k}\left[X > \frac{2d}{|\mathbb{F}|} \ \middle| \ \mathsf{ind}(\vec{y}_1, \ldots, \vec{y}_{k-1})\right] > \frac{1}{|\mathbb{H}|}.$$

Let us say that linearly independent directions $\vec{y}_1, \ldots, \vec{y}_{k-1} \in \mathbb{H}^k$ are *good* if the number of hyperplanes in $\mathcal{H}[\vec{y}_1, \ldots, \vec{y}_{k-1}]$ that land in U is more than $2d$.

It follows from our calculations that there are good linearly independent directions $\vec{y}^1_1, \ldots, \vec{y}^1_{k-1} \in \mathbb{H}^k$. Fix any $\vec{y}^1_k \in \mathbb{H}^k$ that is not spanned by $\vec{y}^1_1, \ldots, \vec{y}^1_{k-1}$ (such necessarily exists since $k-1 < k$). Then there exist at least $(2d+1)$ scalars $c_0, \ldots, c_{2d} \in \mathbb{F}$ such that for every $0 \leq i \leq 2d$ we have $\mathrm{affine}(c_i \vec{y}^1_k; \vec{y}^1_1, \ldots, \vec{y}^1_{k-1}) \in U$.

But recall that we in fact established that, for uniformly distributed linearly independent $\vec{y}_1, \ldots, \vec{y}_{k-1} \in \mathbb{H}^k$, the probability that $\vec{y}_1, \ldots, \vec{y}_{k-1}$ are good is larger than $\frac{1}{|\mathbb{H}|}$. Thus (by using Proposition 3.2), for uniformly distributed linearly independent $\vec{y}^2_1, \ldots, \vec{y}^2_{k-1} \in \mathbb{H}^k$, the probability that $\vec{y}^2_1, \ldots, \vec{y}^2_{k-1}$ are good and $\vec{y}^1_1 \notin span\{\vec{y}^2_1, \ldots, \vec{y}^2_{k-1}\}$ is also positive.

Therefore, there necessarily exists a basis $\vec{y}_1, \ldots, \vec{y}_k \in \mathbb{H}^k$ for $\mathbb{F}^k$ as well as $2 \cdot (2d+1)$ scalars $c_0, \ldots, c_{2d}, c'_0, \ldots, c'_{2d} \in \mathbb{F}$ such that

$$\begin{aligned} s_0 &= \text{affine}(c_0\vec{y}_k; \vec{y}_1, \ldots, \vec{y}_{k-1}) \in U, \\ &\vdots \\ s_{2d} &= \text{affine}(c_{2d}\vec{y}_k; \vec{y}_1, \ldots, \vec{y}_{k-1}) \in U, \\ s'_0 &= \text{affine}(c'_0\vec{y}_1; \vec{y}_2, \ldots, \vec{y}_k) \in U, \\ &\vdots \\ s'_{2d} &= \text{affine}(c'_{2d}\vec{y}_1; \vec{y}_2, \ldots, \vec{y}_k) \in U. \end{aligned}$$

Let us define a polynomial $Q : \mathbb{F}^k \to \mathbb{F}$ such that, for every $0 \le i \le d$, $(Q \equiv \mathcal{A})(s_i)$. This is done by using Lagrange's interpolation formula:

$$Q\left(\sum_{i=1}^{k} \alpha_i \vec{y}_i\right) = \sum_{i=0}^{d} \frac{\prod_{j \in \{0,\ldots,d\}-\{i\}} (\alpha_k - c_j)}{\prod_{j \in \{0,\ldots,d\}-\{i\}} (c_i - c_j)} \cdot \mathcal{A}(s_i) \left(c_i \vec{y}_k + \sum_{j=1}^{k-1} \alpha_j \vec{y}_j\right).$$

The degree of Q in α_k is at most d, and its total degree is $\deg Q \le 2d$.

We would like to argue that, for every $s \in U$, $(Q \equiv \mathcal{A})(s)$. Let $0 \le j \le 2d$. For every line of the form $l = \text{affine}(\sum_{i=1}^{k-1} a_i \vec{y}_i; \vec{y}_k)$ contained in s'_j, the polynomial $Q_{|l}$ has degree at most d. Moreover, for every $0 \le i \le d$, $Q_{|l}$ and $\mathcal{A}(s'_j)$ identify on $l \cap s_i$. By the Schwartz–Zippel lemma, $Q_{|l}$ and $\mathcal{A}(s'_j)$ identify on the entire line l. Thus, for every $0 \le j \le 2d$, Q and $\mathcal{A}$ identify on s'_j. Hence, by the Schwartz–Zippel lemma, for every $0 \le j \le 2d$ (and not only for every $0 \le j \le d$), the polynomial Q (of degree at most $2d$) and $\mathcal{A}$ identify on s_j.

Let $s \in U$. Necessarily, s intersects the s_j's or the s'_j's (or both). Hence, $Q_{|s}$ and $\mathcal{A}(s)$ identify on more than $\frac{2d}{|\mathbb{F}|}$ of the points on s. $Q_{|s}$ is of degree at most $2d$. Thus, by the Schwartz–Zippel lemma, Q and $\mathcal{A}$ identify on s. □

The partition of $G_{\mathcal{A}}$ into cliques yields list decoding,

LEMMA 7.4 (hyperplane vs. hyperplane). *Fix an oracle $\mathcal{A}$ assigning affine subspaces polynomials of degree at most d. Let $G_{\mathcal{A}} = (V, E_{\mathcal{A}})$ be its corresponding consistency graph for affine subspaces of dimension $k \ge 3$. Then for any $\delta \ge 8\sqrt{\frac{d}{|\mathbb{F}|} + \frac{1}{|\mathbb{H}|}}$ there exists a list of polynomials $Q_1, \ldots, Q_t : \mathbb{F}^k \to \mathbb{F}$, $t \le \frac{4}{\delta}$, with $\deg Q_i \le 2d$, such that*

$$\Pr_{s_1, s_2 \in V}\left[(s_1, s_2) \notin E_{\mathcal{A}} \ \vee \ \exists i, \ (Q_i \equiv \mathcal{A})(s_1) \wedge (Q_i \equiv \mathcal{A})(s_2)\right] > 1 - \delta.$$

Proof. Consider the partition of Lemma 7.2. Let $S_1, \ldots, S_l$ denote the small cliques in this partition, i.e., cliques whose size is $|S_i| < \frac{\delta}{4}|V|$. Clearly,

$$\sum_{i=1}^{l} |S_i|^2 < \frac{\delta}{4}|V| \cdot \sum_{i=1}^{l} |S_i| \le \frac{\delta}{4}|V|^2.$$

Hence, $\Pr_{s_1,s_2\in V}[\exists i,\ s_1,s_2\in S_i] < \frac{\delta}{4}$. Let $L_1,\ldots,L_t$ be the set of all large cliques $|L_i| \geq \frac{\delta}{4}|V|$. We have $t \leq \frac{4}{\delta}$. Moreover,

$$\Pr_{s_1,s_2\in V}[(s_1,s_2)\notin E_{\mathcal{A}} \ \vee\ \exists i,\ s_1,s_2\in L_i] > 1-\frac{5}{8}\delta-\frac{1}{4}\delta > 1-\delta.$$

For every $1\leq i\leq t$, let $Q_i:\mathbb{F}^k\to\mathbb{F}$ be the polynomial associated with L_i according to Lemma 7.3. We have $\deg Q_i \leq 2d$ and

$$\Pr_{s_1,s_2\in V}[(s_1,s_2)\notin E_{\mathcal{A}} \ \vee\ \exists i,\ (Q_i\equiv\mathcal{A})(s_1)\wedge(Q_i\equiv\mathcal{A})(s_2)] > 1-\delta. \qquad \square$$

Note that the lemma is meaningful only when the *density* of the graph $|E_{\mathcal{A}}|/|V|^2$ is large enough with respect to δ; otherwise, the list might be empty. This corresponds to the fact that the oracle must assign the affine subspaces somewhat consistent polynomials if we wish to (list) decode.

8. Going up one dimension. Let $\mathcal{A}$ be an oracle assigning polynomials of degree at most d to affine subspaces in $\mathbb{F}^m$. Let us say that $\mathcal{A}$ is γ-*consistent* over subspaces of dimension k (we usually omit the dimensions when they are clear from the context) if

$$\mathop{\mathbf{E}}_{s\in\mathcal{S}_k^m}\left[\Pr_{\vec{x}\in s}[\mathcal{A}(s)(\vec{x})=\mathcal{A}(\vec{x})]\right]\geq\gamma.$$

Fix a dimension $k\geq 3$. Let $\mathcal{A}$ be an oracle assigning polynomials of degree at most d to affine subspaces in $\mathbb{F}^k$. In this section we prove that, if $\mathcal{A}$ is γ-consistent over affine subspaces of dimension $(k-1)$ in $\mathbb{F}^k$, there exists a polynomial $Q:\mathbb{F}^k\to\mathbb{F}$ of degree at most $2d$ that agrees with the oracle on almost γ of the points. This is done in three steps:

1. We use an argument of counting in several ways to transform our setting to one that resembles that of the consistency graph of section 7.
2. We use the analysis of the consistency graph to prove the claim we want while losing in the consistency parameter.
3. We fix the consistency parameter via the consistency consolidation of section 6.

The final result of this section is given in Lemma 8.3. This is also the only lemma in this section that is used outside it. Note that the degree parameter grows from d to $2d$, and we need to take care of that when we use this lemma.

8.1. From hyperplane vs. point to hyperplane vs. hyperplane. We start by showing that γ-consistency over hyperplanes implies that, for an average pair (s_1,s_2) of intersecting hyperplanes, $\mathcal{A}(s_1)$ and $\mathcal{A}(s_2)$ agree (with each other and with $\mathcal{A}$) on at least a γ^2 fraction of the points in the intersection of s_1 and s_2.

The proof uses repeatedly the trick of counting in several ways, which is made possible due to uniformity considerations (see section 4).

For an affine subspace $a\in\mathcal{S}_{k-2}^k$, denote the set of hyperplane pairs that intersect in a by $S_a \stackrel{def}{=} \left\{(s_1,s_2)\ \middle|\ s_1,s_2\in\mathcal{S}_{k-1}^k, s_1\cap s_2 = a\right\}$.

LEMMA 8.1 (counting in several ways). *If, for an oracle $\mathcal{A}$,*

$$\mathop{\mathbf{E}}_{s\in\mathcal{S}_{k-1}^k}\left[\Pr_{\vec{x}\in s}[\mathcal{A}(s)(\vec{x})=\mathcal{A}(\vec{x})]\right]\geq\gamma,$$

then

$$\mathop{\mathbf{E}}_{a\in\mathcal{S}^k_{k-2}}\left[\mathop{\mathbf{E}}_{(s_1,s_2)\in S_a}\left[\Pr_{\vec{x}\in a}\left[\mathcal{A}(s_1)(\vec{x})=\mathcal{A}(\vec{x})=\mathcal{A}(s_2)(\vec{x})\right]\right]\right]\geq\gamma^2-\frac{1}{|\mathbb{H}|}.$$

Proof. For an affine subspace $s\in\mathcal{S}^k_{k-1}$, an affine subspace of it $a\subset s$, $a\in\mathcal{S}^k_{k-2}$, and a point $\vec{x}\in a$, let $I_{s,a,\vec{x}}$ be the indicator variable of the event $\mathcal{A}(s)(\vec{x})=\mathcal{A}(\vec{x})$. By the premise and uniformity,

$$\mathop{\mathbf{E}}_{s}\left[\mathop{\mathbf{E}}_{a\subset s}\left[\mathop{\mathbf{E}}_{\vec{x}\in a}\left[I_{s,a,\vec{x}}\right]\right]\right]\geq\gamma.$$

By uniformity, we can also count in a different way and obtain

$$\mathop{\mathbf{E}}_{a}\left[\mathop{\mathbf{E}}_{\vec{x}\in a}\left[\mathop{\mathbf{E}}_{s\supset a}\left[I_{s,a,\vec{x}}\right]\right]\right]\geq\gamma.$$

By convexity considerations,

$$\mathop{\mathbf{E}}_{a}\left[\mathop{\mathbf{E}}_{\vec{x}\in a}\left[\left(\mathop{\mathbf{E}}_{s\supset a}\left[I_{s,a,\vec{x}}\right]\right)^2\right]\right]\geq\gamma^2,$$

or, in other words,

$$\mathop{\mathbf{E}}_{a}\left[\mathop{\mathbf{E}}_{\vec{x}\in a}\left[\mathop{\mathbf{E}}_{s_1,s_2\supset a}\left[I_{s_1,a,\vec{x}}I_{s_2,a,\vec{x}}\right]\right]\right]\geq\gamma^2.$$

We can change the order of summation once again, and get

$$\mathop{\mathbf{E}}_{a}\left[\mathop{\mathbf{E}}_{s_1,s_2\supset a}\left[\mathop{\mathbf{E}}_{\vec{x}\in a}\left[I_{s_1,a,\vec{x}}I_{s_2,a,\vec{x}}\right]\right]\right]\geq\gamma^2.$$

The lemma follows by using uniformity and noticing that the probability that $s_1=s_2$ given that $s_1,s_2\supset a$ is at most $\frac{1}{|\mathbb{H}|}$. □

8.2. Hyperplane vs. point lemma. Next, we show that considerable consistency between $(k-1)$-dimensional affine subspaces and points implies a significant correspondence of the values assigned to points with a relatively low degree polynomial over $\mathbb{F}^k$. The heart of the proof is the analysis of the consistency graph (Lemma 7.4).

LEMMA 8.2 (hyperplane vs. point). *Assume that $\mathcal{A}$ assigns polynomials of degree at most d to affine subspaces. Fix $\delta\stackrel{def}{=}16\max\{\sqrt{\frac{d}{|\mathbb{F}|}},\sqrt[4]{\frac{1}{|\mathbb{H}|}}\}$. Assume that*

$$\mathop{\mathbf{E}}_{s\in\mathcal{S}^k_{k-1}}\left[\Pr_{\vec{x}\in s}\left[\mathcal{A}(s)(\vec{x})=\mathcal{A}(\vec{x})\right]\right]\geq\gamma.$$

Then there exists a polynomial $Q:\mathbb{F}^k\to\mathbb{F}$, with $\deg Q\leq 2d$, such that

$$\Pr_{\vec{x}\in\mathbb{F}^k}\left[Q(\vec{x})=\mathcal{A}(\vec{x})\right]\geq\gamma^2-3\delta.$$

Proof. Lemma 8.1 allows us to translate the consistency given in this lemma to consistency between pairs of hyperplanes on points:

$$\mathop{\mathbf{E}}_{a\in\mathcal{S}^k_{k-2}}\left[\mathop{\mathbf{E}}_{(s_1,s_2)\in S_a}\left[\Pr_{\vec{x}\in a}\left[\mathcal{A}(s_1)(\vec{x})=\mathcal{A}(\vec{x})=\mathcal{A}(s_2)(\vec{x})\right]\right]\right]\geq\gamma^2-\frac{1}{|\mathbb{H}|}.$$

Lemma 7.4 gives list decoding $Q_1, \ldots, Q_t : \mathbb{F}^k \to \mathbb{F}$, $\deg Q_i \leq 2d$, $t \leq \frac{4}{\delta}$, for consistency among pairs of hyperplanes. We wish to argue that at least one of these polynomials also agrees with the oracle on many of the points.

Let us define an appropriate notation. Choose independently and uniformly at random a subspace $a \in \mathcal{S}^k_{k-2}$, hyperplanes that intersect on a, $(s_1, s_2) \in S_a$, and a point $\vec{x} \in a$. Define the following events:

1. X: $\mathcal{A}(s_1)(\vec{x}) = \mathcal{A}(\vec{x}) = \mathcal{A}(s_2)(\vec{x})$ *(hyperplanes are consistent on (and with) a point).*
2. C: $(s_1, s_2) \in E_{\mathcal{A}}$ *(hyperplanes are consistent).*
3. E: $\exists i\ (Q_i \equiv \mathcal{A})(s_1) \wedge (Q_i \equiv \mathcal{A})(s_2)$ *(hyperplanes are explained).*

By using this notation we have $\Pr_{a,s_1,s_2,\vec{x}}[X] \geq \gamma^2 - \frac{1}{|\mathbb{H}|}$. By uniformity, s_1, s_2 are uniformly distributed over the set of all pairs with $s_1 \cap s_2 \in \mathcal{S}^k_{k-2}$. Since for a uniformly distributed pair $s_1, s_2 \in V$, the probability that $s_1 \cap s_2 \notin \mathcal{S}^k_{k-2}$ is bounded by $\frac{1}{|\mathbb{H}|}$ (see Proposition 3.2), the list decoding translates into

$$\Pr_{s_1,s_2}[\neg C \vee E] \geq 1 - \delta - \frac{1}{|\mathbb{H}|}.$$

$\vec{x}$ is uniformly distributed within $s_1 \cap s_2$. Hence, by the Schwartz–Zippel lemma, $\Pr_{a,s_1,s_2,\vec{x}}[X|\neg C] \leq \frac{d}{|\mathbb{F}|}$. Therefore, the probability that s_1, s_2 are consistent on $\vec{x}$ but not explained is small:

$$\begin{aligned}
\Pr_{a,s_1,s_2,\vec{x}}[X \wedge \neg E] &= \Pr_{a,s_1,s_2,\vec{x}}[C \wedge X \wedge \neg E] + \Pr_{a,s_1,s_2,\vec{x}}[\neg C \wedge X \wedge \neg E] \\
&\leq \Pr_{s_1,s_2}[C \wedge \neg E] + \Pr_{a,s_1,s_2,\vec{x}}[\neg C \wedge X] \\
&\leq 1 - \Pr_{s_1,s_2}[\neg C \vee E] + \Pr_{a,s_1,s_2,\vec{x}}[X|\neg C] \\
&\leq \delta + \frac{1}{|\mathbb{H}|} + \frac{d}{|\mathbb{F}|}.
\end{aligned}$$

Thus, the probability that s_1, s_2 are consistent on $\vec{x}$ and are explained is large:

$$\begin{aligned}
\Pr_{a,s_1,s_2,\vec{x}}[X \wedge E] &\geq \Pr_{a,s_1,s_2,\vec{x}}[X] - \Pr_{a,s_1,s_2,\vec{x}}[X \wedge \neg E] \\
&\geq \gamma^2 - \frac{1}{|\mathbb{H}|} - \delta - \frac{1}{|\mathbb{H}|} - \frac{d}{|\mathbb{F}|} \\
&\geq \gamma^2 - 2\delta. \qquad (8.1)
\end{aligned}$$

Let us define a *distributional oracle* $\widetilde{\mathcal{A}}$, assigning each affine subspace $a \in \mathcal{S}^k_{k-2}$, a distribution over polynomials of degree at most d over a (for clarification of our notion of distributional oracles, see the discussion before Lemma 6.3). To define the distribution $\widetilde{\mathcal{A}}(a)$, we indicate how to sample a polynomial accordingly:

- Pick uniformly at random hyperplanes that intersect on a, $(s_1, s_2) \in S_a$.
- If there is i such that $(Q_i \equiv \mathcal{A})(s_1)$ and $(Q_i \equiv \mathcal{A})(s_2)$, output the restriction of Q_i to a (note that if there are two (or more) such polynomials, they must identify on a).
- Otherwise, output a *null* polynomial.

If $\widetilde{\mathcal{A}}(a)$ is not null, then there exists i such that $(Q_i \equiv \widetilde{\mathcal{A}})(a)$, while if $\widetilde{\mathcal{A}}(a)$ is null, $\widetilde{\mathcal{A}}(a)(\vec{x}) \neq \mathcal{A}(\vec{x})$ for every $\vec{x} \in a$. Thus,

$$\mathop{\mathbf{E}}_{\widetilde{\mathcal{A}}}\left[\mathop{\mathbf{E}}_{a \in \mathcal{S}^k_{k-2}}\left[\Pr_{\vec{x} \in a}\left[\widetilde{\mathcal{A}}(a)(\vec{x}) \neq \mathcal{A}(\vec{x}) \vee\ \exists i\ (Q_i \equiv \widetilde{\mathcal{A}})(a)\right]\right]\right] = 1.$$

By the construction of $\widetilde{\mathcal{A}}$ and inequality (8.1), $\widetilde{\mathcal{A}}$ is $(\gamma^2 - 2\delta)$-consistent with $\mathcal{A}$:

$$\mathop{\mathbf{E}}_{\widetilde{\mathcal{A}}}\left[\mathop{\mathbf{E}}_{a\in\mathcal{S}^k_{k-2}}\left[\Pr_{\vec{x}\in a}\left[\widetilde{\mathcal{A}}(a)(\vec{x}) = \mathcal{A}(\vec{x})\right]\right]\right] \geq \gamma^2 - 2\delta.$$

By Corollary 5.7, the uniform distribution on $\mathcal{S}^k_{k-2}$ samples well: for every set $A \subseteq \mathbb{F}^k$, for every $0 < \varepsilon < 1$,

$$\Pr_{a\in\mathcal{S}^k_{k-2}}\left[\left|\frac{|a\cap A|}{|a|} - \frac{|A|}{|\mathbb{F}^k|}\right| \geq \varepsilon\right] \leq \frac{1}{\varepsilon^2\,|\mathbb{H}|}.$$

Thus, by Lemma 6.3, since $\frac{\delta}{2} \geq t\cdot\frac{4}{\delta^2|\mathbb{H}|}$, there exists $1 \leq i \leq t$ such that

$$\Pr_{\vec{x}\in\mathbb{F}^k}[Q_i(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma^2 - 3\delta. \qquad \square$$

8.3. Consolidating. We can apply consistency consolidation to improve the result of the last subsection. The following summarizes what we establish in this section.

LEMMA 8.3 (consistency consolidated). *Denote* $\theta_0 \stackrel{def}{=} 2^4\cdot(\sqrt[8]{\frac{1}{|\mathbb{H}|}} + \sqrt[4]{\frac{d}{|\mathbb{F}|}})$. *Fix* $k \geq 3$. *Fix an oracle* $\mathcal{A}$ *assigning polynomials of degree at most* d *to all affine subspaces. Assume that*

$$\mathop{\mathbf{E}}_{s\in\mathcal{S}^k_{k-1}}\left[\Pr_{\vec{x}\in s}[\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})]\right] \geq \gamma.$$

Then there exists a polynomial $Q : \mathbb{F}^k \to \mathbb{F}$, *with* $\deg Q \leq 2d$, *such that*

$$\Pr_{\vec{x}\in\mathbb{F}^k}[Q(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma - 2\theta_0.$$

Proof. Assume that $\theta_0 \leq 1$ (otherwise, the claim trivially holds). Denote $\epsilon_0 = \sqrt{\frac{2d}{|\mathbb{F}|}}$ and $\delta_0 = 16\max\{\sqrt{\frac{d}{|\mathbb{F}|}}, \sqrt[4]{\frac{1}{|\mathbb{H}|}}\}$. Define $f(\gamma) \stackrel{def}{=} \gamma^2 - 3\delta_0$. It holds that

$$f(\theta_0 - \epsilon_0) - \epsilon_0 = (\theta_0 - \epsilon_0)^2 - 3\delta_0 - \epsilon_0 \geq \theta_0^2/2,$$

where $\theta_0^2/2 \geq 2\epsilon_0$. Apply Lemma 6.1 on Lemma 8.2 to deduce the existence of $t \leq 4/\theta_0^2$ polynomials $Q_1, \ldots, Q_t : \mathbb{F}^k \to \mathbb{F}$, with $\deg Q_i \leq 2d$, such that

$$\mathop{\mathbf{E}}_{s\in\mathcal{S}^k_{k-1}}\left[\Pr_{\vec{x}\in s}[\mathcal{A}(s)(\vec{x}) \neq \mathcal{A}(\vec{x}) \ \vee\ \exists i(Q_i \equiv \mathcal{A})(s)]\right] \geq 1 - \theta_0.$$

For $\varepsilon = \frac{\theta_0}{2}$, we have $\varepsilon \geq \frac{t}{\varepsilon^2|\mathbb{H}|}$. Thus, by Lemma 6.3 (by using sampling Corollary 5.7), there exists $1 \leq i \leq t$ such that

$$\Pr_{\vec{x}\in\mathbb{F}^k}[Q_i(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma - 2\theta_0. \qquad \square$$

9. The randomness-efficient plane vs. point tester is sound. We wish to show that, if the average consistency between planes and points is large, the oracle assigns points values that are close to a low degree polynomial. Theorem 1 will follow.

LEMMA 9.1 (from dimension 2 to dimension k). *Denote* $\theta_k \stackrel{def}{=} 2^4(\sqrt[4]{\frac{kd}{|\mathbb{F}|}} + \sqrt[8]{\frac{1}{|\mathbb{H}|}})$. *For every dimension* $k \geq 2$, *for every* $0 < \gamma \leq 1$ *and oracle* $\mathcal{A}$, *if*

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^k} \left[\Pr_{\vec{x} \in s} [\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})] \right] \geq \gamma,$$

then there exists a polynomial $Q : \mathbb{F}^k \to \mathbb{F}$ *with* $\deg Q \leq kd$ *such that*

$$\Pr_{\vec{x} \in \mathbb{F}^k} [Q(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma - (8k-10)\theta_k.$$

Proof. We prove the lemma by induction on k. Let us formulate two inductive claims. The second argues what we wish to show. The first argues slightly better consistency but worse degree.

$Claim_1[k]$. For every $0 < \gamma \leq 1$ and oracle $\mathcal{A}$, if

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^k} \left[\Pr_{\vec{x} \in s} [\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})] \right] \geq \gamma,$$

then there exists a polynomial $Q : \mathbb{F}^k \to \mathbb{F}$, with $\deg Q \leq 2(k-1)d$, such that

$$\Pr_{\vec{x} \in \mathbb{F}^k} [Q(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma - (8k-16)\theta_k.$$

$Claim_2[k]$. For every $0 < \gamma \leq 1$ and oracle $\mathcal{A}$, if

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^k} \left[\Pr_{\vec{x} \in s} [\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})] \right] \geq \gamma,$$

then there exists a polynomial $Q : \mathbb{F}^k \to \mathbb{F}$, with $\deg Q \leq kd$, such that

$$\Pr_{\vec{x} \in \mathbb{F}^k} [Q(\vec{x}) = \mathcal{A}(\vec{x})] \geq \gamma - (8k-10)\theta_k.$$

$Claim_1[2]$ holds by taking Q to be $\mathcal{A}(s)$ for the only plane s. Hence, the lemma will follow if we prove that, for every $k \geq 2$,

$$Claim_1[k] \;\Rightarrow\; Claim_2[k] \;\Rightarrow\; Claim_1[k+1].$$

CLAIM 9.1.1. $Claim_1[k] \Rightarrow Claim_2[k]$.

Proof. Fix $0 < \gamma \leq 1$ and oracle $\mathcal{A}$ such that

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^k} \left[\Pr_{\vec{x} \in s} [\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})] \right] \geq \gamma.$$

Assume that $(8k-10)\theta_k \leq 1$ (otherwise, we are done). Denote $\epsilon_0 = \sqrt{2(k-1)d/|\mathbb{F}|}$. Define $f(\gamma) \stackrel{def}{=} \gamma - (8k-16)\theta_k$. Let $\delta = (8k-14)\theta_k$. It holds that

$$f(\delta - \epsilon_0) - \epsilon_0 = (8k-14)\theta_k - \epsilon_0 - (8k-16)\theta_k - \epsilon_0 \geq \theta_k,$$

where $\theta_k \geq 2\epsilon_0$. By Lemmas 6.1 and 6.2 applied on $Claim_1[k]$, there exist $t \leq 2/\theta_k$ polynomials $Q_1, \ldots, Q_t : \mathbb{F}^k \to \mathbb{F}$, $\deg Q_i \leq 2(k-1)d$, such that

1. *(each agrees with many planes)* for every $1 \le i \le t$,

$$\Pr_{s \in \mathcal{S}_2^k}[(Q_i \equiv \mathcal{A})(s)] \ge \frac{\theta_k}{t} > \frac{2(k-1)d}{|\mathbb{F}|} + \frac{1}{|\mathbb{H}|};$$

2. *(all explain almost all of the consistency)*

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^k}\left[\Pr_{\vec{x} \in s}[\mathcal{A}(s)(\vec{x}) \neq \mathcal{A}(\vec{x}) \vee \exists i\ (Q_i \equiv \mathcal{A})(s)]\right] \ge 1 - \delta - \theta_k.$$

By Lemma 6.5, for every $1 \le i \le t$, $\deg Q_i \le kd$. By Lemma 6.3 (by using sampling Corollary 5.7), there exists $1 \le i \le t$ such that

$$\Pr_{\vec{x} \in \mathbb{F}^k}[Q_i(\vec{x}) = \mathcal{A}(\vec{x})] \ge \gamma - (8k-10)\theta_k. \qquad \square$$

CLAIM 9.1.2. *$Claim_2[k] \Rightarrow Claim_1[k+1]$.*

Proof. Fix $0 < \gamma \le 1$ and oracle $\mathcal{A}$ such that

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^{k+1}}\left[\Pr_{\vec{x} \in s}[\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x})]\right] \ge \gamma.$$

Let $s \in \mathcal{S}_k^{k+1}$. Define an oracle relative to s, $\mathcal{A}_s$, as follows: for every affine subspace $s' = \mathrm{affine}(r)$ in $\mathbb{F}^k$ (this includes the points in $\mathbb{F}^k$), let $\mathcal{A}_s(s') \stackrel{def}{=} \mathcal{A}(\mathrm{affine}_s(r))$ (the notation affine_s was introduced in section 4). Let the *consistency within s* be

$$\gamma_s \stackrel{def}{=} \mathop{\mathbf{E}}_{s' \in \mathcal{S}_2^k}\left[\Pr_{\vec{x} \in s'}[\mathcal{A}_s(s')(\vec{x}) = \mathcal{A}_s(\vec{x})]\right].$$

By uniformity, the *average* consistency within $s \in \mathcal{S}_k^{k+1}$ is large:

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_k^{k+1}}[\gamma_s] = \mathop{\mathbf{E}}_{s' \in \mathcal{S}_2^{k+1}}\left[\Pr_{\vec{x} \in s'}[\mathcal{A}(s')(\vec{x}) = \mathcal{A}(\vec{x})]\right] \ge \gamma.$$

$Claim_2[k]$ implies the existence of a new oracle $\mathcal{A}'$ that assigns each hyperplane $s \in \mathcal{S}_k^{k+1}$ a polynomial of degree at most kd that agrees with $\mathcal{A}$ on at least $\gamma_s - (8k-10)\theta_k$ of its points. It holds that

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_k^{k+1}}\left[\Pr_{\vec{x} \in s}[\mathcal{A}'(s)(\vec{x}) = \mathcal{A}(\vec{x})]\right] \ge \mathop{\mathbf{E}}_{s \in \mathcal{S}_k^{k+1}}[\gamma_s - (8k-10)\theta_k] \ge \gamma - (8k-10)\theta_k.$$

By Lemma 8.3, there exists a polynomial $Q : \mathbb{F}^{k+1} \to \mathbb{F}$, with $\deg Q \le 2kd$, such that

$$\Pr_{\vec{x} \in \mathbb{F}^{k+1}}[Q(\vec{x}) = \mathcal{A}(\vec{x})] \ge \gamma - (8(k+1)-16)\theta_{k+1}. \qquad \square$$

Lemma 9.1 follows by induction. $\square$

The soundness of the randomness-efficient plane vs. point tester easily follows.

Proof (of Theorem 1). Assume that

$$\Pr_{\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m}\left[PlanePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2)\right] = \gamma.$$

The probability that $\vec{y}_1, \vec{y}_2$ are linearly dependent is at most $\frac{1}{|\mathbb{H}|^m} + \frac{1}{|\mathbb{H}|^{m-1}} \leq \frac{2}{|\mathbb{H}|}$. Thus,

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^m} \left[\Pr_{\vec{x} \in s} \left[\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x}) \right] \right] \geq \gamma - \frac{2}{|\mathbb{H}|}.$$

By Lemma 9.1, we have decoding: there exists a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q \leq md$, such that

$$\Pr_{\vec{x} \in \mathbb{F}^m} \left[Q(\vec{x}) = \mathcal{A}(\vec{x}) \right] \geq \gamma - \varepsilon.$$

By Lemma 6.1, we have list decoding: for every δ, $\delta > 2\varepsilon$, there exist $t \leq 2/\delta$ polynomials $Q_1, \ldots, Q_t : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q_i \leq md$, such that

$$\mathop{\mathbf{E}}_{s \in \mathcal{S}_2^m} \left[\Pr_{\vec{x} \in s} \left[\mathcal{A}(s)(\vec{x}) \neq \mathcal{A}(\vec{x}) \ \vee \ \exists i \ (Q_i \equiv \mathcal{A})(s) \right] \right] \geq 1 - \delta - 2\varepsilon + \frac{2}{|\mathbb{H}|}.$$

Therefore,

$$\Pr_{\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m} \left[\neg PlanePoint^{\mathcal{A}}(\vec{z}, \vec{y}_1, \vec{y}_2) \ \vee \ \exists i \ (Q_i \equiv \mathcal{A})(\mathrm{affine}(\vec{z}; \vec{y}_1, \vec{y}_2)) \right] \geq 1 - \delta - 2\varepsilon.$$

The proof is complete. □

10. The randomness-efficient subspace vs. point tester is sound. In this section we use the result from the previous section, namely, the soundness of the randomness-efficient plane vs. point tester, to prove the soundness of the subspace vs. point tester.

Consider the distribution $\mathcal{D}$ over three-dimensional affine subspaces induced by the tester: pick uniformly $\vec{z} \in \mathbb{F}^m, \vec{y}_1, \vec{y}_2 \in \mathbb{H}^m$ such that $\vec{z}, \vec{y}_1, \vec{y}_2$ are linearly independent, and output $\mathrm{affine}(\vec{0}; \vec{z}, \vec{y}_1, \vec{y}_2)$.

LEMMA 10.1 (from plane vs. point to subspace vs. point). *Fix dimension $m \geq 3$. Fix $\varepsilon \stackrel{def}{=} 2^7 m(\sqrt[4]{\frac{md}{|\mathbb{F}|}} + \sqrt[8]{\frac{1}{|\mathbb{H}|}})$. If an oracle $\mathcal{A}$ assigning polynomials of degree at most d to affine subspaces satisfies*

$$\mathop{\mathbf{E}}_{s \sim \mathcal{D}} \left[\Pr_{\vec{x} \in s} \left[\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x}) \right] \right] \geq \gamma,$$

then there exists a polynomial $Q : \mathbb{F}^m \to \mathbb{F}$, with $\deg Q \leq md$, such that

$$\Pr_{\vec{x} \in \mathbb{F}^m} \left[Q(\vec{x}) = \mathcal{A}(\vec{x}) \right] \geq \gamma - \varepsilon.$$

Proof. Let us construct a new oracle $\mathcal{A}'$. For every plane $p \in \mathcal{S}_2^m$ that does not contain the origin, let $\mathcal{A}'(p)$ be the restriction of $\mathcal{A}(s)$ to p, where s is the unique three-dimensional linear subspace that contains p. Let $\mathcal{A}'$ identify with $\mathcal{A}$ on all other affine subspaces.

For a subspace $s \sim \mathcal{D}$, $s = \mathrm{affine}(\vec{0}; \vec{z}, \vec{y}_1, \vec{y}_2)$, and a random scalar $\alpha \in \mathbb{F}$, let $s_\alpha = \mathrm{affine}(\alpha\vec{z}; \vec{y}_1, \vec{y}_2)$. Clearly, the premise implies that

$$\mathop{\mathbf{E}}_{s, \alpha} \left[\Pr_{\vec{x} \in s_\alpha} \left[\mathcal{A}(s)(\vec{x}) = \mathcal{A}(\vec{x}) \right] \right] \geq \gamma.$$

The plane s_α is distributed as follows: with probability $\frac{1}{|\mathbb{F}|}$, s_α is uniformly distributed within the planes in $\mathcal{S}_2^m$ that contain the origin; with probability $1-\frac{1}{|\mathbb{F}|}$, s_α is uniformly distributed within the planes in $\mathcal{S}_2^m$ that do not contain the origin.

Hence, by noticing that a uniformly distributed plane in $\mathcal{S}_2^m$ contains the origin with probability $\frac{|\mathbb{F}|^2}{|\mathbb{F}|^m} \leq \frac{1}{|\mathbb{F}|}$,

$$\mathop{\mathbf{E}}_{p\in\mathcal{S}_2^m}\left[\Pr_{\vec{x}\in p}\left[\mathcal{A}'(p)(\vec{x})=\mathcal{A}(\vec{x})\right]\right] \geq \gamma - \frac{1}{|\mathbb{F}|}.$$

By Lemma 9.1, there exists a polynomial $Q:\mathbb{F}^m\to\mathbb{F}$, with $\deg Q \leq md$, such that

$$\Pr_{\vec{x}\in\mathbb{F}^m}\left[Q(\vec{x})=\mathcal{A}(\vec{x})\right] \geq \gamma-\varepsilon. \qquad \square$$

Now we can apply degree consolidation and get the following.

LEMMA 10.2 (degree consolidated). *Fix dimension $m \geq 3$. Fix $\varepsilon \stackrel{def}{=} 2^7 m(\sqrt[4]{\frac{md}{|\mathbb{F}|}} + \sqrt[8]{\frac{1}{|\mathbb{H}|}})$. If an oracle $\mathcal{A}$ assigning polynomials of degree at most d to affine subspaces satisfies*

$$\mathop{\mathbf{E}}_{s\sim\mathcal{D}}\left[\Pr_{\vec{x}\in s}\left[\mathcal{A}(s)(\vec{x})=\mathcal{A}(\vec{x})\right]\right] \geq \gamma,$$

then there exists a polynomial $Q:\mathbb{F}^m\to\mathbb{F}$, with $\deg Q \leq d$, such that

$$\Pr_{\vec{x}\in\mathbb{F}^m}\left[Q(\vec{x})=\mathcal{A}(\vec{x})\right] \geq \gamma - 2\varepsilon.$$

Proof. Assume that $\varepsilon \leq \frac{1}{2}$ (otherwise, we are done). Denote $\epsilon_0 = \sqrt{\frac{md}{|\mathbb{F}|}}$, $\delta = 1.5\varepsilon - \epsilon_0$. By applying Lemmas 6.1 and 6.2 on Lemma 10.1, we know that there exist $t \leq 8/\varepsilon$ polynomials $Q_1,\ldots,Q_t:\mathbb{F}^m\to\mathbb{F}$, with $\deg Q_i \leq md$, such that

1. *(each agrees with many planes)* for every $1\leq i\leq t$,

$$\Pr_{s\sim\mathcal{D}}\left[(Q_i\equiv\mathcal{A})(s)\right] > \frac{\epsilon_0}{t} \geq \frac{md}{|\mathbb{F}|}+\frac{1}{|\mathbb{F}|};$$

2. *(all explain almost all of the consistency)*

$$\mathop{\mathbf{E}}_{s\sim\mathcal{D}}\left[\Pr_{\vec{x}\in s}\left[\mathcal{A}(s)(\vec{x})\neq\mathcal{A}(\vec{x})\vee\exists i(Q_i\equiv\mathcal{A})(s)\right]\right] \geq 1-\delta-\epsilon_0.$$

By Lemma 6.4, for every $1\leq i\leq t$, $\deg Q_i \leq d$. By Corollary 5.9, $\mathcal{D}$ samples well: for every set $A\subseteq\mathbb{F}^m$, for every $0<\epsilon<1$,

$$\Pr_{s\sim\mathcal{D}}\left[\left|\frac{|s\cap A|}{|s|}-\frac{|A|}{|\mathbb{F}^m|}\right|\geq\epsilon\right] \leq \frac{1}{\epsilon^2}\cdot\left(\frac{1}{|\mathbb{H}|}+\frac{1}{|\mathbb{F}|}\right).$$

Hence, by Lemma 6.3, there exists $1\leq i\leq t$ such that

$$\Pr_{\vec{x}\in\mathbb{F}^m}\left[Q_i(\vec{x})=\mathcal{A}(\vec{x})\right] \geq \gamma-2\varepsilon. \qquad \square$$

Our main theorem stating the soundness of the randomness efficient subspace vs. point tester follows.

Proof (of Theorem 2). Assume that

$$\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\vec{y}_2\in\mathbb{H}^m}\left[SpacePoint^{\mathcal{A}}(\vec{z},\vec{y}_1,\vec{y}_2)\right]=\gamma.$$

The probability that $\vec{z},\vec{y}_1,\vec{y}_2$ are linearly dependent is very small:

$$\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\vec{y}_2\in\mathbb{H}^m}\left[\neg\mathsf{ind}(\vec{z},\vec{y}_1,\vec{y}_2)\right]\le\frac{1}{|\mathbb{H}|^m}+\frac{1}{|\mathbb{H}|^{m-1}}+\frac{1}{|\mathbb{F}|^{m-2}}\le\frac{2}{|\mathbb{H}|}+\frac{1}{|\mathbb{F}|}.$$

Hence,

$$\mathop{\mathbf{E}}_{s\sim\mathcal{D}}\left[\Pr_{\vec{x}\in s}\left[\mathcal{A}(s)(\vec{x})=\mathcal{A}(\vec{x})\right]\right]\ge\gamma-\frac{2}{|\mathbb{H}|}-\frac{2}{|\mathbb{F}|}.$$

By Lemma 10.2 we have decoding: there exists a polynomial $Q:\mathbb{F}^m\to\mathbb{F}$, with $\deg Q\le d$, such that

$$\Pr_{\vec{x}\in\mathbb{F}^m}\left[Q(\vec{x})=\mathcal{A}(\vec{x})\right]\ge\gamma-2.5\varepsilon.$$

Lemma 6.1 applied on Lemma 10.2 gives list decoding: there exist $t\le 2/\delta$ polynomials $Q_1,\ldots,Q_t:\mathbb{F}^m\to\mathbb{F}$, with $\deg Q_i\le d$, such that

$$\mathop{\mathbf{E}}_{s\sim\mathcal{D}}\left[\Pr_{\vec{x}\in s}\left[\mathcal{A}(s)(\vec{x})\ne\mathcal{A}(\vec{x})\ \vee\ \exists i\ (Q_i\equiv\mathcal{A})(s)\right]\right]\ge 1-\delta-2.75\varepsilon.$$

Therefore,

$$\Pr_{\vec{z}\in\mathbb{F}^m,\vec{y}_1,\vec{y}_2\in\mathbb{H}^m}\left[\neg SpacePoint^{\mathcal{A}}(\vec{z},\vec{y}_1,\vec{y}_2)\ \vee\ \exists i\ (Q_i\equiv\mathcal{A})(\mathrm{affine}(\vec{0};\vec{z},\vec{y}_1,\vec{y}_2))\right]\ge 1-\delta-3\varepsilon.$$

The proof is complete. □

Appendix A. Combinatorial lemma. Let us prove the lemma of Raz and Safra [17] that we use. First, let us introduce several notations. Given a graph $G=(V,E)$ and a vertex $v\in V$, the *neighbors* of v are $\mathcal{N}_G(v)\stackrel{def}{=}\{u\in V\mid(v,u)\in E\}$. The *degree* of v is $d_G(v)\stackrel{def}{=}|\mathcal{N}_G(v)|$. The *connected component* of v is $C_G(v)\stackrel{def}{=}\{u\in V\mid u\text{ is reachable from }v\}$. The nonneighbors of v within its connected component are denoted by $\mathcal{D}_G(v)\stackrel{def}{=}C_G(v)\setminus(\{v\}\cup\mathcal{N}_G(v))$.

Lemma A.1 (graph partition [17]). *Let $G=(V,E)$ be an undirected graph in which every two nonneighbors $u,v\in V$, $(u,v)\notin E$, have at most $\epsilon|V|$ common neighbors. Then there exists a partition of the vertices into cliques $V=\biguplus_{i=1}^{t}V_i$ such that the following apply:*

1. (All nontrivial cliques are large) *For every $1\le i\le t$, either $|V_i|=1$, or $|V_i|>2\sqrt{\epsilon}\,|V|$.*
2. (Almost all edges are within cliques)

$$\Pr_{u,v\in V}\left[(u,v)\notin E\ \vee\ \exists i\ u,v\in V_i\right]\ge 1-5\sqrt{\epsilon}.$$

Proof. Consider the following operation on graphs, meant to improve transitivity by removing some edges: Pick a vertex $v\in V$.

1. If $d_G(v)\le 2\sqrt{\epsilon}\,|V|$, remove all of the edges that touch v.

2. If $d_G(v) > 2\sqrt{\epsilon}\,|V|$, remove all edges between neighbors of v and nonneighbors of v (these edges are necessarily within v's connected component).

If there is no vertex for which this operation causes removal of edges, then the graph is necessarily transitive, and, moreover, all of its nontrivial cliques are of size more than $2\sqrt{\epsilon}\,|V|$.

Hence, iteratively perform this operation, picking each time an arbitrary vertex for which edges would be removed, until this is no longer possible. Let $v_1, v_2, \ldots, v_l$ denote the picked (not necessarily distinct) vertices. Let $G_1, G_2, \ldots, G_l$ denote the subgraphs obtained in the l iterations. Let I_1 be the set of all indices $1 \le i \le l$ in which step 1 was performed. Let I_2 be the set of all indices $1 \le i \le l$ in which step 2 was performed.

We will bound the total number of edges removed. Observe that, if step 1 is performed for a vertex v_i, its connected component becomes a singleton. Thus, $|I_1| \le |V|$, and we have

$$\sum_{i \in I_1} |\mathcal{N}_{G_i}(v_i)| \le \sum_{i \in I_1} 2\sqrt{\epsilon}\,|V| = |I_1| \cdot 2\sqrt{\epsilon}\,|V| \le 2\sqrt{\epsilon}\,|V|^2 .$$

Observe that, if step 2 is performed for a vertex v_i, after the ith operation, the vertices of $\mathcal{N}_{G_i}(v_i)$ and the vertices of $\mathcal{D}_{G_i}(v_i)$ do not belong to the same connected component. Thus, $\sum_{i \in I_2} |\mathcal{D}_{G_i}(v_i)| \cdot |\mathcal{N}_{G_i}(v_i)| \le |V|^2$ (no pair of vertices appears twice in this sum). By the almost-transitivity, for every $i \in I_2$, every vertex $u \in \mathcal{D}_{G_i}(v_i)$ has at most $\epsilon\,|V|$ neighbors in $\mathcal{N}_{G_i}(v_i)$ (each is a common neighbor of u and v_i). Therefore, we can bound the total number of edges removed in step 2 by

$$\sum_{i \in I_2} |\mathcal{D}_{G_i}(v_i)| \cdot \epsilon\,|V| < \sum_{i \in I_2} |\mathcal{D}_{G_i}(v_i)| \cdot \epsilon \cdot \frac{|\mathcal{N}_{G_i}(v_i)|}{2\sqrt{\epsilon}} \le \frac{\sqrt{\epsilon}}{2} \cdot \sum_{i \in I_2} |\mathcal{D}_{G_i}(v_i)| \cdot |\mathcal{N}_{G_i}(v_i)| \le \frac{\sqrt{\epsilon}}{2}\,|V|^2 .$$

Therefore, the total number of edges removed is at most $2.5\sqrt{\epsilon}\,|V|^2$, and the total number of pairs $u, v \in V$ for which $(u, v) \in E$ but u and v are not in the same clique is at most $5\sqrt{\epsilon}\,|V|^2$. □

Acknowledgments. We are grateful to Muli Safra for many discussions. We also thank Ariel Gabizon, Amir Yehudayoff, and Igor Shparlinski for their suggestions. We thank Ariel Gabizon for allowing us to include his observations regarding our sampling lemma. We also thank the anonymous referees for faithfully doing their job.

REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[2] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[3] S. Arora and M. Sudan, *Improved low-degree testing and its applications*, Combinatorica, 23 (2003), pp. 365–426.

[4] L. Babai, L. Fortnow, and C. Lund, *Nondeterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[5] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Robust PCPs of proximity, shorter pcps and applications to coding*, SIAM J. Comput., 36 (2006), pp. 889–974.

[6] E. Ben-Sasson and M. Sudan, *Simple PCPs with poly-log rate and query complexity*, in Proceedings of the 37th ACM Symposium on Theory of Computing, 2005, pp. 266–275.

[7] E. Ben-Sasson, M. Sudan, S. P. Vadhan, and A. Wigderson, *Randomness-efficient low degree tests and short PCPs via epsilon-biased sets*, in Proceedings of the 34th ACM Symposium on Theory of Computing, 2003, pp. 612–621.

[8] I. Dinur, *The PCP theorem by gap amplification*, J. ACM, 54 (2007), p. 12.
[9] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra, *PCP characterizations of NP: Towards a polynomially-small error-probability*, in Proceedings of the 31st ACM Symposium on Theory of Computing, 1999, pp. 29–40.
[10] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.
[11] K. Ford, *The Distribution of Integers with a Divisor in a Given Interval*, Technical report arXiv:math/0401223, 2004.
[12] K. Friedl and M. Sudan, *Some improvements to low degree tests*, in Proceedings of the 3rd Israel Symposium on Theory and Computing Systems, IEEE, New York, 1995.
[13] O. Goldreich and M. Sudan, *Locally testable codes and PCPs of almost-linear length*, J. ACM, 53 (2006), pp. 558–655.
[14] R. Hall and G. Tenenbaum, *Divisors*, Cambridge Tracts in Math. 90, Cambridge University Press, London, 1988.
[15] D. Moshkovitz and R. Raz, *Sub-Constant Error Probabilistically Checkable Proof of Almost-Linear Size*, Technical report TR07-026, Electronic Colloquium on Computational Complexity, 2007.
[16] A. Polishchuk and D. A. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the 26th ACM Symposium on Theory of Computing, 1994, pp. 194–203.
[17] R. Raz and S. Safra, *A sub-constant error-probability low-degree test and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th ACM Symposium on Theory of Computing, 1997, pp. 475–484.
[18] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 181–206

ON APPROXIMATING RESTRICTED CYCLE COVERS*

BODO MANTHEY†

Abstract. A cycle cover of a graph is a set of cycles such that every vertex is part of exactly one cycle. An L-cycle cover is a cycle cover in which the length of every cycle is in the set L. The weight of a cycle cover of an edge-weighted graph is the sum of the weights of its edges. We come close to settling the complexity and approximability of computing L-cycle covers. On the one hand, we show that, for almost all L, computing L-cycle covers of maximum weight in directed and undirected graphs is APX-hard. Most of our hardness results hold even if the edge weights are restricted to zero and one. On the other hand, we show that the problem of computing L-cycle covers of maximum weight can be approximated within a factor of 2 for undirected graphs and within a factor of 8/3 in the case of directed graphs. This holds for arbitrary sets L.

Key words. approximation algorithms, inapproximability, cycle covers, two-factors

AMS subject classifications. 68Q17, 68Q25, 68R10, 68W25, 90C27

DOI. 10.1137/060676003

1. Introduction.

A cycle cover of a graph is a spanning subgraph that consists solely of cycles such that every vertex is part of exactly one cycle. Cycle covers play an important role in the design of approximation algorithms for the traveling salesman problem [4, 6, 7, 10, 11, 12, 13, 23], the shortest common superstring problem [9, 28], and vehicle routing problems [19].

In contrast to Hamiltonian cycles, which are special cases of cycle covers, cycle covers of maximum weight can be computed efficiently. This is exploited in the aforementioned approximation algorithms, which usually start by computing an initial cycle cover and then join cycles to obtain a Hamiltonian cycle. This technique is called *subtour patching* [16].

Short cycles in a cycle cover limit the approximation ratios achieved by such algorithms. In general, the longer the cycles in the initial cover, the better the approximation ratio. Thus, we are interested in computing cycle covers that do not contain short cycles. Moreover, there are approximation algorithms that perform particularly well if the cycle covers computed do not contain cycles of odd length [6]. Finally, some vehicle routing problems [19] require covering vertices with cycles of bounded length.

Therefore, we consider *restricted cycle covers*, where cycles of certain lengths are ruled out a priori: For $L \subseteq \mathbb{N}$, an *L-cycle cover* is a cycle cover in which the length of each cycle is in L. To fathom the possibility of designing approximation algorithms based on computing cycle covers, we aim to characterize the sets L for which L-cycle covers of maximum weight can be computed, or at least well approximated, efficiently.

*Received by the editors November 27, 2006; accepted for publication (in revised form) October 5, 2007; published electronically March 28, 2008. Parts of this paper appeared in *Proceedings of the 3rd International Workshop on Approximation and Online Algorithms (WAOA 2005)*, Lecture Notes in Comput. Sci. 3879, T. Erlebach and G. Persiano, eds., Springer-Verlag, New York, 2006, pp. 282–295 and *the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2006)*, Lecture Notes in Comput. Sci. 4271, F. V. Fomin, ed., Springer, New York, 2006, pp. 336–347. Work done in part at the Institut für Theoretische Informatik of the Universität zu Lübeck supported by German Research Foundation (DFG) research grant RE 672/3 and at the Department of Computer Science at Yale University supported by the Postdoc-Program of the German Academic Exchange Service (DAAD).

http://www.siam.org/journals/sicomp/38-1/67600.html

†Universität des Saarlandes, Informatik, Postfach 151150, 66041 Saarbrücken, Germany (manthey@cs.uni-sb.de).

Beyond being a basic tool for approximation algorithms, cycle covers are interesting in their own right. Matching theory and graph factorization are important topics in graph theory. The classical matching problem is the problem of finding one-factors, i.e., spanning subgraphs each vertex of which is incident to exactly one edge. Cycle covers of undirected graphs are also known as two-factors because every vertex is incident to exactly two edges. A considerable amount of research has been done on structural properties of graph factors and on the complexity of finding graph factors (cf. Lovász and Plummer [24] and Schrijver [27]). In particular, the complexity of finding restricted two-factors, i.e., L-cycle covers in undirected graphs, has been investigated, and Hell et al. [22] showed that finding L-cycle covers in undirected graphs is NP-hard for almost all L. However, almost nothing is known so far about the complexity of finding directed L-cycle covers.

1.1. Preliminaries. Let $G = (V, E)$ be a graph with vertex set V and edge set E. If G is undirected, then a *cycle cover* of G is a subset $C \subseteq E$ of the edges of G such that all vertices in V are incident to exactly two edges in C. If G is a directed graph, then a cycle cover of G is a subset $C \subseteq E$ such that all vertices are incident to exactly one incoming and one outgoing edge in C. Thus, the graph (V, C) consists solely of vertex-disjoint cycles. The length of a cycle is the number of edges of which it consists. Since we do not allow self-loops or multiple edges, the shortest cycles of undirected and directed graphs are of length three and two, respectively.

We call a cycle of length λ a λ-cycle for short. Cycles of even or odd length will simply be called even or odd cycles, respectively.

An *L-cycle cover* of an undirected graph is a cycle cover in which the length of every cycle is in $L \subseteq \mathcal{U} = \{3, 4, 5, \dots\}$. An L-cycle cover of a directed graph is analogously defined except that $L \subseteq \mathcal{D} = \{2, 3, 4, \dots\}$. A *$k$-cycle cover* is a $\{k, k+1, \dots\}$-cycle cover. In the following, let $\overline{L} = \mathcal{U} \setminus L$ in the case of undirected graphs and $\overline{L} = \mathcal{D} \setminus L$ in the case of directed graphs (whether we consider undirected or directed cycle covers will be clear from the context).

Given edge weights $w : E \to \mathbb{N}$, the *weight* $w(C)$ of a subset $C \subseteq E$ of the edges of G is $w(C) = \sum_{e \in C} w(e)$. In particular, this defines the weight of a cycle cover since we view cycle covers as sets of edges. Let $U \subseteq V$ be any subset of the vertices of G. The *internal edges of U* are all edges of G that have both vertices in U. We denote by $w_U(C)$ the sum of the weights of all internal edges of U that are also contained in C. The *external edges at U* are all edges of G with exactly one vertex in U.

For $L \subseteq \mathcal{U}$, L-UCC is the decision problem whether an undirected graph contains an L-cycle cover as a spanning subgraph.

Max-L-UCC(0,1) is the following optimization problem: Given an undirected complete graph with edge weights zero and one, find an L-cycle cover of maximum weight. We can also consider the graph as being not complete and without edge weights. Then we try to find an L-cycle cover with a minimum number of "nonedges" (nonedges correspond to weight zero edges, edges to weight one edges); i.e., the L-cycle cover should contain as many edges as possible. Thus, Max-L-UCC(0,1) generalizes L-UCC.

Max-L-UCC is the problem of finding L-cycle covers of maximum weight in graphs with arbitrary nonnegative edge weights.

For $k \in \mathcal{U}$, k-UCC, Max-k-UCC(0,1), and Max-k-UCC are defined like L-UCC, Max-L-UCC(0,1), and Max-L-UCC except that k-cycle covers rather than L-cycle covers are sought.

The problems L-DCC, Max-L-DCC(0,1), and Max-L-DCC as well as k-DCC, Max-k-DCC(0,1), and Max-k-DCC are defined for directed graphs like their undirected counterparts except that $L \subseteq \mathcal{D}$ and $k \in \mathcal{D}$.

An instance of Min-Vertex-Cover(λ)is an undirected λ-regular graph $H = (X, F)$; i.e., every vertex in X is incident to exactly λ edges. A vertex cover of H is a subset $\tilde{X} \subseteq X$ such that at least one vertex of every edge in F is in $\tilde{X}$. The aim is to find a subset $\tilde{X} \subseteq X$ of minimum cardinality. Min-Vertex-Cover(λ) is APX-complete for $\lambda \geq 3$ as follows from results by Alimonti and Kann [2] as well as Chlebík and Chlebíková [14].

An instance of λ-XC (exact cover by λ-sets) is a tuple (X, F), where X is a finite set and F is a collection of subsets of X, each of cardinality λ. The question is whether there exists a subcollection $\tilde{F} \subseteq F$ such that for every $x \in X$ there is a unique $a \in \tilde{F}$ with $x \in a$. For $\lambda \geq 3$, λ-XC is NP-complete [15, Problem SP2].

Let Π be an optimization problem, and let I be its set of instances. For an instance $X \in I$, let $\mathrm{opt}(X)$ denote the weight of an optimum solution. We say that Π can be approximated with an approximation ratio of $\alpha \geq 1$ if there exists a polynomial-time algorithm that, for every instance $X \in I$, computes a solution Y of X whose weight $w(Y, X)$ is at most a factor of α away from $\mathrm{opt}(X)$. This means that $w(Y, X) \leq \alpha \cdot \mathrm{opt}(X)$ if Π is a minimization problem and $w(Y, X) \geq \mathrm{opt}(X)/\alpha$ if Π is a maximization problem [3, Definition 3.6].

1.2. Previous results. Max-$\mathcal{U}$-UCC, and thus $\mathcal{U}$-UCC and Max-$\mathcal{U}$-UCC(0,1), can be solved in polynomial time via Tutte's reduction to the classical perfect matching problem [24, section 10.1]. Hartvigsen presented a polynomial-time algorithm that can be used to decide 4-UCC in polynomial time [17]. Furthermore, it can be adapted to solve Max-4-UCC(0,1) as well.

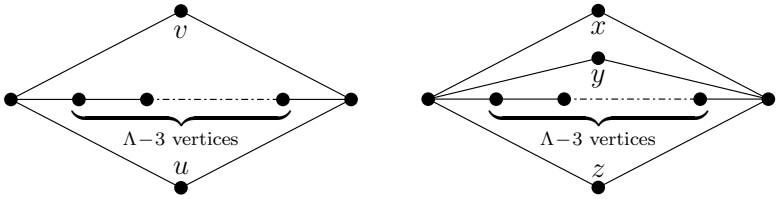
Max-k-UCC admits a simple factor 3/2 approximation for all k: Compute a maximum weight cycle cover, break the lightest edge of each cycle, and join the paths thus obtained to a Hamiltonian cycle. Unfortunately, this algorithm cannot be generalized to work for Max-L-UCC for general L. For the problem of computing k-cycle covers of minimum weight in graphs with edge weights one and two, there exists a factor 7/6 approximation algorithm for all k [8]. Hassin and Rubinstein [20, 21] devised a randomized approximation algorithm for Max-{3}-UCC that achieves an approximation ratio of $83/43 + \epsilon$.

Hell et al. [22] proved that L-UCC is NP-hard for $\overline{L} \not\subseteq \{3, 4\}$. For $k \geq 7$, Max-k-UCC(0,1) and Max-k-UCC are APX-complete [5]. Vornberger showed that Max-5-UCC is NP-hard [29].

The directed cycle cover problems $\mathcal{D}$-DCC, Max-$\mathcal{D}$-DCC(0,1), and Max-$\mathcal{D}$-DCC can be solved in polynomial time by reduction to the maximum weight perfect matching problem in bipartite graphs [1, Chapter 12]. But already 3-DCC is NP-complete [15]. Max-k-DCC(0,1) and Max-k-DCC are APX-complete for all $k \geq 3$ [5].

Similar to the factor 3/2 approximation algorithm for undirected cycle covers, Max-k-DCC has a simple factor 2 approximation algorithm for all k: Compute a maximum weight cycle cover, break the lightest edge of every cycle, and join the cycles to obtain a Hamiltonian cycle. Again, this algorithm cannot be generalized to work for arbitrary L. There is a factor 4/3 approximation algorithm for Max-3-DCC [7] and a factor 3/2 approximation algorithm for Max-k-DCC(0,1) for $k \geq 3$ [5].

The complexity of finding L-cycle covers in undirected graphs seems to be well understood. However, hardly anything is known about the complexity of L-cycle covers in directed graphs and about the approximability of L-cycle covers in both undirected and directed graphs.



(a) L-clamp. (b) L-gadget.

FIG. 1. *An L-clamp and an L-gadget for a set L with* $\max(L) = \Lambda$.

1.3. Our results. We prove that Max-L-UCC(0,1) is APX-hard for all L with $\overline{L} \not\subseteq \{3, 4\}$ (section 2.2) and that Max-L-UCC is APX-hard if $\overline{L} \not\subseteq \{3\}$ (section 2.3). The hardness results for Max-L-UCC hold even if we allow only the edge weights zero, one, and two.

We show a dichotomy for directed graphs: For all L with $L \neq \{2\}$ and $L \neq \mathcal{D}$, L-DCC is NP-hard and Max-L-DCC(0,1) and Max-L-DCC are APX-hard (section 2.5), while all three problems are solvable in polynomial time if $L = \{2\}$ or $L = \mathcal{D}$.

The hardness results for Max-L-UCC(0,1) and Max-L-DCC(0,1) carry over to the problem of computing L-cycle covers of minimum weight in graphs restricted to edge weights one and two. The hardness results for Max-L-UCC for $\overline{L} = \{3, 4\}$ and $\overline{L} = \{4\}$ carry over to the problem of computing L-cycle covers of minimum weight where the edge weights are required to fulfill the triangle inequality.

To show the hardness of directed cycle covers, we show that certain kinds of graphs, called *L-clamps*, exist for nonempty $L \subseteq \mathcal{D}$ if and only if $L \neq \mathcal{D}$ (Theorem 2.10). This graph-theoretical result might be of independent interest.

Finally, we devise approximation algorithms for Max-L-UCC and Max-L-DCC that achieve ratios of 2 and 8/3, respectively (section 3). Both algorithms work for all sets L.

2. The hardness of approximating L-cycle covers.

2.1. Clamps and gadgets. To begin the hardness proofs, we introduce *clamps*, which were defined by Hell et al. [22]. Clamps are crucial for our hardness proof.

Let $K = (U, E)$ be an undirected graph, and let $u, v \in U$ be two vertices of K, which we call the *connectors* of K. We denote by K_{-u} and K_{-v} the graphs obtained from K by deleting u and v, respectively, and their incident edges. K_{-u-v} is obtained from K by deleting both u and v. For $k \in \mathbb{N}$, K^k is the following graph: Let $y_1, \ldots, y_k \notin U$ be new vertices, and add edges $\{u, y_1\}$, $\{y_i, y_{i+1}\}$ for $1 \leq i \leq k-1$, and $\{y_k, v\}$. For $k = 0$, we directly connect u to v.

Let $L \subseteq \mathcal{U}$. The graph K is called an *L-clamp* if the following properties hold:

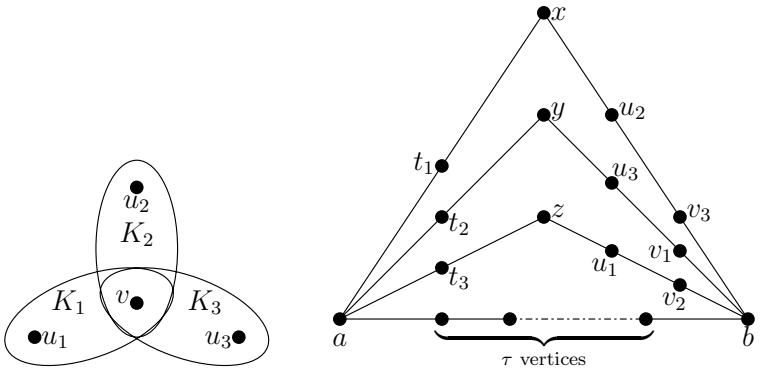
1. Both K_{-u} and K_{-v} contain an L-cycle cover.
2. Neither K nor K_{-u-v} nor K^k for any $k \in \mathbb{N}$ contains an L-cycle cover.

Figure 1(a) shows an example of an L-clamp for a set L with $\Lambda = \max(L)$. Hell et al. [22] proved the following result which we will exploit for our reduction.

LEMMA 2.1 (Hell et al. [22]). *Let $L \subseteq \mathcal{U}$ be nonempty. Then there exists an L-clamp if and only if $\overline{L} \not\subseteq \{3, 4\}$.*

Let G be a graph with vertex set V and $U \subseteq V$. We say that the vertex set U is an L-clamp with connectors $u, v \in U$ in G if the subgraph of G induced by U is an L-clamp and the only external edges of U are incident to u or v.

Let us fix some technical terms. For this purpose, let C be a subset of the edges of G. (In particular, C can be a cycle cover of G.) For any $V' \subseteq V$, we say that V' *is*



(a) A triple L-clamp with connectors u_1, u_2, u_3. The connectors of L-clamp K_i are u_i and v.

(b) An L-gadget with connectors x, y, z. The connectors of triple L-clamps T_i are t_i, u_i, v_i. For legibility, the triple L-clamps are not shown explicitly but only their connectors.

FIG. 2. *A triple L-clamp and an L-gadget.*

isolated in C if there is no edge in C connecting V' to $V \setminus V'$. If C is a cycle cover, then this means that all cycles of C traverse either only nodes of V' or only nodes of $V \setminus V'$. We say that the L-clamp U *absorbs* u *and expels* v if $U \setminus \{v\}$ is isolated in C. This means that each cycle of C traverses either only vertices in $(V \setminus U) \cup \{v\}$ or only vertices in $U \setminus \{v\}$ (which includes u). Analogously, U *absorbs* v *and expels* u if $U \setminus \{u\}$ is isolated in C.

An L-clamp implements an exclusive-or of u and v: In every L-cycle cover, exactly one of them is absorbed, and the other one is expelled. For our purpose of reducing from Min-Vertex-Cover(λ), we need a one-out-of-three behavior. A graph K is called an *L-gadget with connectors* x, y, z if the following property is fulfilled: Let G be an arbitrary graph that contains K as a subgraph such that only x, y, and z are incident to external edges. Then in all L-cycle covers C of G, exactly two of K's connectors are expelled while the third one is absorbed. To put it another way, either K_{-x-y} or K_{-x-z} or K_{-y-z} is isolated in C.

For finite sets L, we obtain an L-gadget, shown in Figure 1(b), by equipping the L-clamp of Figure 1(a) with an additional connector.

For infinite sets L, we first build an intermediate subgraph. A *triple L-clamp* is built from three L-clamps and has three connectors u_1, u_2, u_3. Figure 2(a) shows the construction. Triple L-clamps show a two-out-of-three behavior: Only one connector will be expelled, and the other two will be absorbed. More precisely, one of the three clamps has to absorb v. The other two absorb their connectors u_i, which are also connectors of the triple clamp.

Now we are prepared to build L-gadgets for infinite sets L. These graphs are built from three triple L-clamps T_1, T_2, and T_3, where T_i has connectors u_i, v_i, t_i. Figure 2(b) shows the L-gadget. Since L is infinite, there exists a $\tau \geq 1$, with $\tau + 6 \in L$. Let us argue why the L-gadget behaves as claimed. For this purpose, let C be an arbitrary L-cycle cover of G, where G contains the L-gadget as a subgraph. First, we observe that all $\tau + 2$ vertices of the path connecting a to b must be on the same cycle c in C. The only other vertices to which a is incident are t_1, t_2, and t_3. By symmetry, we assume that t_1 lies also in c. Therefore, T_1 absorbs u_1 and v_1. Hence,

v_2 and u_3 are absorbed by T_2 and T_3, respectively, and c runs through x, u_2, v_3 back to b to form a $(\tau + 6)$-cycle. Thus, x is absorbed by the gadget. T_2 expels u_2 and absorbs u_3, while T_3 expels v_3 and absorbs v_2. Hence, the gadget expels y and z as claimed. The other two cases are symmetric.

To conclude this section about clamps, we transfer the notion of L-gadgets to complete graphs with edge weights zero and one and prove some properties. In section 2.3, we will generalize the notion of L-gadgets to graphs with arbitrary edge weights.

The transformation to graphs with edge weights zero and one is made in the obvious way: Let G be an undirected complete graph with vertex set V and edge weights zero and one. Let $U \subseteq V$. We say that U is an L-gadget with connectors $x, y, z \in U$ if the subgraph of G induced by U restricted to the edges of weight one is an L-gadget with connectors x, y, z.

Let σ be the number of vertices of an L-gadget U with connectors x, y, and z. Let C be a subset of the edges of G (in particular, C can be a cycle cover). We call U *healthy in* C if U absorbs x, y, or z, expels the other two connectors, and $w_U(C) = \sigma - 2$. Since the edge weighted graph is complete, the L-cycle may traverse L-gadgets arbitrarily. The following lemma shows that we cannot gain weight by not traversing them healthily.

LEMMA 2.2. *Let G be an undirected graph with vertex set V and edge weights zero and one, and let $U \subseteq V$ be an L-gadget with connectors x, y, z. Let C be an arbitrary L-cycle cover of G and $|U| = \sigma$. Then the following properties hold:*

1. *$w_U(C) \leq \sigma - 1$.*
2. *If there are 2α external edges at U in C, i.e., edges with exactly one end point in U, then $w_U(C) \leq \sigma - \alpha$.*
3. *Assume that U absorbs exactly one of x, y, or z. Then there exists an L-cycle cover $\tilde{C}$ that differs from C only in the internal edges of U and has $w_U(\tilde{C}) = \sigma - 2$.*
4. *Assume that there are two external edges at U in C that are incident to two different connectors. Then $w_U(C) \leq \sigma - 2$.*

Proof. If $w_U(C) = \sigma$ was true, then U would contain an L-cycle cover consisting solely of weight one edges since $|U| = \sigma$. This would contradict U being an L-gadget.

The second claim follows immediately from $|U| = \sigma$ and the fact that every vertex is incident to exactly two edges in a cycle cover.

Assume without loss of generality that U absorbs x and expels y and z. Since U is an L-gadget, $U \setminus \{y, z\}$ contains an L-cycle cover consisting of $\sigma - 1$ weight one edges, which proves the third claim.

The fourth claim remains to be proved. If there are more than two external edges at U in C, we have at least four external edges and thus $w_U(C) \leq \sigma - 2$. So assume that there are exactly two external edges at U in C incident to, say, x and y. We have $\sigma - 1$ internal edges of U in C. If all of them had weight one, this would contradict the property that in an unweighted L-gadget always $U \setminus \{x, y\}$, $U \setminus \{x, z\}$, or $U \setminus \{y, z\}$ is isolated. □

2.2. The reduction for undirected graphs. The notion of L-reductions was introduced by Papadimitriou and Yannakakis [25] (cf. Ausiello et al. [3, Definition 8.4]). L-reductions can be used to show the APX-hardness of optimization problems. We present an L-reduction from Min-Vertex-Cover(λ) to show the inapproximability of Max-L-UCC(0,1) for $\overline{L} \not\subseteq \{3, 4\}$. The inapproximability of Max-L-UCC for $\overline{L} \not\subseteq \{3\}$ and Max-L-DCC(0,1) for $L \neq \{2\}$ and $L \neq \mathcal{D}$ will be shown in subsequent sections.

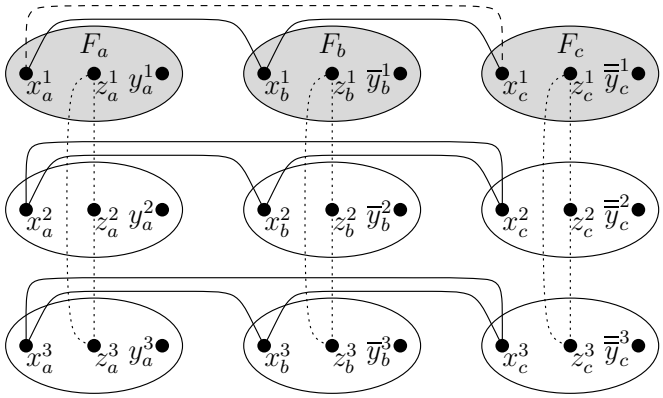


FIG. 3. *The construction for $x \in X$ incident to $a = \{x, y\}, b = \{x, \overline{y}\}, c = \{x, \overline{\overline{y}}\} \in F$ for $\lambda = 3$. F_a, F_b, and F_c are gray. The three ellipses in the second and third rows build G_2 and G_3, respectively. The cycles connecting the z-vertices are dotted. The junctions of x and their copies are solid, except for $\{x_c^1, x_a^1\}$, which has weight zero and is dashed.*

Let $L \subseteq \mathcal{U}$ be nonempty with $\overline{L} \not\subseteq \{3, 4\}$. Thus, L-gadgets exist, and we fix one as in the previous section. Let $\lambda = \min(L)$. (This choice is arbitrary. We could choose any number in L.) We will reduce Min-Vertex-Cover(λ) to Max-L-UCC(0,1). Min-Vertex-Cover(λ) is APX-complete since $\lambda \geq 3$.

Let $H = (X, F)$ be an instance of Min-Vertex-Cover(λ) with $|X| = n$ vertices and $|F| = m = \lambda n/2$ edges. Our instance G for Max-L-UCC(0,1) consists of λ subgraphs $G_1, \ldots, G_\lambda$, each containing σm vertices, where σ is the number of vertices of the L-gadget. We start by describing G_1. Then we state the differences between G_1 and $G_2, \ldots, G_\lambda$ and say to which external edges of $G_1, \ldots, G_\lambda$ weight one is assigned.

Let $a = \{x, y\} \in F$ be any edge of H. We construct an L-gadget F_a for a that has connectors x_a^1, y_a^1, and z_a^1. We call F_a an *edge gadget*.

Now let $x \in X$ be any vertex of H, and let $a_1, \ldots, a_\lambda \in F$ be the λ edges that are incident to x. We connect the vertices $x_{a_1}^1, \ldots, x_{a_\lambda}^1$ to form a path by assigning weight one to the edges $\{x_{a_\eta}^1, x_{a_{\eta+1}}^1\}$ for $\eta \in \{1, \ldots, \lambda - 1\}$. Together with edge $\{x_{a_\lambda}^1, x_{a_1}^1\}$, these edges form a cycle of length $\lambda \in L$, but note that $w(\{x_{a_\lambda}^1, x_{a_1}^1\}) = 0$. These λ edges are called the *junctions of x*. The *junctions at F_a* for some $a = \{x, y\} \in F$ are the junctions of x and y that are incident to F_a. Overall, the graph G_1 consists of σm vertices since every edge gadget consists of σ vertices.

The graphs $G_2, \ldots, G_\lambda$ are almost exact copies of G_1. The graph G_ξ ($\xi \in \{2, \ldots, \lambda\}$) consists of L-gadgets with connectors x_a^ξ, y_a^ξ, and z_a^ξ for each edge $a = \{x, y\} \in F$, just as above. The edge weights are also identical with the single exception that the edge $\{x_{a_\lambda}^\xi, x_{a_1}^\xi\}$ also has weight one. Note that we use the term "edge gadget" only for the subgraphs F_a of G_1 defined above although almost the same subgraphs occur in $G_2, \ldots, G_\lambda$ as well. Similarly, the term "junction" refers only to edges in G_1.

Finally, we describe how to connect $G_1, \ldots, G_\lambda$ with each other. For every edge $a \in F$, there are λ vertices $z_a^1, \ldots, z_a^\lambda$. These are connected to form a cycle consisting solely of weight one edges; i.e., we assign weight one to all edges $\{z_a^\xi, z_a^{\xi+1}\}$ for $\xi \in \{1, \ldots, \lambda - 1\}$ and to $\{z_a^\lambda, z_a^1\}$. Figure 3 shows an example of the whole construction from the viewpoint of a single vertex.

Edges with both vertices in the same gadget are called *internal edges*. Besides junctions and internal edges, the third kind of edges are the *z-edges* of F_a for $a \in F$,

which are the two edges $\{z_a^1, z_a^2\}$ and $\{z_a^1, z_a^\lambda\}$. The fourth kind of edges are *illegal edges*, which are edges that are not junctions but connect any two vertices of two different gadgets. The z-edges, however, are not illegal. Edges within $G_2, \ldots, G_\lambda$ as well as edges connecting G_ξ to $G_{\xi'}$ for $\xi, \xi' \geq 2$ have no special name.

We define the following terms for arbitrary subsets C of the edges of the graph G thus constructed, which includes the case of C being a cycle cover. Let $a = \{x, y\} \in F$ be an arbitrary edge of H. We say that C *legally connects* F_a if the following properties are fulfilled:

(i) C contains either two or four of the junctions at F_a and no illegal edges incident to F_a.
(ii) If C contains exactly two junctions at F_a, then these belong to the same vertex, and the two z-edges at F_a are contained in C.
(iii) If C contains four junctions at F_a, then C does not contain the z-edges at F_a.

We call C *legal* if C legally connects all gadgets. If $\tilde{C}$ is a legal L-cycle cover, then for all $x \in X$ either all junctions of x or no junction of x is in $\tilde{C}$. From a legal L-cycle cover $\tilde{C}$, we obtain the subset $\tilde{X} = \{x \mid \text{the junctions of } x \text{ are in } \tilde{C}\} \subseteq X$. Since at least two junctions at F_a are in $\tilde{C}$ for every $a \in F$, the set $\tilde{X}$ is a vertex cover of H.

The idea behind the reduction is as follows: Consider an edge $a = \{x, y\} \in F$. We interpret x_a^1 being expelled to mean that x is in the vertex cover. (In this case, the junctions of x are in the cycle cover.) Analogously, y is in the vertex cover if y_a^1 is expelled. The vertex z_a^1 is absorbed only if both x and y are in the vertex cover. If only one of x and y is in the vertex cover, z_a^1 forms a λ-cycle together with $z_a^2, \ldots, z_a^\lambda$.

We considered only G_1 when defining the terms "legally connected" and "legal." This is because in G_1 we lose weight one for putting x into the vertex cover since the junction $\{x_{a_\lambda}^1, x_{a_1}^1\}$ weighs zero. The other $\lambda - 1$ copies of the construction are needed only because z_a^1 must be part of some cycle if z_a^1 is not absorbed.

LEMMA 2.3. *Let $\tilde{X}$ be a vertex cover of size $\tilde{n}$ of H. Then G contains an L-cycle cover $\tilde{C}$, with $w(\tilde{C}) = \sigma\lambda m - \tilde{n}$.*

Proof. We start by describing $\tilde{C}$ in G_1. For every vertex $x \in \tilde{X}$, the cycle consisting of all λ junctions is in $\tilde{C}$. Let $a = \{x, y\} \in F$ be any edge. Then either x or y or both are in $\tilde{X}$. If only x is in $\tilde{X}$, we let F_a absorb y_a^1 while z_a^1 is expelled. If only y is in $\tilde{X}$, we let F_a absorb x_a^1 while z_a^1 is again expelled. If both x and y are in $\tilde{X}$, then we let x_a^1 and y_a^1 be expelled while z_a^1 is absorbed.

We perform the same construction as for G_1 for all copies $G_2, \ldots, G_\lambda$. If z_a^1 is expelled, then $z_a^2, \ldots, z_a^\lambda$ are expelled as well. We let them form a λ-cycle in $\tilde{C}$.

Clearly, $\tilde{C}$ is legal. Furthermore, $\tilde{C}$ is an L-cycle cover: Every cycle either has a length of $\lambda \in L$ or lies totally inside a single L-gadget. All L-gadgets are healthy in $\tilde{C}$, and thus $\tilde{C}$ is an L-cycle cover.

All edges of $\tilde{C}$ within $G_2, \ldots, G_\lambda$ have weight one. The only edges that connect different copies G_ξ and $G_{\xi'}$ are edges $\{z_a^\xi, z_a^{\xi+1}\}$ and $\{z_a^\lambda, x_a^1\}$, which have weight one as well. Almost all edges used in G_1 also have weight one; the only exception is one junction of weight zero for each $x \in \tilde{X}$. Since $|\tilde{X}| = \tilde{n}$, there are $\tilde{n}$ edges of weight zero in $\tilde{C}$. The graph G contains $\sigma\lambda m$ vertices, and thus $\tilde{C}$ contains $\sigma\lambda m$ edges, $\sigma\lambda m - \tilde{n}$ of which have weight one. ∎

Let C be an L-cycle cover of G, and let $a \in F$. We define $W_{F_a}(C)$ as the sum of the weights of all internal edges of F_a plus half the number of z-edges in C at F_a. Analogously, $W_{G_\xi}(C)$ is the number of weight one edges with both vertices in G_ξ plus half the number of weight one edges with exactly one vertex in G_ξ.

LEMMA 2.4. *Let C be an L-cycle cover, and let j be the number of weight one junctions in C. Then $w(C) = j + \sum_{a \in F} W_{F_a}(C) + \sum_{\xi=2}^{\lambda} W_{G_\xi}(C)$.*

Proof. Every edge with both vertices in the same G_ξ is counted once. The only edges of weight one between different G_ξ are the edges $\{z_a^\xi, z_a^{\xi+1}\}$ and $\{z_a^\lambda, z_a^1\}$. These are counted with one-half in both $W_{G_\xi}(C)$ and $W_{G_{\xi+1}}(C)$ for $2 \le \xi \le \lambda-1$ or one-half in both $W_{G_\xi}(C)$ and $W_{F_a}(C)$ for $\xi \in \{2, \lambda\}$. □

In a legal L-cycle cover $\tilde{C}$ as described in Lemma 2.3, we have $W_{G_\xi}(\tilde{C}) = \sigma m$ for all $\xi \in \{2, \dots, \lambda\}$ since every vertex in G_ξ is incident only to edges of weight one in $\tilde{C}$ by construction. Now we show that it is always best to traverse the gadgets legally and to keep the gadgets healthy.

LEMMA 2.5. *Given an arbitrary L-cycle cover C, we can compute a legal L-cycle cover $\tilde{C}$ with $w(\tilde{C}) \ge w(C)$ in polynomial time.*

Proof. We proceed as follows to obtain $\tilde{C}$:

1. Let C' be C with all illegal edges removed.
2. For all $x \in X$ in arbitrary order: If at least one junction of x is in C, then put all junctions of x into C'.
3. For all $a = \{x, y\} \in F$ in arbitrary order: If neither the junctions of x nor the junctions of y are in C', choose arbitrarily one vertex of a, say, x, and add all junctions of x to C'.
4. Rearrange C' within G_1 such that all clamps are healthy in C'.
5. Rearrange C' such that all $G_2, \dots, G_\lambda$ are traversed exactly like G_1.
6. For all $a \in F$: If $z_a^1, \dots, z_a^\xi$ are not absorbed, let them form a λ-cycle. Call the result $\tilde{C}$.

The running time of the algorithm is polynomial. Moreover, $\tilde{C}$ is a legal L-cycle cover by construction. What remains is to prove $w(\tilde{C}) \ge w(C)$.

Let $w(C) = j + \sum_{a \in F} W_{F_a}(C) + \sum_{\xi=2}^{\lambda} W_{G_\xi}(C)$ be the weight of C according to Lemma 2.4; i.e., C contains j junctions of weight one. Analogously, let $w(\tilde{C}) = \tilde{j} + \sum_{a \in F} W_{F_a}(\tilde{C}) + \sum_{\xi=2}^{\lambda} W_{G_\xi}(\tilde{C})$; i.e., $\tilde{j}$ is the number of junctions of weight one in $\tilde{C}$.

All illegal edges have weight zero, and we do not remove any junctions. We have $W_{G_\xi}(\tilde{C}) = \sigma m$ for all ξ, which is maximal. Thus, no weight is lost in this way. What remains is to consider the internal edges of the gadgets and the z-edges.

Let $a = \{x, y\}$ be an arbitrary edge of H. If $W_{F_a}(C) \le W_{F_a}(\tilde{C})$, then nothing has to be shown. Those gadgets F_a with $W_{F_a}(C) > W_{F_a}(\tilde{C})$ remain to be considered. We have $W_{F_a}(\tilde{C}) \ge \sigma - 2$ and $W_{F_a}(C) \le \sigma - 1$ according to Lemma 2.2. Thus, $W_{F_a}(C) = \sigma - 1$ and $W_{F_a}(\tilde{C}) = \sigma - 2 = W_{F_a}(C) - 1$ for all $a \in F$ with $W_{F_a}(C) > W_{F_a}(\tilde{C})$. What remains to be proved is that, for all such gadgets, there is a junction of weight one in $\tilde{C}$ that is not in C and can thus compensate for the loss of weight one in F_a. This means that we have to show that $\tilde{j}$ is at least j plus the number of edges a with $W_{F_a}(C) > W_{F_a}(\tilde{C})$.

If $W_{F_a}(C) = \sigma - 1$, then according to Lemma 2.2(4) the junctions at F_a in C (if there are any) belong to the same vertex. Since $W_{F_a}(\tilde{C}) = \sigma - 2$, all four junctions at F_a are in $\tilde{C}$. Thus, while executing the above algorithm, there is a moment at which at least one of, say, y's junctions at F_a is in C', and the junctions of x are added in the next step. We say that a vertex x *compensates* F_a if

1. $\tilde{C}$ contains x's junctions,
2. no junction of x at F_a is in C, and
3. at the moment at which x's junctions are added, C' already contains at least one junction of y at F_a.

Thus, every gadget F_a with $W_{F_a}(\tilde{C}) < W_{F_a}(C)$ is compensated by some vertex $x \in a$.

It remains to be shown that the number of gadgets that are compensated by some vertex is at most equal to the number of weight one junctions added to C'.

Let $\eta \in \{0, \ldots, \lambda\}$ be the number of junctions of x in C. If $\eta = \lambda$, then x does not compensate any gadget. If $\eta = 0$, i.e., C does not contain any of x's junctions, then the junctions of x are added during step 3 of the algorithm because there is some edge $a \in F$ with $x \in a$ such that there is no junction at all in C' at F_a before adding x's junctions. Thus, x does not compensate F_a. At most $\lambda - 1$ gadgets are compensated by x, and $\lambda - 1$ junctions of x have weight one. The case that remains is $\eta \in \{1, \ldots, \lambda - 1\}$. Then $\lambda - \eta$ junctions of x are added, and at least $\lambda - \eta - 1$ of them have weight one. On the other hand, there are at least $\eta + 1$ gadgets F_a such that at least one junction of x at F_a is already in C: Every junction is at two gadgets, and thus η junctions are at $\eta + 1$ or more gadgets. Thus, at most $\lambda - \eta - 1$ gadgets are compensated by x. □

Finally, we prove the following counterpart to Lemma 2.3.

LEMMA 2.6. *Let $\tilde{C}$ be the L-cycle cover constructed as described in the proof of Lemma 2.5, and let $\tilde{X} = \{x \mid x$'s junctions are in $\tilde{C}\}$ be the subset of X obtained from $\tilde{C}$. Choose $\tilde{n}$ such that $w(\tilde{C}) = \sigma\lambda m - \tilde{n}$. Then $|\tilde{X}| = \tilde{n}$.*

Proof. The proof is similar to the proof of Lemma 2.3. We set the weight of all junctions to one. With respect to the modified edge weights, the weight of $\tilde{C}$ is $\sigma\lambda m$. Thus, $\tilde{n}$ is the number of weight zero junctions in $\tilde{C}$, which is just $|\tilde{X}|$. □

Now we are prepared to prove the main theorem of this section.

THEOREM 2.7. *For all $L \subseteq \mathcal{U}$ with $\overline{L} \not\subseteq \{3, 4\}$,* Max-$L$-UCC(0,1) *is* APX*-hard.*

Proof. We show that the reduction presented is an L-reduction. Then the result follows from the APX-hardness of Min-Vertex-Cover(λ). Let $\mathrm{opt}(H)$ be the size of a minimum vertex cover of H and $\mathrm{opt}(G)$ be the weight of a maximum weight L-cycle cover of G. From Lemmas 2.3, 2.5, and 2.6, we obtain that $\mathrm{opt}(G) = \sigma\lambda m - \mathrm{opt}(H) \leq \sigma\lambda m$. Since H is λ-regular, we have $\mathrm{opt}(H) \geq n/2$. Thus,

$$\mathrm{opt}(G) \leq \sigma\lambda m = \sigma\lambda \cdot (\lambda n/2) \leq (\sigma\lambda^2) \cdot \mathrm{opt}(H).$$

Let C be an arbitrary L-cycle cover of G, $\tilde{C}$ be a legal L-cycle cover obtained from C as in Lemma 2.5, and $\tilde{X} \subseteq X$ be obtained from $\tilde{C}$. Then

$$\big||\tilde{X}| - \mathrm{opt}(H)\big| = \big|w(\tilde{C}) - \mathrm{opt}(G)\big| \leq \big|w(C) - \mathrm{opt}(G)\big|,$$

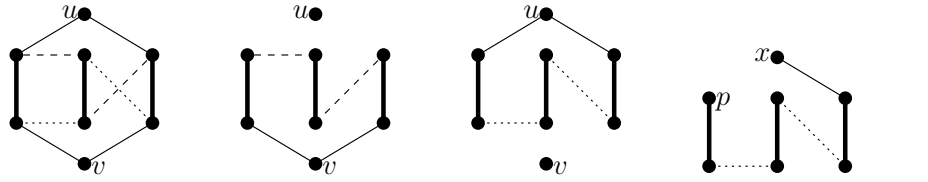
which completes the proof. □

2.3. Adaption of the reduction to Max-*L*-UCC. To prove the APX-hardness of Max-L-UCC for $\overline{L} \not\subseteq \{3\}$, all we have to do is to deal with $\overline{L} = \{4\}$ and $\overline{L} = \{3, 4\}$. For all other sets L, the inapproximability follows from Theorem 2.7. We will adapt the reduction presented in the previous section.

To do this, we have to find an edge weighted analogue of an L-clamp. We do not explicitly define the properties a weighted L-clamp has to fulfill. Instead, we just call the graph shown in Figure 4(a) a *weighted L-clamp* for $\overline{L} = \{3, 4\}$ and $\overline{L} = \{4\}$.

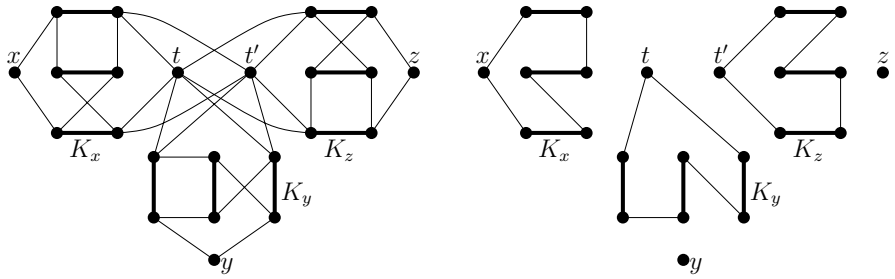
The basic idea is that all three edges of weight two of the weighted clamp have to be traversed in a cycle cover. Since 4-cycles are forbidden, we have to take either the two dotted edges or the two dashed edges. Otherwise, we would have to take an edge of weight zero. Furthermore, if we take the dashed edges, we have to absorb v and to expel u, and if we take the dotted edges, we have to absorb u and to expel v (Figures 4(b) and 4(c)). Again, we would have to take edges of weight zero otherwise.

Using three weighted L-clamps K_x, K_y, K_z, we build an L-gadget as shown in Figure 5(a). Note that both t and t' can serve as a connector for each of the clamps. This weighted L-gadget has essentially the same properties as the L-gadgets of section 2.1,



(a) The clamp. (b) Absorbing v. (c) Absorbing u. (d) Illegal traversal of K_x.

FIG. 4. *A weighted L-clamp for* $\{4\} \subseteq \overline{L} \subseteq \{3,4\}$ *and how to traverse it. Bold edges have weight two; solid, dashed, and dotted edges have weight one.*



(a) The weighted L-gadget. (b) How to absorb x.

FIG. 5. *A weighted L-gadget and how to use it.*

which were stated as Lemma 2.2. The difference is that $\sigma = 32$ is no longer the number of vertices, but the number of vertices plus the number of edges of weight two.

LEMMA 2.8. *Let G be an undirected graph with vertex set V and edge weights zero and one, and let $U \subseteq V$ be a weighted L-gadget with connectors x, y, z in G. Let C be an arbitrary L-cycle cover of G. Then the following properties hold:*

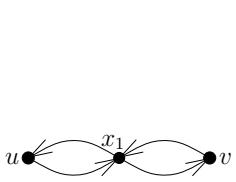
1. $w_U(C) \leq 31$.
2. *If there are 2α external edges at U in C, then $w_U(C) \leq 32 - \alpha$.*
3. *If U absorbs x, then there exists an L-cycle cover $\tilde{C}$ that differs from C only in the internal edges of U and has $w_U(\tilde{C}) = 30$. The same holds if U absorbs y or z.*
4. *Assume that there are two external edges at U in C that are incident to two different connectors. Then $w_U(C) \leq 30$.*

Proof. The only way to achieve $w_U(C) > 31$ is $w_U(C) = 32$, which requires that we have 23 internal edges including all nine edges of weight two. Since 4-cycles are forbidden, such an L-cycle cover does not exist.

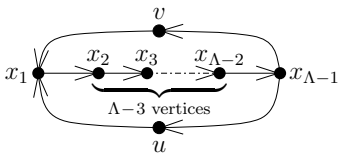
If we have 2α external edges, then we have $23 - \alpha$ internal edges. At most nine of them are of weight two.

If U absorbs x, then we can achieve a weight of 30 by letting K_y and K_z absorb t_1 and t_2, respectively (Figure 5(b)). (We can also connect K_y and K_z via t and t' to obtain a 14-cycle. The weight would be the same.) In the same way, we can achieve weight 30 if U absorbs y or z.

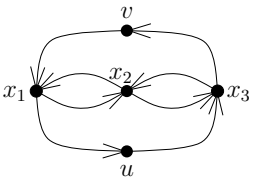
The fourth claim remains to be proved. We have $w_U(C) \leq 31$ and 22 internal edges. If $w_U(C) > 30$, then $w_U(C) = 31$, and C contains all nine edges of weight two and no internal edge of weight zero of U. By symmetry, it suffices to consider the case that x is incident to one external edge. Figure 4(d) shows which edges are mandatory in order to keep all three edges of weight two. Since the cycle that contains x must



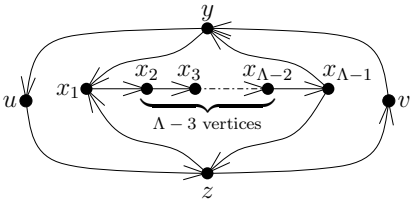
(a) A $\{2\}$-clamp.



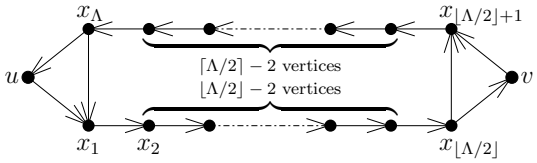
(b) An L-clamp for finite sets L with $\max(L) = \Lambda \geq 3$.



(c) A $\overline{\{2\}}$-clamp.



(d) An L-clamp for $\Lambda \notin L$ and $\Lambda + 2 \in L$ with $\Lambda \geq 3$.



(e) An L-clamp for $\Lambda, \Lambda + 2 \notin L$, and $\Lambda + 1 \in L$ with $\Lambda \geq 4$.

FIG. 6. *Directed L-clamps. The connectors are u and v; the internal vertices are $x_1, x_2, \ldots$ and y, z.*

be continued at p, vertex p is incident to an edge of weight zero in C, which proves the claim. □

Given these properties, we can plug the L-gadget into the reduction described in the previous section to obtain the APX-hardness of Max-L-UCC for $\overline{L} = \{4\}$ and $\overline{L} = \{3, 4\}$. Together with Theorem 2.7, we obtain the following result.

THEOREM 2.9. Max-L-UCC *is* APX*-hard for all L with $\overline{L} \not\subseteq \{3\}$ even if the edge weights are restricted to be zero, one, or two.*

2.4. Clamps in directed graphs. The aim of this section is to prove a counterpart to Lemma 2.1 (for the existence of L-clamps) for directed graphs. Let $K = (V, E)$ be a directed graph and $u, v \in V$. Again, K_{-u}, K_{-v}, and K_{-u-v} denote the graphs obtained by deleting u, v, and both u and v, respectively. For $k \in \mathbb{N}$, K_u^k denotes the following graph: Let $y_1, \ldots, y_k \notin V$ be new vertices, and add edges $(u, y_1), (y_1, y_2), \ldots, (y_k, v)$. For $k = 0$, we add the edge (u, v). The graph K_v^k is similarly defined, except that we now start at v; i.e., we add the edges (v, y_1), $(y_1, y_2), \ldots, (y_k, u)$. K_v^0 is K with the additional edge (v, u).

Now we can define clamps for directed graphs: Let $L \subseteq \mathcal{D}$. A directed graph $K = (V, E)$ with $u, v \in V$ is a *directed L-clamp* with connectors u and v if the following properties hold:

(i) Both K_{-u} and K_{-v} contain an L-cycle cover.

(ii) Neither K, K_{-u-v}, K_u^k, nor K_v^k for any $k \in \mathbb{N}$ contains an L-cycle cover.

Let us now prove that directed L-clamps exist for almost all L.

THEOREM 2.10. *Let $L \subseteq \mathcal{D}$ be nonempty. Then there exists a directed L-clamp if and only if $L \neq \mathcal{D}$.*

Proof. We first prove that directed L-clamps exist for all nonempty sets $L \subseteq \mathcal{D}$ with $L \neq \mathcal{D}$. We start by considering finite L. If L is finite, $\max(L) = \Lambda$ exists. For $L = \{2\}$, the graph shown in Figure 6(a) is a directed L-clamp: Either u or v forms a 2-cycle with x_1, and there are no other possibilities. Otherwise, we have $\Lambda \geq 3$. Figure 6(b) shows a directed L-clamp for this case, which is a directed variant of the undirected clamp shown in Figure 1(a).

Now we consider finite $\overline{L}$. Figure 6(c) shows an L-clamp for $\overline{L} = \{2\}$: x_1, x_2, and x_3 must be on the same path since length two is forbidden. This cycle must include u or v but cannot include both of them.

Otherwise, $\max(\overline{L}) = \Lambda \geq 3$, $\Lambda + 2 \in L$, and the graph shown in Figure 6(d) is an L-clamp: The vertices $x_1, \ldots, x_{\Lambda-1}$ must all be on the same cycle. Thus, either (y, x_1) or (z, x_1) is in the cycle cover. By symmetry, it suffices to consider the first case. Since $\Lambda \notin L$, the edge $(x_{\Lambda-1}, y)$ cannot be in the cycle cover. Thus, (v, y), $(x_{\Lambda-1}, z)$, and hence (z, v) are in the cycle cover.

The case that remains to be considered is that both L and $\overline{L}$ are infinite. We distinguish two subcases. First, there exists a $\Lambda \geq 4$ with $\Lambda, \Lambda+2 \notin L$ and $\Lambda+1 \in L$. In this case, the graph shown in Figure 6(e) is an L-clamp: $x_1, \ldots, x_\Lambda$ must be on the same cycle. Since the lengths Λ and $\Lambda + 2$ are not allowed, either v or u is expelled and the other vertex is absorbed.

Second, if no Λ exists with $\Lambda, \Lambda+2 \notin L$ and $\Lambda+1 \in L$ (but L and $\overline{L}$ are infinite), then there exists a $\Lambda \geq 3$ with $\Lambda \notin L$ and $\Lambda+2 \in L$, and we can use the graph already used for finite $\overline{L}$ (Figure 6(d)) as a directed L-clamp.

Lemma 2.11 below shows that $\mathcal{D}$-clamps do not exist, which completes the proof. □

LEMMA 2.11. *Let $G = (V, E)$ be a directed graph, and let $u, v \in V$. If G_{-u} and G_{-v} both contain a cycle cover, then*

(i) *both G and G_{-u-v} contain cycle covers or*
(ii) *all G_u^k and G_v^k for $k \in \mathbb{N}$ contain cycle covers.*

Proof. Let E_{-u} and E_{-v} be the sets of edges of the cycle covers of G_{-u} and G_{-v}, respectively. We construct two sequences of edges $P = (e_1, e_2, \ldots)$ and $P' = (e'_1, e'_2, \ldots)$. These sequences can be viewed as augmenting paths, and we use them to construct cycle covers of G_{-u-v} and G or G_u^k and G_v^k. The sequence P is given uniquely by traversing edges of E_{-v} forwards and edges of E_{-u} backwards:

(i) $e_1 = (u, x_1)$ is the unique outgoing edge of $u = x_0$ in E_{-v}.
(ii) If $e_i = (x_{i-1}, x_i) \in E_{-v}$, i.e., if i is odd, then $e_{i+1} = (x_{i+1}, x_i) \in E_{-u}$ is the unique incoming edge of x_i in E_{-u}.
(iii) If $e_i = (x_i, x_{i-1}) \in E_{-u}$, i.e., if i is even, then $e_{i+1} = (x_i, x_{i+1}) \in E_{-v}$ is the unique outgoing edge of x_i in E_{-v}.
(iv) If in any of the above steps no extension of P is possible, then stop.

Let $P = (e_1, \ldots, e_\ell)$. We observe two properties of the sequence P.

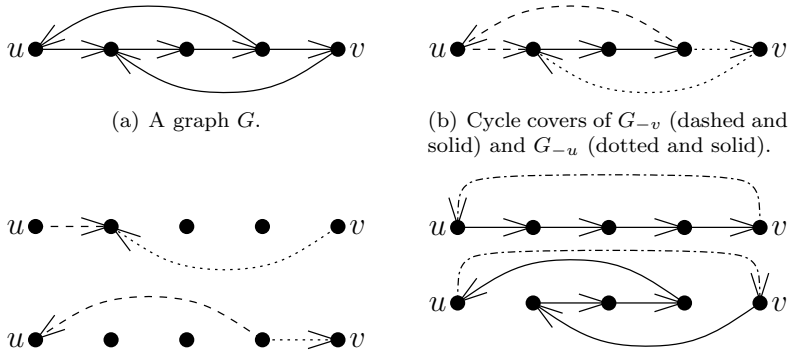
LEMMA 2.12. 1. *No edge appears more than once in P.*

2. *If ℓ is odd, i.e., $e_\ell \in E_{-v}$, then $e_\ell = (x_{\ell-1}, u)$. If ℓ is even, i.e., $e_\ell \in E_{-u}$, then $e_\ell = (v, x_{\ell-1})$.*

Proof. Assume the contrary of the first claim, and let $e_i = e_j$ $(i \neq j)$ be an edge that appears at least twice in P such that i is minimal. If $i = 1$, then $e_j = (u, x_1) \in E_{-v}$. This would imply that $e_{j-1} = (u, x_{j-2}) \in E_{-u}$, a contradiction. If $i > 1$, then assume $e_i = (x_{i-1}, x_i) \in E_{-v}$ without loss of generality. Since exactly one edge leaves x_{i-1} in E_{-u}, the edge $e_{i-1} = e_{j-1}$ is uniquely determined, which contradicts the assumption that i be minimal.

Let us now prove the second claim. Without loss of generality, we assume that the last edge e_ℓ belongs to E_{-v}. Let $e_\ell = (x_{\ell-1}, x_\ell)$. The path P cannot be extended, which implies that there does not exist an edge $(x_{\ell+1}, x_\ell) \in E_{-u}$. Since E_{-u} is a cycle cover of G_{-u}, this implies that $x_\ell = u$ and completes the proof. □

We build the sequence P' analogously, except that we start with the edge $e'_1 = (x'_1, v) \in E_{-u}$. Again, we traverse edges of E_{-v} forwards and edges of E_{-u} backwards. Let $P' = (e'_1, \ldots, e'_{\ell'})$.



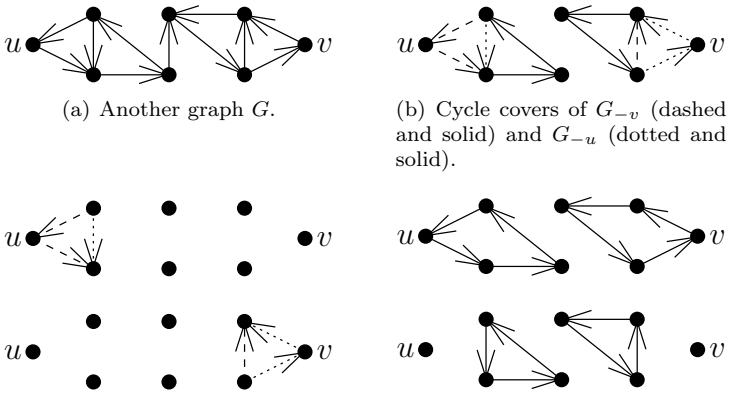
(a) A graph G.

(b) Cycle covers of G_{-v} (dashed and solid) and G_{-u} (dotted and solid).

(c) P (top) and P' (bottom). Dashed and dotted edges belong to the cycle covers of G_{-v} and G_{-u}, respectively.

(d) Cycle covers of G_v^0 (top) and G_u^0 (bottom).

FIG. 7. *Constructing cycle covers of G_v^0 and G_u^0 from the sequences P and P'.*



(a) Another graph G.

(b) Cycle covers of G_{-v} (dashed and solid) and G_{-u} (dotted and solid).

(c) P (top) and P' (bottom).

(d) Cycle covers of G (top) and G_{-u-v} (bottom).

FIG. 8. *Constructing cycle covers of G and G_{-u-v} from the sequences P and P'.*

No edge appears in both P and P' as can be proved similarly to the first claim of Lemma 2.12. Moreover, either P ends at u and P' ends at v or vice versa: We have $e_\ell = (x_{\ell-1}, u)$ if and only if $e'_{\ell'} = (v, x_{\ell'-1})$, and we have $e_\ell = (v, x_{\ell-1})$ if and only if $e'_{\ell'} = (x_{\ell'-1}, u)$. Let $P_{-u} \subseteq E_{-u}$ denote the set of edges of E_{-u} that are part of P. The sets P_{-v}, P'_{-u}, P'_{-v} are defined similarly.

Two examples are shown in Figures 7 and 8: Figures 7(a) and 7(b) show a graph with its cycle covers, while Figure 7(c) depicts P and P', the former starting at u and ending at v and the latter starting at v and ending at u. Figures 8(a), 8(b), and 8(c) show another example graph; this time P starts and ends at u and P' starts and ends at v.

Our aim is now to construct cycle covers of G and G_{-u-v} or of G_u^k and G_v^k. We distinguish two cases. Let us start with the case that P starts at u and ends at v and, consequently, P' starts at v and ends at u. Then

$$E_u^0 = (E_{-v} \setminus P_{-v}) \cup P_{-u} \cup \{(u, v)\}$$

is a cycle cover of G_u^0. To prove this, we have to show $\text{indeg}_{E_u^0}(x) = \text{outdeg}_{E_u^0}(x) = 1$ for all $x \in V$:

(i) We removed the outgoing edge of u in E_{-v}, which is in P_{-v}. The incoming edge of u in E_{-v} is left. P_{-u} does not contain any edge incident to u, and (u, v) is an outgoing edge of u. Thus, $\text{indeg}_{E_u^0}(u) = \text{outdeg}_{E_u^0}(u) = 1$.

(ii) There is no edge incident to v in E_{-v}. P_{-u} contains an outgoing edge of v, and (u, v) is an incoming edge of v. Thus, $\text{indeg}_{E_u^0}(v) = \text{outdeg}_{E_u^0}(v) = 1$.

(iii) For all $x \in V \setminus \{u, v\}$, either both P_{-v} and P_{-u} contain an incoming edge of x or neither of them does. Analogously, either both P_{-v} and P_{-u} contain an outgoing edge of x or neither of them does. Thus, replacing P_{-v} by P_{-u} changes neither $\text{indeg}(x)$ nor $\text{outdeg}(x)$.

By replacing the edge (u, v) by a path $(u, y_1), \ldots, (y_k, v)$, we obtain a cycle cover of G_u^k for all $k \in \mathbb{N}$. A cycle cover of G_v^0 is obtained similarly:

$$E_v^0 = (E_{-u} \setminus P_{-u}) \cup P_{-v} \cup \{(v, u)\}.$$

As above, we get cycle covers of G_v^k by replacing (v, u) by a path $(v, y_1), \ldots, (y_k, u)$. Figure 7(d) shows an example of how the new cycle covers are obtained.

The second case is that P starts and ends at u and P' starts and ends at v. Then

$$(E_{-v} \setminus P_{-u}) \cup P_{-v} \text{ and } (E_{-u} \setminus P'_{-v}) \cup P'_{-u}$$

are cycle covers of G, and

$$(E_{-v} \setminus P_{-v}) \cup P_{-u} \text{ and } (E_{-u} \setminus P'_{-u}) \cup P'_{-v}$$

are cycle covers of G_{-u-v}. The proof is similar to the first case. Figure 8(d) shows an example. □

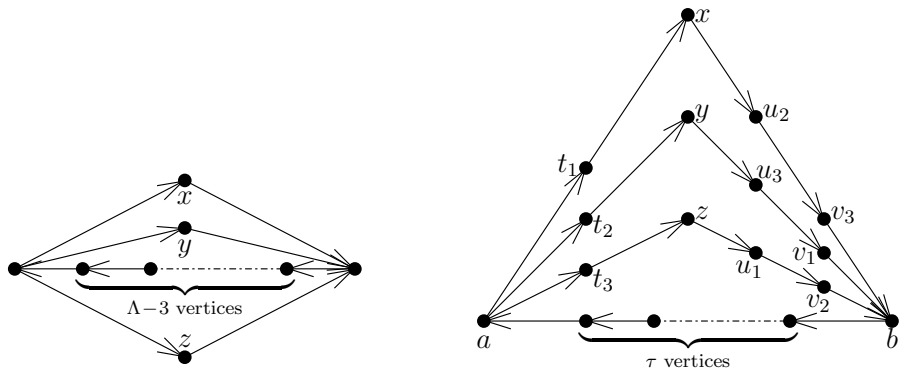
2.5. Intractability for directed graphs. From the hardness results in the previous sections and the work by Hell et al. [22], we obtain the NP-hardness and APX-hardness of L-DCC and Max-L-DCC(0,1), respectively, for all L with $2 \notin L$ and $\overline{L} \not\subseteq \{2, 3, 4\}$: We use the same reduction as for undirected cycle covers and replace every undirected edge $\{u, v\}$ by a pair of directed edges (u, v) and (v, u). However, this does not work if $2 \in L$ and also leaves open the cases when $\overline{L} \subsetneq \{2, 3, 4\}$. $\mathcal{D}$-DCC, Max-$\mathcal{D}$-DCC(0,1), and Max-$\mathcal{D}$-DCC can be solved in polynomial time, but the case $L = \{2\}$ is also easy: Replace two opposite edges (u, v) and (v, u) by an edge $\{u, v\}$ of weight $w(u, v) + w(v, u)$, and compute a matching of maximum weight on the undirected graph thus obtained.

We will settle the complexity of the directed cycle cover problems by showing that $L = \{2\}$ and $L = \mathcal{D}$ are the only tractable cases. For all other L, L-DCC is NP-hard, and Max-L-DCC(0,1) and Max-L-DCC are APX-hard. Let us start by proving the APX-hardness.

THEOREM 2.13. *Let $L \subseteq \mathcal{D}$ be a nonempty set. If $L \notin \{\{2\}, \mathcal{D}\}$, then* Max-$L$-DCC(0,1) *is* APX*-hard.*

Proof. We adapt the proof presented in section 2.2. Since $L \neq \{2\}$, there exists a $\lambda \in L$, with $\lambda \geq 3$. Thus, Min-Vertex-Cover(λ) is APX-complete. All we need is such a λ and a directed L-clamp. Then we can reduce Min-Vertex-Cover(λ) to Max-L-DCC(0,1).

We use the L-clamps to build L-gadgets, which again should have the property that they absorb one of their connectors and expel the other two. In the case of L being finite, the graph shown in Figure 9(a) is a directed L-gadget. In the case of



(a) L-gadget for finite L.

(b) L-gadget for infinite L with $\tau + 6 \in L$. The triple clamps are represented by their connectors t_i, u_i, v_i.

FIG. 9. *Directed L-gadgets with connectors x, y, z.*

infinite L, we can build directed triple L-clamps exactly as for undirected graphs. By using these, we can build directed L-gadgets, which are simply directed variants of their undirected counterparts (Figure 9(b)).

The edge gadgets build the graph G_1: Let $x \in X$ be a vertex of H and $a_1, \ldots, a_\lambda \in F$ be the edges incident to x in H (in arbitrary order). Then we assign weight one to the edges $(x^1_{a_\xi}, x^1_{a_{\xi+1}})$ for all $\xi \in \{1, \ldots, \lambda - 1\}$. The edge $(x^1_{a_\lambda}, x^1_{a_1})$ has weight zero. These λ edges are called the junctions of x.

Again, $G_2, \ldots, G_\lambda$ are exact copies of G_1 except that weight one is assigned also to $(x^\xi_{a_\lambda}, x^\xi_{a_1})$ for all $\xi \in \{2, 3, \ldots, \lambda\}$.

Again, we let the z-vertices form λ-cycles: For all edges $a \in F$, we assign weight one to $(z^\xi_a, z^{\xi+1}_a)$ for $\xi \in \{1, 2, \ldots, \lambda - 1\}$ and to (z^λ_a, z^1_a).

Weight zero is assigned to all edges that are not mentioned.

The remainder of the proof goes along the same lines as the APX-hardness proof for undirected L-cycle covers. □

Note that the NP-hardness of L-DCC for $L \notin \{\{2\}, \mathcal{D}\}$ does not follow directly from the APX-hardness of Max-L-DCC(0,1): A famous counterexample is 2SAT, for which it is APX-hard to maximize the number of simultaneously satisfied clauses [25], although testing whether a 2CNF formula is satisfiable takes only linear time.

THEOREM 2.14. *Let $L \subseteq \mathcal{D}$ be a nonempty set. If $L \notin \{\{2\}, \mathcal{D}\}$, then* L-DCC *is* NP*-hard.*

Proof. All we need is an L-clamp and some $\lambda \in L$, with $\lambda \geq 3$. We present a reduction from λ-XC (which is NP-complete since $\lambda \geq 3$) that is similar to the reduction of Hell et al. [22] used to prove the NP-hardness of L-UCC for $\overline{L} \not\subseteq \{3, 4\}$.

Let (X, F) be an instance of λ-XC. Note that we will construct a directed graph G as an instance of L-DCC; i.e., G is neither complete nor edge-weighted. For each $x \in X$, we have a vertex in G that we again call x. For $a = \{x_1, \ldots, x_\lambda\} \in F$, we construct a λ-cycle consisting of the vertices $a_1, \ldots, a_\lambda$. Then we add λ L-clamps $K^{x_\eta}_a$, with a_η and x_η as connectors for all $\eta \in \{1, \ldots, \lambda\}$. See Figure 10 for an example.

What remains to be shown is that G contains an L-cycle cover if and only if F is a "yes" instance of λ-XC. Assume first that there exists a subset $\tilde{F} \subseteq F$ such that $\bigcup_{a \in \tilde{F}} a = X$ and every element $x \in X$ is contained in exactly one set of $\tilde{F}$. We construct an L-cycle cover of G in which all clamps are healthy: Let $a = \{x_1, \ldots, x_\lambda\} \in F$. If $a \in \tilde{F}$, then let $K^{x_\eta}_a$ expel a_η and absorb x_η for all $\eta \in \{1, \ldots, \lambda\}$, and

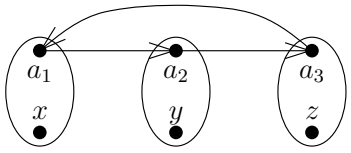


FIG. 10. *The construction for the* NP*-hardness of* L-DCC *from the viewpoint of* $a = \{x, y, z\} \in F$. *Each ellipse represents an* L*-clamp.*

let $a_1, a_2, \ldots, a_\lambda$ form a λ-cycle. If $a \notin \tilde{F}$, let $K_a^{x_\eta}$ expel x_η and absorb a_η for all $\eta \in \{1, \ldots, \lambda\}$. All connectors are absorbed by exactly one clamp or are covered by a λ-cycle since $\tilde{F}$ is an exact cover.

Now we prove the reverse direction. Let C be an L-cycle cover of G. Then every clamp of G is healthy in C; i.e., it absorbs one of its connectors and expels the other one. Let $a = \{x_1, \ldots, x_\lambda\} \in F$, and assume that $K_a^{x_\eta}$ expels a_η. Since a_η must be part of a cycle in C, $(a_{\eta-1}, a_\eta)$ and $(a_\eta, a_{\eta+1})$ must be in C. We obtain either that all $a_1, \ldots, a_\lambda$ are absorbed by $K_a^{x_1}, \ldots, K_a^{x_\lambda}$ or that all are expelled by $K_a^{x_1}, \ldots, K_a^{x_\lambda}$. Now consider any $x \in X$, and let $a_1, a_2, \ldots, a_\ell \in F$ be all of the sets that contain x. All clamps $K_{a_1}^x, \ldots, K_{a_\ell}^x$ are healthy, C is an L-cycle cover of G, and x is not incident to any further edges. Hence, there must be a unique a_i such that $K_{a_i}^x$ absorbs x. Thus,

$$\tilde{F} = \left\{a = \{x_1, \ldots, x_\lambda\} \in F \mid K_a^{x_\eta} \text{ absorbs } x_\eta \text{ for all } \eta \in \{1, \ldots, \lambda\}\right\}$$

is an exact cover of (X, F). □

If the language $\{1^\lambda \mid \lambda \in L\}$ is in NP, then L-DCC is also in NP and therefore NP-complete if $L \notin \{\{2\}, \mathcal{D}\}$: We can nondeterministically guess a cycle cover and then check if $\lambda \in L$ for every cycle length λ occurring in that cover. Conversely, if $\{1^\lambda \mid \lambda \in L\}$ is not in NP, then L-DCC is not in NP either since there is a reduction of $\{1^\lambda \mid \lambda \in L\}$ to L-DCC: On input $x = 1^\lambda$, construct a graph G on λ vertices that consists solely of a Hamiltonian cycle. Then $x \in L$ if and only if G contains an L-cycle cover.

3. Approximation algorithms. The goal of this section is to devise approximation algorithms for Max-L-UCC and Max-L-DCC that work for arbitrary L. The catch is that we have an uncountable number of problems Max-L-UCC and Max-L-DCC, and for most L it is impossible to decide whether some cycle length is in L or not.

Assume, for instance, that we have an algorithm that solves Max-L-UCC for some set L that is not recursively enumerable. We enumerate all instances of Max-L-UCC and run the algorithm on these instances. This yields an enumeration of a subset of L. Since L is not recursively enumerable, there exist $\lambda \in L$ such that the algorithm never outputs λ-cycles. Now consider a graph with λ vertices where all edges have weight zero except for a Hamiltonian cycle of weight one edges. Then the Hamiltonian cycle is the unique optimum solution, but our algorithm does not output the λ-cycle, contradicting the assumption that it solves Max-L-UCC.

One possibility to circumvent this problem would be to restrict ourselves to sets L such that $\{1^\lambda \mid \lambda \in L\}$ is in P. Another possibility to cope with this problem is to include the permitted cycle lengths in the input. However, while such restrictions are necessary for finding optimum solutions, it turns out that they are unnecessary for designing approximation algorithms.

A necessary and sufficient condition for a complete graph with n vertices to have an L-cycle cover is that there exist (not necessarily distinct) lengths $\lambda_1, \ldots, \lambda_k \in L$ for some $k \in \mathbb{N}$ with $\sum_{i=1}^{k} \lambda_i = n$. We call such an n *L-admissible* and define $\langle L \rangle = \{n \mid n \text{ is } L\text{-admissible}\}$. Although L can be arbitrarily complicated, $\langle L \rangle$ always allows efficient membership testing.

LEMMA 3.1. *For all $L \subseteq \mathbb{N}$, there exists a finite set $L' \subseteq L$ with $\langle L' \rangle = \langle L \rangle$.*

Proof. Let $L_{\leq \ell} = \{n \in L \mid n \leq \ell\} \subseteq L$. Let $g_L \in \mathbb{N}$ be the greatest common divisor of all numbers in L. There exists an $\ell_0 \in L$ such that g_L is also the greatest common divisor of $L_{\leq \ell_0}$.

If $g_L \in L$, then $\langle \{g_L\} \rangle = \langle L \rangle$, and we are done. Thus, we assume that $g_L \notin L$. There exist $\xi_1, \ldots, \xi_k \in \mathbb{Z}$ and $\lambda_1, \ldots, \lambda_k \in L_{\leq \ell_0}$ for some $k \in \mathbb{N}$ with $\sum_{i=1}^{k} \xi_i \lambda_i = g_L$. Let $\xi = \min_{1 \leq i \leq k} \xi_i$. We have $\xi < 0$ since $g_L \notin L$. Choose any $\lambda \in L_{\leq \ell_0}$, and let $\ell = -\xi\lambda \cdot \sum_{i=1}^{k} \lambda_i$. Let $n \in \langle L \rangle$ with $n \geq \ell$, let $m = \bmod(n - \ell, \lambda)$, and let $s = \left\lfloor \frac{n-\ell}{\lambda} \right\rfloor$. We can write n as

$$n = \lambda s + m + \ell = \lambda s + \frac{m}{g_L} \cdot \sum_{i=1}^{k} \xi_i \lambda_i - \lambda\xi \cdot \sum_{i=1}^{k} \lambda_i = \lambda s + \sum_{i=1}^{k} (m\xi_i - \lambda\xi) \cdot \lambda_i.$$

Since $m < \lambda$ and $\xi_i \geq \xi < 0$, we have $(m\xi_i - \lambda\xi) \geq 0$ for all i. Hence, $\langle L_{\leq \ell_0} \rangle$ contains all elements $n \in \langle L \rangle$, with $n \geq \ell$. Elements of $\langle L \rangle$ smaller than ℓ are contained in $\langle L_{\leq \ell} \rangle \supseteq \langle L_{\leq \ell_0} \rangle$. Hence, $\langle L_{\leq \ell} \rangle = \langle L \rangle$, and $L' = L_{\leq \ell}$ is the finite set for which we are looking. □

For every fixed L, we can not only test in time polynomial in n whether n is L-admissible, but we can, provided that $n \in \langle L \rangle$, also find numbers $\lambda_1, \ldots, \lambda_k \in L'$ that add up to n, where $L' \subseteq L$ denotes a finite set with $\langle L \rangle = \langle L' \rangle$. This can be done via dynamic programming in time $O(n \cdot |L'|)$, which is $O(n)$ for fixed L.

Although $\langle L \rangle = \langle L' \rangle$, there are clearly graphs for which the weights of an optimal L-cycle cover and an optimal L'-cycle cover differ: Let $\lambda \in L \setminus L'$, and consider a λ-vertex graph where all edge weights are zero except for one Hamiltonian cycle of weight one edges. However, this does not matter for our approximation algorithms.

The two approximation algorithms presented in sections 3.2 and 3.3 are based on a decomposition technique for cycle covers presented in section 3.1.

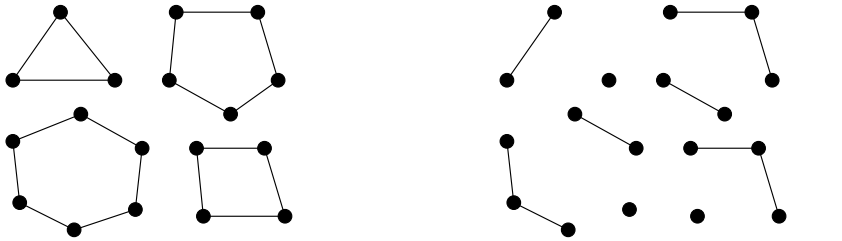
3.1. Decomposing cycle covers. In this section, we present a decomposition technique for cycle covers. The technique can be applied to cycle covers of undirected graphs but also to directed cycle covers that do not contain 2-cycles.

A *single* is a single edge (or a path of length one) in a graph, while a *double* is a path of length two. Our aim is to decompose a cycle cover C on n vertices into roughly $n/6$ singles, $n/6$ doubles, and $n/6$ isolated vertices. If n is not divisible by six, we replace $n/6$ by $\lfloor n/6 \rfloor$ or $\lceil n/6 \rceil$: If $n = 6k + \ell$ for $k, \ell \in \mathbb{N}$ and $\ell \leq 5$, then we take $k + \alpha_\ell$ singles and $k + \beta_\ell$ doubles, where α_ℓ and β_ℓ are given in Table 1. Thus, we retain half of the edges of C. We aim to decompose the cycle covers such that at least half of the weight of the cycle cover is preserved.

The reason why we decompose cycle covers into singles and doubles is the following: We cannot decompose them into longer paths in general since this does not work for $\{3\}$-cycle covers. If we restricted ourselves to decomposing the cycle covers into singles only, then 3-cycles would limit the weight preserved: We would retain only one-third of the edges of the 3-cycles and thus at most one-third of their weight in general. Finally, if we restricted ourselves to doubles, then 5-cycles would limit the weight we could obtain since we would retain only two of their five edges.

TABLE 1
A cycle cover on $n = 6k + \ell$ vertices will be decomposed into $k + \alpha_\ell$ singles and $k + \beta_\ell$ doubles.

ℓ	0	1	2	3	4	5
α_ℓ	0	1	1	0	0	1
β_ℓ	0	0	0	1	1	1



(a) A cycle cover. (b) A decomposition of the cycle cover.

FIG. 11. *An example of a decomposition according to Lemma* 3.2.

In our approximation algorithms, we exploit the following observation: If every cycle cover on n vertices can be decomposed into α singles and β doubles, then, for every L, every L-cycle cover on n vertices can be decomposed in the same way. This implies that we can build cycle covers from such a decomposition: Given α singles and β doubles, and $n - 2\alpha - 3\beta$ isolated vertices, we can join them to form an L-cycle cover. (The only restriction is that n must be L-admissible.)

Let us now state the decomposition lemma.

LEMMA 3.2. *Let $C = (V, E)$ be a cycle cover on $n = 6k + \ell$ vertices such that the length of each cycle is at least three. Let $w : E \to \mathbb{N}$ be an edge weight function.*

Then there exists a decomposition $D \subseteq E$ of C such that (V, D) consists of vertex-disjoint $k + \alpha_\ell$ singles, $k + \beta_\ell$ doubles, and $n - 5k - 3\beta_\ell - 2\alpha_\ell$ isolated vertices and $w(D) \geq w(E)/2$, where α_ℓ and β_ℓ are given in Table 1.

The decomposition can be done in polynomial time.

Figure 11 illustrates how a cycle cover is decomposed into singles and doubles.

Let us first prove some helpful lemmas.

LEMMA 3.3. *Let $\lambda, \alpha, \beta \in \mathbb{N}$, with $\alpha + 2\beta \geq \lambda/2$ and $2\alpha + 3\beta \leq \lambda$. Then every cycle c of length λ can be decomposed into α singles and β doubles such that the weight of the decomposition is at least $w(c)/2$.*

Proof. Every single involves two vertices of c, while every double involves three vertices. Thus, $2\alpha + 3\beta \leq \lambda$ is a necessary condition for c being decomposable into α singles and β doubles. It is also a sufficient condition.

We assign an arbitrary orientation to c. Let $e_0, \ldots, e_{\lambda-1}$ be the consecutive edges of c, where e_0 is chosen uniformly at random among the edges of c. We take α singles $e_0, e_2, \ldots, e_{2\alpha-2}$ and β doubles $(e_{2\alpha}, e_{2\alpha+1}), (e_{2\alpha+3}, e_{2\alpha+4}), \ldots, (e_{2\alpha+3\beta-3}, e_{2\alpha+3\beta-2})$. Since $2\alpha + 3\beta \leq \lambda$, this is a feasible decomposition. The probability that any fixed edge of c is included in the decomposition is $\frac{\alpha+2\beta}{\lambda}$. Thus, the expected weight of the decomposition is $\frac{\alpha+2\beta}{\lambda} \cdot w(c) \geq w(c)/2$. ∎

LEMMA 3.4. *Let $\lambda \in \mathbb{N}$. Suppose that every cycle c of length λ can be decomposed into α singles and β doubles of weight at least $w(c)/2$. Then every cycle c' of length $\lambda + 6$ can be decomposed into $\alpha + 1$ singles and $\beta + 1$ doubles of weight at least $w(c')/2$.*

TABLE 2

The induction basis. The columns α and β show the number of singles and doubles needed, respectively. We denote by $\lambda \rightsquigarrow (\alpha, \beta)$ that a λ-cycle is decomposed into α singles and β doubles. If there are two lines for a case, then the option that yields more weight is chosen.

Length	ℓ	α	β
3	3	0	1
4	4	0	1
5	5	1	1
6	0	1	1
7	1	2	1
8	2	2	1

(a) One cycle.

Lengths	ℓ	α	β	Decomposition
$3+3$	0	1	1	$3\rightsquigarrow(1,0) + 3\rightsquigarrow(0,1)$
$3+5$	2	2	1	$3\rightsquigarrow(1,0) + 5\rightsquigarrow(1,1)$ or $3\rightsquigarrow(0,1) + 5\rightsquigarrow(2,0)$
$3+7$	4	1	2	$3\rightsquigarrow(1,0) + 7\rightsquigarrow(0,2)$ or $3\rightsquigarrow(0,1) + 7\rightsquigarrow(1,1)$
$5+5$	4	1	2	$5\rightsquigarrow(0,1) + 5\rightsquigarrow(1,1)$
$5+7$	0	2	2	$5\rightsquigarrow(2,0) + 7\rightsquigarrow(0,2)$ or $5\rightsquigarrow(1,1) + 7\rightsquigarrow(1,1)$
$7+7$	2	3	2	$7\rightsquigarrow(1,1) + 7\rightsquigarrow(2,1)$

(b) Two odd cycles.

TABLE 3

Induction step.

Length	ℓ	α	β
4	0,3,4,5	0	1
4	1,2	2	0
6	all	1	1
8	0,1,2,5	2	1
8	3,4	0	2

(a) Removing an even cycle.

Lengths	ℓ	α	β	Decomposition
$3+3$	all	1	1	$3\rightsquigarrow(1,0) + 3\rightsquigarrow(0,1)$
$3+7$	0,3,4,5	1	2	$3\rightsquigarrow(1,0) + 7\rightsquigarrow(0,2)$ or $3\rightsquigarrow(0,1) + 7\rightsquigarrow(1,1)$
$3+7$	1,2	3	1	$3\rightsquigarrow(1,0) + 7\rightsquigarrow(2,1)$ or $3\rightsquigarrow(0,1) + 7\rightsquigarrow(3,0)$
$5+5$	0,3,4,5	1	2	$5\rightsquigarrow(0,1) + 5\rightsquigarrow(1,1)$
$5+5$	1,2	3	1	$5\rightsquigarrow(2,0) + 5\rightsquigarrow(1,1)$
$5+7$	all	2	2	$5\rightsquigarrow(2,0) + 7\rightsquigarrow(0,2)$ or $5\rightsquigarrow(1,1) + 7\rightsquigarrow(1,1)$
$7+7$	0,1,2,5	3	2	$7\rightsquigarrow(1,1) + 7\rightsquigarrow(2,1)$
$7+7$	3,4	1	3	$7\rightsquigarrow(1,1) + 7\rightsquigarrow(0,2)$

(b) Removing two odd cycles.

Proof. We have $\alpha+2\beta \geq \lambda/2$ and $2\alpha+3\beta \leq \lambda$. Thus, $\alpha+1+2(\beta+1) \geq (\lambda+6)/2$ and $2(\alpha+1)+3(\beta+1) \leq \lambda+6$. The lemma follows from Lemma 3.3. □

Lemma 3.4 also holds if we consider more than one cycle: Assume that every collection of k cycles of lengths $\lambda_1, \ldots, \lambda_k$ can be decomposed into α singles and β doubles such that the weight of the decomposition is at least half the weight of the cycles. Then k cycles of lengths $\lambda_1+6, \lambda_2, \ldots, \lambda_k$ can be decomposed into $\alpha+1$ singles and $\beta+1$ doubles such that also at least half of the weight of the cycles is preserved. Due to Lemma 3.4, we can restrict ourselves to cycles of length at most eight in the following. The reason for this is the following: If we know how to decompose cycles of length λ, then we also know how to decompose cycles of length $\lambda+6, \lambda+12, \ldots$ from Lemma 3.4.

We are now prepared to prove Lemma 3.2.

Proof of Lemma 3.2. We prove the lemma by induction on the number of cycles. As the induction basis, we consider a cycle cover consisting of either a single cycle or of two odd cycles. See Table 2. Due to Lemma 3.4, we can restrict ourselves to considering cycles of length at most eight. Tables 3(a) and 3(b) show how to decompose a single cycle and two odd cycles, respectively. We always perform the decomposition such that the weight preserved is maximized. In particular, if there are two odd cycles of different length, we have two options in how to decompose these cycles, and we choose the one that yields the larger weight. Overall, we obtain a decomposition with an appropriate number of singles and doubles that preserves at least one half of the weight.

TABLE 4
The complexity of computing L-cycle covers.

	L-UCC	Max-L-UCC(0,1)	Max-L-UCC
$\overline{L} = \emptyset$	in P	in PO	in PO
$\overline{L} = \{3\}$	in P	in PO	
$\overline{L} = \{4\}, \{3,4\}$			APX-complete
$\overline{L} \not\subseteq \{3,4\}$	NP-hard	APX-hard	APX-hard

(a) Undirected cycle covers.

	L-DCC	Max-L-DCC(0,1)	Max-L-DCC
$L = \{2\}, \mathcal{D}$	in P	in PO	in PO
$L \notin \{\{2\}, \mathcal{D}\}$	NP-hard	APX-hard	APX-hard

(b) Directed cycle covers.

As the induction hypothesis, we assume that the lemma holds if the number of cycles is less than r. Assume that we have a cycle cover C consisting of r cycles. Let $n = 6k + \ell$ for the number of its vertices for $k, \ell \in \mathbb{N}$ and $\ell \leq 5$. We remove either an even cycle or two odd cycles. In the following, let C' be the new cycle cover obtained by removing one or two cycles from C. A little more care is needed than in the induction basis: Consider, for instance, the case of removing a 4-cycle. If $\ell = 4$, then C has to be decomposed into k singles and $k+1$ doubles, while we have to take k singles and k doubles from C'. Thus, the 4-cycle has to be decomposed into a double. But if $\ell = 1$, then we need $k + 1$ singles and k doubles from C and $k - 1$ singles and k doubles from C'. Thus, the 4-cycle has to be decomposed into two singles. Overall, the 4-cycle has to be decomposed into a double if $\ell \in \{0, 3, 4, 5\}$ and into two singles if $\ell \in \{1, 2\}$. Similar case distinctions hold for all other cases. How to remove one even or two odd cycles is shown in Tables 4(a) and 4(b), respectively.

To complete the proof, we have to deal with the cases of a 3- and a 5-cycle, which are slightly more complicated and not covered by Table 4(b). We run into trouble if, for instance, $\ell = 3$. In this case, we have to take two doubles. If the 5-cycle is much heavier than the 3-cycle, then it is impossible to preserve half of the weight of the two cycles. But we can avoid this problem: As long as there is an even cycle, we decompose this one. After that, as long as there are at least three odd cycles, we can choose two of them such that we do not have a pair of one $(3 + 6\xi)$-cycle and one $(5 + 6\xi')$-cycle for some $\xi, \xi' \in \mathbb{N}$. The only situation in which it can happen that we cannot avoid decomposing a $(3+6\xi)$-cycle and a $(5+6\xi')$-cycle is when there are only two cycles left. In this case, we have $\ell = 2$, and we have treated this case already in the induction basis. □

If we consider directed graphs where 2-cycles can also occur, only one-third of the weight can be preserved. This can be done by decomposing the cycle cover into a matching of cardinality $\lceil n/3 \rceil$. (Every λ-cycle can be decomposed into a matching of size up to $\lfloor \lambda/2 \rfloor \geq \lceil \lambda/3 \rceil$. The bottleneck is 3-cycles, which yield only one edge.)

An obvious question is whether the decomposition lemma can be improved in order to preserve more than half of the weight or more than one-third of the weight if we additionally allow 2-cycles. Unfortunately, this is not the case.

A generic decomposition lemma states the following: For every $n \in \mathbb{N}$, every k-cycle cover (for $k \in \{2, 3\}$) on n vertices can be decomposed into α singles and β doubles such that at least a fraction r of the weight of the cycle cover is preserved. (As already mentioned, longer paths are impossible due to 3-cycles.) Lemma 3.2 instantiates this generic lemma with $\alpha \approx n/6$, $\beta \approx n/6$, and $r = 1/2$. In case of the

Input: undirected complete graph $G = (V, E)$, $|V| = n$; edge weights $w : E \to \mathbb{N}$
Output: an L-cycle cover C^{apx} of G if n is L-admissible, $\perp$ otherwise
1: **if** $n \notin \langle L \rangle$ **then**
2: **return** $\perp$
3: compute a cycle cover C^{init} in G of maximum weight
4: decompose C^{init} into a set $D \subseteq C^{\text{init}}$ of edges according to Lemma 3.2
5: join the singles and doubles in D to obtain an L-cycle cover C^{apx}
6: **return** C^{apx}

Algorithm 1: A 2 approximation algorithm for Max-L-UCC.

presence of 2-cycles, we have sketched a decomposition with $\alpha \approx n/3$, $\beta = 0$, and $r = 1/3$.

LEMMA 3.5. *No decomposition technique for* 3*-cycle covers can in general preserve more than one-half of the weight of the* 3*-cycle covers.*

Furthermore, no decomposition technique for 2*-cycle covers can in general preserve more than one-third of the weight of the* 2*-cycle covers.*

Proof. We exploit the fact that the fraction of edges that are preserved in a cycle cover decomposition is a lower bound for the fraction of the weight that can be preserved.

Since, in particular, $\{3\}$-cycle covers have to be decomposed, we cannot decompose the cycle cover into paths of length more than two. Now consider decomposing a $\{4\}$-cycle cover. Since paths of length 3 are not allowed, we have to discard two edges of every 4-cycle. Thus, at most two edges of every 4-cycle are preserved, which proves the first part of the lemma.

The second part follows analogously by considering 3-cycles and observing that paths of length two or more are not allowed. ∎

Overall, Lemma 3.5 shows that every approximation algorithm for Max-L-UCC or Max-L-DCC that works for arbitrary sets L and is purely decomposition-based achieves approximation ratios of at best 2 or 3, respectively. We achieve an approximation ratio of $8/3 < 3$ for Max-L-DCC by paying special attention to 2-cycles (section 3.3).

3.2. Undirected cycle covers. Our approximation algorithm for Max-L-UCC (Algorithm 1) directly exploits Lemma 3.2.

THEOREM 3.6. *Algorithm* 1 *is a factor* 2 *approximation algorithm for* Max-L-UCC *for all* $L \subseteq \mathcal{U}$*. Its running time is* $O(n^3)$*.*

Proof. If L is infinite, we replace L by a finite set $L' \subseteq L$, with $\langle L' \rangle = \langle L \rangle$ according to Lemma 3.1. Algorithm 1 returns $\perp$ if and only if $n \notin \langle L \rangle$. Otherwise, an L-cycle cover C^{apx} is returned. Let $C^\star$ denote an L-cycle cover of maximum weight of G. We have $w(C^\star) \leq w(C^{\text{init}}) \leq 2 \cdot w(D) \leq 2 \cdot w(C^{\text{apx}})$. The first inequality holds because L-cycle covers are special cases of cycle covers. The second inequality holds due to the decomposition lemma (Lemma 3.2). The last inequality holds since no weight is lost during the joining. Overall, the algorithm achieves an approximation ratio of 2.

The running time of the algorithm is dominated by the time needed to compute the initial cycle cover, which is $O(n^3)$ [1, Chapter 12]. ∎

3.3. Directed cycle covers. In the following, let C^{opt} be an L-cycle cover of maximum weight. Let w_λ denote the weight of the λ-cycles in C^{opt}; i.e., $w(C^{\text{opt}}) = \sum_{\lambda \geq 2} w_\lambda$.

We distinguish three cases: First, $2 \notin L$; second, $2 \in L$ and $3 \notin L$; and third, $2, 3 \in L$.

We use the decomposition lemma (Lemma 3.2) only if $2 \notin L$. In this case, the weight of an optimal L-cycle cover is at most the weight of an optimal 3-cycle cover C_3^{opt}. Thus, we proceed as follows: First, we compute a 4/3 approximation C_3^{init} for Max-3-DCC, which can be done by using the algorithm of Bläser, Ram, and Sviridenko [7]. We have $w(C_3^{\text{init}}) \geq \frac{3}{4} \cdot w(C_3^{\text{opt}}) \geq \frac{3}{4} \cdot w(C^{\text{opt}})$. Now we decompose C_3^{init} into a collection D of singles and doubles according to Lemma 3.2. Finally, we join the singles, doubles, and isolated vertices of D to form an L-cycle cover C^{apx}. We obtain a factor 8/3 approximation for the case that $2 \notin L$:

$$w(C^{\text{apx}}) \geq w(D) \geq \frac{1}{2} \cdot w(C_3^{\text{init}}) \geq \frac{3}{8} \cdot w(C^{\text{opt}}).$$

Now we consider the case that $2 \in L$ and $3 \notin L$. In this case, a matching-based algorithm achieves an approximation ratio of 5/2: We compute a matching of a certain cardinality, which we will specify in a moment, and then we join the edges of the matching to obtain an L-cycle cover. The cardinality of the matching is chosen such that an L-cycle cover can be built from such a matching. A λ-cycle yields a matching of cardinality $\lfloor \lambda/2 \rfloor$. Thus, a matching of cardinality d in a graph of n vertices can be extended to form an L-cycle cover if and only if $d \leq D(n, L)$, where

$$D(n, L) = \max \left\{ \sum_{i=1}^{k} \lfloor \lambda_i/2 \rfloor \mid k \in \mathbb{N}, \sum_{i=1}^{k} \lambda_i = n, \text{ and } \lambda_i \in L \text{ for } 1 \leq i \leq k \right\} \leq \frac{n}{2}.$$

Given L, we can compute $D(n, L)$ by using dynamic programming. Let us now estimate the weight of a matching of cardinality at most $D(n, L)$ that has maximum weight among all such matchings. From C^{opt}, we obtain a matching with a weight of at least

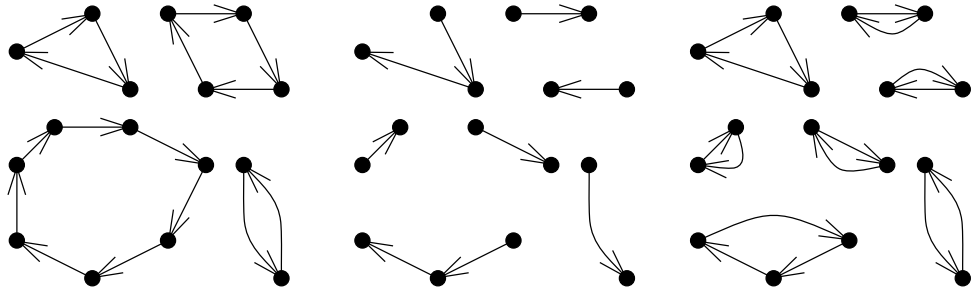
$$\sum_{\lambda \geq 2} \frac{1}{\lambda} \cdot \left\lfloor \frac{\lambda}{2} \right\rfloor \cdot w_\lambda \geq \sum_{\lambda \geq 2} \frac{2}{5} \cdot w_\lambda = \frac{2}{5} \cdot w(C^{\text{opt}}).$$

The reason is that $w_3 = 0$ because $3 \notin L$ and that $\min_{\lambda \in \{2,4,5,6,7,\dots\}} \frac{1}{\lambda} \cdot \lfloor \lambda/2 \rfloor \geq 2/5$. Thus, by computing a maximum-weight matching M of cardinality at most $D(n, L) \geq 2n/5$ and joining the edges to form an L-cycle cover C^{apx}, we obtain a factor 5/2 approximation.

What remains to be considered is the case that $2, 3 \in L$. In this case, we start by computing an initial cycle cover C^{init} (without any restrictions). Then we do the following: For every even cycle, we take every other edge such that at least one-half of its weight is preserved. For every edge thus obtained, we add the converse edge to obtain a collection of 2-cycles. For every odd cycle, we take every other edge and one path of length two such that at least half of the weight is preserved. Then we add edges to obtain 2-cycles and one 3-cycle. In this way, we obtain a $\{2, 3\}$-cycle cover C^{apx}, which is also an L-cycle cover. We have $w(C^{\text{apx}}) \geq \frac{1}{2} \cdot w(C^{\text{init}}) \geq \frac{1}{2} \cdot w(C^{\text{opt}})$. Figure 12 shows an example.

Our approximation algorithm is summarized as Algorithm 2. The running time of the algorithm of Bläser, Ram, and Sviridenko is polynomial [7], and all other steps can be executed in polynomial time as well. Thus, the running time of Algorithm 2 is also polynomial.

THEOREM 3.7. *Algorithm* 2 *is a factor* 8/3 *approximation algorithm for* Max-L-UCC *for all nonempty sets* $L \subseteq \mathcal{D}$. *Its running time is polynomial.*



(a) Initial cycle cover C^{init}. (b) Decomposition of C^{init}. (c) $\{2,3\}$-cycle cover C^{apx}.

FIG. 12. *Sketch of the algorithm for* $\{2,3\} \subseteq L$.

Input: directed complete graph $G = (V, E)$, $|V| = n$; edge weights $w : E \to \mathbb{N}$
Output: an L-cycle cover C^{apx} of G if n is L-admissible, $\perp$ otherwise

if $n \notin \langle L \rangle$ **then**
    return $\perp$
if $2 \in L$ and $3 \in L$ **then**
    compute a cycle cover C^{init} (without restrictions)
    for all even cycles c of C^{init} **do**
        take every other edge of c such that at least one-half of c's weight is preserved
        add the converse edges to obtain 2-cycles; add these cycles to C^{apx}
    for all odd cycles c of C^{init} **do**
        take every other edge and one path of length two of c such that at least one-half of c's weight is preserved
        add edges to obtain 2-cycles plus one 3-cycle; add these cycles to C^{apx}
else if $2 \in L$, $3 \notin L$ **then**
    compute a matching M of maximum weight of cardinality at most $D(n, L)$
    join the edges of M to form an L-cycle cover C^{apx}
else $(2 \notin L)$
    compute a 4/3 approximation C_3^{init} to an optimal 3-cycle cover
    decompose C_3^{init} into a set $D \subseteq C_3^{\text{init}}$ of edges according to Lemma 3.2
    join the singles and doubles in D to obtain an L-cycle C^{apx}
return C^{apx}

Algorithm 2: A factor 8/3 approximation algorithm for Max-L-DCC.

4. Conclusions. For almost all L, finding L-cycle covers is NP-hard, and finding L-cycle covers of maximum weight is APX-hard. Table 4 shows an overview. Although this shows that computing restricted cycle covers is generally very hard, we have proved that L-cycle covers of maximum weight can be approximated within a constant factor in polynomial time for all L.

For directed graphs, we have settled the complexity: If $L = \{2\}$ or $L = \mathcal{D}$, then L-DCC, Max-L-DCC(0,1), and Max-L-DCC are solvable in polynomial time; otherwise, they are intractable. For undirected graphs, the status of only five cycle cover problems remains open: L-UCC and Max-L-UCC(0,1) for $\overline{L} = \{4\}, \{3,4\}$ and Max-4-UCC.

There are some reasons for optimism that L-UCC and Max-L-UCC(0,1) for $\overline{L} = \{4\}, \{3,4\}$ are solvable in polynomial time: Hartvigsen [18] devised a polynomial-time algorithm for finding $\overline{\{4\}}$-cycle covers in bipartite graphs (forbidding 3-cycles does not change the problem for bipartite graphs). Moreover, there are augmenting path theorems for L-cycle covers for all L with $\overline{L} \subseteq \{3,4\}$ [26], which includes the two cases that are known to be polynomial-time solvable. Augmenting path theorems are often a building block for matching algorithms. But there are also augmenting path theorems for $L \subseteq \{3,4\}$ [26], even though these L-cycle cover problems are intractable.

Acknowledgments. I thank Jan Arpe and Martin Böhme for valuable discussions and comments.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] P. Alimonti and V. Kann, *Some APX-completeness results for cubic graphs*, Theoret. Comput. Sci., 237 (2000), pp. 123–134.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, New York, 1999.

[4] M. Bläser, *A 3/4-approximation algorithm for maximum ATSP with weights zero and one*, in Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Lecture Notes in Comput. Sci. 3122, Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, eds., Springer, New York, 2004, pp. 61–71.

[5] M. Bläser and B. Manthey, *Approximating maximum weight cycle covers in directed graphs with weights zero and one*, Algorithmica, 42 (2005), pp. 121–139.

[6] M. Bläser, B. Manthey, and J. Sgall, *An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality*, J. Discrete Algorithms, 4 (2006), pp. 623–632.

[7] M. Bläser, L. S. Ram, and M. I. Sviridenko, *Improved approximation algorithms for metric maximum ATSP and maximum* 3*-cycle cover problems*, in Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 3608, Frank Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, eds., Springer, New York, 2005, pp. 350–359.

[8] M. Bläser and B. Siebert, *Computing cycle covers without short cycles*, in Proceedings of the 9th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 2161, Friedhelm Meyer auf der Heide, ed., Springer, New York, 2001, pp. 368–379. Bodo Siebert is the birth name of Bodo Manthey.

[9] A. L. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis, *Linear approximation of shortest superstrings*, J. ACM, 41 (1994), pp. 630–647.

[10] H. J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger, *Approximation algorithms for the TSP with sharpened triangle inequality*, Inform. Process. Lett., 75 (2000), pp. 133–138.

[11] L. S. Chandran and L. S. Ram, *On the relationship between ATSP and the cycle cover problem*, Theoret. Comput. Sci., 370 (2007), pp. 218–228.

[12] Z.-Z. Chen and T. Nagoya, *Improved approximation algorithms for metric MaxTSP*, J. Comb. Optim., 13 (2007), pp. 321–336.

[13] Z.-Z. Chen, Y. Okamoto, and L. Wang, *Improved deterministic approximation algorithms for Max TSP*, Inform. Process. Lett., 95 (2005), pp. 333–342.

[14] M. Chlebík and J. Chlebíková, *Complexity of approximating bounded variants of optimization problems*, Theoret. Comput. Sci., 354 (2006), pp. 320–338.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[16] P. C. Gilmore, E. L. Lawler, and D. B. Shmoys, *Well-solved special cases*, in The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, eds., John Wiley & Sons, New York, 1985, pp. 87–143.

[17] D. Hartvigsen, *An Extension of Matching Theory*, Ph.D. thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, PA, 1984.
[18] D. Hartvigsen, *Finding maximum square-free* 2*-matchings in bipartite graphs*, J. Combin. Theory Ser. B, 96 (2006), pp. 693–705.
[19] R. Hassin and S. Rubinstein, *On the complexity of the* k*-customer vehicle routing problem*, Oper. Res. Lett., 33 (2005), pp. 71–76.
[20] R. Hassin and S. Rubinstein, *An approximation algorithm for maximum triangle packing*, Discrete Appl. Math., 154 (2006), pp. 971–979.
[21] R. Hassin and S. Rubinstein, *Erratum to "An approximation algorithm for maximum triangle packing"* [Discrete Appl. Math., 154 (2006), pp. 971–979], Discrete Appl. Math., 154 (2006), p. 2620.
[22] P. Hell, D. Kirkpatrick, J. Kratochvíl, and I. Kríz, *On restricted two-factors*, SIAM J. Discrete Math., 1 (1988), pp. 472–484.
[23] H. Kaplan, M. Lewenstein, N. Shafrir, and M. I. Sviridenko, *Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs*, J. ACM, 52 (2005), pp. 602–626.
[24] L. Lovász and M. D. Plummer, *Matching Theory*, North-Holland Math. Stud. 121, Elsevier, New York, 1986.
[25] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[26] M. P. Russell, *Restricted Two-Factors*, Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2001.
[27] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Algorithms Combin. 24, Springer, 2003.
[28] Z. Sweedyk, *A* $2\frac{1}{2}$*-approximation algorithm for shortest superstring*, SIAM J. Comput., 29 (1999), pp. 954–986.
[29] O. Vornberger, *Easy and Hard Cycle Covers*, Technical report, Universität/Gesamthochschule Paderborn, 1980.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 207–225

TESTING GRAPH ISOMORPHISM*

ELDAR FISCHER† AND ARIE MATSLIAH†

Abstract. Two graphs G and H on n vertices are ϵ-far from being isomorphic if at least $\epsilon\binom{n}{2}$ edges must be added or removed from $E(G)$ in order to make G and H isomorphic. In this paper we deal with the question of how many queries are required to distinguish between the case that two graphs are isomorphic and the case that they are ϵ-far from being isomorphic. A query is defined as probing the adjacency matrix of any one of the two graphs, i.e., asking if a pair of vertices forms an edge of the graph or not. We investigate both one-sided and two-sided error testers under two possible settings: The first setting is where both graphs need to be queried, and the second setting is where one of the graphs is fully known to the algorithm in advance. We prove that the query complexity of the best one-sided error testing algorithm is $\widetilde{\Theta}(n^{3/2})$ if both graphs need to be queried, and that it is $\widetilde{\Theta}(n)$ if one of the graphs is known in advance (where the $\widetilde{\Theta}$ notation hides polylogarithmic factors in the upper bounds). For two-sided error testers, we prove that the query complexity of the best tester is $\widetilde{\Theta}(\sqrt{n})$ when one of the graphs is known in advance, and we show that the query complexity lies between $\Omega(n)$ and $\widetilde{O}(n^{5/4})$ if both G and H need to be queried. All of our algorithms are additionally nonadaptive, while all of our lower bounds apply for adaptive testers as well as nonadaptive ones.

Key words. property testing, graph isomorphism, approximation

AMS subject classifications. 68W20, 68Q25

DOI. 10.1137/070680795

1. Introduction.

Combinatorial property testing deals with the following task: For a fixed $\epsilon > 0$ and a fixed property P, distinguish using as few queries as possible (and with probability at least $\frac{2}{3}$) between the case that an input of length m satisfies P and the case that the input is ϵ-far (with respect to an appropriate metric) from satisfying P. The first time a question formulated in terms of property testing was considered was in the work of Blum, Luby, and Rubinfeld [8]. The general notion of property testing was first formally defined by Rubinfeld and Sudan [16], mainly for the context of the algebraic properties (such as linearity) of functions over finite fields and vector spaces. The first investigation in the combinatorial context is that of Goldreich, Goldwasser, and Ron [13], where testing of combinatorial graph properties was first formalized. The "dense" graph testing model that was defined in [13] is also the one that will serve us here. In recent years the field of property testing has enjoyed rapid growth, as witnessed in the surveys [15] and [9].

Formally, our inputs are two functions $g : \{1, 2, \ldots, \binom{n}{2}\} \to \{0, 1\}$ and $h : \{1, 2, \ldots, \binom{n}{2}\} \to \{0, 1\}$, which represent the edge sets of two corresponding graphs G and H over the vertex set $V = \{1, \ldots, n\}$. The *distance* of a graph from a property P is measured by the minimum number of bits that have to be modified in the input in order to make it satisfy P, divided by the input length m, which in our case is taken to be $\binom{n}{2}$. For the question of testing graphs with a constant number of queries, there are many recent advances, such as [4], [11], [3], and [2]. For the properties that we

*Received by the editors January 23, 2007; accepted for publication (in revised form) November 13, 2007; published electronically March 28, 2008. This research was supported in part by Israel Science Foundation grants 55/03 and 1101/06. A preliminary version appeared in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006.

http://www.siam.org/journals/sicomp/38-1/68079.html

†Faculty of Computer Science, Technion–Israel Institute of Technology, Technion City, Haifa 32000, Israel (eldar@cs.technion.ac.il, ariem@cs.technion.ac.il).

consider here the number of required queries is of the form n^α for some $\alpha > 0$, and our interest will be to find bounds as tight as possible on α. We consider the following questions:

1. Given two input graphs G and H, how many queries to G and H are required to test that the two graphs are isomorphic? This property was already used in [1] for proving lower bounds on property testing, and a lower bound of the form n^α was known for quite a while (see, e.g., [9]).
2. Given a graph G_k, which is known in advance (and for which any amount of preprocessing is allowed), and an input graph G_u, how many queries to G_u are required to test that G_u is isomorphic to G_k? Some motivation for this question comes from [10], where upper and lower bounds that correlate this question with the "inherent complexity" of the provided G_k are proved. In this paper, our interest is in finding the bounds for the "worst possible" G_k.

For the case where the testers must have one-sided error, our results show tight (up to logarithmic factors) upper and lower bounds of $\widetilde{\Theta}(n^{3/2})$ for the setting where both graphs need to be queried, and of $\widetilde{\Theta}(n)$ for the setting where one graph is given in advance. The upper bounds are achieved by trivial algorithms of edge sampling and exhaustive search. As we are interested in the number of queries, we make no attempt to optimize the running time. The main work here lies in proving a matching lower bound for the first setting where both graphs need to be queried, as the lower bound for the second setting is nearly trivial.

Unusually for graph properties that involve no explicit counting in their definition, we can do significantly better if we allow our algorithms to have two-sided error. When one graph is given in advance, we show $\widetilde{\Theta}(n^{1/2})$ upper and lower bounds. The upper bound algorithm uses a technique that allows us to greatly reduce the number of candidate bijections that need to be checked, while assuring that for isomorphic graphs one of them will still be close to an isomorphism. For this to work we need to combine it with a distribution testing algorithm from [7], whose lower bound is in some sense the true cause of the matching lower bound here.

For two-sided error testers where the two graphs need to be queried, a gap in the bounds remains. We present here a lower bound proof of $\Omega(n)$ on the query complexity—it is in fact the lower bound proof already known from the literature, only here we analyze it to its fullest potential. The upper bound of $\widetilde{O}(n^{5/4})$ uses the ideas of the algorithm above for the setting where one of the graphs is known, with an additional mechanism to compensate for having to query from both graphs to find matching vertices.

To our knowledge, the best known algorithm for deciding this promise problem in the classical sense (i.e., given two graphs, distinguish whether they are isomorphic or ϵ-far from being isomorphic) requires quasi-polynomial running time [6]. Both our two-sided error testers have the additional property of a quasi-polynomial running time (similarly to the algorithm in [6]) even with the restriction on the number of queries.

Table 1.1 summarizes our results for the query complexity in various settings. We made no effort to optimize the logarithmic factors in the upper bounds, as well as the exact dependence on ϵ (which is at most polynomial).

The rest of the paper is organized as follows. We provide some preliminaries and definitions in section 2. Upper and lower bounds for the one-sided algorithms are proved in section 3, and the upper and lower bounds for the two-sided algorithms are proved in section 4. The final section 5 contains some discussion and concluding comments.

TABLE 1.1

	Upper bound	Lower bound
One-sided error, one graph known	$\widetilde{O}(n)$	$\Omega(n)$
One-sided error, both graphs unknown	$\widetilde{O}(n^{3/2})$	$\Omega(n^{3/2})$
Two-sided error, one graph known	$\widetilde{O}(n^{1/2})$	$\Omega(n^{1/2})$
Two-sided error, both graphs unknown	$\widetilde{O}(n^{5/4})$	$\Omega(n)$

2. Notation and preliminaries. All graphs considered here are undirected and with neither loops nor parallel edges. We also assume (even where not explicitly stated) that the number of vertices of the input graph is large enough, as a function of the other parameters. We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. For a vertex v, $N(v)$ denotes the set of v's neighbors. For a pair of vertices u, v we denote by $N(u) \triangle N(v)$ the symmetric difference between $N(u)$ and $N(v)$. Given a permutation $\sigma : [n] \to [n]$ and a subset U of $[n]$, we denote by $\sigma(U)$ the set $\{\sigma(i) : i \in U\}$. Given a subset U of the vertices of a graph G, we denote by $G(U)$ the induced subgraph of G on U. We denote by $G(n, p)$ the random graph where each pair of vertices forms an edge with probability p, independently of each other.

DEFINITION 2.1. *Given two labeled graphs G and H on the same vertex set V, the* distance *between G and H is the size of the symmetric difference between the edge sets of G and H, divided by $\binom{|V|}{2}$.*

Given a graph G and a graph H on the same vertex set V, we say that H and G are ϵ-far *if the distance between G and any permutation of H is at least ϵ.*

Given a graph G and a graph property (a set of graphs that is closed under graph isomorphisms) P, we say that G is ϵ-far *from satisfying the property P if G is ϵ-far from any graph H on the same vertex set which satisfies P.*

Using this definition of the distance, we give a formal definition of a graph testing algorithm.

DEFINITION 2.2. *An* ϵ-testing algorithm with q queries *for a property P is a probabilistic algorithm that for any input graph G makes up to q queries (a query consisting of finding whether two vertices u, v of G form an edge of G or not) and satisfies the following.*

- *If G satisfies P, then the algorithm accepts G with probability at least $\frac{2}{3}$.*
- *If G is ϵ-far from P, then the algorithm rejects G with probability at least $\frac{2}{3}$.*

A property testing algorithm has one-sided error probability *if it accepts inputs that satisfy the property with probability* 1. *We also call such testers* one-sided error testers.

A property testing algorithm is nonadaptive *if the outcomes of its queries do not affect the choice of the following queries but only the decision of whether to reject or accept the input in the end.*

The following is just an extension of the above definition to properties of pairs of graphs. In our case, we will be interested in the property of two graphs being isomorphic.

DEFINITION 2.3. *An* ϵ-testing algorithm with q queries *for a property P of pairs of graphs is a probabilistic algorithm that for any input pair G, H makes up to q queries in G and H (a query consisting of finding whether two vertices u, v of G (H) form an edge of G (H) or not) and satisfies the following.*

- *If the pair G, H satisfies P, then the algorithm accepts with probability at least $\frac{2}{3}$.*

- *If the pair G, H is ϵ-far from P, then the algorithm rejects with probability at least $\frac{2}{3}$.*

To simplify the arguments when discussing the properties of the query sets, we define *knowledge charts*.

DEFINITION 2.4. *Given a query set Q to the adjacency matrix A of the graph $G = (V, E)$ on n vertices, we define the* knowledge chart *$I_{G,Q}$ of G as the subgraph of G known after making the set Q of queries to A. We partition the pairs of vertices of $I_{G,Q}$ into three classes: Q^1, Q^0, and Q^*. The pairs in Q^1 are those known to be edges of G, the pairs in Q^0 are those that are known not to be edges of G, and all unknown (unqueried) pairs are in Q^*. In other words, $Q^1 = E(G) \cap Q$, $Q^0 = Q \setminus E(G)$, and $Q^* = [V(G)]^2 \setminus Q$. For a fixed q, $0 \leq q \leq n$, and G, we define $I_{G,q}$ as the set of knowledge charts $\{I_{G,Q} : |Q| = q\}$. For example, note that $|I_{G,0}| = |I_{G,\binom{n}{2}}| = 1$.*

We will ask the question of whether two query sets are consistent, i.e., do they provide any evidence for the two graphs being nonisomorphic? We say that the knowledge charts are *knowledge-packable* if the query sets that they represent are consistent. Formally, we have the following definition.

DEFINITION 2.5. *A* knowledge-packing *of two knowledge charts I_{G_1,Q_1}, I_{G_2,Q_2}, where G_1 and G_2 are graphs with n vertices, is a bijection π of the vertices of G_1 into the vertices of G_2 such that for all $v, u \in V(G_1)$, if $\{v, u\} \in E(G_1) \cap Q_1$, then $\{\pi(v), \pi(u)\} \notin Q_2 \setminus E(G_2)$, and if $\{v, u\} \in Q_1 \setminus E(G_1)$, then $\{\pi(v), \pi(u)\} \notin E(G_2) \cap Q_2$.*

In particular, if G_1 is isomorphic to G_2, then for all $0 \leq q_1, q_2 \leq \binom{n}{2}$, every member of I_{G_1,q_1} is knowledge-packable with every member of I_{G_2,q_2}. In other words, if G_1 is isomorphic to G_2, then there is a knowledge-packing of I_{G_1,Q_1} and I_{G_2,Q_2} for any possible query sets Q_1 and Q_2.

LEMMA 2.6. *Any one-sided error isomorphism tester, after completing its queries Q_1, Q_2, must always accept G_1 and G_2 if the corresponding knowledge charts I_{G_1,Q_1} and I_{G_2,Q_2} on which the decision is based are knowledge-packable. In particular, if for some G_1, G_2 and $0 \leq q \leq \binom{n}{2}$, any $I_{G_1,Q_1} \in I_{G_1,q}$ and $I_{G_2,Q_2} \in I_{G_2,q}$ are knowledge-packable, then every one-sided error isomorphism tester which is allowed to ask at most q queries must always accept G_1 and G_2.*

Proof. This is true, since if the knowledge charts I_{G_1,Q_1} and I_{G_2,Q_2} are packable, it means that there is an extension G_1' of G_1's restriction to Q_1 to a graph that is isomorphic to G_2. In other words, given G_1' and G_2 as inputs, there is a positive probability that the isomorphism tester obtained $I_{G_1',Q_1} = I_{G_1,Q_1}$ and I_{G_2,Q_2} after completing its queries, and hence, a one-sided error tester must always accept in this case. □

Proving lower bounds for the two-sided error testers involves Yao's method [17], which for our context informally says that if there is a small enough statistical distance between the distributions of q query results, from two distributions over inputs that satisfy the property and inputs that are far from satisfying the property, then there is no tester for that property which makes at most q queries. We start with definitions that are adapted to property testing lower bounds.

DEFINITION 2.7 (restriction, variation distance). *For a distribution D over inputs, where each input is a function $f : \mathcal{D} \to \{0, 1\}$, and for a subset $\mathcal{Q}$ of the domain $\mathcal{D}$, we define the* restriction *$D|_{\mathcal{Q}}$ of D to $\mathcal{Q}$ to be the distribution over functions of the type $g : \mathcal{Q} \to \{0, 1\}$ that results from choosing a random function $f : \mathcal{D} \to \{0, 1\}$ according to the distribution D and then setting g to be $f|_{\mathcal{Q}}$, the restriction of f to $\mathcal{Q}$.*

Given two distributions D_1 and D_2 of binary functions from $\mathcal{Q}$, we define the variation distance *between D_1 and D_2 as follows: $d(D_1, D_2) = \frac{1}{2} \sum_{g:\mathcal{Q}\to\{0,1\}} |\Pr_{D_1}[g] -$*

$\Pr_{D_2}[g]|$, *where* $\Pr_D[g]$ *denotes the probability that a random function chosen according to* D *is identical to* g.

The next lemma follows from [17] (see, e.g., [9]).

LEMMA 2.8 (see [9]). *Suppose that there exist a distribution* D_P *on inputs over* $\mathcal{D}$ *that satisfy a given property* P *and a distribution* D_N *on inputs that are* ϵ*-far from satisfying the property, and suppose further that for any* $\mathcal{Q} \subset \mathcal{D}$ *of size* q*, the variation distance between* $D_P|_{\mathcal{Q}}$ *and* $D_N|_{\mathcal{Q}}$ *is less than* $\frac{1}{3}$*. Then it is not possible for a nonadaptive algorithm making* q *(or less) queries to* ϵ*-test for* P.

An additional lemma for adaptive testers is proved implicitly in [12], and a detailed proof appears in [9]. Here we strengthen it somewhat, but exactly the same proof still works in our case, too.

LEMMA 2.9 (Fischer, Newman, and Sgall [12]; see [9]). *Suppose that there exists a distribution* D_P *on inputs over* $\mathcal{D}$ *that satisfy a given property* P *and a distribution* D_N *on inputs that are* ϵ*-far from satisfying the property. Suppose further that for any* $\mathcal{Q} \subset \mathcal{D}$ *of size* q *and any* $g : \mathcal{Q} \to \{0,1\}$*, we have* $\Pr_{D_P|_{\mathcal{Q}}}[g] < \frac{3}{2}\Pr_{D_N|_{\mathcal{Q}}}[g]$*. Then it is not possible for any algorithm making* q *(or less) queries to* ϵ*-test for* P*. The conclusion also holds if instead of the above, for any* $\mathcal{Q} \subset \mathcal{D}$ *of size* q *and any* $g : \mathcal{Q} \to \{0,1\}$*, we have* $\Pr_{D_N|_{\mathcal{Q}}}[g] < \frac{3}{2}\Pr_{D_P|_{\mathcal{Q}}}[g]$.

Often, given two isomorphic graphs G, H on n vertices, we want to estimate how many vertices from both graphs need to be randomly chosen in order to get an intersection set of size k with high probability.

LEMMA 2.10. *Given two graphs* G, H *on* n *vertices, a bijection* σ *of their vertices, and two uniformly random subsets* $C_G \subset V(G)$*,* $C_H \subset V(H)$*, the following holds: For any* $0 < \alpha < 1$ *and any positive integers* c, k*, if* $|C_G| = kn^{\alpha}\log^c n$ *and* $|C_H| = n^{1-\alpha}\log^c n$*, then with probability* $1 - o(2^{-\log^c n})$ *the size of* $C_G \cap \sigma(C_H)$ *is greater than* k.

Proof sketch. By the linearity of expectation, the expected size of the intersection set is $\frac{|C_G||C_H|}{n} = k\log^{2c} n$. Using large deviation inequalities, $C_G \cap \sigma(C_H) > k$ with probability $1 - o(2^{-\log^c n})$. □

3. One-sided testers. By Lemma 2.6, one-sided testers for isomorphism look at some query set Q of the input and accept if and only if the restriction of the input to Q is extensible to some input satisfying the property. The main idea is to prove that if the input is far from satisfying the property, then with high probability its restriction Q will provide the evidence for it. To prove lower bounds for one-sided testers, it is sufficient to find an input that is ϵ-far from satisfying the property, but for which the restriction of the input to any possible set Q is extensible to some alternative input that satisfies the property. In this section we prove the following.

THEOREM 3.1. *The query complexity of the best one-sided isomorphism tester is* $\widetilde{\Theta}(n^{3/2})$ *(up to coefficients depending only on the distance parameter* ϵ*) if both graphs are unknown, and it is* $\widetilde{\Theta}(n)$ *if one of the graphs is known in advance.*

We first prove Theorem 3.1 for the case where both graphs are unknown and then move to the proof of the simpler second case where one of the graphs is known in advance.

3.1. One-sided testing of two unknown graphs.

The upper bound.

ALGORITHM 1.

1. For both graphs G_1, G_2 construct the query sets Q_1, Q_2 respectively, by choosing every possible query with probability $\sqrt{\frac{\ln n}{\epsilon n}}$, independently of other queries.

2. If $|Q_1|$ or $|Q_2|$ is larger than $1000n^{3/2}\sqrt{\frac{\ln n}{\epsilon}}$, accept without making the queries. Otherwise make the chosen queries.
3. If there is a knowledge-packing of I_{G_1,Q_1} and I_{G_2,Q_2}, accept. Otherwise reject.

Clearly, the query complexity of Algorithm 1 is $O(n^{3/2}\sqrt{\log n})$ for every fixed ϵ.

LEMMA 3.2. *Algorithm* 1 *accepts with probability* 1 *if* G_1 *and* G_2 *are isomorphic, and if* G_1 *and* G_2 *are* ϵ*-far from being isomorphic, Algorithm* 1 *rejects with probability* $1 - o(1)$.

Proof. Assume first that G_1 and G_2 are isomorphic, and let π be an isomorphism between them. Obviously π is also a knowledge-packing for any pair of knowledge charts of G_1 and G_2. Hence, if the algorithm did not accept in the second stage, then it will accept in the third stage.

Now we turn to the case where G_1 and G_2 are ϵ-far from being isomorphic. Due to large deviation inequalities, the probability that Algorithm 1 terminates in step 2 is $o(1)$, and therefore we can assume in the proof that it reaches step 3 without harming the correctness. Since G_1 and G_2 are ϵ-far from being isomorphic, every possible bijection π of their vertices has a set E_π of at least ϵn^2 pairs of G_1's vertices such that for every $\{u, v\} \in E_\pi$, either $\{u, v\}$ is an edge in G_1 or $\{\pi(u), \pi(v)\}$ is an edge in G_2, but not both. Now we fix π and let $\{u, v\} \in E_\pi$ be one such pair. The probability that $\{u, v\}$ was not queried in G_1 or $\{\pi(u), \pi(v)\}$ was not queried in G_2 is $1 - \frac{\ln n}{\epsilon n}$. Since the queries were chosen independently, the probability that for all $\{u, v\} \in E_\pi$ either $\{u, v\}$ was not queried in G_1 or $\{\pi(u), \pi(v)\}$ was not queried in G_2 is at most $(1 - \frac{\ln n}{\epsilon n})^{\epsilon n^2}$. Using the union bound, we bound the probability of not revealing at least one such pair in both graphs for all possible bijections by $n!(1 - \frac{\ln n}{\epsilon n})^{\epsilon n^2}$. This bound satisfies

$$n! \left(1 - \frac{\ln n}{\epsilon n}\right)^{\epsilon n^2} \leq n! \left(e^{-\frac{\ln n}{\epsilon n}}\right)^{\epsilon n^2} = n! \frac{1}{n^n} = o(1);$$

thus the algorithm rejects graphs that are ϵ-far from being isomorphic with probability $1 - o(1)$. ∎

The lower bound. Here we construct a pair G, H of 1/100-far graphs on n vertices such that every knowledge chart from $I_{G,n^{3/2}/200}$ can be packed with every knowledge chart from $I_{H,n^{3/2}/200}$, and hence by Lemma 2.6, any one-sided algorithm which is allowed to use at most $n^{3/2}/200$ queries must always accept G and H. Note that this holds for nonadaptive as well as adaptive algorithms, since we actually prove that there is no certificate of size $n^{3/2}/200$ for the nonisomorphism of these graphs.

LEMMA 3.3. *For every large enough* n*, there are two graphs* G *and* H *on* n *vertices such that*

1. G *is* 1/100*-far from being isomorphic to* H*;*
2. *every knowledge chart from* $I_{G,n^{3/2}/200}$ *can be knowledge-packed with any knowledge chart from* $I_{H,n^{3/2}/200}$

Proof. We set both G and H to be the union of a complete bipartite graph with a set of isolated vertices. Formally, G has three vertex sets L, R_f, R_e, where $|L| = n/2$, $|R_f| = 26n/100$ and $|R_e| = 24n/100$, and it has the following edges: $\{\{u, v\} : u \in L \wedge v \in R_f\}$. H has the same structure, but with $|R_f| = 24n/100$ and $|R_e| = 26n/100$, as illustrated in Figure 3.1. Clearly, just by the difference in the edge count, G is 1/100-far from being isomorphic to H, so G and H satisfy the first part of Lemma 3.3.

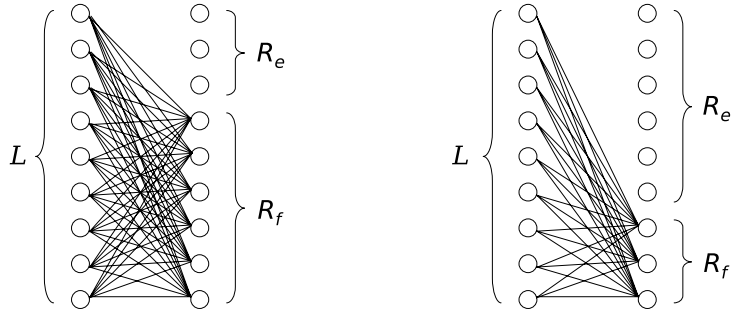


FIG. 3.1. *The graphs G and H (with the difference between them exaggerated).*

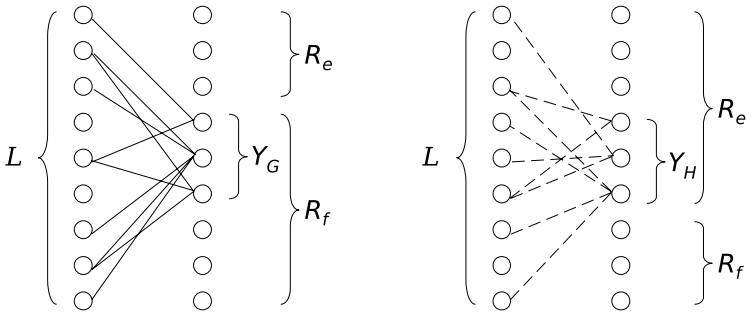


FIG. 3.2. *Finding Y_G and Y_H.*

To prove that the second condition of Lemma 3.3 holds, we will show that for all possible query sets Q_G, Q_H of size $n^{3/2}/200$ there exist sets $Y_G \in R_f(G)$ and $Y_H \in R_e(H)$ that satisfy the following:

- $|Y_G| = |Y_H| = n/50$.
- The knowledge charts I_{G,Q_G} and I_{H,Q_H} restricted to $L(G) \cup Y_G$ and $L(H) \cup Y_H$ can be packed in a way that pairs vertices from $L(G)$ with vertices from $L(H)$.

In Figure 3.2 we illustrate these restricted knowledge charts, where solid lines are known (queried) edges and dashed lines are known (queried) "nonedges." The existence of such Y_G and Y_H implies the desired knowledge-packing, since we can complete the partial packing from the second item by arbitrarily pairing vertices from $R_f(G) \setminus Y_G$ with vertices from $R_f(H)$ and pairing vertices from $R_e(G)$ with vertices from $R_e(H) \setminus Y_H$.

Remark 3.4. Note that there is a trivial algorithm that distinguishes between the two graphs in $O(n)$ queries by sampling vertices and checking their degrees. However, such an algorithm has two-sided error. Any one-sided error algorithm must find evidence of the nonisomorphism of the graphs, i.e., two knowledge charts that cannot be packed (in the sense that there is no isomorphism consistent with them).

Proving the existence of Y_G and Y_H. For every vertex $v \in V(G)$, we define its query degree as

$$d_Q(v) = |\{\{v, u\} : u \in V(G) \wedge \{v, u\} \in Q_G\}|.$$

We also denote by $N_Q(v)$ the set $\{u : \{v, u\} \in E(G) \cap Q_G\}$ and denote by $\overline{N}_Q(v)$ the set $\{u : \{v, u\} \in Q_G \setminus E(G)\}$. In other words, $N_Q(v)$ is the set of known neighbors of

v, $\overline{N}_Q(v)$ is the set of known nonneighbors of v, and $d_Q(v) = |\overline{N}_Q(v)| + |N_Q(v)|$. We define $d_Q(v)$, $N_Q(v)$, and $\overline{N}_Q(v)$ for H's vertices similarly.

Since $|Q_G|, |Q_H| \leq n^{3/2}/200$, there must be two sets of vertices $D_G \in R_f(G)$ and $D_H \in R_e(H)$, both of size $n/10$, such that $\forall_{v \in D_G} : d_Q(v) \leq n^{1/2}/2$ and $\forall_{v \in D_H} : d_Q(v) \leq n^{1/2}/2$.

Now we prove the existence of Y_G and Y_H (as defined above) using a simple probabilistic argument. First we set an arbitrary pairing

$$B_D = \{\{v_G^1, u_H^1\}, \{v_G^2, u_H^2\}, \ldots, \{v_G^{n/10}, u_H^{n/10}\}\}$$

of D_G's and D_H's elements. Then we choose a bijection $B_L : L(G) \to L(H)$ uniformly at random and show that with some positive probability, there are at least $n/50$ consistent (packable) pairs in B_D. Formally, we define

$$Y = \left\{\{v_G, u_H\} \in B_D : B_L(N_Q(v_G)) \cap \overline{N}_Q(u_H) = \emptyset\right\}$$

as the set of consistent pairs and show that $\Pr[|Y| \geq n/50] > 0$.

For a specific pair $\{v \in D_G, u \in D_H\}$, we have

$$\begin{aligned}\Pr_{B_L}[B_L(N_Q(v)) \cap \overline{N}_Q(u) = \emptyset] &\geq \prod_{i=0}^{n^{1/2}/2-1} \left(1 - \frac{n^{1/2}/2}{n/2 - i}\right) \\ &\geq \left(1 - \frac{2n^{1/2}}{n}\right)^{n^{1/2}/2} \geq (e + 0.001)^{-1} \geq 1/3,\end{aligned}$$

and by the linearity of expectation, $\mathrm{E}[|Y|] \geq |D_G|/3 > n/50$. Therefore, there is at least one bijection B_L for which the size of Y is no less than its expectation. We can now set

$$Y_G = \{u : \exists v \in V(H) \text{ such that } \{u, v\} \in Y\}$$

and

$$Y_H = \{v : \exists u \in V(G) \text{ such that } \{u, v\} \in Y\},$$

concluding the proof. □

3.2. One-sided testing where one of the graphs is known in advance. The algorithm for testing isomorphism between an unknown graph and a graph that is known in advance is similar to Algorithm 1. In this case the algorithm makes a quasi-linear number of queries, to accept with probability 1 if the graphs are isomorphic and reject with probability $1 - o(1)$ if they are ϵ-far from being isomorphic. We also prove an almost matching nearly trivial lower bound for this problem.

The upper bound. Denote by G_k and G_u the known and the unknown graphs, respectively.

ALGORITHM 2.

1. Construct a query set Q by choosing every possible query from G_u with probability $\frac{\ln n}{\epsilon n}$, independently at random.
2. If $|Q|$ is larger than $\frac{10n \ln n}{\epsilon}$, accept without making the queries. Otherwise make the chosen queries.

3. If there is a knowledge-packing of $I_{G_u,Q}$ and $I_{G_k,[V(G_k)]^2}$, accept. Otherwise reject.

Clearly the query complexity of Algorithm 2 is $O(n \log n)$, and it rejects in step 2 with probability $o(1)$.

LEMMA 3.5. *Algorithm* 2 *always accepts isomorphic graphs, and it rejects* ϵ*-far graphs with probability* $1 - o(1)$.

Proof. The proof is almost identical to that of Lemma 3.2. It is clear that isomorphic graphs are always accepted by Algorithm 2. Now we assume that the graphs G_k and G_u are ϵ-far and that the algorithm reached step 3 (as it stops at step 2 with probability $o(1)$). Given a bijection π, the probability that no violating pair $\{u, v\} \in E_\pi$ was queried is at most $(1 - \frac{\ln n}{\epsilon n})^{\epsilon n^2} \leq e^{-n \ln n} = n^{-n}$. Applying the union bound over all $n!$ possible bijections, the acceptance probability is bounded by $n!/n^n = o(1)$. ∎

The lower bound. As before, to give a lower bound on one-sided error algorithms it is sufficient to show that for some G_k and G_u that are far, no "proof" of their nonisomorphism can be provided with $\Omega(n)$ queries. First we formulate the second part of Lemma 2.6 for the special case where one of the graphs is known in advance.

LEMMA 3.6. *If for some* G_k, G_u*, where* G_k *is known in advance, and some fixed* $0 \leq q \leq \binom{n}{2}$*,* $I_{G_k,[v(G_k)]^2}$ *is knowledge-packable with every* $I_{G_u,Q} \in I_{G_u,q}$*, then every one-sided error isomorphism tester which is allowed to ask at most* q *queries must always accept* G_k *and* G_u*.*

We set G_k to be a disjoint union of $K_{n/2}$ and $n/2$ isolated vertices and set G_u to be a completely edgeless graph.

OBSERVATION 3.7. G_k *and* G_u *are* 1/4*-far, and every* $I_{G_u,Q} \in I_{G_u,\frac{n}{4}}$ *is knowledge-packable with* $I_{G_k,[V(G_k)]^2}$*.*

Proof. Clearly, just by the difference in the edge count, G_k is 1/4 far from being isomorphic to G_u. But since $n/4$ queries cannot involve more than $n/2$ vertices from G_u (all isolated), and G_k has $n/2$ isolated vertices, the knowledge charts are packable. ∎

Together with Lemma 3.6, we get the desired lower bound. This concludes the proof of the last part of Theorem 3.1.

4. Two-sided testers. In the context of graph properties, two-sided error testers are usually not known to achieve significantly lower query complexity than the one-sided error testers, apart from the properties that explicitly involve counting, such as *Max-Cut* and *Max-Clique* [13]. However, in our case two-sided error isomorphism testers have substantially lower query complexity than their one-sided error counterparts.

4.1. Two-sided testing where one of the graphs is known in advance.

THEOREM 4.1. *The query complexity of two-sided error isomorphism testers is* $\widetilde{\Theta}(\sqrt{n})$ *if one of the graphs is known in advance and the other needs to be queried.*

We prove the lower bound first. This way it will be easier to understand why certain stages of the upper bound testing algorithm are necessary.

The lower bound.

LEMMA 4.2. *Any isomorphism tester that makes at most* $\frac{\sqrt{n}}{4}$ *queries to* G_u *cannot distinguish between the case that* G_k *and* G_u *are isomorphic and the case that they are* 1/32*-far from being isomorphic, where* G_k *is known in advance.*

We begin with a few definitions.

DEFINITION 4.3. *Given a graph G and a set W of $\frac{n}{2}$ vertices of G, we define the* clone $G^{(W)}$ *of G in the following way:*

- *The vertex set of $G^{(W)}$ is defined as $V(G^{(W)}) = W \cup \{w' : w \in W\}$*
- *The edge set of $G^{(W)}$ is defined as*

$$\begin{aligned} E(G^{(W)}) = &\Big\{\{v,u\} : \{v,u\} \in E(G)\Big\} \\ \cup &\Big\{\{v',u\} : \{v,u\} \in E(G)\Big\} \\ \cup &\Big\{\{v',u'\} : \{v,u\} \in E(G)\Big\}. \end{aligned}$$

In other words, $G^{(W)}$ is the product of the subgraph of G induced on W with the graph K_2.

For the two copies $v, v' \in V(G^{(W)})$ of $v \in W$, we say that v is the source *of both v and v'.*

LEMMA 4.4. *Let $G \sim G(n, 1/2)$ be a random graph. With probability $1 - o(1)$ the graph G is such that for every subset $W \subset V(G)$ of size $n/2$, the clone $G^{(W)}$ of G is $1/32$-far from being isomorphic to G.*

Proof. Let G be a random graph according to $G(n, 1/2)$, and let $W \subset V(G)$ be an arbitrary subset of G's vertices of size $n/2$. First we show that for an arbitrary bijection $\sigma : V(G^{(W)}) \to V(G)$ the graphs $G^{(W)}$ and G are $1/32$-close under σ with probability at most $2^{-\Omega(n^2)}$, and then we apply the union bound on all bijections and every possible subset W.

We split the bijection $\sigma : V(G^{(W)}) \to V(G)$ into two injections $\sigma_1 : W \to V(G)$ and $\sigma_2 : V(G^{(W)}) \setminus W \to V(G) \setminus \sigma_1(W)$. Note that either $|W \setminus \sigma_1(W)| \geq n/4$ or $|W \setminus \sigma_2(W)| \geq n/4$. Assume without loss of generality that the first case holds, and let U denote the set $W \setminus \sigma_1(W)$. Since every edge in G is chosen at random with probability $1/2$, the probability that for some pair $u, v \in U$ either $\{u, v\}$ is an edge in G and $\{\sigma(u), \sigma(v)\}$ is not an edge in G or $\{u, v\}$ is not an edge in G and $\{\sigma(u), \sigma(v)\}$ is an edge in G is exactly $1/2$. Therefore, using large deviation inequalities, the probability that in the set U there are less than $\binom{n}{2}/32$ such pairs is at most $2^{-\Omega(n^2)}$ (as these events are all independent). There are at most $n!$ possible bijections and $\binom{n}{n/2}$ possible choices for W, so using the union bound, the probability that for some W the graph $G \sim G(n, 1/2)$ is not $1/32$-far from being isomorphic to $G^{(W)}$ is at most $2^{-\Omega(n^2)}\binom{n}{n/2}n! = o(1)$. ∎

Given a graph G satisfying the assertion of Lemma 4.4, we set $G_k = G$ and define two distributions over graphs, from which we choose the unknown graph G_u:

- D_P: A permutation of G_k, chosen uniformly at random.
- D_N: A permutation of $G_k^{(W)}$, where both W and the permutation are chosen uniformly at random.

According to Lemmas 4.4 and 2.9, it is sufficient to show that the distributions D_P and D_N restricted to a set of $\sqrt{n}/4$ queries are close. In particular, we intend to show that for any $\mathcal{Q} \subset \mathcal{D} = V^2$ of size $\sqrt{n}/4$ and any $Q : \mathcal{Q} \to \{0, 1\}$, we have $\Pr_{D_P|_{\mathcal{Q}}}[Q] < \frac{3}{2}\Pr_{D_N|_{\mathcal{Q}}}[Q]$. This will imply a lower bound for adaptive (as well as nonadaptive) testing algorithms.

OBSERVATION 4.5. *For a set U of $G^{(W)}$'s vertices, define the event E_U as the event that there is no pair of copies w, w' of any one of G's vertices in U. For a given set Q of pairs of vertices, let U_Q be the set of all vertices that belong to some pair in*

Q. Then the distribution $D_N|_{\mathcal{Q}}$ conditioned on the event E_{U_Q} (defined above) and the unconditioned distribution $D_P|_{\mathcal{Q}}$ are identical.

Proof. In D_N, if no two copies of any vertex were involved in the queries, then the source vertices of the queries to G_u are in fact a uniformly random sequence (with no repetition) of the vertices of G_k, and this (together with G_k) completely determines the distribution of the answers to the queries. This is the same as the unconditioned distribution induced by D_P. □

Intuitively, the next lemma states that picking two copies of the same vertex in a randomly permuted $G^{(W)}$ requires many samples, as per the well-known birthday problem.

LEMMA 4.6. *For a fixed set Q of at most $\sqrt{n}/4$ queries and the corresponding set $U = U_Q$ of vertices, the probability that the event E_U did not happen is at most $1/4$.*

Proof. The bound on $|Q|$ implies that $|U| \leq \sqrt{n}/2$. Now we examine the vertices in U as if we add them one by one. The probability that a vertex v that is added to U is a copy (with respect to the original graph G) of some vertex u that was already inserted into U (or vice versa) is at most $\frac{\sqrt{n}}{2n}$. Hence, the probability that eventually (after $\sqrt{n}/2$ insertions) we have two copies of the same vertex in U is at most $\frac{\sqrt{n}}{2n} \cdot \sqrt{n}/2 = 1/4$. □

From Observation 4.5, the distribution $D_N|_{\mathcal{Q}}$ conditioned on the event E_U and the unconditioned distribution $D_P|_{\mathcal{Q}}$ are identical. By Lemma 4.6 it follows that $\Pr[E_U] > 2/3$. Therefore, for any $g : \mathcal{Q} \to \{0,1\}$ we have

$$\begin{aligned}\Pr_{D_N|_{\mathcal{Q}}}[g] &= \Pr[E_U] \cdot \Pr_{D_P|_{\mathcal{Q}}}[g] + (1 - \Pr[E_U]) \cdot \Pr_{D_N|_{\mathcal{Q}}}[g] \\ &\geq \Pr[E_U] \cdot \Pr_{D_P|_{\mathcal{Q}}}[g] > \frac{2}{3} \cdot \Pr_{D_P|_{\mathcal{Q}}}[g]\end{aligned}$$

or, equivalently,

$$\Pr_{D_P|_{\mathcal{Q}}}[g] < \frac{3}{2}\Pr_{D_N|_{\mathcal{Q}}}[g];$$

hence the distributions D_P and D_N satisfy the conditions of Lemma 2.9. The following corollary completes the proof of Lemma 4.2.

COROLLARY 4.7. *It is not possible for any algorithm (adaptive or not) making $\sqrt{n}/4$ (or less) queries to test for isomorphism between a known graph and a graph that needs to be queried.*

The upper bound. We start with a few definitions. Given a graph G and a subset C of $V(G)$, we define the *C-labeling* of G's vertices as follows: Every vertex $v \in V(G)$ gets a label according to the set of its neighbors in C. Note that there are $2^{|C|}$ possible labels for a set C, but even if $2^{|C|} > n$, still at most n of the labels occur, since there are only n vertices in the graph. On the other hand, it is possible that several vertices will have the same label according to C. Such a labeling implies the following distribution over the vertices of G.

DEFINITION 4.8. *Given a graph G and a C-labeling of its vertices (according to some $C \subset V(G)$), we denote by D_C the distribution over the actual labels of the C-labeling (at most n labels), in which the probability of a certain label γ is calculated from the number of vertices from $V(G)$ having the label γ under the C-labeling, divided by n.*

Given a graph G on n vertices and a graph C on $k < n$ vertices, we say that a one-to-one function $\eta : V(C) \to V(G)$ is an *embedding* of C in G. We also call

$\eta(V(C))$ the *placement* of C in G. With a slight abuse of notation, from now on by a placement $\eta(V(C))$ we mean also the correspondence given by η and not just the set.

Given graphs G, H on n vertices, a subset C_G of $V(G)$ and a placement C_H of C_G in H under an embedding η, we define the *distance* between the C_G-labeling of G and the C_H-labeling of H as

$$\frac{1}{2}\sum_{\gamma\in 2^{C_G}} \big|\,|\{u\in V(G): N(u)\cap C_G=\gamma\}| - |\{v\in V(H): N(v)\cap \eta(C_G)=\gamma\}|\,\big|.$$

This distance measure is equal to the usual variation distance between D_{C_G} and D_{C_H} multiplied by n. We are now ready to prove the upper bound.

LEMMA 4.9. *Given an input graph G_u and a known graph G_k (both of order n), there is a property tester A_{ku} that accepts with probability at least $2/3$ if G_u is isomorphic to G_k and rejects with probability at least $2/3$ if G_u is ϵ-far from G_k. Furthermore, A_{ku} makes $\widetilde{O}(\sqrt{n})$ queries to G_u.*

We first outline the algorithm: The test is performed in two main phases. In Phase 1 we randomly choose a small subset C_u of G_u's vertices, and try all possible placements of C_u in the known graph G_k. The placements that imply a large distance between the labeling of G_u and G_k are discarded. After filtering the good placements of C_u in G_k, we move to Phase 2. In Phase 2 every one of the good placements is tested separately, by defining a random bijection $\pi : V(G_u) \to V(G_k)$ and testing whether π is close to being an isomorphism. Finally, if one of the placements passed both Phase 1 and Phase 2, the graphs are accepted. Otherwise they are rejected.

Phase 1. In the first phase we choose at random a core set C_u of $\log^2 n$ vertices from G_u (the unknown graph). For every embedding η of C_u in G_k and the corresponding placement $C_k \in G_k$, we examine the distributions D_{C_u} and D_{C_k} as in Definition 4.8. Since the graph G_k is known in advance, we know exactly which are the actual labels according to C_k (in total no more than n labels), so from now on we will consider the restriction of both distributions to these actual labels only. Next we test for every embedding of C_u whether D_{C_u} is statistically close to D_{C_k}. Note that the distribution D_{C_k} is explicitly given, and the distribution D_{C_u} can be sampled by choosing a vertex v from $V(G_u)$ uniformly at random and making all queries $\{v\} \times C_u$. If the label of some $v \in V(G_u)$ does not exist in the C_k-labeling of G_k, we immediately reject this placement and move to the next one. Now we use the following lemma from [7], which states that $\widetilde{O}(\sqrt{n})$ samples are sufficient for testing if the sampled distribution is close to the explicitly given distribution.

LEMMA 4.10. *There is an algorithm that given two distributions D_K, D_U over n elements and a distance parameter ϵ, where D_K is given explicitly and D_U is given as a black box that allows sampling according to the distribution, satisfies the following: If the distributions D_K and D_U are identical, then the algorithm accepts with probability at least $1 - 2^{-\log^7 n}$, and if the variation distance between D_K and D_U is larger than $\epsilon/10$, then the algorithm accepts with probability at most $2^{-\log^7 n}$. For a fixed ϵ, the algorithm uses $\widetilde{O}(\sqrt{n})$ many samples.*

Actually, this is an amplified version of the lemma from [7], which can be achieved by independently repeating the algorithm provided there polylog(n) many times and taking the majority vote. This amplification allows us to reuse the same $\widetilde{O}(\sqrt{n})$ samples for all possible placements of the core set. As a conclusion of Phase 1, the algorithm rejects the placements of C_u that imply a large variation distance between the above distributions and passes all other placements of C_u to Phase 2. Naturally,

if Phase 1 rejects all placements of C_k due to distribution test failures or due to the existence of labels in G_u that do not exist in G_k, then G_u is rejected without moving to Phase 2 at all. First we observe the following.

OBSERVATION 4.11. *With probability* $1 - o(1)$*, all of the placements that passed Phase* 1 *imply* $\epsilon/10$*-close distributions, and all placements that imply identical distributions passed Phase* 1*. In other words, the distribution test did not err on any of the placements.*

Proof. There are at most $2^{\log^3 n}$ possible placements of C_u. Using the union bound with Lemma 4.10, we conclude that Phase 1 will not err with probability $1-o(1)$. ∎

Phase 2. Following Observation 4.11, we need to design a test such that given a placement C_k of C_u in G_k that implies close distributions, the test satisfies the following conditions:

1. If the graphs are isomorphic and the embedding of C_u is expandable to some isomorphism, then the test accepts with probability at least 3/4.
2. If the graphs G_u and G_k are ϵ-far, then the test accepts with probability at most $o(2^{-\log^3 n})$.

If our test in Phase 2 satisfies these conditions, then we get the desired isomorphism tester. From now on, when we refer to some placement of C_u we assume that it has passed Phase 1 and hence implies close distributions.

In Phase 2 we choose a set W_u of $\log^4 n$ vertices from $V(G_u)$ and retrieve their labels according to C_u by making the queries $W_u \times C_u$. Additionally, we split W_u into $\frac{1}{2}\log^4 n$ pairs $\{\{u_1, v_1\}, \ldots, \{u_{\frac{1}{2}\log^4 n}, v_{\frac{1}{2}\log^4 n}\}\}$ randomly and make all $\frac{1}{2}\log^4 n$ queries according to these pairs. This is done once, and the same set W_u is used for all the placements of C_u that are tested in Phase 2. Then, for every placement C_k of C_u, we would like to define a random bijection $\pi_{C_u,C_k} : V(G_u) \to V(G_k)$ as follows. For every label γ, the bijection π_{C_u,C_k} pairs the vertices of G_u having label γ with the vertices of G_k having label γ uniformly at random. There might be labels for which one of the graphs has more vertices than the other. We call these remaining vertices *leftovers*. Note that the amount of leftovers from each graph is equal to the distance between the C_k-labeling and the C_u-labeling. Finally, after π_{C_u,C_k} pairs all matching vertices, the leftover vertices are paired arbitrarily. In practice, since we do not know the labels of G_u's vertices, we instead define a partial bijection $\widetilde{\pi}_{C_u,C_k}(W_u) \to V(G_k)$ as follows. Every vertex $v \in W_u$ that has the label γ_v is paired uniformly at random with one of the vertices of G_k which has the same label γ_v and was not paired yet. If this is impossible, we reject the current placement of C_u and move to the next one.

Denote by δ_{C_u,C_k} the fraction of the queried pairs from W_u for which exactly one of $\{u_i, v_i\}$ and $\{\widetilde{\pi}_{C_u,C_k}(u_i), \widetilde{\pi}_{C_u,C_k}(v_i)\}$ is an edge. If $\delta_{C_u,C_k} \le \epsilon/2$, then G_u is accepted. Otherwise we move to the next placement of C_u. If none of the placements was accepted, G_u is rejected.

Correctness. A crucial observation in our proof is that with high probability, any two vertices that have many distinct neighbors in the whole graph will also have distinct neighbors within a "large enough" random core set.

Formally, given a graph G and a subset C of its vertices, we say that C is β-*separating* if for every pair of vertices $u, v \in V(G)$ such that $d_{uv} \triangleq \frac{1}{n}|\{N(u) \triangle N(v)\}| \ge \beta$ the vertices u and v have different labels under the C-labeling of G.

CLAIM 4.12. *Let* $\beta > 0$ *be fixed, let* G *be a graph of order* n*, and let* $C \subset V(G)$ *be a uniformly chosen random subset of size* $\log^2 n$*. Then* C *is* β*-separating with probability* $1 - o(1)$*.*

Proof. Fix a pair $u, v \in V(G)$. If u, v are such that $d_{uv} > \beta$, then the probability that they share exactly the same neighbors in C is bounded by $(1-\beta)^{\log^2 n} \leq e^{-\beta \log^2 n} = n^{-\beta \log n}$. Using the union bound, with probability $1 - o(1)$ every pair u, v of vertices with $d_{uv} > \beta$ will not have exactly the same neighbors in C; i.e., the vertices will have different labels under the C-labeling. □

LEMMA 4.13 (completeness). *Conditioned over the event that C_u is $\epsilon/8$-separating, if the graphs G_u and G_k are isomorphic and the placement C_k of C_u is expandable to some isomorphism, then $\Pr[\delta_{C_u,C_k} \leq \epsilon/2] = 1 - o(1)$, and hence C_k is accepted in Phase 2 with probability $1 - o(1)$.*

Proof. Let $\phi : V(G_u) \to V(G_k)$ be an isomorphism to which the placement of C_u is expandable. By definition, for every pair v_1, v_2 of G_u's vertices, $\{v_1, v_2\}$ is an edge in G_u if and only if $\{\phi(v_1), \phi(v_2)\}$ is an edge in G_k. In addition, for every vertex $v \in V(G_u)$, the vertices v and $\phi(v)$ have exactly the same labels. Let σ be the permutation such that π_{C_u,C_k} is the composition of σ and the isomorphism ϕ. In the rest of this proof, by distance we mean the absolute distance between two labeled graphs (which is between 0 and $\binom{n}{2}$).

First we show that the distance from $\sigma(G_u)$ to G_k is almost the same as the distance from $\phi(G_u)$ to G_k (which is zero since ϕ is an isomorphism), and then we apply large deviation inequalities to conclude that $\Pr[\delta_{C_u,C_k} \leq \epsilon/2] = 1 - o(1)$.

To prove that the distance from $\sigma(G_u)$ to G_k is close to zero we show a transformation of ϕ into π_{C_u,C_k} by performing "swaps" between vertices that have the same label. Namely, we define a sequence of permutations ϕ_i, starting from $\phi_0 = \phi$ and ending with $\phi_t = \pi_{C_u,C_k}$. In each step, if there is some vertex v_0 such that $\phi_i(v_0) = u_1$ while $\pi_{C_u,C_k}(v_0) = u_0$, then we find a vertex v_1 such that $\phi_i(v_1) = u_0$, and set $\phi_{i+1}(v_0) = u_0$ and $\phi_{i+1}(v_1) = u_1$. The rest of the vertices are mapped by ϕ_{i+1} as they were mapped by ϕ_i.

Since in each step we swap only between vertices with the same label, and since the core set C_u is $\epsilon/8$-separating, every such swap can increase the distance by at most $\epsilon n/8$, so eventually the distance between $\sigma(G_u)$ and G_k is at most $\epsilon n^2/8$. Therefore, by large deviation inequalities, δ_{C_u,C_k} as defined in Phase 2 is at most $\epsilon/2$ with probability $1 - o(1)$, and so the placement C_k is accepted. □

We now turn to the case where G_u and G_k are ϵ-far. Note that until now we did not use the fact that C_u and C_k imply close distributions. To understand why this closeness is important, recall the pairs of graphs from the lower bound proof. If we give up the distribution test in Phase 1, then these graphs will be accepted with high probability, since the algorithm cannot reveal two copies of the same vertex when sampling $o(\sqrt{n})$ vertices (recall that $|W_u| = O(\log^4 n)$). Intuitively, the problem is that in these pairs of graphs, the partial random bijection $\widetilde{\pi}_{C_u,C_k}$ will not simulate a restriction of the random bijection π_{C_u,C_k} to a set of $\log^4 n$ vertices. In the lower bound example, $\widetilde{\pi}_{C_u,C_k}$ will have no leftovers with high probability, even though π_{C_u,C_k} will always have $\Omega(n)$ leftovers. The reason is that in the cloned graph G_u, for each of about half of the labels from C_k there are two times more vertices, while for the second half there are no vertices at all. The distribution test in Phase 1 actually checks whether the clustering of the vertices according to the labels is into subsets of almost equal sizes in both G_u and G_k. If it is so, then the partial random bijection $\widetilde{\pi}_{C_u,C_k}$ is indeed similar to the restriction of a bijection π_{C_u,C_k} to a set of $\log^4 n$ vertices.

LEMMA 4.14 (soundness). *If the graphs G_u and G_k are ϵ-far, and the placement C_k implies $\epsilon/10$-close distributions, then $\Pr[\delta_{C_u,C_k} \leq \epsilon/2] \leq o(2^{-\log^3 n})$, and hence*

C_k is accepted in Phase 2 with probability at most $o(2^{-\log^3 n})$.

Proof. Assume that for a fixed C_k the random bijection π_{C_u,C_k} is ϵ-far from isomorphism. We then need to show that δ_{C_u,C_k} as defined in Phase 2 is larger than $\epsilon/2$ with probability $1 - o(2^{-\log^3 n})$.

Since the variation distance between the distributions D_{C_u} and D_{C_k} is at most $\epsilon/10$, the amount of leftovers (which is exactly the distance between the C_u-labeling of G_u and the C_k-labeling of G_k) is at most $\epsilon n/10$. Therefore, even if we first remove those $\epsilon n/10$ (or less) leftovers, the fraction of pairs u, v for which exactly one of $\{u, v\}$ and $\{\widetilde{\pi}_{C_u,C_k}(u), \widetilde{\pi}_{C_u,C_k}(v)\}$ is an edge is not smaller by more than $4\epsilon/10$ from that of π_{C_u,C_k}.

Let $\widetilde{\pi}_{C_u,C_k}$ be the random partial bijection as defined above. The distribution test of Phase 1 guarantees that $\widetilde{\pi}_{C_u,C_k}$ is a random restriction of a function that is $\epsilon/10$-close to some bijection π_{C_u,C_k}. Since G_u is ϵ-far from G_k, the bijection π_{C_u,C_k} must be ϵ-far from being an isomorphism, and hence $\widetilde{\pi}_{C_u,C_k}$ must exhibit a $6\epsilon/10$-fraction of mismatching edges. Note that the acceptance probability of C_k given $\widetilde{\pi}_{C_u,C_k}$ is equal to the probability that δ_{C_u,C_k} as defined in Phase 2 is at most $\epsilon/2$. Large deviation inequalities show that this probability is at most $2^{-\Omega(\log^4 n)} = o(2^{-\log^3 n})$. □

In conclusion, if G_k and G_u are isomorphic, then the probability that C_u is not $\epsilon/8$-separating is at most $o(1)$, and for a correct (under some isomorphism) embedding of C_u in G_k, the probability that the distribution test will fail is also $o(1)$; thus in summary algorithm A_{ku} accepts with probability greater than $2/3$. In the case that G_k and G_u are ϵ-far from being isomorphic, with probability $1 - o(1)$ all placements that are passed to Phase 2 imply close label distributions. Then each such placement is rejected in Phase 2 with probability $1 - o(2^{-\log^3 n})$, and by the union bound over all possible placements the graphs are accepted with probability less than $1/3$. Algorithm A_{ku} makes $\widetilde{O}(\sqrt{n})$ queries in Phase 1 and $\widetilde{O}(n^{1/4})$ queries in Phase 2. This completes the proof of Lemma 4.9 and thus of Theorem 4.1.

4.2. Two-sided testing of two unknown graphs.

THEOREM 4.15. *The query complexity of two-sided error isomorphism testers is between $\Omega(n)$ and $\widetilde{O}(n^{5/4})$ if both graphs need to be queried.*

The upper bound.

LEMMA 4.16. *Given two unknown graphs G and H on n vertices, there is a property tester A_{uu} that accepts with probability at least $2/3$ if G is isomorphic to H, and rejects with probability at least $2/3$ if G is ϵ-far from H. Furthermore, A_{uu} makes $\widetilde{O}(n^{5/4})$ queries to G and H.*

We use here ideas similar to those used in the upper bound proof of Lemma 4.9, but with several modifications. The main difference between this case and the case where one of the graphs is known in advance is that here we cannot write all label distributions with all possible core sets in either one of the unknown graphs (because doing that would require $\Omega(n^2)$ queries). We overcome this difficulty by sampling from both graphs in a way that with high probability will make it possible to essentially simulate the test for isomorphism where one of the graphs is known in advance.

Phase 1. First we randomly pick a set U_G of $n^{1/4}\log^3(n)$ vertices from G and a set U_H of $n^{3/4}\log^3(n)$ vertices from H. Then we make all $n^{5/4}\log^3(n)$ possible queries in $U_G \times V(G)$. Note that if G and H have an isomorphism σ, then according to Lemma 2.10 with probability $1 - o(1)$ the size of $U_G \cap \sigma(U_H)$ will exceed $\log^2(n)$.

For all subsets C_G of U_G of size $\log^2 n$ we try every possible placement $C_H \subset U_H$ of C_G. There are at most $2^{\log^3 n}$ subsets C_G, and at most $2^{\log^3 n}$ possible ways to

embed each C_G in U_H. Since we made all $n^{5/4}\log^3(n)$ possible queries in $U_G \times V(G)$, for every $C_G \subset U_G$ the corresponding distribution D_{C_G} is entirely known.

So now for every possible placement of C_G in U_H we test if the variation distance between the distributions D_{C_G} and D_{C_H} is at most $\epsilon/10$. Since we know the entire distributions D_{C_G}, we need only sample the distribution D_{C_H}; therefore we can still use the amplified distribution test of Lemma 4.10. The test there requires $\widetilde{O}(\sqrt{n})$ samples, so similarly to the proof of Lemma 4.9, we take a random set S of $\widetilde{O}(\sqrt{n})$ vertices from H and make all $n^{5/4}\text{polylog}(n)$ queries in $S \times U_H$.

We reject the pairs of a set C_G and a placement C_H that were rejected by the distribution test for D_{C_G} and D_{C_H}, and pass all other pairs to Phase 2. If Phase 1 rejects all possible pairs, then the graphs G and H are rejected without moving to Phase 2. The following observation is similar to the one we used in the case where one of the graphs is known in advance.

OBSERVATION 4.17. *With probability* $1 - o(1)$*, all of the placements that passed Phase* 1 *imply* $\epsilon/10$*-close distributions, and all placements that imply identical distributions passed Phase* 1*. In other words, the distribution test did not err on any of the placements.*

Phase 2. As in Lemma 4.9, we need to design a test which, given a placement C_H of C_G in H that implies close distributions, satisfies the following conditions:

1. If the graphs are isomorphic and the embedding of C_H is expandable to some isomorphism, then the test accepts with probability at least 3/4.
2. If the graphs G and H are ϵ-far, then the test accepts with probability at most $o(2^{-2\log^3 n})$.

In Phase 2 we choose at random a set W_G of $n^{1/2}\log^{13} n$ vertices from $V(G)$, and a set W_H of $n^{1/2}\log^6 n$ vertices from $V(H)$. We retrieve the labels in W_H according to any C_H by making the queries $W_H \times U_H$. Additionally, we make all queries inside W_H and all queries inside W_G. This is done once, and the same sets W_G, W_H are used for all of the pairs C_G, C_H that are tested in Phase 2. According to Lemma 2.10, if the graphs are isomorphic under some isomorphism σ, then $|W_H \cap \sigma(W_G)| > \log^7 n$ with probability $1 - o(1)$.

Then, similarly to what is done in Lemma 4.9, for every pair C_G, C_H, we would like to define a random bijection $\pi_{C_G,C_H} : V(G) \to V(H)$ as follows. For every label γ, π_{C_G,C_H} pairs the vertices of G having label γ with the vertices of H having label γ uniformly at random. After π_{C_G,C_H} pairs all matching vertices, the leftover vertices are paired arbitrarily. Then again, since we do not know the labels of H's vertices, we define a partial bijection $\widetilde{\pi}_{C_G,C_H}(W_H) \to V(G)$ instead, in which every vertex $v \in W_H$ that has the label γ_v is paired uniformly at random with one of the vertices of G which has the same label γ_v and is not paired yet. If this is impossible, we reject the current pair C_G, C_H and move to the next one.

Denote by I_H the set $\widetilde{\pi}_{C_G,C_H}(W_H) \cap W_G$, and denote by S_H the set $\widetilde{\pi}^{-1}_{C_G,C_H}(I_H)$. According to Lemma 2.10, $|I_H| > \log^7 n$ with probability $1 - o(2^{-\log^6 n})$; that is, with probability $1 - o(1)$ we have $|I_H| > \log^7 n$ for every pair C_G, C_H (if this is not the case, we terminate the algorithm and answer arbitrarily). Next we take $\frac{1}{2}\log^7 n$ pairs $\{\{u_1, v_1\}, \ldots, \{u_{\frac{1}{2}\log^7 n}, v_{\frac{1}{2}\log^7 n}\}\}$ randomly from S_H, and denote by δ_{C_G,C_H} the fraction of S_H's pairs for which exactly one of $\{u_i, v_i\}$ and $\{\widetilde{\pi}_{C_G,C_H}(u_i), \widetilde{\pi}_{C_G,C_H}(v_i)\}$ is an edge. If $\delta_{C_G,C_H} \leq \epsilon/2$, then the graphs are accepted. Otherwise we move to the next pair C_G, C_H. If none of the pairs accepted, then the graphs are rejected.

As noted above, if G and H are isomorphic, then according to Lemma 2.10 with probability $1 - o(1)$, the size of $U_G \cap \sigma(U_H)$ is at least $\log^2(n)$. Therefore with probability $1 - o(1)$ for some pair C_H, C_G, the placement C_H of C_G is expandable to an isomorphism. We now need to show that in this case the pair C_H, C_G is accepted with sufficient probability.

LEMMA 4.18 (completeness). *If the graphs G and H are isomorphic and σ is an isomorphism between them, then with probability at least $3/4$ there exists $C_G \subset U_G$ with a placement $C_H \subset U_H$ which is expandable to σ, and for which $\delta_{C_G,C_H} \leq \epsilon/2$.*

Proof sketch. First we look at the set $\Delta = U_G \cap \sigma^{-1}(U_H)$. By Lemma 2.10 the size of Δ is at least $\log^2 n$ with probability $1-o(1)$. Conditioned on this event, we pick $C_G \subseteq \Delta \subseteq U_G$ uniformly from all subsets of Δ with size $\log^2 n$, and set $C_H = \sigma(C_G)$ to be its placement in U_H. We now prove that, conditioned on the event that Δ is large enough, C_G and C_H will be as required with probability $1 - o(1)$.

Our main observation is that if we condition only on the event that Δ is large enough, then C_G is distributed uniformly among all subsets with this size of $V(G)$, so we proceed similarly to the case where one of the graphs is known in advance. We observe that if two vertices have many distinct neighbors, then with high probability they will not share exactly the same neighbors within a random core set of size $\log^2 n$ (see Lemma 4.12), so C_G has a separating property. When this happens, it is possible to switch between the vertices with identical labels and still retain a small enough bound on δ_{C_G,C_H}. □

LEMMA 4.19 (soundness). *If the graphs G and H are ϵ-far, and the pair C_G, C_H implies close distributions, then $\Pr[\delta_{C_G,C_H} \leq \epsilon/2] \leq o(2^{-\log^6 n})$, and hence the pair C_G, C_H is accepted in Phase 2 with probability at most $o(2^{-\log^6 n})$.*

Proof sketch. As before, assume that for a fixed pair C_G, C_H the random bijection π_{C_G,C_H} is ϵ-far from isomorphism. We then need to show that δ_{C_G,C_H} as defined in Phase 2 is at most $\epsilon/2$ with probability only $o(2^{-\log^6 n})$.

Since the variation distance between the distributions D_{C_G} and D_{C_H} is at most $\epsilon/10$, the amount of leftovers (which is exactly the distance between the C_G-labeling and the C_H-labeling) is at most $\epsilon n/10$. After removing those $\epsilon n/10$ (or less) leftovers, the fraction of pairs u, v for which exactly one of $\{u, v\}$ and $\{\widetilde{\pi}_{C_G,C_H}(u), \widetilde{\pi}_{C_G,C_H}(v)\}$ is an edge is still not smaller than that of π_{C_G,C_H} by more than $4\epsilon/10$. Now the distribution test of Phase 1 guarantees that $\widetilde{\pi}_{C_G,C_H}$ is $\epsilon/10$-close to the restriction of some random bijection π_{C_G,C_H}. Since the graph G is ϵ-far from being isomorphic to the graph H, the bijection π_{C_G,C_H} must be ϵ-far from an isomorphism. Hence $\widetilde{\pi}_{C_G,C_H}$ must exhibit a $6\epsilon/10$-fraction of incompatible edges, and the acceptance probability of the pair C_G, C_H given $\widetilde{\pi}_{C_G,C_H}$ is equal to the probability that δ_{C_G,C_H} as defined in Phase 2 is at most $\epsilon/2$. Applying large deviation inequalities shows that this probability is at most $2^{-\Omega(\log^7 n)} = o(2^{-\log^6 n})$. □

The isomorphism testing algorithm A_{uu} makes $\widetilde{O}(n^{5/4})$ queries in total, completing the proof of Theorem 4.15.

The lower bound. A lower bound of $\Omega(n)$ queries is implicitly stated in [9] following [1]. Here we provide the detailed proof for completeness.

LEMMA 4.20. *Any adaptive (as well as nonadaptive) testing algorithm that makes at most $\frac{n}{4}$ queries cannot distinguish between the case that the unknown input graphs G and H are isomorphic and the case that they are $\frac{1}{8}$-far from being isomorphic.*

Proof. We construct two distributions over pairs of graphs. The distribution D_P is constructed by letting the pair of graphs consist of a random graph $G \sim G(n, 1/2)$ and

a graph H that is a random permutation of G. The distribution D_N is constructed by letting the pair of graphs consist of two independently chosen random graphs $G, H \sim G(n, 1/2)$.

Clearly D_P satisfies the property with probability 1. By large deviation inequalities, it is also clear that in an input chosen according to D_N, the graphs G and H are $\frac{1}{8}$-far with probability $1 - 2^{\Omega(n^2)}$. The next step is to replace D_N with D'_N, in which the graphs are $\frac{1}{8}$-far from being isomorphic with probability 1. We just set D'_N to be the distribution that results from conditioning D_N on the event that G is indeed $\frac{1}{8}$-far from H.

We now consider any fixed set $Q = \{p_1, \ldots, p_{\frac{n}{4}}\}$ of vertex pairs, some from the first graph, and others from the second graph. For an input chosen according to the distribution D_N, the values of these pairs (the answers for corresponding queries) are $\frac{n}{4}$ uniformly and independently chosen random bits. We now analyze the distribution D_P. Let $e_1, \ldots, e_k$ and $f_1, \ldots, f_l$ be all vertex pairs of the first and the second graph, respectively, that appear in Q. Clearly $k, l \leq |Q| = \frac{n}{4}$. Let $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be the permutation according to which the second graph is chosen in D_P. Let E denote the event that $\sigma(e_i) \neq f_j$ for every $1 \leq i \leq k$ and $1 \leq j \leq l$, where for $e = \{u, v\}$ we denote by $\sigma(e)$ the pair $\{\sigma(u), \sigma(v)\}$. Clearly, if E occurs, then $\{p_1, \ldots, p_{\frac{n}{4}}\}$ will be a set of $\frac{n}{4}$ uniformly and independently chosen random bits.

CLAIM 4.21. *The event E as defined above occurs with probability at least $3/4$.*

Proof. For a single pair e_i and a random permutation σ, the probability that $e_i = \sigma(f_j)$ for some $1 \leq j \leq l$ is bounded by $\frac{n}{2\binom{n}{2}}$. Hence by the union bound,

$$\Pr[E] \geq 1 - \frac{kn}{2\binom{n}{2}} > 3/4. \qquad \square$$

Since E occurs with probability at least $3/4$ and since the event upon which we conditioned D_N to get D'_N occurs with probability $1 - 2^{-\Omega(n^2)} = 1 - o(2^{-|Q|})$, we get that for any $g : \mathcal{Q} \to \{0, 1\}$, $\Pr_{D'_N|_Q}[g] < \frac{3}{2}\Pr_{D_P|_Q}[g]$, and therefore the distributions D_P and D'_N satisfy the conditions of Lemma 2.9. $\square$

5. Concluding remarks. While our two-sided error algorithms run in time quasi-polynomial in n (like the general approximation algorithm of [6]), the one-sided algorithms presented here require an exponential running time. It would be interesting to reduce the running time of the one-sided algorithms to be quasi-polynomial while still keeping them one-sided.

Another issue goes back to [1]. There, the graph isomorphism question was used to prove that certain first order graph properties are impossible to test with a constant number of queries. However, in view of the situation with graph isomorphism, the question now is whether every first order graph property is testable with $O(n^{2-\alpha})$ many queries for some $\alpha > 0$ that depends on the property to be tested.

Finally, it would be interesting to close the remaining gap between $\Omega(n)$ and $\widetilde{O}(n^{5/4})$ in the setting of two graphs that need to be queried and a two-sided error algorithm. It appears (with the aid of martingale analysis on the same distributions D_P, D_N as above) that at least for nonadaptive algorithms the lower bound can be increased a little to a bound of the form $\Omega(n \log^{\alpha} n)$, but we are currently unable to give tighter bounds on the power of n.

Acknowledgments. We would like to thank Salil Vadhan and two anonymous referees for their useful comments.

REFERENCES

[1] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, Combinatorica, 20 (2000), pp. 451–476.

[2] N. Alon, E. Fischer, I. Newman, and A. Shapira, *A combinatorial characterization of the testable graph properties: It's all about regularity*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), 2006, pp. 251–260.

[3] N. Alon and A. Shapira, *A characterization of the (natural) graph properties testable with one-sided error*, SIAM J. Comput., 37 (2008), pp. 1703–1727.

[4] N. Alon and A. Shapira, *Every monotone graph property is testable*, SIAM J. Comput., to appear.

[5] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley-Interscience (John Wiley & Sons), New York, 1992 (1st ed.) and 2000 (2nd ed.).

[6] S. Arora, A. Frieze, and H. Kaplan, *A new rounding procedure for the assignment problem with applications to dense graph arrangement problems*, Math. Program., 92 (2002), pp. 1–36.

[7] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White, *Testing random variables for independence and identity*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 442–451.

[8] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[9] E. Fischer, *The art of uninformed decisions: A primer to property testing*, in Current Trends in Theoretical Computer Science: The Challenge of the New Century, Vol. I, G. Paun, G. Rozenberg, and A. Salomaa, eds., World Scientific, River Edge, NJ, 2004, pp. 229–264.

[10] E. Fischer, *The difficulty of testing for isomorphism against a graph that is given in advance*, SIAM J. Comput., 34 (2005), pp. 1147–1158.

[11] E. Fischer and I. Newman, *Testing versus estimation of graph properties*, SIAM J. Comput., 37 (2007), pp. 482–501.

[12] E. Fischer, I. Newman, and J. Sgall, *Functions that have read-twice constant width branching programs are not necessarily testable*, Random Structures Algorithms, 24 (2004), pp. 175–193.

[13] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[14] P. Hajnal and M. Szegedy, *On packing bipartite graphs*, Combinatorica, 12 (1992), pp. 295–301.

[15] D. Ron, *Property testing (a tutorial)*, in Handbook of Randomized Computing, Vol. II, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 597–649.

[16] R. Rubinfeld and M. Sudan, *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.

[17] A. C. Yao, *Probabilistic computations toward a unified measure of complexity*, in Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 226–240

IMPROVING THE STRETCH FACTOR OF A GEOMETRIC NETWORK BY EDGE AUGMENTATION*

MOHAMMAD FARSHI†, PANOS GIANNOPOULOS‡, AND JOACHIM GUDMUNDSSON§

Abstract. Given a Euclidean graph G in $\mathbb{R}^d$ with n vertices and m edges, we consider the problem of adding an edge to G such that the stretch factor of the resulting graph is minimized. Currently, the fastest algorithm for computing the stretch factor of a graph with positive edge weights runs in $\mathcal{O}(nm+n^2\log n)$ time, resulting in a trivial $\mathcal{O}(n^3m+n^4\log n)$-time algorithm for computing the optimal edge. First, we show that a simple modification yields the optimal solution in $\mathcal{O}(n^4)$ time using $\mathcal{O}(n^2)$ space. To reduce the running time we consider several approximation algorithms.

Key words. computational geometry, approximation algorithms, geometric networks

AMS subject classifications. 65D18, 68U05, 68Q25

DOI. 10.1137/050635675

1. Introduction.

Consider a set V of n points in $\mathbb{R}^d$. A network on V can be modeled as an undirected graph G with vertex set V of size n and an edge set E of size m where every edge (u, v) has a positive weight $w(u, v)$. A Euclidean network is a geometric network where the weight of the edge (u, v) is equal to the Euclidean distance $|uv|$ between its two endpoints u and v.

For two vertices u, v in a weighted graph G we use $\delta_G(u, v)$ to denote a shortest path between u and v in G, and the length of the path is denoted by $d_G(u, v)$. Consider a weighted graph $G = (V, E)$ and a graph $G' = (V, E')$ on the same vertex set but with edge set $E' \subseteq E$. We say that G' is a *t-spanner* of G if for each pair of vertices $u, v \in V$ we have that $d_{G'}(u, v) \leq t \cdot d_G(u, v)$. The minimum t such that G is a t-spanner for V is called the stretch factor, or dilation, of G.

We say that a Euclidean network $G = (V, E)$ is a *t-spanner* if $G = (V, E)$ is a t-spanner of the complete network on V. In other words, for any two points $p, q \in V$ the graph distance in G is at most t times the Euclidean distance between the two points.

Complete graphs represent ideal communication networks, but they are expensive to build; sparse spanners represent low-cost alternatives. The weight of the spanner network is a measure of its sparseness; other sparseness measures include the number of edges, the maximum degree, and the number of Steiner points. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and many other areas. Recently spanners found interesting practical applications

*Received by the editors July 11, 2005; accepted for publication (in revised form) November 20, 2007; published electronically March 28, 2008.

http://www.siam.org/journals/sicomp/38-1/63567.html

†School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada (mfarshi@cg.scs.carleton.ca) and Department of Computer Science, Yazd University, P.O. Box 89195-741, Yazd, Iran (mfarshi@yazduni.ac.ir). This author's research was supported by the Ministry of Science, Research and Technology of I.R. Iran.

‡Institut für Informatik, Humboldt-Universität zu Berlin, D-10099 Berlin, Germany (panos@informatik.tu-berlin.de). This author's research was partially conducted at IICS, Utrecht University, The Netherlands.

§IMAGEN Program, National ICT Australia Ltd., NSW Alexandria 1435, Sydney, Australia (joachim.gudmundsson@nicta.com.au). NICTA is funded through the Australian government's Backing Australia's Ability initiative, in part through the Australian Research Council.

TABLE 1
Complexity bounds for the algorithms presented in the paper.

Input graph	Approximation factor	Time complexity	Space	Section
Weighted graph	1	$\mathcal{O}(n^3m + n^4 \log n)$	$\mathcal{O}(m)$	2.1
Weighted graph	1	$\mathcal{O}(n^4)$	$\mathcal{O}(n^2)$	2.1
Euclidean graph	$1+\varepsilon$	$\mathcal{O}(n^3/\varepsilon^d)$	$\mathcal{O}(n^2)$	2.2
Weighted graph	3	$\mathcal{O}(nm + n^2 \log n)$	$\mathcal{O}(m)$	3
Euclidean graph	$2+\varepsilon$	$\mathcal{O}(nm + n^2(\log n + 1/\varepsilon^{3d}))$	$\mathcal{O}(n^2)$	4
Euclidean t-spanner	$1+\varepsilon$	$\mathcal{O}((t^7/\varepsilon^4)^d \cdot n^2)$	$\mathcal{O}(m + (t^3/\varepsilon^2)^d\, n \log(tn))$	5

in areas such as metric space searching [29, 30] and broadcasting in communication networks [2, 14, 25].

Several well-known theoretical results also use the construction of t-spanners as a building block; for example, Rao and Smith [32] made a breakthrough by showing an optimal $\mathcal{O}(n \log n)$-time approximation scheme for the well-known Euclidean *traveling salesperson problem*, using t-spanners (or banyans). Similarly, Czumaj and Lingas [7] showed approximation schemes for minimum-cost multiconnectivity problems in geometric graphs. The problem of constructing geometric spanners has received considerable attention from a theoretical perspective; see [1, 3, 4, 5, 8, 9, 10, 17, 20, 21, 23, 24, 33, 36], the surveys [12, 16, 34], and the book by Narasimhan and Smid [28]. Note that considerable research has also been done in the construction of spanners for general graphs; see, for example, the book by Peleg [31] or the recent work by Elkin and Peleg [11] and Thorup and Zwick [35].

All the existing algorithms construct a network from scratch, but in many applications the network is already given, and the problem at hand is to extend the network with an additional edge, or edges, while minimizing the stretch factor of the resulting graph. The problem was first stated by Narasimhan [26] and, surprisingly, it had not been studied earlier, to the best of the authors' knowledge. In this paper we study the following problem.

Problem. Given a graph G, construct a graph G' by adding an edge to G such that the stretch factor of G' is minimized.

The results presented in this paper are summarized in Table 1. Note that some of the presented bounds hold for any graph with positive edge weights (*weighted graphs*), while some hold only for Euclidean graphs.

Finally, throughout this paper we will use $G_{\mathcal{P}}$ to denote the optimal solution, while $t_{\mathcal{P}}$ and t denote the stretch factor of $G_{\mathcal{P}}$ and the input graph G, respectively.

2. Three simple algorithms. A naive approach to deciding which edge to add is to test every possible candidate edge. The number of such edges is obviously $(\frac{n(n-1)}{2} - m) = \mathcal{O}(n^2)$. Testing a candidate edge e entails computing the stretch factor of the graph $G' = (V, E \cup \{e\})$, denoted the candidate graph. Therefore we briefly consider the problem of computing the stretch factor of a given graph with

positive edge weights. This problem has recently received considerable attention; see, for example, [13, 22, 27].

2.1. Exact algorithms. We consider the problem of computing an optimal solution $G_{\mathcal{P}}$. That is, we are given a t-spanner $G = (V, E)$, and the aim is to compute a $t_{\mathcal{P}}$-spanner $G_{\mathcal{P}} = (V, E \cup \{e\})$.

A trivial upper bound is obtained by computing the length of the shortest paths between every pair of vertices in G'. This can be done by running Dijkstra's algorithm —implemented using Fibonacci heaps—n times, resulting in an $\mathcal{O}(mn+n^2 \log n)$-time algorithm using linear space. This approach is quite slow, and we would like to be able to compute the stretch factor more efficiently, but no faster algorithm is known for any graphs except planar graphs, paths, cycles, stars, and trees [13, 22, 27]. Applying the stated bound to the problem of computing the exact stretch factor of G' gives that $G_{\mathcal{P}}$ can be computed in time $\mathcal{O}(n^3(m + n \log n))$ using linear space.

A small improvement can be obtained by observing that when an edge (u, v) is about to be tested, we do not have to check all possible shortest paths between two vertices $x, y \in V$ again; it suffices to check whether there is a shorter path using the edge (u, v). That is, we only have to compute $d_G(x, u) + w(u, v) + d_G(v, y)$, $d_G(x, v) + w(v, u) + d_G(u, y)$, and $d_G(x, y)$, which can be done in constant time since the length of a shortest path between every pair of vertices in G has already been computed (provided that we store this information). Hence, by first computing all-pair-shortest paths of G we obtain the following lemma.

LEMMA 1. *Given a graph G with positive edge weights, an optimal solution $G_{\mathcal{P}}$ can be computed in time $\mathcal{O}(n^4)$ using $\mathcal{O}(n^2)$ space.*

Proof. Computing the all-pair-shortest path requires cubic time, and all the distances are stored in an $n \times n$ matrix. The $\mathcal{O}(n^2)$ edges are tested for insertion: for each candidate edge compute the length of the shortest path between every pair of points in G, each of which can be done in constant time as described above. □

2.2. A $(1+\varepsilon)$-approximation for Euclidean graphs. In the previous section we showed that an optimal solution can be obtained by testing a quadratic number of candidate edges. Testing each candidate edge entails $\mathcal{O}(n^2)$ distance queries, where a distance query asks for the length of a shortest path in the graph between two query points. One way to speed up the computation is to compute an approximate stretch factor. t' is said to be a β-approximate stretch factor of G if $t_G \leq t' \leq \beta \cdot t_G$, where t_G is the stretch factor of G. The problem of computing an approximate stretch factor of a geometric graph was considered by Narasimhan and Smid in [27]. They showed the following fact.

Fact 1 (Narasimhan and Smid [27]). Given a Euclidean graph G and a real value $\tau > 0$, a $(1 + \tau)^2$-approximative stretch factor of G can be computed by performing $\mathcal{O}(n/\tau^d)$ many $(1 + \gamma)$-approximate distance queries, where γ is a positive constant smaller than τ.

The algorithm is almost as stated in the previous section with the exception that when the stretch factor of the candidate graph is computed we approximate it by only performing $\mathcal{O}(n/\tau^d)$ shortest path queries as stated in Fact 1. As a result the time to compute the stretch factor decreases from $\mathcal{O}(n^2)$ to $\mathcal{O}(n/\tau^d)$; thus the total running time decreases from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^3/\tau^d)$.

THEOREM 2. *Given a Euclidean graph $G = (V, E)$ and a real constant $\varepsilon > 0$, one can in $\mathcal{O}(n^3/\varepsilon^d)$ time, using $\mathcal{O}(n^2)$ space, compute a t'-spanner $G' = (V, E \cup \{e\})$ such that $t' \leq (1 + \varepsilon) \cdot t_{\mathcal{P}}$.*

Proof. The time bound follows from the above discussion setting $\tau = \sqrt{1+\varepsilon} - 1$, where τ is as stated in Fact 1. It remains to prove that G' has stretch factor $((1+\varepsilon)\cdot t_{\mathcal{P}})$.

For each candidate graph G'_i, let t'_i be its approximate stretch factor as computed by the algorithm, and let t_i be its exact stretch factor. From Fact 1 it follows that for each candidate graph G'_i, $t'_i \leq (1+\tau)^2 \cdot t_i$. Assume that $t_{\mathcal{P}} = t_j$ and that $t' = t'_k = \min_i t'_i$, for some indices j and k. As a result we have

$$t' = t'_k \leq t'_j \leq (1+\tau)^2 \cdot t_j = (1+\tau)^2 \cdot t_{\mathcal{P}} = (1+\varepsilon)\cdot t_{\mathcal{P}} \quad \text{and} \quad t_{\mathcal{P}} \leq t_k \leq t'_k = t'.$$

Thus, $t_{\mathcal{P}} \leq t' \leq (1+\varepsilon)\cdot t_{\mathcal{P}}$. □

3. Adding a bottleneck edge. Consider a graph $G = (V, E)$ with positive edge weights and stretch factor t. In this section we analyze the following simple algorithm: Add an edge between a pair of vertices in G with stretch factor t; this edge is called a *bottleneck edge* of G.

Let $G_{\mathcal{B}}$ be a graph obtained from G by adding a bottleneck edge, and let $t_{\mathcal{B}}$ be the stretch factor of $G_{\mathcal{B}}$. Note that $G_{\mathcal{B}}$ can be computed in the same time as the stretch factor of G can be decided, i.e., in $\mathcal{O}(mn + n^2 \log n)$ time for graphs with positive edge weights.

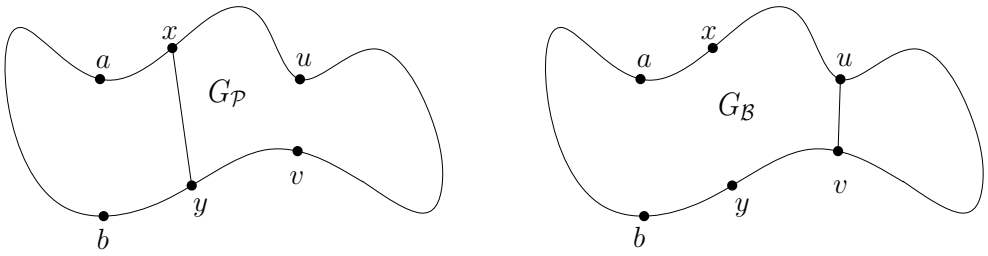


FIG. 1. (x, y) *is the optimal edge added to* G, *and* (u, v) *is a bottleneck edge.*

LEMMA 3. *Given a graph* G *with positive edge weights, it holds that* $t_{\mathcal{B}} < 3t_{\mathcal{P}}$.

Proof. Recall that t denotes the stretch factor of G and that $G_{\mathcal{P}}$ denotes the optimal graph. Let (x, y) be the edge added to G to obtain $G_{\mathcal{P}}$, and let (u, v) be the edge added to G to obtain $G_{\mathcal{B}}$; i.e., (u, v) is a bottleneck edge of G, as illustrated in Figure 1.

First note that if $t_{\mathcal{P}} > t/3$, then the lemma holds and we are done. Thus we may assume that $t_{\mathcal{P}} \leq t/3$. The proof of the lemma is done by considering a pair of vertices, denoted (a, b), that are endpoints of a bottleneck edge of $G_{\mathcal{B}}$. Fix a path $\delta_{G_{\mathcal{P}}}(a, b)$. If this path does not include the edge (x, y), then $d_{G_{\mathcal{P}}}(a, b) = d_G(a, b) \geq d_{G_{\mathcal{B}}}(a, b)$ and we are done. Therefore, we may assume that the path $\delta_{G_{\mathcal{P}}}(a, b)$ includes (x, y). Also, we will assume without loss of generality that a shortest path in $G_{\mathcal{P}}$ from a to b goes from a to x and then to b via y; otherwise the labels a and b may be switched. Note that $\delta_{G_{\mathcal{P}}}(u, v)$ must pass through (x, y); otherwise we have $t_{\mathcal{P}} \geq d_{G_{\mathcal{P}}}(u, v)/|uv| = d_G(u, v)/|uv| = t$, which means that $t = t_{\mathcal{P}}$, which contradicts the assumption that $t_{\mathcal{P}} \leq t/3$. Furthermore, we assume that a shortest path in $G_{\mathcal{P}}$ from u to v goes from u to x and then to v via y; otherwise the labels u and v may be switched.

As a first step we bound the distance between the endpoints of the bottleneck edge u and v. This is done by bounding the length of the path in G between x and y

as follows (see Figure 1):

$$\begin{aligned} d_G(u,v) &\le d_{G_{\mathcal{P}}}(u,v) - |xy| + d_G(x,y) \\ &\le t_{\mathcal{P}} \cdot |uv| - |xy| + t \cdot |xy| \\ &\le \frac{t}{3} \cdot |uv| - |xy| + t \cdot |xy| \\ &< \frac{t}{3} \cdot |uv| + t \cdot |xy|. \end{aligned}$$

Since $d_G(u,v) = t \cdot |uv|$ it follows that

$$|uv| < 3/2 \cdot |xy|. \tag{1}$$

Also,

$$\begin{aligned} t \cdot |uv| &= d_G(u,v) \\ &\le d_G(u,a) + d_G(a,b) + d_G(b,v) \\ &\le d_G(u,a) + t \cdot |ab| + d_G(b,v), \end{aligned}$$

which implies that

$$t \cdot (|uv| - |ab|) \le d_G(u,a) + d_G(b,v), \tag{2}$$

and

$$\begin{aligned} d_G(a,u) + 2|xy| + d_G(v,b) &\le d_G(a,x) + d_G(x,u) + 2|xy| + d_G(v,y) + d_G(y,b) \\ &= d_{G_{\mathcal{P}}}(a,b) + d_{G_{\mathcal{P}}}(u,v) \\ &\le t_{\mathcal{P}}(|ab| + |uv|), \end{aligned} \tag{3}$$

which gives that

$$d_G(a,u) + d_G(v,b) \le t_{\mathcal{P}}(|ab| + |uv|) - 2|xy|. \tag{4}$$

By putting together (2) and (4) we have

$$\begin{aligned} t(|uv| - |ab|) &\le d_G(a,u) + d_G(v,b) \\ &\le t_{\mathcal{P}}(|ab| + |uv|) - 2|xy| \\ &< t_{\mathcal{P}}(|ab| + |uv|), \end{aligned}$$

which implies that

$$|ab|(t_{\mathcal{P}} + t) > |uv|(t - t_{\mathcal{P}})$$

and

$$|ab| > \frac{t - t_{\mathcal{P}}}{t_{\mathcal{P}} + t} \cdot |uv| > \frac{t - \frac{t}{3}}{\frac{t}{3} + t} \cdot |uv| = \frac{1}{2} \cdot |uv|. \tag{5}$$

Now we are ready to put together the results:

$$\begin{aligned} t_{\mathcal{B}} \cdot |ab| &= d_{G_{\mathcal{B}}}(a,b) \\ &\le d_G(a,u) + |uv| + d_G(v,b) \\ &< d_G(a,u) + \frac{3}{2}|xy| + d_G(v,b) && \text{(from (1))} \\ &< d_G(a,u) + 2|xy| + d_G(v,b) \\ &\le t_{\mathcal{P}}\,(|ab| + |uv|) && \text{(from (3))} \\ &< 3t_{\mathcal{P}} \cdot |ab| && \text{(from (5))}. \end{aligned}$$

This completes the proof of the lemma since $t_{\mathcal{B}} < 3t_{\mathcal{P}}$. ∎

We conclude by stating the main result of this section followed by a lower bound for the bottleneck approach.

THEOREM 4. *Given a graph $G = (V, E)$ with positive edge weights, a $t_{\mathcal{B}}$-spanner $G' = (V, E \cup \{e\})$ with $t_{\mathcal{B}} < 3t_{\mathcal{P}}$ can be computed in $\mathcal{O}(mn + n^2 \log n)$ time using $\mathcal{O}(m)$ space.*

OBSERVATION 1. *There exists a Euclidean graph G such that $(2-\varepsilon)\cdot t_{\mathcal{P}} \leq t_{\mathcal{B}}$ for any $0 < \varepsilon < 1$.*

Proof. Consider the graph G, as in Figure 2(a). More specifically, G is a graph with ten vertices $p_i = ((i-1) \mod 5, \lfloor (i-1)/5 \rfloor \cdot \delta)$, $1 \leq i \leq 10$, and nine edges (p_5, p_{10}) and (p_j, p_{j+1}) for $1 \leq j \leq 4$ and $6 \leq j \leq 9$. For any value $\delta \leq 1$, (p_1, p_6) is a bottleneck edge in G and $t_{\mathcal{B}} = \frac{4+\delta}{\delta}$; see Figure 2(b).

In the case where edge (p_2, p_7) is added to G, as shown in Figure 2(c), the resulting graph has stretch factor $(2+\delta)/\delta$. Combining the upper and lower bounds gives $\frac{t_{\mathcal{B}}}{t_{\mathcal{P}}} \geq \frac{4+\delta}{2+\delta} = (2-\varepsilon)$, where the last equality follows if we set $\delta = \min\{1, \frac{2\varepsilon}{1-\varepsilon}\}$. ∎

Grüne [15] improved the lower bound in Observation 1 to $(3-\varepsilon)$, so the upper bound stated in Lemma 3 is tight.

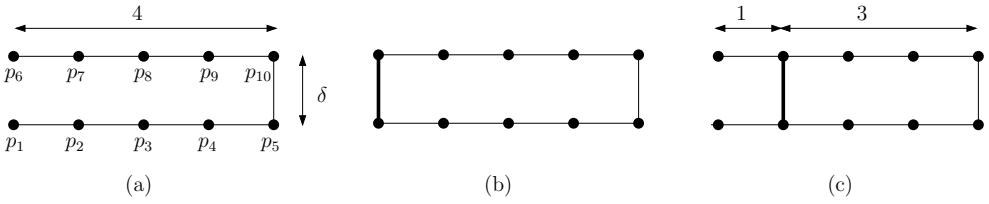


FIG. 2. (a) *The input graph G,* (b) *the graph $G_{\mathcal{B}}$, and* (c) *the graph $G_{\mathcal{P}}$.*

4. A $(2+\varepsilon)$-approximation for Euclidean graphs. In the remainder of the paper we will develop approximation algorithms for Euclidean graphs. In this section we present a fast approximation algorithm which guarantees an approximation factor of $(2+\varepsilon)$. The algorithm is similar to the algorithms presented in section 2 in the sense that it tests candidate edges. Testing a candidate edge entails computing the stretch factor of the input graph augmented with the candidate edge. The main difference is that we will show, in section 4.2, that only a linear number of candidate edges need to be tested to obtain a solution that gives a $(2+\varepsilon)$-approximation, instead of a quadratic number of edges.

Moreover, in section 4.3 we show that the same approximation bound can be achieved by performing only a linear number of shortest path queries for each candidate edge. The candidate edges are selected by using the well-separated pair decomposition, which we briefly define below.

4.1. Well-separated pair decomposition. Our algorithm uses the well-separated pair decomposition defined by Callahan and Kosaraju [6]. We briefly review this decomposition before we state the algorithms.

DEFINITION 5 (see [6]). *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well separated *with respect to s if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that* (i) *C_A contains the bounding box $R(A)$ of A,* (ii) *C_B contains the bounding box $R(B)$ of B, and* (iii) *the minimum distance between C_A and C_B is at least s times the radius of C_A.*

The parameter s will be referred to as the *separation constant*. The next lemma follows easily from Definition 5.

LEMMA 6 (see [6]). *Let A and B be two finite sets of points that are well separated with respect to s, let x and p be points of A, and let y and q be points of B. Then* (i) $|xy| \leq (1 + 4/s) \cdot |pq|$, *and* (ii) $|px| \leq (2/s) \cdot |pq|$.

DEFINITION 7 (see [6]). *Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A* well-separated pair decomposition *(WSPD) for S with respect to s is a sequence of pairs of nonempty subsets of S, $(A_1, B_1), \ldots, (A_m, B_m)$, such that*

1. *$A_i \cap B_i = \emptyset$ for all $i = 1, \ldots, m$,*
2. *for any two distinct points p and q of S, there is exactly one pair (A_i, B_i) in the sequence, such that* (i) *$p \in A_i$ and $q \in B_i$, or* (ii) *$q \in A_i$ and $p \in B_i$,*
3. *A_i and B_i are well separated with respect to s for $1 \leq i \leq m$.*

The integer m is called the size *of the WSPD.*

Callahan and Kosaraju showed that a WSPD of size $m = \mathcal{O}(s^d n)$ can be computed in $\mathcal{O}(s^d n + n \log n)$ time.

4.2. Linear number of candidate edges. In this section we show how to obtain a $(2 + \varepsilon)$-approximation in cubic time. As mentioned above, the algorithm is similar to the algorithm presented in section 2 in the sense that it tests candidate edges. Here we will show that only a linear number of candidate edges need to be tested to obtain a solution that gives a $(2 + \varepsilon)$-approximation.

The approach is straightforward. First the algorithm computes the length of the shortest path in G between every pair of points in V. The distances are saved in a matrix M. Next, the WSPD is computed. Note that, in step 5, the candidate edges will be chosen using the WSPD. In step 6, the function StretchFactor returns the stretch factor of the graph on V with edge set $E \cup (a_i, b_i)$; i.e., in steps 5–8, a candidate edge is tested by computing the stretch factor of G with the candidate edge (a_i, b_i) added to G.

ALGORITHM ExpandGraph(G, ε)
Input: Euclidean graph $G = (V, E)$ and a real constant $\varepsilon > 0$.
Output: Euclidean graph $G' = (V, E \cup \{e\})$.
1. $M \leftarrow$All-Pairs-Shortest-Path distance matrix of G.
2. $\{(A_i, B_i)\}_{i=1}^{k} \leftarrow$WSPD of the set V with respect to separation constant $s = \frac{256}{\varepsilon^2}$.
3. $t' \leftarrow \infty$.
4. **for** $i \leftarrow 1$ to k
5. Select arbitrary points $a_i \in A_i$ and $b_i \in B_i$.
6. $t_i \leftarrow$StretchFactor(a_i, b_i, M).
7. **if** $t_i < t'$
8. **then** $t' \leftarrow t_i$ and $e \leftarrow (a_i, b_i)$
9. **return** $G' = (V, E \cup \{e\})$.

Next, we bound the running time of the approximation algorithm and then prove the approximation bound.

LEMMA 8. *Algorithm* ExpandGraph *requires $\mathcal{O}(n^3/\varepsilon^{2d})$ time and $\mathcal{O}(n^2)$ space.*

Proof. The complexity of all steps of the algorithm, except step 6, is straightforward to calculate. Recall that step 1 requires $\mathcal{O}(mn + n^2 \log n)$ time and quadratic space, and step 2 requires $\mathcal{O}(n/\varepsilon^{2d} + n \log n)$ time according to section 4.1. Thus, it remains to consider step 6 of the algorithm. Note that the number of times step 6 is executed is $\mathcal{O}(n/\varepsilon^{2d})$.

Let $G_i = (V, E \cup \{(a_i, b_i)\})$. Since we computed the all-pair-shortest distances of G and stored the results in a matrix M, it holds that shortest path distance queries in G_i can be computed in constant time. That is, for a query (p, q) return $\min\{M[p,q], M[p,a_i]+|a_ib_i|+M[b_i,q], M[p,b_i]+|b_ia_i|+M[a_i,q]\}$. For each candidate edge, a quadratic number of queries are performed; thus summing up we get $\mathcal{O}(\frac{n}{\varepsilon^{2d}} \cdot n^2)$, as stated in the lemma. □

It remains to analyze the quality of the solution obtained from Algorithm ExpandGraph. We need to compare the graph resulting from adding an optimal edge to G and the graph G' resulting from ExpandGraph. Let $e = (a, b)$ be an optimal edge, and let (A_i, B_i) be the well-separated pair such that $a \in A_i$ and $b \in B_i$. At first sight, it seems that the edge (a_i, b_i) tested by the algorithm should be a good candidate. However, the separation constant of our WSPD depends only on ε, which implies that the shortest path between a and a_i, and between b and b_i, could be very long compared to the distance between a and b. In Lemma 9, we show the existence of a "short" edge e' that is a good approximation of the optimal edge and then, in Lemma 10, we show that ExpandGraph computes a good approximation of e'.

Let $\Delta(p, q)$ denote the set of point pairs in V such that the point pair (u, v) belongs to $\Delta(p, q)$ if and only if $(p, q) \in \delta_{G\cup\{(p,q)\}}(u, v)$. That is, $\Delta(p, q)$ is the set of point pairs for which a shortest path between them in $G \cup \{(p, q)\}$ passes through (p, q).

Lemma 9. *For any given constant $0 < \lambda \leq 1$, there exists a point pair $p, q \in V$ such that*

(I) $|uv| \geq \frac{\lambda}{2}|pq|$ *for every pair* $(u, v) \in \Delta(p, q)$, *and*

(II) *the stretch factor of* $G \cup \{(p, q)\}$ *is bounded by* $(2 + \lambda) \cdot t_{\mathcal{P}}$.

Proof. The proof is done in two steps. First a point pair $p_j, q_j \in V$ is selected that fulfills (I). Then we prove that this pair will also fulfill (II), i.e., that the stretch factor of $G \cup \{(p_j, q_j)\}$ is bounded by $(2 + \lambda) \cdot t_{\mathcal{P}}$.

Consider an optimal solution $G_1 = G \cup \{(p_1, q_1)\}$. If (p_1, q_1) fulfills (I), then we are done; i.e., we have found the point pair $(p = p_1, q = q_1)$ we are searching for. Otherwise, let $e_2 = (p_2, q_2)$ denote the closest pair in $\Delta(p_1, q_1)$. Since there exists a pair $(u, v) \in \Delta(p_1, q_1)$ such that $|uv| < \frac{\lambda}{2} \cdot |p_1q_1|$ and since (p_2, q_2) is the closest pair in $\Delta(p_1, q_1)$ we have $|p_2q_2| < \frac{\lambda}{2} \cdot |p_1q_1|$, as illustrated in Figure 3(a).

If (p_2, q_2) fulfills (I), then $(p = p_2, q = q_2)$ and we are done. Otherwise, let $e_3 = (p_3, q_3)$ denote the closest pair in $\Delta(p_2, q_2)$. We continue this procedure until we find a point pair (p_j, q_j) that satisfies (I). Since, for each $i > 0$, $|p_{i+1}q_{i+1}| < \frac{\lambda}{2} \cdot |p_iq_i|$, the process must terminate.

Now for each $1 \leq i \leq j$, let $G_i = G \cup \{(p_i, q_i)\}$ where (p_i, q_i) are the point pairs constructed above. We claim that G_j has stretch factor at most $(2 + \lambda) \cdot t_{\mathcal{P}}$. Before we continue we need to prove

$$d_{G_i}(p_{i+1}, q_{i+1}) \leq t_{\mathcal{P}} \cdot |p_{i+1}q_{i+1}|. \tag{6}$$

The inequality is obviously true for $i = 1$. For $i > 1$ it holds that $|p_{i+1}q_{i+1}| < |p_2q_2|$ which implies that $(p_{i+1}, q_{i+1}) \notin \Delta(p_1, q_1)$ since (p_2, q_2) is the closest pair in $\Delta(p_1, q_1)$. This, in turn, implies that $d_G(p_{i+1}, q_{i+1}) = d_{G_1}(p_{i+1}, q_{i+1}) \leq t_{\mathcal{P}} \cdot |p_{i+1}, q_{i+1}|$. Since G is a subgraph of G_i, the length of the shortest path in G_i between p_{i+1} and q_{i+1} must be bounded by the length of the shortest path in G between p_{i+1} and q_{i+1}, which is bounded by $t_{\mathcal{P}} \cdot |p_{i+1}q_{i+1}|$. Thus, inequality (6) holds.

We continue with the second part of the proof. If $(u, v) \notin \Delta(p_1, q_1)$, then we are done since $d_{G_j}(u, v) \leq d_G(u, v) = d_{G_1}(u, v)$. Otherwise, if $(u, v) \in \Delta(p_1, q_1)$, the

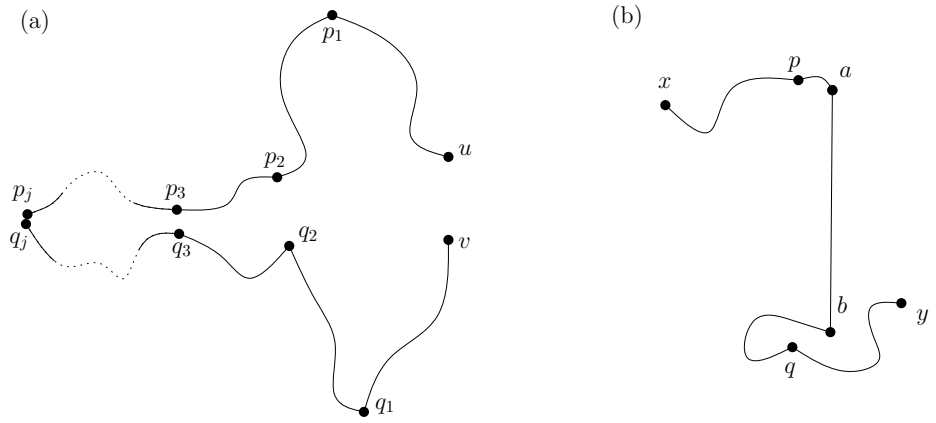


FIG. 3. (a) *Illustrating the proof of Lemma* 9. (b) *Illustrating the proof of Lemma* 10.

following holds (see Figure 3(a) for an illustration):

$$\begin{aligned}
d_{G_j}(u,v) &\le d_{G_1}(u,v) - |p_1q_1| + (d_{G_1}(p_2,q_2) - |p_1q_1|) + \cdots \\
&\quad + (d_{G_{j-1}}(p_j,q_j) - |p_{j-1}q_{j-1}|) + |p_jq_j| \\
&< t_{\mathcal{P}} \cdot |uv| - |p_1q_1| + (t_{\mathcal{P}} \cdot |p_2q_2| - |p_1q_1|) + \cdots \\
&\quad + (t_{\mathcal{P}} \cdot |p_jq_j| - |p_{j-1}q_{j-1}|) + |p_jq_j| && \text{(cf. (6))} \\
&= t_{\mathcal{P}} \cdot |uv| - 2|p_1q_1| + ((t_{\mathcal{P}}-1) \cdot |p_2q_2|) + \cdots + ((t_{\mathcal{P}}-1) \cdot |p_jq_j|) + 2|p_jq_j| \\
&< t_{\mathcal{P}} \cdot |uv| + (t_{\mathcal{P}}-1)(|p_2q_2| + \cdots + |p_jq_j|) && \text{(since } |p_jq_j| < |p_1q_1|\text{)} \\
&< t_{\mathcal{P}} \cdot |uv| + t_{\mathcal{P}} \cdot \sum_{i=2}^{j} \left(\frac{\lambda}{2}\right)^{i-2} |p_2q_2| && \text{(since } |p_{i+1}q_{i+1}| \le (\lambda/2) \cdot |p_iq_i|\text{)} \\
&\le t_{\mathcal{P}} \cdot |uv| + t_{\mathcal{P}} \cdot |uv| \cdot \sum_{i=0}^{j-2} \left(\frac{\lambda}{2}\right)^{i} && \text{(since } |p_2q_2| \le |uv|\text{)} \\
&= 2t_{\mathcal{P}} \cdot |uv| + t_{\mathcal{P}} \cdot |uv| \cdot \sum_{i=1}^{j-2} \left(\frac{\lambda}{2}\right)^{i} \\
&\le 2t_{\mathcal{P}} \cdot |uv| + t_{\mathcal{P}} \cdot |uv| \cdot \lambda \cdot \sum_{i=1}^{j-2} \frac{1}{2^i} && \text{(since } \lambda \le 1\text{)} \\
&< (2+\lambda) \cdot t_{\mathcal{P}} \cdot |uv|.
\end{aligned}$$

Thus, $t_j < (2+\lambda) \cdot t_{\mathcal{P}}$, which concludes the lemma. □

In the previous lemma we showed the existence of a "short" candidate edge (p,q) for which the resulting graph has small stretch factor. Note that Algorithm EXPAND-GRAPH might not test (p,q). However, in the following lemma it will be shown that Algorithm EXPANDGRAPH will test an edge (a,b) that is almost as good as (p,q).

LEMMA 10. *For any given constant* $0 < \varepsilon \le 1$ *it holds that the graph* G' *returned by Algorithm* EXPANDGRAPH *has stretch factor at most* $(2+\varepsilon) \cdot t_{\mathcal{P}}$.

Proof. According to Lemma 9, there exists an edge (p,q) such that for every pair $(u,v) \in \Delta(p,q)$ it holds that $|uv| \ge \frac{\lambda}{2}|pq|$, and the stretch factor t_H of $H = G \cup \{(p,q)\}$ is bounded by $(2+\lambda) \cdot t_{\mathcal{P}}$. Let (A_i, B_i) be the well-separated pair computed in step 2 of the algorithm such that $p \in A_i$ and $q \in B_i$. According to Definition 7 such a

well-separated pair must exist. Next, consider the candidate edge (a_i, b_i) tested by the algorithm, such that $a_i, p \in A_i$ and $b_i, q \in B_i$. For simplicity of writing we will use a and b to denote a_i and b_i, respectively.

Our claim is that the stretch factor t' of $G' = G \cup \{(a, b)\}$ is bounded by $(1+\varepsilon/4) \cdot t_H$. Thus, setting $\lambda = \varepsilon/4$ would then prove the lemma since $(2+\varepsilon/4)(1+\varepsilon/4) < (2+\varepsilon)$ for $\varepsilon \leq 1$.

Now we are ready to prove the claim. To compute the stretch factor of G' the algorithm performs a shortest path distance query between each pair of points in V. If it holds that $(x, y) \notin \Delta(p, q)$ for every pair of points $x, y \in V$, then the claim is obviously true; thus we have to consider only the pairs x, y for which it holds that $(x, y) \in \Delta(p, q)$; see Figure 3(b). Now the claim is

$$d_G(a, p) = d_H(a, p) \quad \text{and} \quad d_G(b, q) = d_H(b, q). \tag{7}$$

Lemma 9 states that if $(x', y') \in \Delta(p, q)$, then $|x'y'| \geq \frac{\varepsilon}{8}|pq|$. But by Lemma 6 the distances $|ap|$ and $|bq|$ are less than $\frac{2}{s}|pq| = \frac{\varepsilon^2}{128}|pq|$, which is less than $\frac{\varepsilon}{8}|pq|$ since $\varepsilon \leq 1$. As a consequence, $(a, p) \notin \Delta(p, q)$ and $(b, q) \notin \Delta(p, q)$; thus $(p, q) \notin \delta_H(a, p)$ and $(p, q) \notin \delta_H(b, q)$. Hence, claim (7) holds, which we will need below.

Next, we consider the length of the path in G' between x and y as illustrated in Figure 3(b). Recall that x and y are two arbitrary points of V for which it holds that $(x, y) \in \Delta(p, q)$. Without loss of generality we have

$$\begin{aligned}
d_{G'}(x, y) &\leq d_G(x, p) + d_G(p, a) + |ab| + d_G(b, q) + d_G(q, y) \\
&\leq d_G(x, p) + d_H(p, a) + |ab| + d_H(b, q) + d_G(q, y) && \text{(cf. (7))} \\
&\leq d_G(x, p) + |ab| + d_G(q, y) + t_H \cdot (|pa| + |bq|) \\
&\leq d_G(x, p) + (1 + 4/s) \cdot |pq| + d_G(q, y) + \frac{4t_H}{s} \cdot |pq| && \text{(Lemma 6)} \\
&\leq d_H(x, y) + \frac{8t_H}{s} \cdot |pq| \\
&\leq d_H(x, y) + \frac{64t_H}{\varepsilon s} \cdot |xy| && \text{(Lemma 9)} \\
&= d_H(x, y) + \frac{\varepsilon}{4} \cdot t_H \cdot |xy|.
\end{aligned}$$

The stretch factor of the path in G' between x and y is

$$\frac{d_{G'}(x, y)}{|xy|} \leq \frac{d_H(x, y)}{|xy|} + \frac{\frac{\varepsilon}{4} t_H |xy|}{|xy|} \leq \left(1 + \frac{\varepsilon}{4}\right) \cdot t_H.$$

Finally, according to Lemma 9 and the fact that $\lambda = \varepsilon/4$, it holds that $t_H \leq (2 + \varepsilon/4) \cdot t_{\mathcal{P}}$. This completes the lemma since $(2 + \varepsilon/4)(1 + \varepsilon/4) < (2 + \varepsilon)$. □

We may now conclude this section with the following theorem.

THEOREM 11. *Given a Euclidean graph $G = (V, E)$ in $\mathbb{R}^d$ one can in time $\mathcal{O}(n^3/\varepsilon^{2d})$, using $\mathcal{O}(n^2)$ space, compute a t'-spanner $G' = (V, E \cup \{e\})$, where $t' \leq (2 + \varepsilon) \cdot t_{\mathcal{P}}$.*

4.3. Speeding up Algorithm ExpandGraph. In the previous section we showed that a $(2+\varepsilon)$-approximate solution can be obtained by testing a linear number of candidate edges. Testing each candidate edge entails $\mathcal{O}(n^2)$ shortest path queries. One way to speed up the computation is to compute an approximate stretch factor. As in section 2.2 we will use Fact 1 by Narasimhan and Smid [27].

Their idea is to compute a WSPD of size $\mathcal{O}(s^d n)$ with respect to $s = 4(1+\tau)/\tau$, and then for each well-separated pair (A_i, B_i) select an arbitrary pair $a_i \in A_i$ and $b_i \in B_i$. They prove that these are the only pairs for which the $(1+\tau)^2$-approximate stretch factor needs to be computed.

We will use their idea to speed up step 6 of ExpandGraph from $\mathcal{O}(n^2)$ to $\mathcal{O}(n/\varepsilon^d)$; i.e., we check a linear number of pairs in order to compute an approximate stretch factor using Fact 1. However, we will not use the fact that only approximate distance queries are needed; instead the exact shortest distance will be computed, and thus $\gamma = 0$ where γ is as stated in Fact 1. There will be two main changes in the ExpandGraph algorithm; two WSPDs will be computed, and the computation of the stretch factor will be different. Instead of computing the exact stretch factor of G with the candidate edge (a_i, b_i) added to G, we compute the approximate stretch factor. This is done by a call to ApproximateStretchFactor, or ASF for short, with parameters (a_i, b_i), M, and $\mathcal{S}$. The ASF algorithm is stated in more detail below. Note that the number of point pairs in $\mathcal{S}$ is bounded by $\mathcal{O}(n/\varepsilon^d)$.

Algorithm ExpandGraph2(G, ε)
Input: Euclidean graph $G = (V, E)$ and a real constant $\varepsilon > 0$.
Output: Euclidean graph $G' = (V, E \cup \{e\})$.
1. $M \leftarrow$All-Pairs-Shortest-Path distance matrix of G.
2. $\{(A_i, B_i)\}_{i=1}^{k} \leftarrow$WSPD of the set V with respect to $s = 256/\varepsilon^2$.
3. $\{(C_j, D_j)\}_{j=1}^{\ell} \leftarrow$WSPD of the set V with respect to $s' = 4(1+\varepsilon)/\varepsilon$.
4. **for** $j \leftarrow 1$ **to** ℓ
5. Select an arbitrary point c_j of C_j and an arbitrary point d_j of D_j.
6. $\mathcal{S} = \{(c_1, d_1), \ldots, (c_\ell, d_\ell)\}$.
7. $t' \leftarrow \infty$.
8. **for** $i \leftarrow 1$ to k
9. Select an arbitrary point a_i of A_i and an arbitrary point b_i of B_i.
10. $t_i \leftarrow \mathrm{ASF}((a_i, b_i), M, \mathcal{S})$.
11. **if** $t_i < t'$
12. **then** $t' \leftarrow t_i$ and $e \leftarrow (a_i, b_i)$
13. **return** $G' = (V, E \cup \{e\})$.

For completeness we also state the ASF algorithm.

Algorithm ASF$((a, b), M, \mathcal{S})$
Input: Vertex pair $(a, b) \in V^2$, distance matrix M, and a set of point pairs $\mathcal{S}$.
Output: A real value $\mathcal{D}$.
1. $\mathcal{D} \leftarrow 1$
2. **for** each point pair (c_j, d_j) in $\mathcal{S}$
3. dist $\leftarrow \min\{M[c_j, d_j],\ \ M[c_j, a] + |ab| + M[b, d_j],\ \ M[c_j, b] + |ba| + M[a, d_j]\}$
4. $\mathcal{D} \leftarrow \max\{\mathcal{D}, \text{dist}/|c_j d_j|\}$
5. **return** $\mathcal{D}$.

Theorem 12. *Given a Euclidean graph $G = (V, E)$ and a real constant $\epsilon > 0$, one can in $\mathcal{O}(nm + n^2(\log n + 1/\epsilon^{3d}))$ time, using $\mathcal{O}(n^2)$ space, compute a t'-spanner $G' = (V, E \cup \{e\})$ such that $t' \leq (2+\epsilon) \cdot t_{\mathcal{P}}$.*

Proof. The complexity of all steps of the algorithm, except step 10, is as in Lemma 8. Steps 1–7 require $\mathcal{O}(mn + n^2 \log n + n/\varepsilon^{2d})$ time. It remains to consider

step 10 of the algorithm. Note that the number of times step 10 is executed is $\mathcal{O}(n/\varepsilon^{2d})$. Procedure ASF performs $\mathcal{O}(n/\varepsilon^d)$ shortest path queries, instead of $\mathcal{O}(n^2)$, thus the total time needed by step 10 is $\mathcal{O}(\frac{n}{\varepsilon^{2d}} \cdot \frac{n}{\varepsilon^d})$. Summing up the running times gives the stated time complexity.

In Lemma 10 it was proven that the solution returned by algorithm ExpandGraph had a stretch factor that was at most a factor $(2+\varepsilon)$ worse than the stretch factor of an optimal solution. Since the modified algorithm does not compute the exact stretch factor of a candidate graph, but instead computes a $(1+\varepsilon)^2$-approximate stretch factor it is not hard to verify that the same arguments as in Lemma 10 can be applied to prove that the algorithm ExpandGraph2 returns a graph with stretch factor at most $(1+\varepsilon)^2 \cdot (2+\varepsilon) \cdot t_{\mathcal{P}}$. Setting $\varepsilon = \min\{\epsilon/10, 1\}$ concludes the proof of the theorem. □

5. A special case: G has constant stretch factor. In the special case when the stretch factor of a graph G is known to be constant there are well-known tools that we can use to decrease both the time complexity and the space complexity of the algorithms and improve the approximation factor.

Fact 2 (see [18]). Let V be a set of n points in $\mathbb{R}^d$, let $t > 1$ and $0 < \varepsilon \leq 1$ be real numbers, and let $G = (V, E)$ be a t-spanner for V. In $\mathcal{O}(m + \frac{nt^{5d}}{\varepsilon^{2d}}(\log n + (t/\varepsilon)^d))$ time, we can preprocess G into a data structure of size $\mathcal{O}(\frac{t^{3d}}{\varepsilon^{2d}} n \log tn)$ such that for any two distinct points p and q in V, a $(1+\varepsilon)$-approximation to the shortest path distance between p and q in G can be computed in time $\mathcal{O}\big((t^5/\varepsilon^2)^d\big)$.

The query structure in Fact 2 is denoted M' and is constructed by algorithm QueryStructure. We have to use a modified version of ASF, denoted ASF′, that takes the query structure M' as input instead of the matrix M. The shortest path distance queries using M in ASF are replaced in ASF′ by performing approximate shortest path distance queries using M'.

Next we state the main algorithm. Recall that the parameter t is a constant and an upper bound on the stretch factor of the input graph G. Also note that this algorithm only needs one WSPD.

Algorithm ExpandGraph3(G, t, ε)
Input: Euclidean t-spanner $G = (V, E)$ and two real constants $t > 1$ and $\varepsilon > 0$.
Output: Euclidean graph $G' = (V, E \cup \{e\})$.
1. $M' \leftarrow$QueryStructure(G, t, ε) using Fact 2.
2. $\{(A_i, B_i)\}_{i=1}^k \leftarrow$WSPD of V with respect to the separation constant $s = 8(t+1)/\varepsilon$.
3. **for** $j \leftarrow 1$ **to** k
4. Select an arbitrary point a_j of A_j and an arbitrary point b_j of B_j.
5. $\mathcal{S} = \{(a_1, b_1), \ldots, (a_k, b_k)\}$.
6. $t_{\mathcal{C}} \leftarrow \infty$.
7. **for** $i \leftarrow 1$ **to** k
8. $t_i \leftarrow$ASF′$((a_i, b_i), M', \mathcal{S})$.
9. **if** $t_i < t_{\mathcal{C}}$
10. **then** $t_{\mathcal{C}} \leftarrow t_i$ and $e_C \leftarrow (a_i, b_i)$
11. **return** $G' = (V, E \cup \{e_C\})$.

Lemma 13. *Algorithm* ExpandGraph3 *runs in* $\mathcal{O}((t^7/\varepsilon^4)^d \cdot n^2)$ *time and uses* $\mathcal{O}((t^3/\varepsilon^2)^d\, n \log(tn))$ *space.*

Proof. The time complexity of steps 1–3 is dominated by step 1; thus $\mathcal{O}(m + n(t^5/\varepsilon^2)^d(\log n+(t/\varepsilon)^d))$ time. Step 8 is executed $\mathcal{O}((t/\varepsilon)^d n)$ times, and each iteration requires $\mathcal{O}((t/\varepsilon)^d n \cdot (t^{5d}/\varepsilon^{2d}))$ time according to Facts 1 and 2. Summing up the time bounds gives the time bound stated in the algorithm.

The space bound follows since the approximate distance oracle stated in Fact 2 uses only $\mathcal{O}((t^3/\varepsilon^2)^d\, n \log tn)$ space instead of the quadratic space needed earlier. ∎

Now, we show that this algorithm computes a $(1+\varepsilon)$-approximation of the optimal solution. Note that in ExpandGraph3 the separation constant depends on both ε and t, which is the main difference compared to the previous algorithms. This allows us to improve the approximation factor.

Lemma 14. *Let $G = (V, E)$ be a Euclidean graph with constant stretch factor t and a positive real constant ε, and let $\{(A_i, B_i)\}_{i=1}^k$ be a WSPD of V with respect to $s = \frac{8(t+1)}{\varepsilon}$. For every pair (A_i, B_i) and any elements $a_1, a_2 \in A_i$ and $b_1, b_2 \in B_i$, let $G_1 = (V, E \cup \{(a_1, b_1)\})$ and $G_2 = (V, E \cup \{(a_2, b_2)\})$, and let t_1 and t_2 denote the stretch factor of G_1 and G_2, respectively. It holds that $t_1 \leq (1 + \varepsilon)t_2$.*

Proof. It suffices to prove that for every pair of points $(u, v) \in \Delta(a_2, b_2)$ there exists a path in G_1 of length at most $(1 + \varepsilon) \cdot d_{G_2}(u, v)$. Without loss of generality we may assume that the shortest path between u and v in G_2 goes from u to a_2 and to v via b_2. We have

$$\begin{aligned}
d_{G_1}(u, v) &\leq d_G(u, a_2) + d_G(a_2, a_1) + |a_1b_1| + d_G(b_1, b_2) + d_G(b_2, v) \\
&\leq d_G(u, a_2) + t|a_1a_2| + |a_1b_1| + t|b_1b_2| + d_G(b_2, v) \\
&\leq d_G(u, a_2) + \frac{4t}{s}|a_2b_2| + (1 + 4/s) \cdot |a_2b_2| + d_G(b_2, v) \\
&< d_G(u, a_2) + |a_2b_2| + d_G(b_2, v) + \frac{8t}{s}|a_2b_2| \\
&= d_{G_2}(u, v) + \frac{t\varepsilon}{t+1}|a_2b_2| \\
&< (1 + \varepsilon) \cdot d_{G_2}(u, v).
\end{aligned}$$

In the second inequality we used Lemma 6, in the fifth inequality we used the fact that $s = 8(t + 1)/\varepsilon$, and in the final step we used that $d_{G_2}(u, v) \geq |a_2b_2|$ since $(u, v) \in \Delta(a_2, b_2)$. The lemma follows. ∎

Lemma 15. *Algorithm* ExpandGraph3 *returns a graph with stretch factor at most $(1 + \varepsilon)^3 \cdot t_{\mathcal{P}}$.*

Proof. Assume that $t_{\mathcal{P}}$ is the stretch factor of an optimal solution $G \cup \{(p, q)\}$, and let G' with stretch factor $t_{\mathcal{C}}$ be the output of the above algorithm.

We will use the same notation as in the algorithm. For each i let t_i^* be the stretch factor of $G_i = G \cup \{(a_i, b_i)\}$. According to Fact 1 we have $t_i^* \leq t_i \leq (1 + \varepsilon)^2 \cdot t_i^*$ for each i.

Let (A_j, B_j) be the pair in the WSPD such that $p \in A_j$ and $q \in B_j$, or $p \in B_j$ and $q \in A_j$. From Lemma 14 it follows that $t_j^* \leq (1 + \varepsilon) \cdot t_{\mathcal{P}}$. As a result it follows that $t_{\mathcal{C}} \leq t_j \leq (1 + \varepsilon)^2 \cdot t_j^* \leq (1 + \varepsilon)^3 \cdot t_{\mathcal{P}}$. Therefore $t_{\mathcal{P}} \leq t_{\mathcal{C}} \leq (1 + \varepsilon)^3 \cdot t_{\mathcal{P}}$, which completes the lemma. ∎

The following theorem follows by setting $\varepsilon = \min\{\varphi/15, 1\}$ and combining Lemmas 13 and 15.

Theorem 16. *Let V be a set of n points in $\mathbb{R}^d$, let $t > 1$ and $\varphi > 0$ be real numbers, and let $G = (V, E)$ be a t-spanner of V. One can in $\mathcal{O}((t^7/\varphi^4)^d \cdot n^2)$ time, using $\mathcal{O}((t^3/\varphi)^d\, n \log tn)$ space, compute a t'-spanner $G' = (V, E \cup \{e\})$ such that $t' \leq (1 + \varphi) \cdot t_{\mathcal{P}}$.*

6. Concluding remarks. We considered the problem of adding an edge to a Euclidean graph such that the stretch factor of the resulting graph is minimized and gave several algorithms. Our main result is a $(2+\varepsilon)$-approximation algorithm with running time $\mathcal{O}(nm + n^2(\log n + 1/\varepsilon^{3d}))$ using $\mathcal{O}(n^2)$ space. Several problems remain open:

1. Is there an exact algorithm with running time $o(n^4)$ using linear space?
2. Can we achieve a $(1+\varepsilon)$-approximation within the same time bound as in Theorem 12?
3. A natural extension is to allow more than one edge to be added. Can we generalize our results to this case?

Acknowledgments. The authors would like to thank René van Oostrum for fruitful discussions during the early stages of this work, Mohammad Ali Abam for discussions about section 2.2, and Sergio Cabello for simplifying the algorithm in section 5. Finally, we thank the anonymous referees for many insightful comments and suggestions on how to improve the paper.

REFERENCES

[1] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[2] K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder, *Geometric spanners for wireless ad hoc networks*, IEEE Trans. Parallel Distrib. Syst., 14 (2003), pp. 408–421.

[3] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid, *Euclidean spanners: Short, thin, and lanky*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 489–498.

[4] S. Arya, D. M. Mount, and M. Smid, *Randomized and deterministic algorithms for geometric spanners of small diameter*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 703–712.

[5] P. Bose, J. Gudmundsson, and P. Morin, *Ordered theta graphs*, Comput. Geom., 28 (2004), pp. 11–18.

[6] P. B. Callahan and S. R. Kosaraju, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.

[7] A. Czumaj and A. Lingas, *Fast approximation schemes for Euclidean multi-connectivity problems*, in Proceedings of the 27th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 856–868.

[8] G. Das, P. Heffernan, and G. Narasimhan, *Optimally sparse spanners in 3-dimensional Euclidean space*, in Proceedings of the Ninth Annual ACM Symposium on Computational Geometry, 1993, pp. 53–62.

[9] G. Das and G. Narasimhan, *A fast algorithm for constructing sparse Euclidean spanners*, Internat. J. Comput. Geom. Appl., 7 (1997), pp. 297–315.

[10] G. Das, G. Narasimhan, and J. Salowe, *A new way to weigh malnourished Euclidean graphs*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 1995, pp. 215–222.

[11] M. Elkin and D. Peleg, *$(1+\epsilon,\beta)$-spanner constructions for general graphs,* SIAM J. Comput., 33 (2004), pp. 608–631.

[12] D. Eppstein, *Spanning trees and spanners*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 425–461.

[13] D. Eppstein and K. Wortman, *Minimum dilation stars*, Comput. Geom., 37 (2007), pp. 27–37.

[14] A. M. Farley, A. Proskurowski, D. Zappala, and K. J. Windisch, *Spanners and message distribution in networks*, Discrete Appl. Math., 137 (2004), pp. 159–171.

[15] A. Grüne, *Tightness of Upper Bound on t_B/t_P*, manuscript, 2005.

[16] J. Gudmundsson and C. Knauer, *Dilation and detour in geometric networks*, in Handbook on Approximation Algorithms and Metaheuristics, T. Gonzales, ed., Chapman & Hall/CRC, Boca Raton, FL, 2007, pp. 52-1–52-17.

[17] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan, *Fast greedy algorithms for constructing sparse geometric spanners*, SIAM J. Comput., 31 (2002), pp. 1479–1500.

[18] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid, *Approximate distance oracles revisited*, in Proceedings of the 13th International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 2518, Springer-Verlag, Berlin, 2002, pp. 357–368.

[19] J. Gudmundsson, G. Narasimhan, and M. Smid, *Fast pruning of geometric spanners*, in Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 3404, Springer-Verlag, Berlin, 2005, pp. 508–520.

[20] J. M. Keil, *Approximating the complete Euclidean graph*, in Proceedings of the 1st Scandinavian Workshop on Algorithmic Theory, Lecture Notes in Comput. Sci. 318, Springer-Verlag, Berlin, 1988, pp. 208–213.

[21] J. M. Keil and C. A. Gutwin, *Classes of graphs which approximate the complete Euclidean graph*, Discrete Comput. Geom., 7 (1992), pp. 13–28.

[22] S. Langerman, P. Morin, and M. A. Soss, *Computing the maximum detour and spanning ratio of planar paths, trees, and cycles*, in Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2285, Springer-Verlag, Berlin, 2002, pp. 250–261.

[23] C. Levcopoulos and A. Lingas, *There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees*, Algorithmica, 8 (1992), pp. 251–256.

[24] C. Levcopoulos, G. Narasimhan, and M. Smid, *Improved algorithms for constructing fault-tolerant spanners*, Algorithmica, 32 (2002), pp. 144–156.

[25] X.-Y. Li, *Applications of computational geometry in wireless ad hoc networks*, in Ad Hoc Wireless Networking, X.-Z. Cheng, X. Huang, and D.-Z. Du, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003, pp. 197–264.

[26] G. Narasimhan, *Geometric Spanner Networks: Open Problems*, Invited talk at the 1st Utrecht-Carleton Workshop on Computational Geometry, 2002.

[27] G. Narasimhan and M. Smid, *Approximating the stretch factor of Euclidean graphs*, SIAM J. Comput., 30 (2000), pp. 978–989.

[28] G. Narasimhan and M. Smid, *Geometric Spanner Networks*, Cambridge University Press, Cambridge, UK, 2007.

[29] G. Navarro and R. Paredes, *Practical construction of metric t-spanners*, in Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments, SIAM, Philadelphia, 2003, pp. 69–81.

[30] G. Navarro, R. Paredes, and E. Chàvez, *t-spanners as a data structure for metric space searching*, in Proceeding of the 9th International Symposium on String Processing and Information Retrieval, Lecture Notes in Comput. Sci. 2476, Springer-Verlag, Berlin, 2002, pp. 298–309.

[31] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, Philadelphia, 2000.

[32] S. Rao and W. D. Smith, *Approximating geometrical graphs via "spanners" and "banyans*," in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 540–550.

[33] J. S. Salowe, *Constructing multidimensional spanner graphs*, Internat. J. Comput. Geom. Appl., 1 (1991), pp. 99–107.

[34] M. Smid, *Closest-point problems in computational geometry*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 877–935.

[35] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 183–192.

[36] P. M. Vaidya, *A sparse graph almost as good as the complete graph on points in K dimensions*, Discrete Comput. Geom., 6 (1991), pp. 369–381.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 241–256

EQUITABLE COST ALLOCATIONS VIA PRIMAL–DUAL-TYPE ALGORITHMS*

KAMAL JAIN† AND VIJAY V. VAZIRANI‡

Abstract. Perhaps the strongest notion of truth-revealing in a cost sharing mechanism is group strategyproofness. However, matters are not so clear-cut on fairness, and many different, sometimes even conflicting, notions of fairness have been proposed which have relevance in different situations. We present a large class of group strategyproof cost sharing methods, for submodular cost functions, satisfying a wide range of fairness criteria, thereby allowing the service provider to choose a method that best satisfies the notion of fairness that is most relevant to its application. Our class includes the Dutta–Ray egalitarian method as a special case. It also includes a new cost sharing method, which we call the *opportunity egalitarian method*.

Key words. cost sharing methods, submodular cost functions, fairness in cost sharing, group strategyproof mechanism, opportunity egalitarian method

AMS subject classifications. 68Q25, 68W40, 91A12

DOI. 10.1137/060658448

1. Introduction.

Distributing the cost of a shared resource in a fair and truth-revealing manner is a central problem in cooperative game theory. Perhaps the strongest notion of truth-revealing is *group strategyproofness*, under which the dominant strategy of users is to reveal their true utility, even if they collude. However, matters are not so clear-cut on fairness—many different, sometimes conflicting, notions have been proposed which have relevance in different situations. Let us clarify that we are not necessarily postulating a service provider who is inherently "fair," but that in the long run, it is in the best interest of the service provider to subscribe to some form of fairness in choosing its cost allocations. We will assume that the cost function is submodular, a natural economies-of-scale condition. Equivalently, these results also apply to the situation of profit sharing under a convex transferable utility game; e.g., see [23]. In this paper, we will be concerned with fully budget-balanced methods; i.e., the total amount accrued from the users should be exactly equal to the cost of the shared resource.

As shown by Moulin [23], a *cross-monotone cost sharing method* for the given cost function gives rise to a group strategyproof mechanism, and for submodular cost functions, essentially the converse holds as well. Informally, a cost sharing method is cross-monotone, also called population monotone, if the cost share of any user can only decrease if a superset is being served.

Two well-known cross-monotone cost sharing methods for submodular cost functions are the Shapley value and the egalitarian method of Dutta and Ray [6] (the former requires that the cost function be *nondecreasing* as well; i.e., the cost of serving a set should not be larger than the cost of serving any of its supersets). Both these methods have been extensively studied; for the latter, see [12, 11, 13, 16, 21, 3, 2, 5, 7].

*Received by the editors April 28, 2006; accepted for publication (in revised form) November 21, 2007; published electronically March 28, 2008.
http://www.siam.org/journals/sicomp/38-1/65844.html
†Microsoft, One Microsoft Way, Redmond, WA 98052 (kamalj@microsoft.com).
‡Corresponding author. College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280 (vazirani@cc.gatech.edu). This author's research was supported by NSF grants CCR-0220343 and CCR-0515186.

These two methods satisfy different fairness criteria. The Shapley value charges higher amounts from users who are more expensive to serve. The egalitarian method attempts to charge equal amounts from all users subject to the coalition participation constraint that the method lies in the core of the game; i.e., no subset is charged more than its stand-alone cost (thereby precluding the possibility of its seceding).

Consider the following situation in which neither of these criteria appears to be fair. Suppose that two users, Fred Smith and Gill Bates, are proximally located so that they are equally expensive to serve. For concreteness, assume that the service provider is transmitting valuable financial data and needs to recover a cost of $2000, regardless of whether it serves one or both users. Fred Smith and Gill Bates derive widely different utilities on receiving this data—the former is a man of modest means, and the latter is a multimillionaire. Hence, they are also willing to pay different amounts for this data. In this situation, the Shapley value as well as the egalitarian method will assign cost shares of $1000 each for the service, an amount that is not acceptable to Fred Smith. However, Gill Bates considers this data useful for wisely managing his vast acquisitions and ends up paying the entire $2000 for the service. If, instead, the cost sharing method were to take into consideration the relative paying powers of the two users and charge differentially, it may be able to find an outcome that Pareto dominates the previous outcome. For instance, if it charges Fred Smith $100 and Gill Bates $1900, both accept the service and both are better off. In addition, the service provider is also better off since it has a larger and more satisfied pool of customers.

This form of differential pricing, sometimes also called price discrimination, is widely resorted to and is in fact crucial to the survival of many industries [25, 27, 28, 29]. For instance, it provides mechanisms to the airline industry to charge higher fares from business travelers, who can afford to pay higher fares, than from casual travelers. Clearly, the fate of the airline industry, which has been on the brink of bankruptcy numerous times, would be dire without such a mechanism. Another common example is differential subscription rates for journals charged from students, professionals, and institutions.

Can the service provider resort to differential pricing and still ensure that the mechanism is strategyproof or, better, group strategyproof? In this paper, we provide a formal setting to accomplish this. We present a large class of group strategyproof mechanisms for submodular cost functions, satisfying a wide range of fairness criteria—hence the name "equitable." Our class includes the Dutta–Ray egalitarian method as a special case. It also includes a new cost sharing method, which we call the *opportunity egalitarian method*. Assuming that individual utilities are drawn from probability distributions which are known to the service provider, this method finds cost shares that attempt to equalize the users' probabilities of accepting the service, subject to core constraints. The above-stated examples of differential cost shares can be viewed as approximations of the opportunity egalitarian method.

Each equitable cost sharing method is parameterized by n *equalizing functions* which encode the fairness criterion chosen. The method ensures that w.r.t. the chosen criterion, the cost shares satisfy min-max as well as max-min fairness; i.e., no one underpays, and no one overpays. Thus, in the case of the egalitarian method (opportunity egalitarian method), among all cost allocations in the core, the chosen allocation minimizes the maximum cost shares (probability of accepting service) as well as maximizes the minimum cost shares (for fairness). Precise definitions appear in sections 3, 5.1, and 6. Max-min fairness has been used in the networking community for tackling issues of bandwidth allocation [4, 15] and has also been algorithmically studied in the context of routing in networks [22, 1, 20]. Approximate versions of this

notion have also been studied [9].

Our algorithms are inspired by the primal-dual schema from the field of approximation algorithms (see [30]). In the latter setting, it is natural to view the dual program as "paying" for the primal, and the algorithm as progressive bidding to get access to a shared resource (this viewpoint is particularly clear in the primal-dual algorithm presented in [18]). The equalizing functions determine the rates at which individual users increase their "bids." Each iteration of our algorithm runs in polynomial time. We utilize recently discovered polynomial time algorithms for the minimization of a submodular function [14, 26]. The precise number of iterations depends upon the accuracy needed.

We have recently found some rather unexpected applications of the results of this paper. The cross-monotonic cost sharing method developed in this paper for submodular cost functions has been used for proving competition monotonicity for submodular utility allocation markets in [19]. Our max-min and min-max fairness results for these cost sharing methods have also been used in [19] for establishing that the equilibrium allocations for submodular utility allocation markets are max-min fair.

It is interesting to note that independently, though somewhat preceding our work, Hokari [11] generalized Dutta–Ray solutions to give a class of cost sharing methods that turns out to be identical to ours. He calls his methods *monotone path cost allocations*. Hokari's formalization and point of view are quite different from ours—the definitions of the cost sharing methods are strikingly different,[1] and so are the algorithms for computing them (Hokari does not address issues of algorithmic efficiency). We believe that this class deserves further study—in the past, notions derived from diverse considerations have turned out to be particularly robust and canonical.

Mutuswami [24] proved the following interesting fact about the Dutta–Ray egalitarian method. If the utilities of individual users are independently and identically distributed (i.i.d.) (and the distribution satisfies the monotone hazard rate condition; see section 7 for a formal definition), then for every set S of users, the egalitarian method maximizes the probability that all members of S accept the service, among all cost sharing methods in the core. We generalize this result by removing the restriction that all utilities come from the *same* distribution. We show that for each choice of distributions from which the utilities are picked (provided they satisfy the strict monotone hazard rate condition), there is an equitable cost sharing method that maximizes, for every set $S \subseteq U$, the probability that all members of S accept the service, among all cost sharing methods for S in the core.

2. Basic definitions. Let $U = \{1, \ldots, n\}$ denote the set of users and cost : $2^U \to \mathbf{R}^+$ denote the function that gives the cost of serving any subset of the users. We will assume that this function is *submodular*; i.e.,

$$\forall S, T \subseteq U, \quad \text{cost}(S) + \text{cost}(T) \geq \text{cost}(S \cup T) + \text{cost}(S \cap T).$$

Following is an equivalent definition that makes it clear that such cost functions satisfy a natural economies-of-scale condition. The marginal cost of including a new user can only be smaller if a superset is being served:

$$\forall S \subset T \subset U,\ i \notin T, \quad \text{cost}(S + i) - \text{cost}(S) \geq \text{cost}(T + i) - \text{cost}(T).$$

[1] See section 8 for an implication that is easier to derive from our formulation.

We consider the following game. The service provider picks a *mechanism* for deciding the set $S \subseteq U$ of users that get the service and the individual cost shares of users in S so as to retrieve the cost of serving them, $\text{cost}(S)$. It obtains from the users their utilities for receiving the service. Thus each user's strategy is simply the utility he reports. The service provider is not allowed to charge a user more than his reported utility (otherwise, the user will refuse the service).

We will say that the service provider's mechanism is *strategyproof* if the dominant strategy of each user is to report his true utility. It is *group strategyproof* if the above holds despite collusions among users. Let us make this precise. Consider a coalition C of users. Let $\boldsymbol{u}$ and $\boldsymbol{u}'$ be two vectors of bids (we will think of the former as agents' true values and $\boldsymbol{u}'$ as strategically chosen bids). Assume that $\boldsymbol{u}_j = \boldsymbol{u}'_j$ for all $j \notin C$. Let $(\boldsymbol{q}, \boldsymbol{x})$ and $(\boldsymbol{q}', \boldsymbol{x}')$ denote the users served and costs at $\boldsymbol{u}$ and $\boldsymbol{u}'$, respectively. Now, group strategyproofness requires that if the inequality

$$\boldsymbol{u}'_i \boldsymbol{q}_i - \boldsymbol{x}_i \geq \boldsymbol{u}'_i \boldsymbol{q}'_i - \boldsymbol{x}'_i$$

holds for all $i \in C$, then it must hold with equality for all $i \in C$ as well; i.e., if no member of C is made worse off by misreporting of their utility values, then no member of C is made better off either. Moulin [23] showed that it is sufficient for the service provider to pick a cross-monotone cost sharing method in order to obtain a group strategyproof mechanism. His procedure for obtaining such a mechanism from the method is recapitulated below.

A *cost sharing method*, ξ, specifies how to distribute, for any set $S \subseteq U$, $\text{cost}(S)$ among the users in S. It satisfies the following:

1. Users will not be paid for receiving service; i.e.,

$$\forall S \subseteq U,\ i \in S, \quad \xi(S, i) \geq 0.$$

2. Budget balance

$$\forall S \subseteq U, \quad \sum_{i \in S} \xi(S, i) = \text{cost}(S).$$

3. Users not being served will not be charged; i.e.,

$$\forall S \subseteq U,\ i \notin S, \quad \xi(S, i) = 0.$$

For any set $S \subseteq U$, the slice of ξ at S, i.e., $\xi(S, \cdot)$, will be denoted by ξ^S. Thus, $\xi^S : S \to \mathbf{R}^+$ specifies the cost shares of users in S, assuming that S is the set being served. We will say that ξ is *cross-monotonic* if it satisfies the following economies of scale condition:

$$\forall S \subset T \subseteq U,\ \forall i \in S, \quad \xi^S(i) \geq \xi^T(i);$$

i.e., the cost share of a user can only be smaller if a superset is being served.

Let ξ be a cross-monotone cost sharing method. Consider the following mechanism. Initialize $S \leftarrow U$. If for each user $i \in S$, his cost share $\xi(S, i)$ is at most his utility, HALT. Else, drop users whose utilities are smaller than their cost shares, update S, and repeat.

THEOREM 1 (Moulin [23]). *If ξ is a cross-monotone cost sharing method, then the mechanism specified above is group strategyproof.*

A cost allocation for set $S \subseteq U$, $\alpha : S \to \mathbf{R}^+$ is said to satisfy the *coalition participation constraint* if it satisfies the following:

1. Budget balance, i.e.,

$$\sum_{i \in S} \alpha(i) = \text{cost}(S).$$

2. No subset $S' \subset S$ is charged more than the stand-alone cost of serving S'; i.e.,

$$\forall S' \subset S, \quad \sum_{i \in S'} \alpha(i) \leq \text{cost}(S').$$

The core is usually defined as the set of all cost allocations for U (i.e., the grand coalition) satisfying the coalition participation constraint. In this paper, we will define the *core* to consist of all cost allocations satisfying the coalition participation constraint for all sets $S \subseteq U$, rather than only U. We will say that *a cost sharing method ξ is in the core* if for all $S \subseteq U$, ξ^S is in the core.

3. Equitable cost sharing methods. Let $\boldsymbol{q}$ and $\boldsymbol{r}$ be n-dimensional vectors with nonnegative coordinates. We will denote by $\boldsymbol{q}_{INC}$ the vector obtained by sorting the components of $\boldsymbol{q}$ in increasing order. Thus $\boldsymbol{q}_{INC}(i) \leq \boldsymbol{q}_{INC}(i+1)$ for $1 \leq i \leq n-1$. Define a partial order as follows: say that $\boldsymbol{q}$ *max-min dominates* $\boldsymbol{r}$ if $\boldsymbol{q}_{INC}$ is lexicographically larger than $\boldsymbol{r}_{INC}$, i.e., if there is an i such that $\boldsymbol{q}_{INC}(i) > \boldsymbol{r}_{INC}(i)$ and $\boldsymbol{q}_{INC}(j) = \boldsymbol{r}_{INC}(j)$ for $j < i$. Clearly, $\boldsymbol{q}_{INC} = \boldsymbol{r}_{INC}$ may hold even though $\boldsymbol{q} \neq \boldsymbol{r}$.

An equitable cost sharing method is parameterized by n strictly increasing, continuous, and unbounded functions from $\mathbf{R}^+$ to $\mathbf{R}^+$, $f_1, \ldots, f_n$ satisfying further that $f_i(0) = 0$. These will be called *equalizing functions*. The *equitable cost sharing method* corresponding to this set of equalizing functions, say ξ, is defined as follows. We will specify ξ^S for each set $S \subseteq U$. Without loss of generality assume that S consists of users $1, \ldots, s$. Let α be a cost allocation for S that lies in the core. Let $\boldsymbol{t}(\alpha)$ denote the s-dimensional vector whose ith component is $f_i^{-1}(\alpha(i))$. We will show in Theorem 6 that there is a unique cost allocation for set S in the core, say β, such that $\boldsymbol{t}(\beta)$ max-min dominates $\boldsymbol{t}(\alpha)$ for all other allocations, α, for S in the core. We will define $\xi^S = \beta$.

If all n equalizing functions are picked to be the identity function, the resulting cost sharing method will be the egalitarian method of Dutta and Ray (strictly speaking, this is not the way they defined their method; see section 5.2). The egalitarian method maximizes the minimum cost shares, i.e., ensures that no one underpays, subject to core constraints—in that sense, it tries to make the cost shares of individual users as equal as possible. Our generalization optimizes the max-min objective function relative to the equalizing functions $f_1, \ldots, f_n$, which encode the particular fairness criterion chosen. As shown in Theorem 10, equitable methods (including the egalitarian method) optimize the min-max objective as well, ensuring that no one overpays.

4. The algorithm. We now present a primal–dual-type algorithm for obtaining ξ^S for any set S. First we give some definitions. Let $x : S \to \mathbf{R}^+$ be a function assigning costs to users in S. Set $A \subseteq S$ will be said to be *tight* if $\sum_{i \in A} x_i = \text{cost}(A)$. It will be said to be *overtight* if $\sum_{i \in A} x_i > \text{cost}(A)$. We will say that x is *feasible* if no subset of S is overtight. Note that we have not imposed the condition that $\sum_{i \in S} x_i = \text{cost}(S)$. The algorithm will utilize the following properties.

LEMMA 2. *Let* cost *be submodular and x be feasible for S. If $A, B \subseteq S$ are both tight, then $A \cup B$ is also tight.*

Proof. By submodularity,

$$\text{cost}(A \cup B) \leq \text{cost}(A) + \text{cost}(B) - \text{cost}(A \cap B).$$

Since x is feasible for S,

$$\sum_{i \in (A \cap B)} x_i \leq \text{cost}(A \cap B).$$

Combining this with the fact that A and B are both tight, we get

$$\text{cost}(A \cup B) \leq \sum_{i \in A} x_i + \sum_{i \in B} x_i - \sum_{i \in (A \cap B)} x_i = \sum_{i \in (A \cup B)} x_i.$$

Therefore, $A \cup B$ must also be tight. □

COROLLARY 3. *If* cost *is submodular and x is feasible for S, then there is a unique maximal tight set. It is given by $\{i \in S \mid i$ belongs to some tight set$\}$.*

For each set $S \subseteq U$, the algorithm below computes a cost allocation for S.

ALGORITHM 1. We will associate a notion of time with our algorithm. Initially, the time t is set to zero. As the algorithm proceeds, we raise cost shares of users in S in proportion to their respective functions f_i; thus, at time t, the cost share of a user i is $f_i(t)$. Whenever a set $A \subseteq S$ goes tight, the cost shares of all users in A are frozen at the current value. The cost shares of the remaining users keep increasing with time as before. The algorithm terminates when the cost shares of all users are frozen. For each user $i \in S$, define $\xi^S(i)$ to be i's cost share at termination.

By Corollary 3, at any time, there is a unique maximal tight set. This tight set can be found in polynomial time using a submodular function minimization algorithm [14, 26] as follows. The difference of a submodular function and a modular function is a submodular function. Hence, the following function, defined on 2^S, is submodular:

$$\text{cost}'(A) = \text{cost}(A) - \sum_{i \in A} f_i(t_i)$$

for $A \subseteq S$, where t_i's are fixed for each element $i \in S$. For an element i that is already frozen, fix t_i to be the time at which i froze. For an element i that is not yet frozen, let $t_i = t$. Now we will do a binary search on t to find the smallest time at which there is a set $A \subseteq S$ such that $\text{cost}'(A)$ is a small negative number. At that value of time, the set whose cost$'$ is minimum will clearly be the maximal set to go tight next.

REMARK 4. *Observe that $\boldsymbol{t}(\xi^S)$ is precisely the vector of times at which individual elements went tight ($\boldsymbol{t}$ is defined at the beginning of section 3).*

LEMMA 5. *For each set $S \subseteq U$, the cost allocation given by ξ^S lies in the core.*

Proof. Clearly, ξ^S is feasible for S and no subset of S is overtight. Furthermore, by Corollary 3, at termination, set S must be tight. □

THEOREM 6. *For any set $S \subseteq U$, the cost allocation, ξ^S, found by Algorithm 1 is such that $\boldsymbol{t}(\xi^S)$ max-min dominates $\boldsymbol{t}(\alpha)$ for all other cost allocations, α, for S in the core.*

Proof. Let α be an allocation for set S that lies in the core. Suppose that $\boldsymbol{t}(\xi^S)$ does not max-min dominate $\boldsymbol{t}(\alpha)$. Then we will show that ξ^S and α are in fact the same allocation, hence proving the theorem.

Let $A_1 \subset A_2 \subset \cdots \subset S$ be the sequence of maximal sets that go tight when the algorithm is run on set S. We will show by induction on i that all users in A_i must have the same cost allocation in α and ξ^S. Observe that all elements in $A_i - A_{i-1}$ go tight at the same time, and hence the components corresponding to them in $\boldsymbol{t}(\xi^S)$ are identical.

Clearly,

$$\sum_{i \in A_1} \alpha(i) \leq \sum_{i \in A_1} \xi^S(i) = \text{cost}(A_1).$$

If this inequality is strict, there exists $i \in A_1$ such that $\alpha(i) < \xi^S(i)$. Since users $i \in A_1$ give rise to the smallest entries of $\boldsymbol{t}_{INC}(\xi^S)$, $\boldsymbol{t}(\xi^S)$ max-min dominates $\boldsymbol{t}(\alpha)$, leading to a contradiction. Therefore, this inequality must hold with equality. If for some user $i \in A_1$, $\alpha(i) > \xi^S(i)$, then for some other user $j \in A_1$, $\alpha(j) < \xi^S(j)$, and again $\boldsymbol{t}(\xi^S)$ max-min dominates $\boldsymbol{t}(\alpha)$, leading to a contradiction. Therefore,

$$\forall i \in A_1, \quad \alpha(i) = \xi^S(i).$$

The idea for the induction step is the same as for the basis. □

REMARK 7. *Observe that the precise manner of "dual" increase in the algorithm was essential for proving Theorem* 6.

For a definition of equitable cost sharing method, see the beginning of section 3.

COROLLARY 8. *The cost sharing method, ξ, found by Algorithm* 1 *is the equitable cost sharing method for equalizing functions $f_1, \ldots, f_n$.*

THEOREM 9. *The cost sharing method ξ is cross-monotonic.*

Proof. Suppose that $S \subset T \subseteq U$. Let us call the two runs of the algorithm S-run and T-run, respectively. It suffices to prove that at each time t, the tight set in the T-run is a superset of the tight set in the S-run, because then each user $i \in S$ can be frozen only at an earlier time in the T-run and hence can have only a smaller cost share under the T-run.

Consider time t, and let A and B be the tight sets in the S- and T-runs, respectively. Let x_i denote the cost share of $i \in S$ at time t under the S-run, and let x'_i denote the cost share of $i \in T$ at time t under the T-run.

By submodularity,

$$\text{cost}(A \cup B) + \text{cost}(A \cap B) \leq \text{cost}(A) + \text{cost}(B).$$

Since x is feasible for S, we have

$$\sum_{i \in (A \cap B)} x_i \leq \text{cost}(A \cap B).$$

Using the additional fact that A and B are tight in the S- and T-runs, respectively, we get

$$\text{cost}(A \cup B) + \sum_{i \in (A \cap B)} x_i \leq \sum_{i \in A} x_i + \sum_{i \in B} x'_i.$$

Therefore,

$$\text{cost}(A \cup B) \leq \sum_{i \in (A - B)} x_i + \sum_{i \in B} x'_i.$$

Observe that at time t, the users in $A - B$ are frozen in the S-run but not in the T-run. Therefore for each $i \in A - B$, $x_i \leq x'_i$. Therefore,

$$\text{cost}(A \cup B) \leq \sum_{i \in (A \cup B)} x'_i.$$

Therefore, $A \cup B$ is tight at time t in the T-run. Hence $A \subseteq B$, and the theorem follows. □

5. Alternative characterizations of equitable methods. In this section, we will give two alternative characterizations of equitable methods. The proofs of both characterizations appeal to Algorithm 1, and we do not know of direct proofs. These alternative characterizations are as basic as the definition of equitable methods and could have been taken as alternative definitions of these methods.

5.1. Min-max domination. Let $\boldsymbol{q}$ and $\boldsymbol{r}$ be n-dimensional vectors with nonnegative coordinates. We will denote by $\boldsymbol{q}_{DEC}$ the vector obtained by sorting the components of $\boldsymbol{q}$ in decreasing order and will say that $\boldsymbol{q}$ *min-max dominates* $\boldsymbol{r}$ if $\boldsymbol{q}_{DEC}$ is lexicographically smaller than $\boldsymbol{r}_{DEC}$, i.e., if there is an i such that $\boldsymbol{q}_{DEC}(i) < \boldsymbol{r}_{DEC}(i)$ and $\boldsymbol{q}_{DEC}(j) = \boldsymbol{r}_{DEC}(j)$ for $j < i$.

THEOREM 10. *For any set $S \subseteq U$, the cost allocation, ξ^S, found by Algorithm 1 is such that $\boldsymbol{t}(\xi^S)$ min-max dominates $\boldsymbol{t}(\alpha)$ for all other cost allocations, α, for S in the core.*

Proof. The proof is similar to that of Theorem 6. Let α be an allocation for set S that lies in the core. Suppose that $\boldsymbol{t}(\xi^S)$ does not min-max dominate $\boldsymbol{t}(\alpha)$. Then we will show that ξ^S and α are in fact the same allocation, hence proving the theorem.

Let $S = A_1 \supset A_2 \supset \cdots \supset \emptyset$ be the *reverse* order in which sets go tight when the algorithm is run on set S. We will show by induction on i that all users in $A_i - A_{i+1}$ must have the same cost allocation in α and ξ^S. Observe that all elements in $A_i - A_{i+1}$ go tight at the same time, and hence the components corresponding to them in $\boldsymbol{t}(\xi^S)$ are identical.

Clearly,

$$\sum_{i \in A_2} \alpha(i) \leq \sum_{i \in A_2} \xi^S(i) = \text{cost}(A_2).$$

Therefore,

$$\sum_{i \in A_1 - A_2} \alpha(i) \geq \sum_{i \in A_1 - A_2} \xi^S(i).$$

If this inequality is strict, there exists $i \in A_1 - A_2$ such that $\alpha(i) > \xi^S(i)$. Since users $i \in A_1 - A_2$ give rise to the largest entries in $\boldsymbol{t}_{INC}(\xi^S)$, $\boldsymbol{t}(\xi^S)$ min-max dominates $\boldsymbol{t}(\alpha)$, leading to a contradiction. Therefore, this inequality must hold with equality. If for some user $i \in A_1 - A_2$, $\alpha(i) < \xi^S(i)$, then for some other user $j \in A_1 - A_2$, $\alpha(j) > \xi^S(j)$, and again $\boldsymbol{t}(\xi^S)$ min-max dominates $\boldsymbol{t}(\alpha)$, leading to a contradiction. Therefore,

$$\forall i \in A_1 - A_2, \quad \alpha(i) = \xi^S(i).$$

The idea for the induction step is the same as for the basis. □

5.2. *L*-domination. The next characterization is along the lines of Dutta and Ray's definition of the egalitarian method which uses the notion of Lorentz orderings. Following is their definition when applied to the case of a submodular cost function (for the complete definition, which involves a recursive construct, see [6]).

Let $\alpha_1 \leq \cdots \leq \alpha_n$ and $\beta_1 \leq \cdots \leq \beta_n$ be such that $\alpha_1 + \cdots + \alpha_n = \beta_1 + \cdots + \beta_n$. We will say that $(\alpha_1, \ldots, \alpha_n)$ *Lorentz dominates* $(\beta_1, \ldots, \beta_n)$ if for $1 \leq k \leq n$, we have

$$\alpha_1 + \cdots + \alpha_k \geq \beta_1 + \cdots + \beta_k,$$

and the inequality is strict for at least one k.

Dutta and Ray [6] showed that if the underlying cost function is submodular, there is a core cost allocation for S, say $\boldsymbol{q}$, such that $\boldsymbol{q}_{INC}$ Lorentz dominates $\boldsymbol{r}_{INC}$ for all other cost allocations, $\boldsymbol{r}$, for S in the core. $\boldsymbol{q}$ is the egalitarian cost allocation for S.

Note that the definition of Lorentz ordering may compare cost shares of different users, since $\boldsymbol{q}_{INC}(i)$ and $\boldsymbol{r}_{INC}(i)$ may be cost shares of different users. In our setting, the equalizing functions for different users may be very different, thus making such comparisons meaningless. We give below an ordering that takes this into consideration. This ordering is not a generalization of Lorentz ordering—if all f_i's are the identity function, it does not necessarily reduce to the Lorentz ordering, but it does preserve the property established in Lemma 11.

Let V be the set of all nonnegative s-dimensional vectors $\boldsymbol{q}$ such that $f_1(\boldsymbol{q}(1)) + \cdots + f_s(\boldsymbol{q}(s)) = \text{cost}(S)$. Let $V_c \subseteq V$ be the set of vectors $\boldsymbol{q}$ such that $(f_1(\boldsymbol{q}(1)), \ldots, f_s(\boldsymbol{q}(s)))$ forms a cost allocation for S lying in the core. For $\boldsymbol{q}, \boldsymbol{r} \in V$, say that $\boldsymbol{q}$ *L-dominates* $\boldsymbol{r}$ if there exists a permutation π such that

1. $\boldsymbol{q}(\pi(1)) \leq \cdots \leq \boldsymbol{q}(\pi(s))$,
2. for $1 \leq i \leq s$, we have

$$\sum_{k=1}^{i} f_{\pi(k)}(\boldsymbol{q}(\pi(k))) \geq \sum_{k=1}^{i} f_{\pi(k)}(\boldsymbol{r}(\pi(k))),$$

and the inequality is strict for at least one i.

It is easy to construct examples showing that the relation defined above is not necessarily transitive. However, it is *acyclic* in the following sense: if $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k \in V$, then it cannot be the case that $\boldsymbol{q}_i$ L-dominates $\boldsymbol{q}_{i+1}$, for $1 \leq i \leq k$, and $\boldsymbol{q}_k$ L-dominates $\boldsymbol{q}_1$. The definition of L-dominates requires that vectors $\boldsymbol{q}$ and $\boldsymbol{r}$ be ordered according to the *same* permutation—it is for this reason that this notion does not generalize the notion of Lorentz ordering.

LEMMA 11. *Let $\boldsymbol{q}, \boldsymbol{r} \in V$, and suppose that $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$. Then $\boldsymbol{q}_{INC}$ is lexicographically larger than $\boldsymbol{r}_{INC}$.*

Proof. Without loss of generality, assume that the permutation π showing that $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$ is the identity permutation. Let i be the smallest index, $1 \leq i \leq s$, such that $\sum_{k=1}^{i} f_k(\boldsymbol{q}(k)) > \sum_{k=1}^{i} f_k(\boldsymbol{r}(k))$. Clearly, $\boldsymbol{q}(k) = \boldsymbol{r}(k)$, for $k < i$, and $\boldsymbol{q}(i) > \boldsymbol{r}(i)$. Therefore, $\boldsymbol{r}(1) \leq \cdots \leq \boldsymbol{r}(i)$. By assumption, $\boldsymbol{q} = \boldsymbol{q}_{INC}$. On the other hand, the first i components of $\boldsymbol{r}_{INC}$ can be only smaller than the corresponding components of $\boldsymbol{r}$. Hence $\boldsymbol{q}_{INC}$ is lexicographically larger than $\boldsymbol{r}_{INC}$. □

Since lexicographic domination is acyclic, we get the following corollary.

COROLLARY 12. *The relation of L-domination is acyclic.*

We will need the following technical lemma for the main result.

LEMMA 13. *Let $M, a_1, \ldots, a_l, m, b_1, \ldots, b_l$ be nonnegative real numbers satisfying*

- $M \geq m$,
- $M + a_1 + \cdots + a_l \geq m + b_1 + \cdots + b_l$.

Then, there is a permutation π over $[1, \ldots, l]$ such that for $1 \leq i \leq l$,

$$M + \sum_{k=1}^{i} a_{\pi(k)} \geq m + \sum_{k=1}^{i} b_{\pi(k)}.$$

Proof. Let us show how to construct π. Let us first determine $\pi(1)$. We claim that there exists k, $1 \leq k \leq l$, such that

$$M + a_k \geq m + b_k.$$

Suppose not. Then, for $1 \leq k \leq l$,

$$M + a_k < m + b_k.$$

Adding these l inequalities, we get

$$(l-1)M + (M + a_1 + \cdots + a_l) < (l-1)m + (m + b_1 + \cdots + b_l).$$

But this contradicts the assumptions made on these numbers, proving existence of k satisfying the inequality above. Set $\pi(1) = k$. The idea for constructing the rest of π is the same. □

THEOREM 14. *Let $\boldsymbol{q} \in V_c$ correspond to the equitable cost allocation χ^S, and let $\boldsymbol{r} \in V_c$ be any other vector. Then $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$.*

Proof. We need to construct permutation π that shows that $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$. Let $A_1 \subset A_2 \subset \cdots \subset S$ be the sequence in which sets go tight when Algorithm 1 computes cost shares for S. In the simple case that each of $A_{i+1} - A_i$ is a singleton, let π be the order in which elements go tight. Then, for $1 \leq i \leq s$,

$$\sum_{k=1}^{i} f_{\pi(k)}(\boldsymbol{q}(\pi(k))) = \text{cost}(A_i) \geq \sum_{k=1}^{i} f_{\pi(k)}(\boldsymbol{r}(\pi(k))).$$

The inequality holds because the cost allocation corresponding to $\boldsymbol{r}$ lies in the core. Since $\boldsymbol{q} \neq \boldsymbol{r}$, one of these inequalities must be strict, thereby showing that $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$.

In the general case, we will order elements of A_1 first, the elements of $A_2 - A_1$ next, and so on. This ensures that

$$\sum_{k \in A_i} f_k(\boldsymbol{q}(k)) = \text{cost}(A_i) \geq \sum_{k \in A_i} f_k(\boldsymbol{r}(k)).$$

Next, let us specify the precise order given to elements of $A_{i+1} - A_i = \{j_1, \ldots, j_l\}$. For this, we will use Lemma 13 with

$$M = \sum_{k \in A_i} f_k(\boldsymbol{q}(k)) = \text{cost}(A_i), \quad m = \sum_{k \in A_i} f_k(\boldsymbol{r}(k))$$

and for $1 \leq k \leq l$,

$$a_k = f_{j_k}(\boldsymbol{q}(j_k)) \quad \text{and} \quad b_k = f_{j_k}(\boldsymbol{r}(j_k)). \qquad \square$$

Observe that the proof given above uses the fact that the functions f_j are *strictly* increasing.

6. The opportunity egalitarian method. For each user $i \in U$, let $G_i : \mathbf{R}^+ \to [0,1]$ be the cumulative probability distribution function from which i's utility is drawn; assume that these distributions are independent. Assume that G_i is monotonically increasing.

Let α be any cost allocation for $S \subseteq U$ that lies in the core. User i will accept the service only if his utility turns out to be at least $\alpha(i)$, his cost share. The probability of this event is $1 - G_i(\alpha(i))$. Let $\boldsymbol{p}(\alpha)$ denote the vector whose ith component is this probability. We will say that a cost sharing method ξ is the *opportunity egalitarian method* for cumulative distribution functions $G_1, \ldots, G_n$ if for each set $S \subseteq U$, ξ^S is the cost allocation in the core such that

1. $\boldsymbol{p}(\xi^S)$ max-min dominates $\boldsymbol{p}(\alpha)$ and
2. $\boldsymbol{p}(\xi^S)$ min-max dominates $\boldsymbol{p}(\alpha)$

for all other cost allocations, α, for S in the core. Theorem 15 shows that there is a unique such cost allocation. Observe that ξ is attempting to equalize the probabilities of users receiving the service, subject to core constraints. The two characterizations, min-max and max-min, ensure that both extremes are avoided.

Let $f_i : [0,1] \to \mathbf{R}^+$ denote the inverse of G_i. Let ξ be the equitable cost sharing method for functions $f_1, \ldots, f_n$.

THEOREM 15. *ξ is the opportunity egalitarian method for probability density functions $G_1, \ldots, G_n$.*

Proof. Consider any set $S \subseteq U$, and let α be a cost allocation for S lying in the core. Clearly, for $i \in S$,

$$\boldsymbol{t}(\alpha)(i) = f_i^{-1}(\alpha(i)) = G_i(\alpha(i)) = 1 - \boldsymbol{p}(\alpha)(i);$$

i.e., this is the probability that user i does not accept service under cost allocation α. Since ξ is the equitable method for $f_1, \ldots, f_n$, $\boldsymbol{t}(\xi^S)$ max-min dominates $\boldsymbol{t}(\alpha)$ for all cost allocations, α, for S in the core. Equivalently, $\boldsymbol{p}(\xi^S)$ min-max dominates $\boldsymbol{p}(\alpha)$ for all core cost allocations for S. Its uniqueness follows from Theorem 6. By Theorem 10, $\boldsymbol{p}(\xi^S)$ max-min dominates $\boldsymbol{p}(\alpha)$ for all other cost allocations, α, for S in the core. Hence, ξ is the opportunity egalitarian method for cumulative distribution functions $G_1, \ldots, G_n$. □

By Theorem 9, the opportunity egalitarian method is cross-monotone.

7. Maximizing acceptance probability. As in the last section, for each user $i \in U$, let $G_i : \mathbf{R}^+ \to [0,1]$ be the cumulative probability distribution function from which i's utility is drawn; assume that these distributions are independent. Let g_i be the corresponding probability density function, i.e., $g_i(x) = \frac{\partial G_i(x)}{\partial x}$. Assume that $g_i(0) = 0$. We further assume that G_i satisfies the *strict monotone hazard rate condition*, i.e.,

$$\frac{\partial}{\partial x}\left[\frac{g_i(x)}{1 - G_i(x)}\right] > 0.$$

If g_i represents the failure probability of a component as a function of time, the above condition says that the failure rate, conditioned on the component still being intact, strictly increases with age. The monotone hazard rate condition is satisfied by most standard probability distributions and is a standard assumption in the Bayesian mechanism design literature [8].

Let $\lambda_i : \mathbf{R}^+ \to \mathbf{R}^+$ be the function

$$\lambda_i(x) = \frac{g_i(x)}{1 - G_i(x)}.$$

Observe that under the assumption of strict monotone rate hazard rate condition, λ_i is an invertible function. Let f_i be the inverse of this function, and let χ be the equitable method corresponding to equalizing functions $f_1, f_2, \ldots, f_n$.

Let ξ be any cost sharing method in the core. Let $P(\xi^S)$ denote the probability that all users in S accept service when it is offered at cost shares ξ^S. Clearly,

$$P(\xi^S) = \prod_{i \in S} [1 - G_i(\xi_i^S)].$$

THEOREM 16. *Let χ be the equitable cost sharing method defined above, and ξ be any cost sharing method in the core. Then, for each set of users $S \subseteq U$, $P(\chi^S) \geq P(\xi^S)$.*

This is a generalization of Mutuswami's result [24], which deals with the case that G_i's are i.i.d. (on the other hand, he requires only monotone hazard rate condition—not necessarily *strict*). In this case, χ is the Dutta–Ray egalitarian method. Mutuswami's proof uses the original definition of Dutta and Ray, in terms of Lorentz orderings, and a theorem of Hardy, Littlewood, and Polya [10], giving a condition that is equivalent to Lorentz domination. To prove the generalization, we prove a characterization for L-domination, in Lemma 18, in the style of the Hardy–Littlewood–Polya theorem. The proof of Theorem 16 is given below.

By Theorem 16, χ *simultaneously* maximizes the probability of all users accepting service, for each set S of users, among all cost sharing methods in the core. One may be led to believe that if the mechanism of Theorem 1 is run with χ, then the expected size of the set served is maximized, over all cost sharing methods in the core. However, this is not true, as shown in the example below.

Example. Let $U = \{a, b\}$ and the cost function be $\text{cost}(a, b) = 10$, $\text{cost}(a) = 8$, $\text{cost}(b) = 6$, $\text{cost}(\emptyset) = 0$. Suppose the utilities of a and b are picked from the uniform distribution over the interval $[0, 20]$. In this case, χ will be the egalitarian method:

$$\chi(\{a, b\}, a) = \chi(\{a, b\}, b) = 5, \quad \chi(\{a\}, a) = 8, \quad \chi(\{b\}, b) = 6.$$

The expected size of set picked by the mechanism is 1.45. However, using the following cross-monotonic cost sharing method, ξ, the expected size of set picked is 1.45125:

$$\xi(\{a, b\}, a) = 5.5, \quad \xi(\{a, b\}, b) = 4.5, \quad \xi(\{a\}, a) = 8, \quad \xi(\{b\}, b) = 6.$$

For completeness, we first state the following theorem.

THEOREM 17 (Hardy, Littlewood, and Polya [10]). *Let $\alpha_1 \leq \cdots \leq \alpha_n$ and $\beta_1 \leq \cdots \leq \beta_n$ be such that $\alpha_1 + \cdots + \alpha_n = \beta_1 + \cdots + \beta_n$. The following two statements are equivalent.*

1. *$(\alpha_1, \ldots, \alpha_n)$ Lorentz dominates $(\beta_1, \ldots, \beta_n)$.*
2. *$(\alpha_1, \ldots, \alpha_n)$ can be obtained from $(\beta_1, \ldots, \beta_n)$ by applying the following transformations to $(\beta_1, \ldots, \beta_n)$ a finite number of times:*
   (a) *Find $i < j$ such that $\beta_i < \beta_j$ and the next step can be performed.*
   (b) *Increase β_i and decrease β_j by a small $\epsilon > 0$.*

Following is an analogous fact for L-domination. As defined in section 5.2, let V be the set of all nonnegative s-dimensional vectors $\boldsymbol{q}$ such that $f_1(\boldsymbol{q}(1)) + \cdots + f_s(\boldsymbol{q}(s)) = \text{cost}(S)$.

LEMMA 18. *Let $\boldsymbol{q}, \boldsymbol{r} \in V$. The following two statements are equivalent.*

1. *$\boldsymbol{q}$ L-dominates $\boldsymbol{r}$, with π being the identity permutation.*
2. *$\boldsymbol{q}$ can be obtained from $\boldsymbol{r}$ by applying the following transformations to $\boldsymbol{r}$ a nonzero number of times:*

(a) *Find $i < j$ such that $\boldsymbol{r}(i) < \boldsymbol{r}(j)$ and the next step can be performed.*
(b) *Increase $\boldsymbol{r}(i)$ and decrease $\boldsymbol{r}(j)$ by small positive amounts such that*
- *$f_i(\boldsymbol{r}(i)) + f_j(\boldsymbol{r}(j))$ is preserved,*
- *$\boldsymbol{r}(i) \le \boldsymbol{r}(j)$ is preserved.*

Proof. We will prove the forward direction only; the reverse direction is straightforward. Let i be the smallest index such that $\boldsymbol{q}(i) \neq \boldsymbol{r}(i)$ (if such an index i does not exist, then $\boldsymbol{q}$ and $\boldsymbol{r}$ are identical). Since $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$, $\boldsymbol{q}(i) > \boldsymbol{r}(i)$. By definition,

$$f_1(\boldsymbol{q}(1)) + \cdots + f_s(\boldsymbol{q}(s)) = f_1(\boldsymbol{r}(1)) + \cdots + f_s(\boldsymbol{r}(s)).$$

Let j be the smallest index $> i$ such that

$$f_1(\boldsymbol{q}(1)) + \cdots + f_j(\boldsymbol{q}(j)) = f_1(\boldsymbol{r}(1)) + \cdots + f_j(\boldsymbol{r}(j)).$$

By the choice of j, it must be the case that

$$f_1(\boldsymbol{q}(1)) + \cdots + f_{j-1}(\boldsymbol{q}(j-1)) > f_1(\boldsymbol{r}(1)) + \cdots + f_{j-1}(\boldsymbol{r}(j-1));$$

therefore $\boldsymbol{q}(j) < \boldsymbol{r}(j)$. Since $i < j$, $\boldsymbol{q}(i) \le \boldsymbol{q}(j)$. Now we have $\boldsymbol{r}(i) < \boldsymbol{q}(i) \le \boldsymbol{q}(j) < \boldsymbol{r}(j)$. Increase $\boldsymbol{r}(i)$ and decrease $\boldsymbol{r}(j)$ as much as possible so that the following are preserved:

1. $f_i(\boldsymbol{r}(i)) + f_j(\boldsymbol{r}(j))$.
2. $\boldsymbol{r}(i) \le \boldsymbol{r}(j)$.
3. For $1 \le k \le s$,

$$f_1(\boldsymbol{q}(1)) + \cdots + f_k(\boldsymbol{q}(k)) \ge f_1(\boldsymbol{r}(1)) + \cdots + f_k(\boldsymbol{r}(k)).$$

Clearly, $\boldsymbol{r}(i)$ must increase by a positive amount, and $\boldsymbol{r}(j)$ must decrease by a positive amount. Therefore, in the limit, $\boldsymbol{q}$ can be obtained from $\boldsymbol{r}$ by applying such steps. □

Proof of Theorem 16. Consider any set of users $S \subseteq U$. Let $\boldsymbol{q}, \boldsymbol{r} \in V_c$ correspond to χ^S and ξ^S, respectively. By Theorem 14, $\boldsymbol{q}$ L-dominates $\boldsymbol{r}$. Now, $\boldsymbol{q}$ can be obtained from $\boldsymbol{r}$ by steps specified in Lemma 18. Finally, by Lemma 19 given below, each of these steps will only increase the probability that all users accept service. Hence $P(\chi^S) \ge P(\xi^S)$. □

LEMMA 19. *Let $f_1, \ldots, f_s$ be the equalizing functions specified above. Let $x_i \ge 0$, $1 \le i \le s$, and define $\boldsymbol{v} = (f_1(x_1), \ldots, f_s(x_s))$. Define $P(\boldsymbol{v}) = \prod_{i=1}^{s}[1 - G_i(f_i(x_i))]$. Suppose there exist i and j such that $x_i < x_j$. Then,*

$$dP(\boldsymbol{v}) = \sum_{k=1}^{s} \frac{\partial P(\boldsymbol{v})}{\partial f_k(x_k)} df_k(x_k) \ge 0,$$

where $0 < df_i(x_i) = -df_j(x_j)$ and $df_k(x_k) = 0$ if $k \neq i, j$.

Proof.

$$dP(\boldsymbol{v}) = \sum_{k=1}^{s} \frac{\partial P(\boldsymbol{v})}{\partial f_k(x_k)} df_k(x_k)$$

$$= \sum_{k=1}^{s} \frac{P(\boldsymbol{v})}{1 - G(f_k(x_k))} \times (-g(f_k(x_k))) \times df_k(x_k)$$

$$= P(\boldsymbol{v}) \left[\frac{g_j(f_j(x_j))}{1 - G_j(f_j(x_j))} - \frac{g_i(f_i(x_i))}{1 - G_i(f_i(x_i))} \right] dx$$

$$= P(\boldsymbol{v})[\lambda_j(f_j(x_j)) - \lambda_i(f_i(x_i))]dx$$

$$= P(\boldsymbol{v})[x_j - x_i]dx \geq 0,$$

where $dx = df_i(x_i) = -df_j(x_j)$.

The second-to-last step uses the fact that f_i is the inverse of λ_i and the last step follows from the assumption that $x_i < x_j$. □

8. Discussion. Submodular cost functions admit a rich class of cross-monotone cost sharing methods. Of these, equitable methods capture a large subclass, though not all, as evidenced by the example below. Our definition as well as Hokari's [11] appears quite natural, and despite differences, they lead to the same class of cost sharing methods. This raises the following question: how to impose an economic criterion on cross-monotone methods so as to obtain precisely the class of equitable methods.

Example. Consider the following cross-monotone cost sharing method (the costs of individual sets are simply the sum of cost shares of their elements).
$\xi(\{a,b,c,d\},a) = 2$,
$\xi(\{a,b,c,d\},b) = 2$, $\xi(\{a,b,c,d\},c) = 1$, $\xi(\{a,b,c,d\},d) = 1$,
$\xi(\{a,b,c\},a) = 2$, $\xi(\{a,b,c\},b) = 3$, $\xi(\{a,b,c\},c) = 1$,
$\xi(\{a,b,d\},a) = 3$, $\xi(\{a,b,d\},b) = 2$, $\xi(\{a,b,d\},d) = 1$,
$\xi(\{a,c,d\},a) = 2$, $\xi(\{a,c,d\},c) = 2$, $\xi(\{a,c,d\},d) = 2$,
$\xi(\{b,c,d\},b) = 2$, $\xi(\{b,c,d\},c) = 2$, $\xi(\{b,c,d\},d) = 2$,
$\xi(\{a,b\},a) = 3$, $\xi(\{a,b\},b) = 3$,
$\xi(\{a,c\},a) = 3$, $\xi(\{a,c\},c) = 3$,
$\xi(\{a,d\},a) = 3$, $\xi(\{a,d\},d) = 3$,
$\xi(\{b,c\},b) = 3$, $\xi(\{b,c\},c) = 3$,
$\xi(\{b,d\},b) = 3$, $\xi(\{b,d\},d) = 3$,
$\xi(\{c,d\},c) = 3$, $\xi(\{c,d\},d) = 3$,
$\xi(\{a\},a) = 5$, $\xi(\{b\},b) = 5$, $\xi(\{c\},c) = 5$, $\xi(\{d\},d) = 5$.
We will show, by contradiction, that this is not an equitable cost sharing method. First run Algorithm 1 on $S = \{a,b,c\}$. In order to produce the above method, S must be the first set to go tight. Suppose this happens at time t. Clearly, none of the proper subsets of S is tight at time t. Therefore, $f_a(t) = 2$, and $f_b(t) = 3$. Next, run Algorithm 1 on set $S' = \{a,b,d\}$. The first set to go tight must be the entire set S', at time t', say. Therefore, $f_a(t') = 3$, and $f_b(t') = 2$. But then at least one of f_a or f_b is not monotonically increasing.

Observe that Algorithm 1 is very explicitly trying to impose equality among users, as specified by the equalizing functions. Of course, the precise notion of "equality" or "fairness" imposed depends on these functions. An exciting research direction is to characterize the notions of fairness captured by particular choices of the equalizing functions.

This very explicit seeking of "equality" is also the chief difference between our definition and Hokari's definition and algorithm. Hokari defines *sequential monotone path* cost sharing methods by partitioning the set of users, ordering the partitions sequentially, and applying monotone path methods within each partition, with an incremental method applied on the ordered list of partitions. In our setting it is easy

to see that the resulting method is also equitable, and so this operation does not lead to new cost sharing methods.

Considering the fact that for submodular cost functions, our definition is a natural generalization of the Dutta–Ray egalitarian method, one wonders whether there is a similar generalization for nonsubmodular cost functions as well. L-orderings, presented in section 5.2, may lead to such a definition. At present it is not clear how to generalize Algorithm 1 to this setting—if the cost function is not submodular, then every element may be in a tight set, without the entire set being tight. Algorithm 1 will halt at this point, and the resulting cost allocation will not be budget balanced.

Next, assume that the cost function is nondecreasing and submodular. Let σ be a permutation on $1, \ldots, n$. The *incremental cost sharing method*, ξ_σ, corresponding to permutation σ is defined as follows. Let $S \subseteq U$, $|S| = k$, and let $i_1, \ldots, i_k$ be the users in S ordered according to σ. Then, $\xi_\sigma(S, i_1) = \text{cost}(i_1)$, and for $2 \leq j \leq k$, $\xi_\sigma(S, i_j) = \text{cost}(\{i_1, \ldots, i_j\}) - \text{cost}(\{i_1, \ldots, i_{j-1}\})$.

In [17] we showed that the class of cross-monotone methods for a nondecreasing submodular cost function form a polytope and that incremental cost sharing methods form corner points of this polytope. We also gave an example of a cross-monotone method that is not in the convex hull of these corner points (corresponding to incremental cost sharing methods) and left the open problem of characterizing the rest of the corner points. We show below that this example is in fact an equitable method. It is easy to see that all incremental methods are also equitable. Do equitable methods capture all corner points of this polytope? Are equitable methods closed under convex combinations? Since not all cost sharing methods are equitable, the answers to both these questions cannot be "Yes."

Example. Consider the following cost sharing method:
$\xi(\{a, b, c\}, a) = 2$, $\xi(\{a, b, c\}, b) = 3$, $\xi(\{a, b, c\}, c) = 4$,
$\xi(\{a, b\}, a) = 4$, $\xi(\{a, b\}, b) = 3$,
$\xi(\{a, c\}, a) = 3$, $\xi(\{a, c\}, c) = 4$,
$\xi(\{b, c\}, b) = 3$, $\xi(\{b, c\}, c) = 4$,
$\xi(\{a\}, a) = 4$, $\xi(\{b\}, b) = 4$, $\xi(\{c\}, c) = 4$.
In [17] we showed that this cross-monotone method is not a convex combination of incremental cost sharing methods. However, it is an equitable cost sharing method corresponding to equalizing functions f_a, f_b, f_c satisfying

$$f_a(2) = 0, \quad f_b(1) = 0, \quad f_b(2) = 3, \quad f_b(3) = 3, \quad f_b(4) = 4, \quad \text{and } f_c(1) = 4.$$

Finally, the example given in section 7 raises the question of identifying the cost sharing method that maximizes the expected size of the set served.

Acknowledgments. We wish to thank Bhaskar Dutta, Suresh Mutuswami, and Christos Papadimitriou for valuable comments and for providing references to the literature. Thanks also go to the anonymous STOC referee for valuable suggestions on the paper.

REFERENCES

[1] Y. Afek, Y. Mansour, and Z. Ostfeld, *Convergence complexity of optimistic rate based flow control algorithms*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996.

[2] J. Arin, J. Kupers, and D. Vermeulen, *An Axiomatic Approach to Egalitarianism in TU Games*, Discussion paper, Department of Mathematics, Maastricht University, Maastricht, The Netherlands, 2000.

[3] J. Arin, J. Kupers, and D. Vermeulen, *Some Characterization of Egalitarian Solutions on Classes of TU Games*, mimeo, Maastricht University, Maastricht, The Netherlands, 2000.
[4] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Upper Saddle River, NJ, 1987.
[5] B. Dutta, *The egalitarian solution and reduced game properties in convex games*, Internat. J. Game Theory, 19 (1990), pp. 153–169.
[6] B. Dutta and D. Ray, *A concept of egalitarianism under participation constraints*, Econometrica, 57 (1989), pp. 615–635.
[7] B. Dutta and D. Ray, *Constrained egalitarian allocations*, Games Econom. Behav., 3 (1991), pp. 403–422.
[8] D. Fudenberg and J. Tirole, *Game Theory*, MIT Press, Cambridge, MA, 1991.
[9] A. Goel, A. Meyerson, and S. Plotkin, *Approximate majorization and fair online load balancing*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 384–390.
[10] G. Hardy, H. Littlewood, and G. Polya, *Inequalities*, Cambridge University Press, Cambridge, UK, 1934.
[11] T. Hokari, *Monotone-path Dutta-Ray solutions on convex games*, Soc. Choice Welf., 19 (2002), pp. 825–844.
[12] T. Hokari, *Weighted Dutta-Ray Solutions on Convex Games*, mimeo, University of Rochester, Rochester, NY, 1998.
[13] J. L. Hougard, B. Peleg, and L. P. Osterdal, *The Dutta-Ray Solution on the Class of Convex Games: A Generalisation and Its Monotonicity Properties*, mimeo, University of Copenhagen, Copenhagen, Denmark, 2000.
[14] S. Iwata, L. Fleischer, and S. Fujishige, *A strongly polynomial-time algorithm for minimizing submodular functions*, J. ACM, 48 (2001), pp. 761–777.
[15] J. Jaffe, *Bottleneck flow control*, IEEE Trans. Comm., 29 (1981), pp. 954–962.
[16] J. Y. Jaffray and P. Mongin, *Constrained egalitarianism in a simple redistribution model*, Theory and Decision, 54 (2003), pp. 33–56.
[17] K. Jain and V. V. Vazirani, *Applications of approximation algorithms to cooperative games*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, 2001, pp. 364–372.
[18] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.
[19] K. Jain and V. V. Vazirani, *Eisenberg-Gale markets: Algorithms and structural properties*, in Proceedings of the 39th Annual ACM Symposium on Theory of Computing, 2007, pp. 364–373.
[20] J. Kleinberg, Y. Rabani, and E. Tardos, *Fairness in routing and load balancing*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, p. 568.
[21] F. Klijn, M. Slikker, S. Tijs, and J. Zarzuelo, *The egalitarian solution for convex games: Some characterizations*, Math. Social Sci., 40 (2000), pp. 111-121.
[22] N. Megiddo, *Optimal flows in networks with multiple sources and sinks*, Math. Programming, 7 (1974), pp. 97–107.
[23] H. Moulin, *Incremental cost sharing: Characterization by coalition strategy-proofness*, Soc. Choice Welf., 16 (1999), pp. 279–320.
[24] S. Mutuswami, *Strategyproof Mechanism for Cost Sharing*, Economics Discussion Papers 520, University of Essex, Colchester, UK, 2000.
[25] A. M. Odlyzko, *Internet pricing and the history of communications*, Computer Networks, 36 (2001), pp. 493–517.
[26] A. Schrijver, *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, J. Combin. Theory Ser. B, 80 (2000), pp. 346–355.
[27] H. Varian, *Pricing electronic journals*, D-Lib Magazine, June (1996); available online from http://www.dlib.org/dlib/june96/06varian.html.
[28] H. Varian, *Differential pricing and efficiency*, First Monday, 1 (2) (1996).
[29] H. Varian, *Pricing information goods*, in Proceedings of Research Libraries Group Symposium on "Scholarship in the New Information Environment," Harvard Law School, Cambridge, MA, 1995.
[30] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 257–275

VERTICAL RAY SHOOTING AND COMPUTING DEPTH ORDERS FOR FAT OBJECTS*

MARK DE BERG† AND CHRIS GRAY†

Abstract. We present new results for three problems dealing with a set $\mathcal{P}$ of n convex constant-complexity fat polyhedra in 3-space. (i) We describe a data structure for vertical ray shooting in $\mathcal{P}$ that has $O(\log^2 n)$ query time and uses $O(n \log^2 n)$ storage. (ii) We give an algorithm to compute in $O(n \log^3 n)$ time a depth order on $\mathcal{P}$ if it exists. (iii) We give an algorithm to verify in $O(n \log^3 n)$ time whether a given order on $\mathcal{P}$ is a valid depth order. All three results improve on previous results.

Key words. computational geometry, depth orders, fat objects, ray shooting

AMS subject classification. 68U05

DOI. 10.1137/060672261

1. Introduction.

Motivation. Many algorithms and data structures developed in computational geometry have a rather poor worst-case performance. This behavior is often caused by sets of objects in very intricate configurations, such as many long and thin objects packed closely together. For example, the worst-case running time of the best known general motion-planning algorithm is $\Theta(n^f)$, where f is the number of degrees of freedom of the robot, because for certain configurations of obstacles the free space has complexity $\Theta(n^f)$. The configurations that determine the worst-case behavior, however, are usually rather artificial constructions; one would expect that in practice the input is much more well behaved, and that better performance is possible than the worst-case analysis suggests.

These considerations have led to the study of geometric problems in so-called *realistic input models* [9]. Here one places certain restrictions on the shape and/or distribution of the input objects, so that hypothetical worst-case examples are excluded. The hope is that this will enable proving much stronger theoretical results than are possible for arbitrary inputs, while the results are still general enough for practical applications. One of the most widely studied realistic input models assumes that the input objects are *fat*, that is, they are not arbitrarily long and skinny—see the next section for a formal definition. For motion planning this has proved to be quite successful: the free space for a not-too-large robot moving amidst a set of fat obstacles has only linear complexity [26], which has enabled the development of motion-planning algorithms with near-linear running times in this setting [25].

In this paper we study two problems arising in computer graphics in the context of realistic input models: the vertical ray-shooting problem and the depth-order problem for fat polytopes in $\mathbb{R}^3$.

Problem statement and previous results. Let $\mathcal{P}$ be a set of n disjoint objects in $\mathbb{R}^3$.

*Received by the editors October 13, 2006; accepted for publication (in revised form) January 10, 2008; published electronically April 11, 2008. A preliminary version of this paper appeared in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 494–503. This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project 639.023.301.

http://www.siam.org/journals/sicomp/38-1/67226.html

†Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands (mdberg@win.tue.nl, cgray@win.tue.nl).

The first problem we study is the *ray-shooting problem*: preprocess the set $\mathcal{P}$ such that ray-shooting queries (i.e., what is the first object in $\mathcal{P}$ hit by a query ray?) can be answered efficiently. Ray-shooting queries are important for realistic image synthesis: they form the basis of ray-tracing algorithms and can be used in radiosity methods to approximate form factors. Hence, data structures for ray shooting have received ample attention, in both computer graphics and computational geometry; the book by De Berg [5] and the survey by Pellegrini [24] discuss many of the (computational geometry) solutions. As for the motion-planning problem, the worst-case bounds that have been obtained for the general ray-shooting problem are rather disappointing. For ray-shooting queries with arbitrary rays in a collection of n disjoint triangles, for example, the best known structures that achieve $O(\log n)$ query time use $O(n^{4+\epsilon})$ storage [5, 23], and the best structures with near-linear storage have roughly $O(n^{3/4})$ query time [3]. For vertical ray shooting in a collection of disjoint triangles the situation is still not very rosy: to obtain $O(\log n)$ query time one needs $O(n^{2+\epsilon})$ storage, and with near-linear storage the query time becomes roughly $O(\sqrt{n})$ [5].

Given the prominence of the ray-shooting problem in computational geometry and the fact that general inputs do not seem to admit very efficient (near-linear) solutions, it is not surprising that ray shooting has already been studied from the perspective of realistic input models. In particular, the vertical ray-shooting problem—here the query ray is required to be vertical—has been studied for fat convex polyhedra. For this case Katz [19] presented a data structure that uses $O(n \log^3 n)$ storage and has $O(\log^4 n)$ query time. (In fact, Katz's solution works for polygons whose projections onto the xy-plane are fat, but it is not difficult to see that it works for fat three-dimensional polytopes as well.) Using the techniques of Efrat et al. [17], it is possible to improve the storage bound to $O(n \log^2 n)$ and the query time to $O(\log^3 n)$ [20]. Recently De Berg [6] presented a structure with $O(\log^2 n)$ query time; his structure uses $O(n \log^3 n (\log \log n)^2)$ storage.

The second problem we study is the *depth-order problem*: compute a depth order for the set $\mathcal{P}$, that is, an ordering $P_1, \ldots, P_n$ of the objects in $\mathcal{P}$ such that if P_i is below P_j, then $i < j$. Here we say that P_i is below P_j, denoted by $P_i \prec P_j$, if there are points $(x, y, z_i) \in P_i$ and $(x, y, z_j) \in P_j$ with $z_i < z_j$. In other words, a depth order is a linear extension of the $\prec$-relation. Since there can be cycles in the $\prec$-relation—we then say there is *cyclic overlap* among the objects—a depth order does not always exist. In that case the algorithm should report that there is cyclic overlap. Depth orders are useful in several applications. For example, they can be used to render scenes with the Painter's algorithm [18] or to do hidden-surface removal with the algorithm of Katz, Overmars, and Sharir [21]. Depth orders also play a role in assembly planning [1].

The depth-order problem for arbitrary sets of triangles in 3-space does not seem to admit a near-linear solution; the best known algorithm runs in $O(n^{4/3+\varepsilon})$ time [11]. This has led researchers to also study this problem for fat objects. Agarwal, Katz, and Sharir [2] gave an algorithm for computing the depth order of a set of triangles whose projections onto the xy-plane are fat; their algorithm runs in $O(n \log^5 n)$ time. However, their algorithm cannot detect cycles—when there are cycles it reports an incorrect order. A subsequent result by Katz [19] produced an algorithm that runs in $O(n \log^5 n)$ time and that can detect cycles. In this case though, the constant of proportionality depends on the minimum overlap of the projections of the objects that do overlap. If there is a pair of objects whose projections barely overlap, then the running time of the algorithm increases greatly. One advantage that this algorithm has is that it can deal with convex curved objects.

Our results. First, we present a new data structure for vertical ray shooting in a collection of n convex constant-complexity fat polyhedra[1] in $\mathbb{R}^3$. Our structure uses $O((1/\beta)n\log^2 n)$ storage and has $O((1/\beta^2)\log^2 n)$ query time. Compared to Katz's structure [20] it has a better query time (while the storage is the same) and compared to De Berg's structure [6] it has a better storage bound (while keeping the same query time).

Second, we present a new algorithm for computing a depth order on a collection of n convex constant-complexity fat polyhedra in $\mathbb{R}^3$. Our algorithm runs in $O((1/\beta^3)n\log^3 n)$ time, improving the result of Agarwal, Katz, and Sharir [2] by two logarithmic factors. Like their algorithm, our algorithm unfortunately does not detect cyclic overlap. Hence, we also study the problem of verifying a given depth order. We give an algorithm that checks in $O((1/\beta^2)n\log^3 n)$ time[2] whether a given ordering for a set of fat convex polyhedra is a valid depth order. This is the first result for this problem. Until now, the only algorithm for verifying a given depth order was an algorithm for arbitrary triangles [11], which runs in $O(n^{4/3+\varepsilon})$ time.

2. Preliminaries. In this section we introduce some basic definitions and terminology.

For a three-dimensional object o, we use $vol(o)$ to denote the volume of o and we use $proj(o)$ to denote the vertical projection of o onto the xy-plane. For a two-dimensional object, we use $area(o)$ to denote its area. We use the following definition of fatness [9].

DEFINITION 1. *An object o in $\mathbb{R}^d$ is defined to be β-fat if for any ball b whose center lies in o and that does not fully contain o, we have $vol(b\cap o)\geq\beta\cdot vol(b)$.*

(For convex objects—the case considered in this paper—this definition is equivalent, up to constant factors, to other definitions of fatness that have been proposed.) It is not hard to show that the projection of a fat object is also fat, as proved by De Berg [6] and made precise in the following lemma.

LEMMA 1 (see [6]). *If an object P is a β-fat object in three dimensions, then $proj(P)$ has fatness $\Omega(\beta)$ in two dimensions.*

Define the *size* of an object o, denoted by $size(o)$ to be the radius of its smallest enclosing ball. Note that the size of a ball is simply its radius.

Finally, we need a result that will allow us to stab a set of relatively large fat objects that all intersect some region R using only a few points. Similar results have been proved earlier [7].

LEMMA 2. *Let R be a bounded region in the plane, and let c be a constant with $0<c\leq 1$. Then there is a collection Q of $O(1/(c\beta)^2)$ points with the following property: any β-fat object o with $size(o)\geq c\cdot size(R)$ that intersects R contains at least one point from Q.*

Proof. Let U be a bounding square of R, and let $\widehat{U}$ be the square twice the size of U and with the same center. Consider a β-fat object o with $size(o)\geq c\cdot size(R)$ that intersects R. Then $area(o\cap\widehat{U})\geq c'c\beta\cdot area(\widehat{U})$ for a suitable constant c' (cf. Van der Stappen's thesis [25, Theorem 2.9]). Hence, a regular grid on $\widehat{U}$ with $\lceil M\rceil^2$ cells, where $M=2/(c'c\beta)$, must have at least one grid point inside P, because the area of any convex object missing all grid points is less than $2\cdot area(\widehat{U})/M$. □

The following lemma was proved in a more general setting by Van Kreveld [22].

[1] Though results are presented in terms of fat polyhedra, all our results also work in the more general setting of objects that project to fat polygons.

[2] This is an improvement over the $O(n\log^4 n)$ bound that we had in the preliminary version of the paper [8], which was published in SODA 2006.

However, his definition of fatness is different from ours. Therefore we have proved it independently using our definition. The proof can be found in the appendix. In the lemma below, an α-fat triangle refers to a triangle all of whose angles are at least α. (Such a triangle is α'-fat, according to Definition 1, for some $\alpha' = \Omega(\alpha)$.)

LEMMA 3. *Let P be a β-fat convex polygon with n vertices. There is a set T of α-fat triangles that cover P, where $|T| = O(n)$ and $\alpha = \Theta(\beta)$.*

We will also need the following lemma.

LEMMA 4. *Let P_1 and P_2 be simple polygons. Let e_1 be an edge of P_1 and e_2 be an edge of P_2. If P_1 intersects P_2 so that there is no vertex of P_1 inside P_2 and no vertex of P_2 inside P_1, then there is an intersection of edges e_3 of P_1 and e_4 of P_2 such that $e_3 \neq e_1$ and $e_4 \neq e_2$.*

Proof. Let e of P_1 and e' of P_2 be edges that intersect. If $e \neq e_1$ and $e' \neq e_2$, then we are done. If $e \neq e_1$ and $e' = e_2$, then there must be an intersection between e and a different edge of P_2 (since there are no vertices of P_1 inside P_2), meaning that we have found an intersection between e and some edge $e'' \neq e_2$, and we are done. Similarly, we are done if $e = e_1$ and $e' \neq e_2$. Finally, suppose $e = e_1$ and $e' = e_2$. Since e_1 enters P_2, it must exit it, and that implies that there must be an intersection between e_1 and some edge $e'' \neq e_2$. This puts us in the previous case, so we are done. ∎

3. Vertical ray shooting. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a collection of n constant-complexity convex β-fat polyhedra that we wish to preprocess for vertical ray shooting. We start by studying the simpler case where all the objects are intersected by a common vertical line. After that we will show how to use this structure to obtain an efficient solution to the general problem.

Agarwal, Katz, and Sharir [2] already described a data structure for the case where all objects are intersected by a common vertical line and project to triangles. We observe that it is possible to apply fractional cascading to their structure to obtain the following result.

LEMMA 5. *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n disjoint convex constant-complexity β-fat polyhedra that are all stabbed by a vertical line ℓ and that all project to fat triangles. Then there is a data structure such that vertical ray shooting queries on $\mathcal{P}$ can be answered in $O(\log n)$ time. The structure uses $O((1/\beta)n \log n)$ storage and can be built in $O((1/\beta)n \log n)$ time.*

Proof. As stated above, all we need to do is apply fractional cascading to the structure of Agarwal, Katz, and Sharir [2]. For completeness, we briefly describe their solution and explain how to apply fractional cascading.

The structure is a balanced binary tree $\mathcal{T}$ with the objects in the leaves, sorted by their position along ℓ; the lowest object is in the leftmost leaf, the second lowest object in the next leaf, and so on. Since the objects are nonintersecting and convex, this ordering is well defined.

For a node ν, let $\mathcal{P}(\nu)$ denote the set of objects stored in the leaves of the subtree rooted at ν. At each nonleaf node ν of $\mathcal{T}$, we store the union $U(\nu)$ of the projections of the objects in $\mathcal{P}(\nu)$. We preprocess $U(\nu)$ for point-enclosure queries, that is, queries that ask whether a point q in the xy-plane lies inside $U(\nu)$, as follows. Let p_ℓ be the point where ℓ intersects the xy-plane. Then all projections contain p_ℓ, and since they are convex, $U(\nu)$ is star-shaped with p_ℓ in the kernel. Hence, if we partition the plane into cones by drawing half-lines from p_ℓ through all breakpoints on the boundary of $U(\nu)$, then a point-enclosure query can be answered in $O(1)$ time after we have determined in which cone the query point lies.

To perform a query with a vertical ray starting above all objects, we walk down the tree as follows. Suppose we reach a node ν. When the point p where the ray hits

the xy-plane lies inside the union of the right child of ν, we proceed to the right child; otherwise we proceed to the left child. The leaf we reach must store the first object hit (if any object is hit at all). When the starting point of the ray does not lie above all objects, things are more complicated. However, Agarwal, Katz, and Sharir have shown that a query can still be answered by walking down the tree, although now up to four nodes per level may be visited. In any case, we visit $O(\log n)$ nodes in total, and at each node we have to do a point-enclosure query. As explained above, a point-enclosure query can be answered in $O(1)$ time after we have determined in which cone the query point lies. Finding the right cone can be done in $O(\log n)$ time by binary search, but this can be made faster: using fractional cascading [13, 14], finding the cones can be done in $O(1)$ time, except for the search at the root. Since the application of fractional cascading is completely standard in this setting, we omit further details.

To build the structure, we sort the objects along ℓ in $O(n \log n)$ time, and then we construct the unions to be stored at each node in a bottom-up fashion. Hence, when we arrive at a node ν, we have to merge the two unions of the children of ν. Because the unions are star-shaped with respect to the same point, computing the union of these unions boils down to merging the two circularly sorted lists of breakpoints. Hence, this can be done in linear time. The total time to construct all unions is therefore equal to the total size of the data structure, which is $\sum_\nu O(|\mathcal{P}(\nu)|) = O(n \log n)$. Adding the additional pointers for the fractional cascading does not increase the preprocessing time or the amount of storage asymptotically. □

Now consider the general case, where the objects in $\mathcal{P}$ are not necessarily stabbed by a vertical line. We can cover each object by $O(1)$ subobjects whose projections are fat triangles using the technique of Lemma 3, so we can assume without loss of generality that all objects project to fat triangles. We shall make use of balanced aspect ratio trees (BAR-trees). *BAR-trees* are a special type of binary space partition (BSP) trees for point sets. They were introduced by Duncan, Goodrich, and Kobourov [15, 16], who showed that BAR-trees have excellent performance for approximate range searching and approximate nearest-neighbor searching. A BSP tree $\mathcal{T}$ for a set S of points contained in some bounding square σ is a recursive partitioning of σ by splitting lines, such that the final cells of the subdivision do not contain any points in their interior. Each node ν of $\mathcal{T}$ corresponds to a region $\mathit{region}(\nu) \subset \sigma$, which is defined recursively as follows. The region $\mathit{region}(\mathit{root}(\mathcal{T}))$ is the whole square σ. Furthermore, if the splitting line stored at a node ν is $\ell(\nu)$, then $\mathit{region}(\mathit{leftchild}(\nu)) = \mathit{region}(\nu) \cap \ell(\nu)^-$, where $\ell(\nu)^-$ is the half-plane below $\ell(\nu)$. Similarly, $\mathit{region}(\mathit{rightchild}(\nu)) = \mathit{region}(\nu) \cap \ell(\nu)^+$, where $\ell(\nu)^+$ is the half-plane above $\ell(\nu)$.

The special properties of BAR-trees that are relevant for us are the following. First, a BAR-tree on a set S of points has depth $O(\log |S|)$ and size $O(|S|)$. Furthermore, the regions corresponding to a node in a BAR-tree have bounded aspect ratio, which implies they are c-fat for some constant c. It has been shown by De Berg and Streppel [12] that this implies the following.

LEMMA 6 (see [12]). *Let o be a β-fat object. Then there is a set $G(o)$ of 12 points—we call these points* guards*—such that for any BAR-tree region R that intersects o but does not contain a guard from $G(o)$ in its interior we have $\mathit{size}(o) = \Omega(\mathit{size}(R))$.*

De Berg and Streppel [12] used this to design a so-called object BAR-tree: this is a BAR-tree that can be used for approximate range searching in a set of objects rather than in a point set. Our ray-shooting structure combines BAR-trees and the lemma above in a different way, as described next.

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n constant-complexity β-fat polyhedra. Let $G_i = G(proj(P_i))$ be a set of guards for the projection of P_i, as in Lemma 6. Our data structure for vertical ray shooting on $\mathcal{P}$ is defined as follows.

- The main tree $\mathcal{T}$ is a BAR-tree for the set $G = G_1 \cup \cdots \cup G_n$.
- Let ν be a node in $\mathcal{T}$. We say that an object P_i is *large* at ν if (i) $proj(P_i)$ intersects $region(\nu)$, and (ii) $region(parent(\nu))$ contains a guard from G_i in its interior, but $region(\nu)$ does not. Note that Lemma 6 implies that $size(P_i) = \Omega(size(region(\nu)))$ if P_i is large at ν. Let $\mathcal{P}(\nu) \subset \mathcal{P}$ be the subset of objects that are large at ν.
  Let $Q(\nu)$ be a set of points such that for any $P_i \in \mathcal{P}(\nu)$, there is a point $q \in Q(\nu)$ with $q \in proj(P_i)$. By Lemma 2 there exists such a set $Q(\nu)$ of size $O(1/\beta^2)$. Assign each object $P_i \in \mathcal{P}(\nu)$ arbitrarily to one of the points $q \in Q(\nu)$ contained in its projection. Let $\mathcal{P}(q)$ denote the set of objects assigned to q. We store the set $\mathcal{P}(q)$ in a data structure $\mathcal{T}(q)$ for vertical ray shooting according to Lemma 5. Thus each node ν has $|Q(\nu)|$ associated structures.

Let us first see how to answer a vertical ray shooting query with this structure.

LEMMA 7. *A vertical ray-shooting query can be answered in $O((1/\beta^2)\log^2 n)$ time.*

Proof. Let p be the point where the line through the query ray intersects the xy-plane. Search with p down the tree $\mathcal{T}$. At every node ν on the search path, perform a query in the associated structure $\mathcal{T}(q)$ of each $q \in Q(\nu)$. A query thus takes $O(\log n \cdot \log n \cdot (1/\beta^2))$ time, that is, the length of every search path, times the query time in the associated data structures along the search path, times the size of $Q(\nu)$.

To prove the correctness, it suffices to argue that any object P_i whose projection contains p must be large at one of the nodes on the search path of p. To see this, we observe that $region(root(\mathcal{T}))$ contains all guards from G_i, while the leaf regions do not contain any guards in their interior. It follows that when we follow the path of p, the object P_i must become large at some node. □

We can now prove our final result on vertical ray shooting.

THEOREM 1. *Let $\mathcal{P}$ be a collection of n convex disjoint constant-complexity β-fat polyhedra in $\mathbb{R}^3$. Then there is a data structure such that vertical ray-shooting queries on $\mathcal{P}$ can be answered in $O((1/\beta^2)\log^2 n)$ time. The structure uses $O((1/\beta)n\log^2 n)$ storage and can be built in $O((1/\beta)n\log^2 n)$ time.*

Proof. The correctness of the query procedure and the query time have been shown in Lemma 7.

It remains to prove the bound on the construction time; the storage bound then follows trivially. Computing the guards for each object takes constant time per object, and constructing the BAR-tree takes $O(n\log n)$ time [16]. We claim that an object P_i is large at $O(\log n)$ nodes. Indeed, any guard is contained in the regions of the nodes on a single path down the tree, and an object can be large at a node only if the parent region contains one of its guards. Hence, $\sum_\nu |\mathcal{P}(\nu)| = O(n\log n)$. We can generate the sets $\mathcal{P}(\nu)$ in $O(n\log n)$ time by filtering the objects down the tree $\mathcal{T}$. The set $Q(\nu)$ can be constructed in $O(|Q(\nu)|)$ time, and associating the objects with the points in $Q(\nu)$ can be done in a brute-force way in $O(|Q(\nu)| \cdot |\mathcal{P}(\nu)|)$. Finally, constructing the associated structures of ν takes time

$$\sum_{q \in Q(\nu)} O((1/\beta)|\mathcal{P}(q)|\log|\mathcal{P}(q)|) \;=\; O((1/\beta)|\mathcal{P}(\nu)|\log|\mathcal{P}(\nu)|)$$

by Lemma 5. Hence, the overall construction time is

$$\begin{aligned}&\textstyle\sum_\nu O(|\mathcal{P}(\nu)| \cdot (|Q(\nu)| + (1/\beta)\log|\mathcal{P}(\nu)|))\\ =\ & O((1/\beta)n\log^2 n + (1/\beta^2)n\log n)\\ =\ & O((1/\beta)n\log^2 n). \qquad \square\end{aligned}$$

4. The size of the transitive reduction of depth-order graphs. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of disjoint objects in $\mathbb{R}^3$. Recall that we say that P_i is below P_j, denoted by $P_i \prec P_j$, if there are points $(x, y, z_i) \in P_i$ and $(x, y, z_j) \in P_j$ with $z_i < z_j$. We define the *depth-order graph of* $\mathcal{P}$ to be the graph $\mathcal{G}(\mathcal{P}) = (\mathcal{P}, E)$, where $(P_i, P_j) \in E$ if and only if $P_i \prec P_j$. Hence, a depth order for $\mathcal{P}$ corresponds to a topological order on $\mathcal{G}(\mathcal{P})$.

In general it is too costly to compute $\mathcal{G}(\mathcal{P})$ explicitly, since it can have $\Omega(n^2)$ arcs. When computing depth orders for segments in the plane, this can be circumvented by looking only at pairs of segments that "see" each other, that is, that can be connected vertically without crossing another segment. For objects in 3-space, however, the number of pairs that see each other can be quadratic, even when the objects are fat. In this section we therefore study the size of the transitive reduction of depth-order graphs, since the transitive reduction is the smallest subgraph that is sufficient to topologically sort a graph. The main result is that the number of arcs in the transitive reduction of the depth-order graph of a set of fat objects is linear. Then in the next section we will compute a superset of the arcs in the transitive reduction.

We define the *separation* of two nodes in the depth-order graph, denoted $sep(P_i, P_j)$, to be the length of the longest path from P_i to P_j. Notice that if the graph contains cycles, $sep(P_i, P_j)$ can be infinite. We define $\mathcal{G}^{(1)}(\mathcal{P}) = (\mathcal{P}, E^{(1)})$ to be the subgraph of the depth-order graph $\mathcal{G}(\mathcal{P})$, where $(P_i, P_j) \in E^{(1)}$, if and only if $sep(P_i, P_j) = 1$ in $\mathcal{G}(\mathcal{P})$. In other words, $(P_i, P_j) \in E^{(1)}$ if there exists a vertical line that intersects both objects, and every such line does not intersect any other object between P_i and P_j.

LEMMA 8. *If $\mathcal{G}(\mathcal{P})$ is acyclic, the transitive closure of $\mathcal{G}^{(1)}(\mathcal{P})$ is the transitive closure of $\mathcal{G}(\mathcal{P})$.*

Proof. We have to prove that there is a path $P_i \rightsquigarrow P_j$ in $\mathcal{G}(\mathcal{P})$ if and only if there is a path $P_i \rightsquigarrow P_j$ in $\mathcal{G}^{(1)}(\mathcal{P})$. The "if" part is obvious since $\mathcal{G}^{(1)}(\mathcal{P})$ is a subgraph of $\mathcal{G}(\mathcal{P})$. We prove the "only if" part by induction on $sep(P_i, P_j)$.

If $sep(P_i, P_j) = 1$, the arc (P_i, P_j) exists in $\mathcal{G}^{(1)}(\mathcal{P})$ by construction. Now assume there is a path in $\mathcal{G}^{(1)}(\mathcal{P})$ between all nodes with separation m. Take P_i and P_j in $\mathcal{G}(\mathcal{P})$ which have separation $m + 1$. Then there is a node x such that $sep(P_i, x) = 1$ and $sep(x, P_j) = m$. By the induction hypothesis, we then have a path $P_i \rightarrow x \rightsquigarrow P_j$ in $\mathcal{G}^{(1)}(\mathcal{P})$. $\square$

For arbitrary triangles in 3-space, the number of arcs in $\mathcal{G}^{(1)}(\mathcal{P})$ can still be $\Theta(n^2)$. For some special classes of objects, however, the number of arcs is linear. For example, one can show that this number is linear for a set of disjoint polyhedra whose projections form a set of polygonal pseudodisks [10]. Here we concentrate on the case where the objects in the given set $\mathcal{P}$ project onto fat convex objects. We show that in this case the number of arcs is also linear. Since fat convex objects project to fat objects, showing this also shows that the number of arcs in $\mathcal{G}^{(1)}(\mathcal{P})$ is small if the input is a set of fat objects. We start with an auxiliary lemma.

LEMMA 9. *Let $P_i \in \mathcal{P}$ be an object and let $\mathcal{P}^+(i)$ be the subset of objects $P_j \in \mathcal{P}$ that are above P_i and where $sep(P_i, P_j) = 1$. Then the projections $proj(P_j)$ of the objects $P_j \in \mathcal{P}^+(i)$ are pairwise disjoint.*

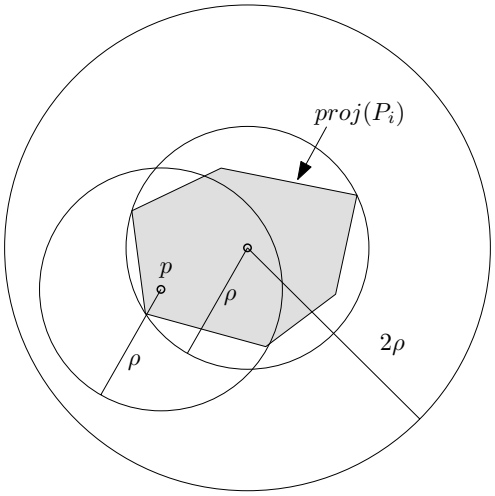


FIG. 1. *Illustration of the packing argument.*

Proof. Suppose the projections are not pairwise disjoint. Then there are objects $P_j, P_k \in \mathcal{P}^+(i)$ such that $proj(P_j) \cap proj(P_k) \neq \emptyset$ and $sep(P_i, P_j) = 1$ and $sep(P_i, P_k) = 1$. Since $proj(P_j)$ and $proj(P_k)$ intersect, they must share at least one point, so there must be an arc between P_j and P_k in $\mathcal{G}(\mathcal{P})$. Therefore, either $sep(P_i, P_j) > 1$ or $sep(P_i, P_k) > 1$, either case being a contradiction. □

THEOREM 2. *Let $\mathcal{P}$ be a collection of n disjoint objects in $\mathbb{R}^3$ that project to convex β-fat objects. Then the number of edges in $\mathcal{G}^{(1)}(\mathcal{P})$ is $O(n/\beta)$.*

Proof. We will charge each arc in $\mathcal{G}^{(1)}(\mathcal{P})$ to an object and then use a packing argument to show that the number of arcs in $\mathcal{G}^{(1)}(\mathcal{P})$ charged to each object is $O(1/\beta)$.

We project all objects onto the xy-plane, making them convex fat objects. In this setting, we say that one object is above another if the original objects satisfy this relationship.

Recall that for a planar object o, its size is defined as the radius of its smallest enclosing disk. Consider an arc (P_i, P_j) in $\mathcal{G}^{(1)}(\mathcal{P})$. We charge the arc to the smaller of the two objects. That is, we charge the arc to P_i if $size(proj(P_i)) < size(proj(P_j))$ and to P_j otherwise. We claim that any object is charged $O(1/\beta)$ arcs. To prove this, take an arbitrary object P_j such that (P_i, P_j) is charged to P_i. Let $\rho = size(proj(P_i))$. If there is an arc in $\mathcal{G}^{(1)}(\mathcal{P})$ between P_i and P_j, then $proj(P_j)$ intersects $proj(P_i)$. Let p be a point in this intersection. Then a circle centered at p with radius ρ is centered in $proj(P_j)$ and does not fully enclose $proj(P_j)$, or else $proj(P_j)$ would have a smallest enclosing circle that is smaller than or equal to that of $proj(P_i)$. Thus, this circle contains at least $\beta\pi\rho^2$ units of area of $proj(P_j)$ by the definition of fatness. Also, this circle is completely enclosed in a circle of radius 2ρ centered at the center of the smallest enclosing disk of $proj(P_i)$. This is illustrated in Figure 1.

Since all objects $proj(P_j)$ where P_j is above P_i and $sep(P_i, P_j) = 1$ must be disjoint by Lemma 9, and because each must have at least $\beta\pi\rho^2$ units of area inside a disk that has $4\pi\rho^2$ units of area, there can be only $4/\beta$ edges of $\mathcal{G}^{(1)}(\mathcal{P})$ charged to P_i. We must double this number to account for objects P_j below P_i such that (P_j, P_i) is charged to P_i. Therefore, we get an upper bound on the number of arcs charged to P_i of $8/\beta$. Finally, since there are n objects, $\mathcal{G}^{(1)}(\mathcal{P})$ can have at most $8n/\beta$ edges, which is $O(n/\beta)$. □

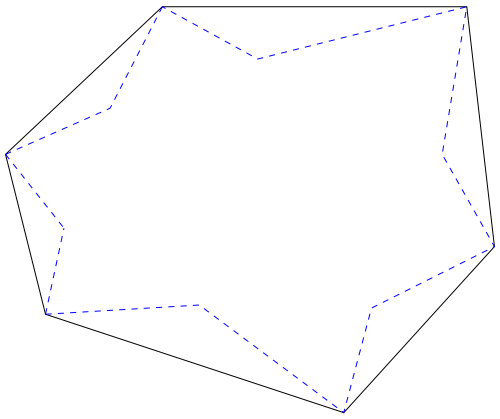


FIG. 2. *A projection of a polyhedron with witness edges added.*

5. Computing depth orders. We now present the algorithm for finding the depth order of a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n disjoint β-fat convex polyhedra. In contrast to the proof of Theorem 2, we require the complexity of the projection of each object to be constant.

Witness edges. One of the basic steps that we need to perform repeatedly in our algorithm will be to find polyhedra that are above a query polyhedron. To facilitate this, we will add so-called *witness edges* inside the projection of each P_i. They are defined as follows.

Let β' be defined so that each member of $\{proj(P_i)|P_i \in \mathcal{P}\}$ is β'-fat. By Lemma 1 we know that $\beta' = \Omega(\beta)$. Also let $\mathcal{C} = \{0, \alpha, 2\alpha, \ldots, c\alpha\}$, where $\alpha = (\beta'\pi)/8$ and $c = \lfloor 2\pi/\alpha \rfloor$. We call the directions in $\mathcal{C}$ *canonical directions*. We require the witness edges to have the following properties. Let W_i and W_j be the sets of witness edges constructed for P_i and P_j, respectively.

(i) Each witness edge has one of the canonical directions.

(ii) For any pair of polyhedra P_i and P_j, we have that $proj(P_i)$ intersects $proj(P_j)$ if and only if at least one of the following is true:

- A vertex of $proj(P_i)$ is inside $proj(P_j)$, or a vertex of $proj(P_j)$ is inside $proj(P_i)$.
- A witness edge in W_i crosses a witness edge in W_j.

The construction of the set W_i of witness edges for P_i is done as follows. For each edge $e = vw$ of $proj(P_i)$ we add to W_i two witness edges e' and e'' that are incident to v and w, respectively, extend into the interior of P_i, and form a triangle with e. The directions of the witness are chosen from the canonical directions, such that the angles that e' and e'' make with e are minimal; see Figure 2. We claim that if we add the witness edges in this manner, they have the required properties. The first property holds by construction, so it remains to prove the second property. We first argue that the witness edges lie completely inside $proj(P_i)$, which implies that the "if" part of the second property holds.

LEMMA 10. *The witness edges in W_i lie completely inside $proj(P_i)$.*

Proof. Let e be the edge for which we are adding witness edges. Let p be the midpoint of e and consider the circle C with center p and diameter equal to the length of e. Suppose an edge of $proj(P_i)$ intersects the region bounded by e' and e''. Note that this region must be inside the lighter region in Figure 3 by the minimal-angle

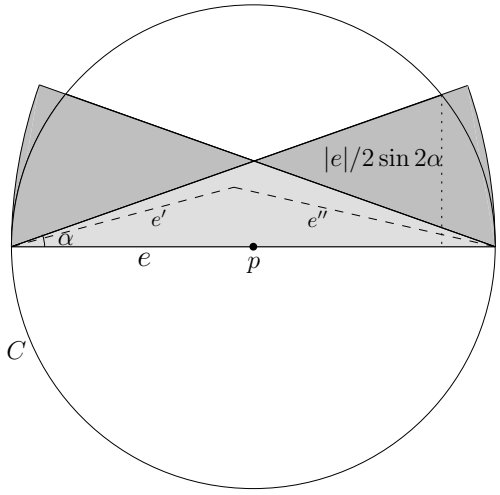


FIG. 3. *No edge of the polygon may enter the light gray region.*

condition which implies that the angles that e makes with e' and e'' are at most α. Then, by convexity of $proj(P_i)$, we know that $proj(P_i) \cap C$ must be completely inside the union of the triangular wedges in Figure 3. These wedges have area at most $\beta'\pi|e|^2/8$ inside C. Hence, $area(proj(P_i) \cap C) < \beta'\pi|e|^2/4$, contradicting our assumption that $proj(P_i)$ is β'-fat. □

The following lemma, which follows directly from Lemma 4, finishes the proof that the witness edges have the required properties.

LEMMA 11. *If $proj(P_i)$ intersects $proj(P_j)$ and $proj(P_i)$ does not contain a vertex of $proj(P_j)$ or vice versa, then a witness edge from P_i intersects a witness edge from P_j.*

The algorithm. The general idea of our algorithm is as follows. By Lemma 8 it is sufficient to find all pairs of objects P_i, P_j of separation 1 in the depth-order graph. Such a pair of objects must, of course, intersect in the projection. Thus, ideally we would like to find among all pairs P_i, P_j whose projections intersect the pairs of separation 1. Our algorithm does not quite achieve this—it will find more pairs, but the number of extra pairs we find will be small. Lemma 11 suggests that the task of finding the intersecting pairs of projections can be broken into two parts: finding pairs for which there is a vertex of the projection of one polyhedron inside the projection of another and finding crossing pairs of witness edges.

Below we give a more detailed description of the algorithm. The algorithm will find a set A of arcs—a superset of the arcs (P_i, P_j) for objects of separation 1—and then topologically sort the graph $\mathcal{G}^* = (\mathcal{P}, A)$. Initially A is empty.

1. For every vertex v of each object $P_i \in \mathcal{P}$, find the objects $P^b(v)$ and $P^a(v)$ that are directly below and above v, respectively, and add the arcs $(P^b(v), P_i)$ and $(P_i, P^a(v))$ to A.
2. Sort the objects by decreasing size so that $size(proj(P_1)) \geq \cdots \geq size(proj(P_n))$, and define $\mathcal{S}_i = \{P_1, \ldots, P_i\}$.
3. For every witness edge e associated with each P_i, find a set $\mathcal{P}(e)$ consisting of objects $P_j \in \mathcal{S}_{i-1}$ with the following properties:
   (P1) Each $P_j \in \mathcal{P}(e)$ has a witness edge that intersects e.
   (P2) Each $P_j \in \mathcal{P}(e)$ is above P_i.

(P3) Each $P_j \in \mathcal{S}_{i-1}$ with $sep(P_i, P_j) = 1$ that satisfies (P1) and (P2) is a member of $\mathcal{P}(e)$.

For each P_i, add the set of arcs $\{(P_i, P_j) : P_j \in \mathcal{P}(e)$ and e is a witness edge of $P_i\}$ to A.

4. Repeat step 3 with "below" substituted for "above" and the directions of the arcs added reversed.
5. Topologically sort the graph $\mathcal{G}^* = (\mathcal{P}, A)$ and report the order.

LEMMA 12. *The order reported by the algorithm is a valid depth order for $\mathcal{P}$ if a depth order exists.*

Proof. Assume a depth order exists for $\mathcal{P}$. It follows directly from the construction that every arc added to the set A is also an arc in the depth-order graph $\mathcal{G}(\mathcal{P})$. It remains to argue that A is a superset of the set of arcs in the graph $\mathcal{G}^{(1)}(\mathcal{P})$.

Consider an arc (P_i, P_j) in $\mathcal{G}^{(1)}(\mathcal{P})$. If there is a vertex of $proj(P_i)$ in $proj(P_j)$ (or vice versa) then, because $sep(P_i, P_j) = 1$, that vertex is directly below P_j (resp., above P_i). Hence, the arc is found in step 1. By Lemma 11, the remaining case is that a witness edge of $proj(P_i)$ intersects a witness edge from $proj(P_j)$. Without loss of generality, assume P_i is smaller than P_j. Hence, $P_j \in \mathcal{S}_{i-1}$. Since (P_i, P_j) is an arc in $\mathcal{G}^{(1)}(\mathcal{P})$, $sep(P_i, P_j) = 1$. By condition (P3), the arc will be found in step 3 or 4, depending on whether P_j is above or below P_i. □

Step 1 can be carried out efficiently using the ray-shooting data structure presented in the previous section. Hence, it remains to describe step 3 in more detail. This step will be performed as follows. We will treat each $P_2, \ldots, P_n$ in order. When we have to handle P_i, we will make sure we have a data structure available that we can query with each witness edge e of P_i and that will then report the set $\mathcal{P}(e)$. After having queried with all witness edges of P_i, we insert P_i into the data structure and proceed with P_{i+1}. Next we describe this data structure.

The witness-edge-intersection data structure. Consider the set of all witness edges of the objects in $\mathcal{P}_{i-1}$. These witness edges have canonical directions, so we can partition them into $|\mathcal{C}|$ subsets depending on their directions. The query segment e has one of the canonical directions as well. Hence, we construct for each subset $|\mathcal{C}|$ different data structures, one for each query direction. We now describe the structure for one such subset, call it W, and a fixed query direction.

Assume without loss of generality that the witness edges in W are all horizontal, and that the query edge e is vertical. The structure is a multilevel data structure defined as follows.

- The top level of the data structure is a segment tree $\mathcal{T}$ on the projections of the edges in W onto the x-axis. Note that each node ν in $\mathcal{T}$ corresponds to a vertical slab in the plane.
- Let $W(\nu)$ denote the edges in W whose projection is in the canonical subset of ν. Such an edge crosses the slab of ν but not the slab of the parent of ν. We store the edges in $W(\nu)$ in a balanced binary tree $\mathcal{T}(\nu)$, ordered according to their y-coordinates. We call this the "slab tree." So far our structure is just a standard two-level tree to perform intersection queries with vertical segments in a set of horizontal segments in the plane [10].
- Let μ be a node in $\mathcal{T}(\nu)$. Let $\mathcal{P}(\mu)$ denote the subset of objects that have a witness edge in the subtree rooted at μ. The node μ represents a rectangular[3] region $R(\mu)$ that is bounded by two slab boundaries and the topmost and

[3]This is only true because we assumed the edges in W are horizontal and the query edge is vertical. In general, μ will represent a parallelogram, but this does not influence the arguments.

bottommost edge stored in the subtree rooted at μ. We associate with μ a reduced subset $\overline{\mathcal{P}(\mu)} \subset \mathcal{P}(\mu)$ of the objects, in the following way: $P_j \in \overline{\mathcal{P}(\mu)}$ if and only if $P_j \in \mathcal{P}(\mu)$ and $size(proj(P_j)) \geq size(R(\mu))/2\sqrt{2}$.

By Lemma 2 we can find a set $Q(\mu)$ consisting of $O(1/\beta^2)$ points such that the projection of any object $P_j \in \overline{\mathcal{P}(\mu)}$ is stabbed. We arbitrarily assign each $P_j \in \overline{\mathcal{P}(\mu)}$ to one of the points q it contains, and we associate a balanced binary search tree $\mathcal{T}(q)$ with each point q on the associated objects, where the sorting order is defined by the height of the objects along the vertical line through q.

This finishes the description of the data structure. Next we describe the algorithms to query the structure and to insert an object.

LEMMA 13. *With the structure described above, we can find the set $\mathcal{P}(e)$ referred to in Step 3 of the depth-order algorithm in $O((1/\beta^3)\log^3 n)$ time. Furthermore, the set $\mathcal{P}(e)$ contains $O((1/\beta^3)\log^2 n)$ objects.*

Proof. Recall that we actually have to query $|\mathcal{C}| = O(1/\beta)$ different versions of the structure. We focus on the time spent in one of these structures.

To perform a query with a witness edge e belonging to an object P_i, we search with e in the first two levels of the tree in the standard way. This gives us $O(\log^2 n)$ nodes μ whose subtrees contain exactly those edges that intersect e. At each node μ, we use the trees $\mathcal{T}(q)$ for $q \in Q(\mu)$ to find the lowest object that is above P_i. We can search in $\mathcal{T}(q)$ since P_i is known to intersect all objects in $\mathcal{P}(q)$ in the projection. Hence, at μ, we find $|Q(\mu)|$ objects in $O(|Q(\mu)|\log n)$ time in total. The query time and the bound on the size of $\mathcal{P}(e)$ follow.

It remains to argue that the reported set has the required properties. Properties (P1) and (P2) follow immediately from the definition of the data structure and query algorithm. Furthermore, when we query a tree $\mathcal{T}(q)$ we can indeed restrict our attention to the lowest object that is above P_i, because the other objects P_j will either be below P_i or have $sep(P_i, P_j) > 1$. Hence, to prove (P3) it is sufficient to argue that any P_j satisfying (P1) and (P2) and with $sep(P_i, P_j) = 1$ will be a member of one of the sets $\overline{\mathcal{P}(\mu)}$. We know that the object will be a member of $\mathcal{P}(\mu)$ for a visited node μ.

Suppose for a contradiction that $P_j \notin \overline{\mathcal{P}(\mu)}$. This means that we must have $size(proj(P_j)) < size(R(\mu))/2\sqrt{2}$. This can happen only when $size(proj(P_j))$ is less than $d/2$, where d is the distance between the top and bottom edges of $R(\mu)$, because P_j crosses the slab of which $R(\mu)$ is a part. On the other hand, when we reach a node μ in the slab tree by querying with a witness edge e of $\mathcal{P}_i$, we have $size(proj(P_i)) \geq \text{length}(e)/2 \geq d/2$. This contradicts that when we query with a witness edge e of P_i, all objects P_j in the data structure have $size(proj(P_j)) \geq size(proj(P_i))$. □

LEMMA 14. *An object P_i can be inserted in $O((1/\beta)\log^2 n(\log n + 1/\beta^2))$ time into the structure.*

Proof. Each of the $O(1)$ witness edges of P_i has to be inserted into $|\mathcal{C}| = O(1/\beta)$ structures. To insert a witness edge, we first find each node μ in a slab tree whose canonical subset contains the witness edge. We test if $size(P_j) \geq size(R(\mu))/2$ and, if so, find a point $q \in Q(\mu)$ that is contained in $proj(P_i)$ and insert P_i into the tree $\mathcal{T}(q)$. This takes $O(\log^2 n(\log n + 1/\beta^2))$ time per structure, thus $O((1/\beta)\log^2 n(\log n + 1/\beta^2))$ time in total. □

From Lemmas 13 and 14, we see that steps 3 and 4 of the depth-order algorithm can be performed in $O((1/\beta^3)n\log^3 n)$ time in total. We get the following theorem.

THEOREM 3. *Let $\mathcal{P}$ be a collection of n disjoint constant-complexity β-fat convex polyhedra in $\mathbb{R}^3$. Then we can compute a depth order for $\mathcal{P}$ in time $O((1/\beta^3)n\log^3 n)$ if it exists.*

6. Verifying depth orders. In order for our algorithm to be complete, it should output the correct depth order if one exists, but it should also not output an incorrect depth order if no depth order exists. Unfortunately the algorithm of the previous section does not necessarily detect cycles in the $\prec$-relation. Hence, we present an algorithm for checking whether a given order is correct.

We use the general approach by De Berg, Overmars, and Schwarzkopf [11] for verifying depth orders. Let $L = P_1, \ldots, P_n$ be the given order. Define $L_1 = \{P_1, \ldots, P_{\lfloor n/2 \rfloor}\}$ and $L_2 = \{P_{\lfloor n/2 \rfloor + 1}, \ldots, P_n\}$. We first check if any object in L_1 is above any object in L_2. Clearly, if the answer is "yes," then the given ordering is not valid. Otherwise, we verify the lists L_1 and L_2 recursively. If $T(\beta, n)$ is the amount of time to check the objects in L_1 against those in L_2, then the overall algorithm takes $O(T(\beta, n)\log n)$ time. We shall see that $T(\beta, n) = O((1/\beta^2)n\log^2 n)$, so the algorithm for verifying the depth order takes $O((1/\beta^2)n\log^3 n)$ time. Next we describe how to check the objects in L_1 against those in L_2.

First we introduce a new type of witness edge. The difference from the witness edges in section 5 is that the new witness edges will have canonical directions in three dimensions, rather than in the projection. In order to achieve this we need the following lemma from Aronov, De Berg, and Gray [4].

LEMMA 15 (see [4]). *Let $\sigma := \lceil 54\sqrt{3}/\beta \rceil$. For any convex β-fat object o in $\mathbb{R}^3$, there exist concentric axis-aligned cubes $C^-(o)$ and $C^+(o)$ with $C^-(o) \subseteq o \subseteq C^+(o)$ such that*

$$\frac{size(C^+(o))}{size(C^-(o))} = \sigma\,.$$

Assume we are given $C^-(o)$ and $C^+(o)$ for object o. We partition each face of $C^+(o)$ into squares with side length equal to the side length of $C^-(o)$. For each facet f of $C^-(o)$ and each square on the corresponding facet of $C^+(o)$, we sweep f so that it coincides with the square; see Figure 4(i). The sweeping directions form the set of canonical directions. There are at most σ^2 different directions that a facet of $C^-(o)$ can be swept in, so we have $O(1/\beta^2)$ canonical directions. We denote an arbitrary member of this set of directions by $\vec{d}$. Note that the set of canonical directions thus obtained does not depend on o, only on the value σ, which is specified by the fatness factor β.

For each P_i we construct a set W_i of witness edges as follows. First, we add the edges of $C^-(P_i)$ to W_i. Second, for each silhouette vertex v of P_i (a silhouette vertex is a vertex whose projection is a boundary vertex of the projection of P_i), we add an edge e_v that connects v to one of the facets of $C^-(P_i)$. This edge is allowed to be in any of the canonical directions as long as it reaches a facet of $C^-(P_i)$. We can be certain that at least one direction works for v since there must be at least one pair consisting of a facet f of $C^-(P_i)$ and a square on a facet of $C^+(P_i)$ such that v is hit when sweeping f to that square.

We also add some vertices to P_i that we call *witness vertices* as follows (see Figure 4(ii)). For each witness edge e, we add the intersection point of $proj(e)$ and $\partial\, proj(C^-(P_i))$, lifted back to e, to the set of witness vertices for P_i. Moreover, if the projected witness edges of two consecutive silhouette vertices intersect, then we lift

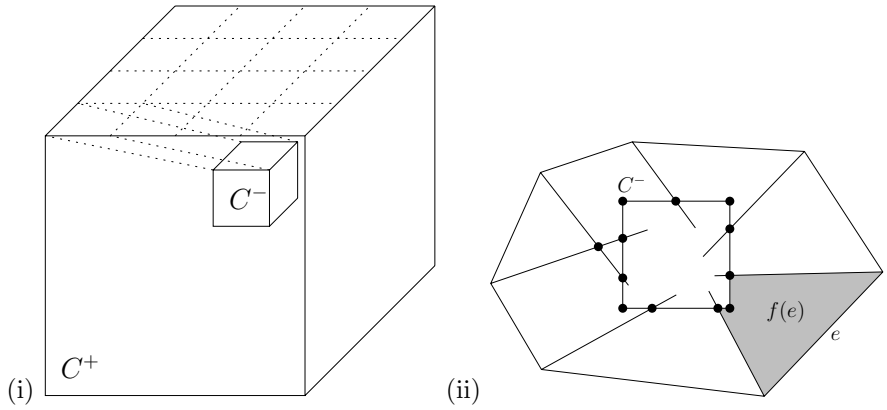


FIG. 4. (i) *One of the canonical directions.* (ii) *Projection of the new witness edges and witness vertices.*

those intersection points to either of the witness edges and make the resulting point a witness vertex. Finally, we add the vertices of $C^-(P_i)$ to the set of witness vertices.

LEMMA 16. *The witness edges satisfy the following properties.*

(i) *Each witness edge has one of the canonical directions.*

(ii) *For any pair of polyhedra P_i and P_j, we have that $proj(P_i)$ intersects $proj(P_j)$ if and only if at least one of the following is true:*

- *A projected witness or silhouette vertex of P_i is inside $proj(P_j)$, or a projected witness or silhouette vertex of P_j is inside $proj(P_i)$.*
- *A projected witness edge in W_i crosses a projected witness edge in W_j.*

Proof. Property (i) is satisfied by construction. Also, if one of the two conditions in property (ii) is satisfied, then the projections of P_i and P_j must intersect since they share a point. Therefore, we will concentrate on proving that a projected witness edge in W_i crosses a projected witness edge in W_j assuming that $proj(P_i) \cap proj(P_j) \neq \emptyset$, and that no projected witness or silhouette vertex of P_i is contained in $proj(P_j)$ (or vice versa).

Since $proj(P_i)$ intersects $proj(P_j)$ and no projected silhouette vertex of one is inside the projection of the other, there must be silhouette edges of $proj(P_i)$ and $proj(P_j)$ that cross. Take one such pair of edges and call them e_i and e_j. Consider the arrangement induced by the projections of the silhouette edges and the witness edges of P_i, and let $f(e_i)$ denote the (bounded) face in this arrangement with e_i on its boundary; see Figure 4(ii). Define $f(e_j)$ similarly for the arrangement induced by the projections of the silhouette edges and the witness edges of P_j. By Lemma 4, there must be an intersection between a pair of edges from $f(e_i)$ and $f(e_j)$, neither of which is $proj(e_i)$ or $proj(e_j)$. Hence, there must be an intersection between two projected witness edges. □

It follows from Lemma 16 that there is an object from L_1 below an object from L_2 when at least one of the following conditions holds for some pair P_i, P_j with $P_i \in L_1$ and $P_j \in L_2$: a witness or silhouette vertex of P_i is below P_j, or a witness or silhouette vertex of P_j is above P_i, or a witness edge of P_i is below a witness edge of P_j. To test for the first condition, we construct a data structure for vertical ray shooting on the objects in L_2 and query it with upward rays from the witness and silhouette vertices of the objects in L_1. The second condition can be tested similarly, namely, by constructing a data structure for vertical ray shooting on the objects in L_1 and

querying it with downward rays from the witness and silhouette vertices of the objects in L_2. By Theorem 1 and the fact that the total number of witness and silhouette vertices is $O(n)$, this will take $O((1/\beta^2)\log^2 n)$ in total. To test the third condition we proceed as follows. Let $W(L_1)$ and $W(L_2)$ denote the set of all witness edges defined for the objects in L_1 and L_2, respectively. We will preprocess $W(L_2)$ into a data structure for querying with witness edges from $W(L_1)$, according to the following lemma.

LEMMA 17. *We can preprocess the set $W(L_2)$ in $O((1/\beta^2)n\log n)$ time into a data structure of size $O((1/\beta^2)n\log n)$ such that, for any witness edge $e \in W(L_1)$, we can determine in $O((1/\beta^2)\log^2 n)$ time whether any witness edge from $W(L_2)$ is above e.*

Proof. Let $W_{\vec{d}}(L_2) \subset W(L_2)$ denote the subset of witness edges of canonical direction $\vec{d}$. Note that

$$\sum_{\vec{d}} |W_{\vec{d}}(L_2)| = |W(L_2)| = O(n).$$

Define $W_{\vec{d}}(L_1)$ similarly. For each pair of directions $\vec{d}_1, \vec{d}_2$ we build a data structure on $W_{\vec{d}_1}(L_2)$ for querying with edges from $W_{\vec{d}_2}(L_1)$. (In fact, the structure can be queried with any segment with direction $\vec{d}_2$.) Assume without loss of generality that $\vec{d}_1$ is parallel to the x-axis and $\vec{d}_2$ is parallel to the y-axis. The structure is a multi-level data structure similar to the structure of section 5. The first two levels are exactly the same as for the structure in section 5: the first level is a segment tree on the x-ranges of the witness edges, and the second level is a balanced binary search tree on their y-values (in section 5 this was called the slab tree). For each canonical subset of a node in the slab tree, we store its highest witness edge. Note that the concept of "highest" is well defined since the witness edges in the canonical subset all have the same direction and the query edge will have a fixed direction as well.

A query with a witness edge $e \in W_{\vec{d}_2}(L_1)$ can be answered in $O(\log^2 n)$ time, as follows: query with the x-coordinate of e in the segment tree; for each node ν on the path, query with the y-range of e in the associated slab tree $\mathcal{T}(\nu)$; and for each selected node μ in $\mathcal{T}(\nu)$, check if the witness stored there is above e.

When we are querying with an edge e, we actually have to query in the sets $W_{\vec{d}}(L_2)$ for each canonical direction $\vec{d}$. Since there are $O(1/\beta^2)$ canonical directions, this means that the total query time is $O((1/\beta^2)\log^2 n)$.

Building the structure on $W_{\vec{d}_1}(L_2)$ for a given query direction $\vec{d}_2$ can be done in $O(|W_{\vec{d}_1}(L_2)|\log|W_{\vec{d}_1}(L_2)|)$ time. This is because the associated structures of the segment tree (the slab trees) can be built in linear time if we presort the witness edges by y-coordinate. After that we compute the highest edge for each node in a slab tree in a bottom-up fashion (the highest edge for a node is the higher of the highest edges of its two children) in linear time. Hence, the overall preprocessing time is the same as the amount of storage, which is $O(|W_{\vec{d}_1}(L_2)|\log|W_{\vec{d}_1}(L_2)|)$. Overall, the preprocessing is therefore

$$\begin{aligned}
&\textstyle\sum_{\vec{d}_1,\vec{d}_2} O(|W_{\vec{d}_1}(L_2)|\log|W_{\vec{d}_1}(L_2)|)\\
&= O(1/\beta^2)\cdot\textstyle\sum_{\vec{d}_1} O(|W_{\vec{d}_1}(L_2)|\log|W_{\vec{d}_1}(L_2)|)\\
&= O((1/\beta^2)n\log n). \qquad \square
\end{aligned}$$

Putting everything together, we get the following theorem.

THEOREM 4. *We can verify whether a given order on a set of n disjoint convex constant-complexity β-fat polyhedra in $\mathbb{R}^3$ is a valid depth order in $O((1/\beta^2)n\log^3 n)$ time.*

7. Concluding remarks. We have presented new and improved solutions to three problems on fat convex polyhedra in 3-space: vertical ray shooting, computing depth orders, and verifying depth orders. One open problem is to see if the results can be extended to fat nonconvex polyhedra or fat curved objects.

Our algorithm for verifying depth orders uses a collection of witness edges that have canonical directions in three dimensions and allow us to capture (together with a certain set of points in the objects) the above-below relation between the objects. It would be interesting to investigate if these witness edges can be useful for other problems on convex fat objects as well.

Appendix. This appendix contains a proof that was omitted.

LEMMA 3. *Let P be a β-fat convex polygon with n vertices. There is a set T of α-fat triangles that cover P where $|T| = O(n)$ and $\alpha \geq \beta/128$.*

Proof. Recall that for triangles, we use the definition that the fatness is given by the smallest angle in the triangle.

Let S be the largest possible square contained in P. Any convex subset of P that contains all of S is at least β'-fat, where $\beta' = \Theta(\beta)$, by Lemma 19 below.

We extend the edges of S until they intersect P and add vertices to P at the intersection points (see Figure 5). We let P_a denote the part of P above the (extended) top edge of S, let P_b denote the part below the bottom edge of S, let P_c denote the part to the right of the right edge of S, and let P_d denote the part to the left of the left edge of S. We will show how to cover P_a. The three other parts of P are covered similarly, and S is covered with two triangles that each have a fatness of 45°.

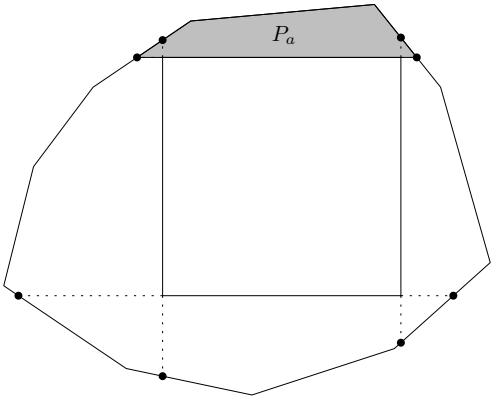


FIG. 5. *One of the subpolygons of P induced by S.*

An ear of a polygon P consists of two consecutive edges of P that have the property that a straight edge connecting the first and last vertex of the edges stays completely inside the polygon. In a convex polygon, any two consecutive edges are ears.

We cover P_a by choosing an arbitrary ear from it (except any ear that also contains the top edge of S), covering it using Lemma 18, and then replacing P by P with that ear removed. Since no part of S is ever removed, P remains fat. Thus we keep removing ears from P_a until it exactly coincides with the extended edge of S.

Since we cover the ears that we remove using the procedure from Lemma 18, we add a constant number of triangles to T per vertex, implying that $|T| = O(n)$. The exact bound on α is given by combining Lemmas 18 and 19. ∎

LEMMA 18. *An ear of a β-fat polygon P can be covered with at most four α-fat triangles that all stay inside P where $\alpha = (\beta\pi)/16$.*

Proof. In a convex polygon, an ear is a triangle formed by three consecutive vertices. Consider the ear defined by vertices v_{i-1}, v_i, and v_{i+1}. Let ϕ_{i-1}, ϕ_i, and ϕ_{i+1} be the angles at the respective vertices—see Figure 6(i). Because P is β-fat, we know that the angle between any two adjacent edges of P, and in particular the angle ϕ_i, is at least $\beta/(2\pi)$. There are three possibilities for the other two angles, ϕ_{i-1} and ϕ_{i+1}: either they are both at least α, they are both less than 2α, or one is larger than 2α and one is smaller than α. Note that these cases overlap.

Case (i). $\phi_{i-1} \geq \alpha$ and $\phi_{i+1} \geq \alpha$. In this case, the ear is trivial to cover: it is already an α-fat triangle that can be covered by a copy of itself.

Case (ii). $\phi_{i-1} < 2\alpha$ and $\phi_{i+1} < 2\alpha$. First, we add triangles to the edges $v_{i-1}v_i$ and v_iv_{i+1} where the angles of the edges of the triangles with respect to the boundary edges are at least 2α. By Lemma 10, these triangles must stay inside P as long as $\alpha \leq (\beta\pi)/16$. However, it is clear that the nonboundary vertex of these triangles must be outside the ear that we are covering. Therefore, we can place a triangle at the middle vertex of the ear with two sides that correspond to the sides of the two triangles that we just added and whose third side is the edge of the ear that goes between these two edges. This triangle completes the covering of the ear.

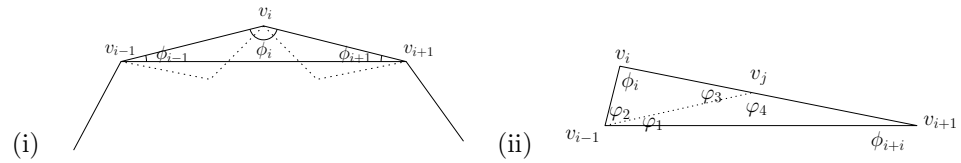


FIG. 6. (i) *Case* (ii). (ii) *Case* (iii).

Case (iii). $\phi_{i-1} > 2\alpha$ and $\phi_{i+1} < \alpha$ (or the symmetric case). See Figure 6(ii). In this case, we add an edge between the vertex that is at the large angle (v_{i-1}, without loss of generality) and the edge across from it, making vertex v_j. This splits ϕ_i into two angles φ_1 and φ_2. We place v_j such that φ_1 is exactly α. Thus, $\varphi_3 = \alpha + \phi_{i+1} > \alpha$. By assumption, $\varphi_2 > \alpha$. Thus, we can cover the triangle $v_{i-1}v_iv_j$ with a copy of itself. Triangle $v_{i-1}v_jv_{i+1}$ can be covered according to the procedure outlined for Case (ii) above. Note that in all cases, we have covered the ear with at most four α-fat triangles. ∎

LEMMA 19. *Let P be a convex β-fat polygon in $\mathbb{R}^2$ and S be the largest square contained in P. Then any convex subset P' such that $S \subseteq P' \subseteq P$ is β'-fat where $\beta' \geq \beta/(8\pi)$.*

Proof. By the results of section 3.2.1 of Van der Stappen's thesis [25], the side length of S is at least $\beta\rho/(2\sqrt{2})$, where ρ is the diameter of P.

Let $d = \overline{p_1p_2}$ be the diameter of P'. Let $S' \subseteq S$ be the largest square contained in S that has an edge parallel to d. The side length of S' is at least $\sqrt{2}/2$ times the side length of S. Let p_3 and p_4 denote the midpoints of the sides of S' parallel to d; see Figure 7.

We will make two triangles: $p_1p_3p_4$ and $p_2p_3p_4$. By convexity, both of these triangles must be completely inside P'. The sum of the area of these triangles is not

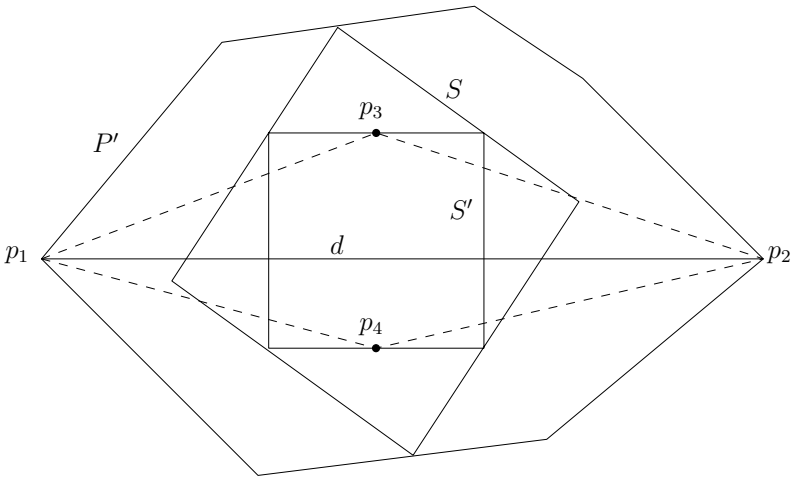


FIG. 7. *P' must be fat.*

dependent on the placement of S'—it is always $d \cdot s/2$, where s is the side length of S'.

Since P' is convex, the fatness of P' is determined by a circle placed at p_1 with radius d [25]. The area of that circle is πd^2. Thus the fatness of P' is

$$\beta' = \frac{\frac{d \cdot s}{2}}{\pi d^2} = \frac{s}{2d\pi} \geq \frac{\beta\rho}{8d\pi} \geq \frac{\beta}{8\pi}$$

since $d \leq \rho$. □

REFERENCES

[1] P. K. AGARWAL, M. DE BERG, D. HALPERIN, AND M. SHARIR, *Efficient generation of k-directional assembly sequences*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1996, pp. 122–131.

[2] P. K. AGARWAL, M. J. KATZ, AND M. SHARIR, *Computing depth orders for fat objects and related problems*, Comput. Geom., 5 (1995), pp. 187–206.

[3] P. K. AGARWAL AND J. MATOUŠEK, *On range-searching with semi-algebraic sets*, Discrete Comput. Geom., 11 (1993), pp. 393–418.

[4] B. ARONOV, M. DE BERG, AND C. GRAY, *Ray shooting and intersection searching amidst fat convex polyhedra in 3-space*, in Proceedings of the 22nd Annual ACM Symposium on Computational Geometry, 2006, pp. 88–94.

[5] M. DE BERG, *Ray Shooting, Depth Orders and Hidden Surface Removal,* Lecture Notes in Comput. Sci. 703, Springer-Verlag, Berlin, 1993.

[6] M. DE BERG, *Vertical ray shooting for fat objects*, in Proceedings of the 21st Annual ACM Symposium on Computational Geometry, 2005, pp. 288–295.

[7] M. DE BERG, H. DAVID, M. J. KATZ, M. OVERMARS, A. F. VAN DER STAPPEN, AND J. VLEUGELS, *Guarding scenes against invasive hypercubes*, Comput. Geom., 26 (2003), pp. 99–117.

[8] M. DE BERG AND C. GRAY, *Vertical ray shooting and computing depth orders for fat objects*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 494–503.

[9] M. DE BERG, M. KATZ, F. VAN DER STAPPEN, AND J. VLEUGELS, *Realistic input models for geometric algorithms*, Algorithmica, 34 (2002), pp. 81–97.

[10] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

[11] M. DE BERG, M. OVERMARS, AND O. SCHWARZKOPF, *Computing and verifying depth orders*, SIAM J. Comput., 23 (1994), pp. 437–446.

[12] M. de Berg and M. Streppel, *Approximate range searching using binary space partitions*, in Proceedings of the IARCS Conference on Foundation Software Technology and Theoretical Computational Science (FSTTCS), 2004, pp. 110–121.
[13] B. Chazelle and L. J. Guibas, *Fractional cascading*: I. *A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.
[14] B. Chazelle and L. J. Guibas, *Fractional cascading*: II. *Applications*, Algorithmica, 1 (1986), pp. 163–191.
[15] C. A. Duncan, *Balanced Aspect Ratio Trees,* Ph.D. thesis, The Johns Hopkins University, Baltimore, MD, 1999.
[16] C. A. Duncan, M. T. Goodrich, and S. Kobourov, *Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 300–309.
[17] A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir, *Dynamic data structures for fat objects and their applications*, Comput. Geom., 15 (2000), pp. 215–227.
[18] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics. Principles and Practice*, 2nd ed., Addison-Wesley, Bonn, Germany, 1990.
[19] M. J. Katz, 3-*D vertical ray shooting and* 2-*D point enclosure, range searching, and arc shooting amidst convex fat objects*, Comput. Geom., 8 (1998), pp. 299–316.
[20] M. J. Katz, *personal communication*, 2005.
[21] M. J. Katz, M. Overmars, and M. Sharir, *Efficient hidden surface removal for objects with small union size*, Comput. Geom., 2 (1992), pp. 223–234.
[22] M. van Kreveld, *On fat partitioning, fat covering and the union size of polygons*, Comput. Geom., 9 (1998), pp. 197–210.
[23] M. Pellegrini, *Ray shooting on triangles in* 3*-space*, Algorithmica, 9 (1993), pp. 471–494.
[24] M. Pellegrini, *Ray shooting and lines in space*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC, Boca Raton, FL, 1997, pp. 599–614.
[25] A. F. van der Stappen, *Motion Planning Amidst Fat Obstacles*, Ph.D. thesis, Utrecht University, Utrecht, The Netherlands, 1994.
[26] A. F. van der Stappen, D. Halperin, and M. H. Overmars, *The complexity of the free space for a robot moving amidst fat obstacles*, Comput. Geom., 3 (1993), pp. 353–373.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 276–302

CONVERGENCE OF AUTONOMOUS MOBILE ROBOTS WITH INACCURATE SENSORS AND MOVEMENTS*

REUVEN COHEN† AND DAVID PELEG‡

Abstract. A number of recent studies concern algorithms for distributed control and coordination in systems of autonomous mobile robots. The common theoretical model adopted in these studies assumes that the positional input of the robots is obtained by perfectly accurate visual sensors, that robot movements are accurate, and that internal calculations performed by the robots on (real) coordinates are perfectly accurate as well. The current paper concentrates on the effect of weakening this rather strong set of assumptions and replacing it with the more realistic assumption that the robot sensors, movement, and internal calculations may have slight inaccuracies. Specifically, the paper concentrates on the ability of robot systems with inaccurate sensors, movements, and calculations to carry out the task of convergence. The paper presents several impossibility theorems, limiting the inaccuracy levels that still allow convergence, and prohibiting a general algorithm for gathering, namely, meeting at a point, in a finite number of steps. The main positive result is an algorithm for convergence under bounded measurement, movement, and calculation errors.

Key words. autonomous robots, fault tolerance, inaccurate sensors

AMS subject classifications. 68Q22, 70B15

DOI. 10.1137/060665257

1. Introduction: Background.

Distributed systems consisting of autonomous mobile robots (also known as *robot swarms*) are motivated by the idea that instead of using a single, highly sophisticated and expensive robot, it may be advantageous in certain situations to employ a group of small, simple, and relatively cheap robots. This approach is of interest for a number of reasons. Multiple robot systems may be used to accomplish tasks that *cannot* be achieved by a single robot. Such systems usually have decreased cost due to the simpler individual robot structure. These systems can be used in a variety of environments where the acting (human or artificial) agents may be at risk, such as military operations, exploratory space missions, cleanups of toxic spills, fire fighting, search and rescue missions, and other hazardous tasks. In such situations, a multiple robot system has a better chance of successfully carrying out its mission (while possibly accepting the loss or destruction of some of its robots) than a single irreplaceable robot. Such systems may also be useful for carrying out simple repetitive tasks that humans may find extremely boring or tiresome.

Subsequently, studies of autonomous mobile robot systems can be found in different disciplines, from engineering to artificial intelligence (e.g., [17, 23, 24]). Notable engineering efforts in these directions include the cellular robotic system [18],

*Received by the editors July 17, 2006; accepted for publication (in revised form) October 7, 2007; published electronically April 11, 2008. A preliminary version of this paper appeared in *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science*, 2006, pp. 549–560.

http://www.siam.org/journals/sicomp/38-1/66525.html

†Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 (reuven@mit.edu). Current address: Department of Mathematics, Bar-Ilan University, Ramat-Gan 52900, Israel (reuven@macs.biu.ac.il). Part of this work was done while this author was visiting the Weizmann Institute of Science. This author's research was supported in part by the Pacific Theaters Foundation.

‡Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel (david.peleg@weizmann.ac.il). This author's research was supported by the Israel Science Foundation (grant 693/04).

swarm intelligence [5], and the self-assembly machine [20]. Relevant studies in the realm of artificial intelligence include social interaction and intelligent behavior [19], behavior-based robot systems [4, 21, 22], multirobot learning [23, 24], and ant robotics [30]. (A survey on this area is presented in [6].)

A number of recent studies on autonomous mobile robot systems focus on algorithms for distributed control and coordination. Most of the studies mentioned above handled control and coordination issues following experimental, empirical, or architectural approaches, which resulted in the design of heuristic protocols. Algorithmic aspects were handled implicitly, with little or no emphasis on formal analysis of the correctness, termination, or complexity properties of the algorithms. During the last decade, various coordination related issues have been studied from a distributed computing point of view (cf. [2, 15, 26, 27, 28]). The approach is to propose suitable computational models and analyze the minimal capabilities the robots must possess in order to achieve their common goals. The basic model studied in these papers can be summarized as follows. The robots execute a given algorithm in order to achieve a prespecified task. Each robot in the system is assumed to operate individually in simple cycles consisting of three steps:

(1) *Look*. Determine the current configuration by identifying the locations of all visible robots and marking them on your private coordinate system (the model may assume perfect or limited visibility range);

(2) *Compute*. Execute the given algorithm, resulting in a goal point p_G; and

(3) *Move*. Travel towards the point p_G.

Weak and strong model assumptions. Due to the focus on cheap robot design and the minimal capabilities allowing the robots to perform some tasks, most papers in this area (cf. [8, 14, 15, 27]) assume the robots to be rather limited. Specifically, the robots are assumed to be indistinguishable, so when looking at the current configuration, a robot cannot tell the identity of the robots at each of the points (apart from itself). Furthermore, the robots are assumed to have no means of direct communication. This gives rise to challenging "distributed coordination" problems since the only permissible communication is based on "positional" or "geometric" information exchange, yielding an interesting variant of the classical (direct communication-based) distributed model.

Moreover, the robots are also assumed to be *oblivious* (or memoryless); namely, they cannot remember their previous states, their previous actions, or the previous positions of the other robots. Hence the algorithm employed by the robots for the Compute step cannot rely on information from previous cycles, and its only input is the current configuration. While this is admittedly an overrestrictive and unrealistic assumption, developing algorithms for the oblivious model still makes sense in various settings, for two reasons. First, solutions that rely on nonobliviousness, namely, storing information regarding history, become more complex when they need to be applied in a dynamic environment where the robots have different start times, i.e., they are activated in different cycles, or robots might be added to or removed from the system dynamically. In dynamic environments an algorithm relying on the outcome of previous rounds may enter an inconsistent state and fail to reach the desired result. In contrast, oblivious solutions are insensitive to changing conditions and thus require no modification in dynamic settings. Second, any algorithm that works correctly for oblivious robots is inherently self-stabilizing; i.e., it withstands transient errors that alter the robots' local states.

On the other hand, the robot model studied in the literature includes the following *overly strong* assumptions:

- when a robot observes its surroundings, it obtains a perfect map of the locations of the other robots relative to itself;
- when a robot performs internal calculations on (real) coordinates, the outcome is exact (infinite precision) and suffers no numerical errors; and
- when a robot decides to move to a point p, it progresses on the straight line connecting its current location to p, stopping either precisely at p or at some earlier point on the straight line segment leading to it.

All of these assumptions are unrealistic. In practice, the robot measurements suffer from nonnegligible inaccuracies in both distance and angle estimations. (The most common range sensors in mobile robots are sonar sensors. The accuracy in range estimation of the common models is about $\pm 1\%$, and the angular separation is about 3°; see, e.g., [25]. Other possible range detectors are based on laser range detection, which is usually more accurate than the sonar, and on stereoscopic vision, which is usually less accurate.) The same applies to the precision of robot movements. Due to various mechanical factors such as unstable power supply, friction, and force control, the exact distance a robot traverses in a single cycle is hard to control or to even predict to a high degree of accuracy. This makes most previous algorithms proposed in the literature inapplicable in most practical settings. Finally, the robots' internal calculations cannot be assumed precise, for a variety of well-understood reasons such as convergence rates of numerical procedures, truncated numeric representations, rounding errors, and more.

In this paper we address the issue of imperfections in robot measurements, calculations, and movements. Specifically, we replace the unrealistic assumptions described above with more appropriate ones, allowing for measurement, calculation, and movement inaccuracies, and show that efficient algorithmic solutions can still be obtained in the resulting model.

We focus on the *gathering* and *convergence* problems, which have been extensively studied in the common (fully accurate) model (cf. [7, 8, 15, 27, 28]). The *gathering* problem is defined as follows. Starting from any initial configuration, the robots should occupy a single point within a finite number of steps. The closely related *convergence* problem requires the robots to converge to a single point, rather than reach it (namely, for every $c > 0$ there must be a time t_c by which all robots are within a distance of at most c of each other).

It is important to note that analyzing the effect of errors is not merely of theoretical value. In section 3.1 we show that gathering cannot be guaranteed in environments with errors. In section 3.2 we illustrate how certain existing geometric algorithms, including ones designed for fault tolerance, fail to guarantee even convergence in the presence of small errors. We also show (in Theorem 6.10) that the standard center-of-gravity algorithm may also fail to converge when errors occur.

Related work. A number of problems concerning coordination in autonomous mobile robot systems have been considered so far in the literature. The gathering problem was first discussed in [27, 28] in the semisynchronous model. It was proved that it is impossible to achieve gathering of *two* oblivious autonomous mobile robots that have no common sense of orientation under the semisynchronous model. Also, an algorithm was presented in [28] for gathering $N \geq 3$ robots in the semisynchronous model. In the asynchronous model, an algorithm for gathering $N = 3, 4$ robots is brought in [8, 15], and an algorithm for gathering $N \geq 5$ robots has recently been described in [7]. Fault tolerant gathering algorithms (in the crash and Byzantine fault models) were studied in [1]. The gathering problem was also studied in a system where the robots have limited visibility. The visibility conditions are modeled by means of

a *visibility graph*, representing the (symmetric) visibility relation of the robots with respect to one another; i.e., an edge exists between robots i and j if and only if i and j are visible to each other. (Note that in this model visibility is a boolean predicate and does not involve imprecisions; namely, if robot j is visible to robot i, then its precise coordinates are measured accurately.) It was shown that the problem is unsolvable in the case that the visibility graph is not connected [14]. In [2] a convergence algorithm was provided for any N in limited visibility systems. The effect of sensor and control errors was also studied numerically in [2]. The natural gravitational algorithm based on going to the center of gravity and its convergence properties were studied in [10, 11] in the semisynchronous and asynchronous models, respectively.

Another class of problems studied, e.g., in [12, 13, 26, 27, 28], concerns the formation of geometric patterns. The robots are required to arrange themselves (approximately) in a simple geometric form (such as a circle, a simple polygon, or a line segment) within a finite number of cycles. Algorithms were presented for enabling a group of robots to achieve such self-arrangement and even spread itself nearly evenly along the form shaped. Flocking (or "following the leader") is yet another task studied in the literature, where the robots are required to follow the movements of a predefined leader [15]. Distributed *search* by a group of robots after a (static or moving) target in a specified region is a potentially useful application for mobile robot systems. An important subtask in this context is achieving an *even distribution* of the robots, namely, requiring the robots to spread out uniformly over a specified region of a simple geometric shape. This problem has been studied in [26]. A related task of interest is *partitioning*, where the robots are required to organize themselves into a number of groups. An algorithm for this problem was also presented in [26]. A final example is the *wake-up* task, where a single initially awake robot must wake up all the others (with the help of those already wakened). The *freeze-tag* problem, which is a paradigm for the distributed wakeup of a group of robots, was presented in [3, 29] and given a number of approximation algorithms.

Our results. In this paper we study the convergence problem in the common semisynchronous model where the robots' only inputs are obtained by inaccurate visual sensors, and their movements and internal calculations may be inaccurate as well. In section 3 we present several impossibility theorems in a model allowing measurement errors, limiting the inaccuracy allowing convergence, and prohibiting a general algorithm for gathering in a finite number of steps. In section 4 we present an algorithm for convergence under bounded measurement error and prove its correctness, first in the fully synchronous model and then in the semisynchronous model. In section 5 we describe how movement and calculation errors can be treated. In section 6 we consider the fully asynchronous model, where we analyze our algorithm in the one-dimensional case and compare it with the ordinary center-of-gravity algorithm. The main contributions are summarized in Table 1. We remark that the paper does not explicitly concern the question of establishing the convergence rate of our algorithm. Note, however, that our convergence proofs do imply certain lower bounds on the convergence rate (that is, upper bounds on the time to halve some measure of the dispersion). Some additional comments on this issue are deferred to the conclusions section.

2. The model. Robot operation cycle. Each of the N robots i in the system is assumed to operate individually in simple cycles. Every cycle consists of three steps, Look, Compute, and Move.

- *Look*. Identify the locations of all robots in i's private coordinate system;

TABLE 1
The contribution of this paper. Results of convergence or divergence for the `Go_to_COG` *and* `RCG` *algorithms under different timing models for robots with inaccurate sensors.*

Algorithm	`Go_to_COG`	`RCG`
$\mathcal{FSYNC}$	converges (Lemma 4.2)	converges (Thm. 4.8)
$\mathcal{SSYNC}$	?	converges (Thm. 4.10)
$\mathcal{ASYNC}$	diverges (Thm. 6.10)	converges in 1 dim (Thm. 6.8)

the result of this step is a multiset of points $P = \{p_1, \ldots, p_N\}$ defining the current *configuration*. The robots are indistinguishable, so each robot i knows its own location p_i, but does not know the identity of the robots at each of the other points.

- *Compute*. Execute the given algorithm, resulting in a goal point p_G.
- *Move*. Move towards the point p_G.

A common assumption made in a number of papers dealing with this model, known as *premature stopping*, is that the robot might stop before reaching its goal point p_G, but is guaranteed to traverse at least some minimal distance s (unless it has reached the goal first). For ease of presentation, we assume throughout most of this paper (particularly, in sections 3 and 4, which deal with measurement errors), a slightly simpler model where the Move step of a robot is ensured to bring it to its goal point p_G. We handle premature stopping in section 5, when we discuss movement errors.

Note that the Look and Move steps are carried out identically in every cycle, independently of the algorithm used. The differences between different algorithms occur in the Compute step. Moreover, the procedure carried out in the Compute step is identical for all robots. If the robots are oblivious, then the algorithm cannot rely on information from previous cycles; thus the procedure can be fully specified by describing a single Compute step, and its only input is the current configuration P, giving the locations of the robots.

The synchronization model. As mentioned earlier, our computational model for studying and analyzing problems of coordinating and controlling a set of autonomous mobile robots follows the well-studied *semisynchronous* ($\mathcal{SSYNC}$) model. This model is partially synchronous in the sense that all robots operate according to the same clock cycles, but not all robots are necessarily active in all cycles. Those robots which are awake at a given cycle take a measurement of the positions of all other robots. Then they may make a computation and move instantaneously accordingly. The activation of the different robots can be thought of as managed by a hypothetical scheduler, whose only fairness obligation is that each robot must be activated and given a chance to operate infinitely often in any infinite execution. On the way to establishing the result on the $\mathcal{SSYNC}$ model, we prove it first in the fully synchronous ($\mathcal{FSYNC}$) model, where each robot moves at each cycle. In section 6.2 we also discuss its performance in the fully asynchronous ($\mathcal{ASYNC}$) model, in which there is no global clock and the robots operate independently of each other, at arbitrary and possibly nonuniform rates.

Modeling measurement imprecisions. Our model assumes that the robot's location estimation is imprecise, where, in general, this imprecision can affect both distance and angle estimations. However, we make the following restrictive assumption.

Bounded imprecision assumption. The distance imprecision is bounded by some accuracy parameter ε_{d} known at the robot's design. Similarly, the imprecision in angle measurements is bounded by an accuracy parameter ε_θ (where it can always be assumed that $\varepsilon_\theta \leq \pi$).

Formally, the bound on distance imprecision means that if the true location of an observed point in i's coordinate system is $\overline{V}$ and the measurement taken by i is $\bar{v}$, then this measurement will satisfy $(1-\varepsilon_{\mathrm{d}})V < v < (1+\varepsilon_{\mathrm{d}})V$. (Throughout, for a vector $\bar{v}$, we denote by v its scalar length, $v = |\bar{v}|$. Also, capital letters are used for exact quantities, whereas lowercase letters denote the robots' views.) Similarly, the bound on angle imprecision means that the angle θ between the actual distance vector $\overline{V}$ and the measured distance vector $\bar{v}$ satisfies $\theta \leq \varepsilon_\theta$, or, alternatively, $\cos\theta = \frac{\overline{V}\bar{v}}{Vv} \geq \cos\varepsilon_\theta$.

In what follows, we consider the model $\mathcal{ERR}$, in which both types of imprecision are possible, and the model $\mathcal{ERR}^-$, where only distance estimates are inaccurate. This gives rise to six composite timing/error models, denoted $\langle \mathcal{T}, \mathcal{E} \rangle$, where $\mathcal{T}$ is the timing model under consideration ($\mathcal{FSYNC}$, $\mathcal{SSYNC}$, or $\mathcal{ASYNC}$) and $\mathcal{E}$ is the error model ($\mathcal{ERR}$ or $\mathcal{ERR}^-$).

While in reality each robot uses its own private coordinate system, for simplicity of presentation it is convenient to assume the existence of a global coordinate system (which is unknown to the robots) and use it for our notation. Throughout, we denote by $\bar{R}_j$ the location of robot j in the global coordinate system. In addition, for every two robots i and j, denote by $\overline{V}^i_j = \bar{R}_j - \bar{R}_i$ the true location of robot j from the position of robot i (i.e., the true vector from i to j), and by $\bar{v}^i_j$ the location of robot j as measured by i, *translated* to the global coordinate system. Likewise, our algorithm and its analysis will be described in the global coordinate system, although each of the robots will apply it in its own local coordinate system. As the functions computed by the algorithm are all invariant under translations and rotations, this representation does not violate the correctness of our analysis.

If the robots may have inaccuracies in distance estimation but not in directions, then i will measure $\overline{V}^i_j$ as $\bar{v}^i_j = (1+\sigma^i_j)\overline{V}^i_j$, where $-\varepsilon_{\mathrm{d}} < \sigma^i_j < \varepsilon_{\mathrm{d}}$ is the actual local error factor in distance estimation at robot i. For robots with inaccuracy in angle measurement as well, if the true distance is V^i_j, then i will measure it as $v^i_j = (1+\sigma^i_j)V^i_j$, where $-\varepsilon_{\mathrm{d}} < \sigma^i_j < \varepsilon_{\mathrm{d}}$ and the angle θ between $\overline{V}^i_j$ and $\bar{v}^i_j$ will satisfy $|\theta| \leq \varepsilon_\theta$.

Throughout, values computed at time slot t are denoted by a parameter $[t]$. Also, the actual error factor is time dependent, and its value at time t is denoted by $\sigma^i_j[t]$. The parameter t is omitted whenever it is clear from the context.

Modeling movement and calculation imprecisions. Other than inaccuracies in measurements, inaccuracies in movement and calculations should also be taken into account. For movement, we may assume that if the robot wants to move from its current location $\bar{R}_i$ to some goal point p_G, then it will move on a vector in angle of at most $\varepsilon^{\mathrm{mv}}_\theta$ from the vector $\overline{r_i p_G}$ and to any distance d in the interval $d \in [1-\varepsilon^{\mathrm{mv}}_{\mathrm{d}}, 1+\varepsilon^{\mathrm{mv}}_{\mathrm{d}}]|\overline{r_i p_G}|$. When it calculates a goal point $p_G = (x, y)$, it will have a multiplicative error of up to ε_{c}. Since in the center-of-gravity algorithms presented below, the calculation error is bounded linearly in the calculated terms, these inaccuracies can be treated as global inaccuracies in the measurement instead with the same effect. It can be seen that relative movement and calculation errors can be replaced with errors in measurement causing the same effect, so these errors can be treated using the same algorithm by recalibrating ε_{d}. Absolute errors in movement or calculation cannot be treated, since even if the robots have almost converged, they

may lead to wider spreading. Therefore, throughout most of the ensuing technical development, we will assume only measurement inaccuracies, where the treatment of movement and calculation inaccuracies can be conducted by assimilating them into the measurement errors.

Another often ignored aspect of robot motion is the existence of a consistent sense of direction. As mentioned earlier, following the commonly used model of [27, 28], it is assumed in this paper that the robots do not share a common sense of orientation. Let us remark, however, that this is an expected feature of real robots, since the robot's body is positioned in an angle dependent on the previous movement direction. Hence in practice, this feature may allow the usage of the robot's direction as a finite state memory between robot movements. The algorithms presented herein do not rely on sense of direction. However, one of the impossibility results presented below (Theorem 3.5) relies on the inability to use sense of direction.

Technical lemmas. We make use of the following technical lemmas. The first lemma says that if a point $\bar{b}$ outside the unit circle moves towards the center of the circle and stops at its boundary, then it gets closer to any point $\bar{a}$ inside the circle. (See Figure 1.)

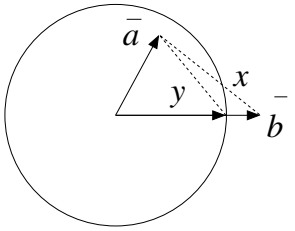


FIG. 1. *Illustration for Lemma* 2.1.

LEMMA 2.1. *For two vectors* $\bar{a}$ *and* $\bar{b}$ *with* $a \leq 1 \leq b$, *let* $x = |\bar{a} - \bar{b}|$ *and* $y = |\bar{a} - \bar{b}/b|$. *Then*

(1) $x^2 - y^2 \geq (b-1)^2 + 2(1-a)(b-1) \geq (b-1)^2$,

(2) $y \leq x$.

Proof. Note that $|\bar{a} - \bar{b}|^2 = a^2 + b^2 - 2\bar{a}\bar{b}$ and $|\bar{a} - \bar{b}/b|^2 = a^2 + 1 - 2\bar{a}\bar{b}/b$. Thus

$$\begin{aligned} |\bar{a} - \bar{b}|^2 - |\bar{a} - \bar{b}/b|^2 &= b^2 - 1 - 2\bar{a}\bar{b}(1 - 1/b) \\ &\geq b^2 - 1 - 2a(b-1) \;=\; (b-1)^2 + 2(1-a)(b-1) \\ &\geq b^2 - 1 - 2(b-1) \;=\; (b-1)^2 \,, \end{aligned}$$

and part (1) of the lemma follows. Part (2) follows from part (1), as $(b-1)^2 \geq 0$. □

LEMMA 2.2. *For vectors* $\overline{V}$ *and* $\bar{v}$, *if* $(1-\varepsilon_d)V < v < (1+\varepsilon_d)V$ *and* $\frac{\overline{V}\bar{v}}{Vv} \geq \cos(\varepsilon_\theta)$, *then* $|\overline{V} - \bar{v}| < V\sqrt{2(1+\varepsilon_d)(1-\cos\varepsilon_\theta) + {\varepsilon_d}^2}$.

Proof. Let $A = v/V$. Then $\bar{v}\overline{V} = AV^2\cos\theta$, where θ is the angle between the vectors; hence $(\bar{v} - \overline{V})^2 = \bar{v}^2 + \overline{V}^2 - 2\bar{v}\overline{V} = (1 + A^2)V^2 - 2AV^2\cos\theta$. In the range $A \in [1 - \varepsilon_{\rm d}, 1 + \varepsilon_{\rm d}]$ and $\theta \leq \varepsilon_\theta$, this function attains its maximum for $\theta = \varepsilon_\theta$ and $A = 1 + \varepsilon_{\rm d}$, whereby $(\bar{v} - \overline{V})^2 = \left(1 + (1+\varepsilon_{\rm d})^2 - 2(1+\varepsilon_{\rm d})\cos\varepsilon_\theta\right)V^2$, implying the lemma. □

The following lemma is used in conjunction with Lemma 2.1 to show that moving some fraction of the distance to the circle perimeter will reduce the squared distance by at least the same fraction of the total improvement.

LEMMA 2.3. *For any two vectors $\bar{A}$ and $\bar{B}$ satisfying $A^2 - B^2 \geq c$, for some constant c, and for any $0 \leq \mu \leq 1$, the parameterized difference $X(\mu) = A^2 - |\mu\bar{B} + (1-\mu)\bar{A}|^2$ satisfies $X(\mu) \geq \mu c$.*

Proof.

$$\begin{aligned} X(\mu) &= A^2 - |\mu\bar{B} + (1-\mu)\bar{A}|^2 \;\geq\; A^2 - (1-\mu)^2A^2 - \mu^2B^2 - 2\mu(1-\mu)AB \\ &= \mu(A^2 - B^2) + (\mu - \mu^2)(A-B)^2 \;\geq\; \mu(A^2 - B^2) \;=\; \mu c. \qquad \square \end{aligned}$$

3. The effect of measurement errors. To appreciate the importance of error analysis one must realize two facts. First, computers are limited in their computational power and therefore cannot perform perfect precision calculations. This may seem insignificant, since floating point arithmetic can be made to very high accuracy with modern computers. However, this may prove to be a practical problem. For instance, the point that minimizes the sum of distances to the robots' locations, also known as the Weber point [31], may be used to achieve gathering. However, this point is not computable, due to its infinite sensitivity to location errors [9].

Second, the correctness of algorithms that use geometric properties of the plane is usually proved using theorems from Euclidean geometry. However, these theorems are, in many cases, no longer applicable when measurement or calculation errors occur.

3.1. Impossibility results. We start with some impossibility results. The proofs of these results are based on the ability of the adversary to partition the space of possible initial configurations into countably many regions, each of uncountably many configurations (say, on the basis of the initial distance between the robots), such that within each region, the outcome of the algorithm (i.e., the instructions to the robots on how far to move in each round) is the same. Note that our impossibility proofs for gathering (Theorems 3.1 and 3.3) exploit the requirement of meeting at a point, and do not preclude the possibility of convergence. In contrast, Theorem 3.5 shows that assuming sufficiently large inaccuracies in angle measurement, and in the absence of consistent sense of direction, even convergence is impossible, and the adversary can cause the robots to diverge under any algorithm.

Although this paper does not concern the issue of memory complexity and all the algorithms presented hereafter are deterministic, it may be of interest to note that the following two impossibility theorems hold even in a rather strong setting where the timing model is fully synchronous and the robots have unlimited memory and are allowed to use randomness. The main idea of the proof is to divide the line into a set of segments, where each segment is small enough such that each robot may see each of its points as the location of the other robot due to measurement errors. Since the algorithm must perform the same movement (or a random movement chosen from the same distribution), it will fail to meet the other robot, or reach the next landmark for the algorithm, starting from almost any point in the segment.

THEOREM 3.1. *Gathering is impossible for two robots on the line with inexact distance measurements even in the strong setting outlined above.*

Proof. Consider two robots 1 and 2 on the line in the $\mathcal{FSYNC}$ model and a potential gathering algorithm `ALG`. At each round t, the algorithm `ALG` at robot j has only one input $v^j_{j'}[t]$, namely, the result of the distance measurement taken by j to the other robot j'. Assuming the adversary ensures that in each round t the two robots obtain the same measurement result, i.e., $v^{j'}_j[t] = v^j_{j'}[t] = m[t]$, the inter-robot distance change $o(t)$ (namely, the distance reduction between the two robots) on round t becomes some function of $m[t]$, i.e., $o[t] = \varphi(m[t])$, and hence the total inter-robot

distance change $L(n)$ after n cycles is the sum of n outputs of the algorithm, $L(n) = \sum_{t \le n} o[t]$.

The adversary policy exploits this observation as follows. Partition the positive reals into infinitely many disjoint segments $(a_i, b_i]$, for $i \ge 1$, such that $b_i/a_i < (1+\varepsilon_{\mathrm{d}})/(1-\varepsilon_{\mathrm{d}})$ for each i. The adversary initially selects for every i a single possible measurement result m_i in the range $(1-\varepsilon_{\mathrm{d}})b_i < m_i < (1+\varepsilon_{\mathrm{d}})a_i$. During the execution of the algorithm, whenever a robot j takes a measurement where the true inter-robot distance $V^j_{j'}$ falls in the ith segment, $V^j_{j'} \in (a_i, b_i]$, the (inaccurate) measurement result will be $v^j_{j'} = m_i$.

This policy allows the introduction of only countably many distinct measurement results. Hence the total inter-robot distance change $L[t]$ after n cycles is the sum of a finite number of reals from a countable basis. The set of possible inter-robot distance changes is therefore still countable. As the set of possible initial inter-distances between the robots in the starting configuration is uncountable, it follows that gathering is never achieved from almost every starting configuration.

Observe that adding memory (allowing robots to store their previous measurements) or allowing the use of randomness will not help, since now the output of the algorithm on round t is a function of a finite number of variables, $o[t] = \varphi(\langle m(t'), \rho(1,t'), \rho(2,t')\rangle_{t' \le t})$, representing the entire history of the execution till this round, where $\rho(j,t')$ is the random number drawn by robot j on round t'. The results of each measurement are again taken from a countable set, and the values of ρ for each run form a countable set. It follows that, fixing the sequence of random number pairs $\langle \rho(1,t'), \rho(2,t')\rangle$ drawn in the execution and considering all uncountably many possible starting configurations, the robots will achieve gathering only from a countable number of starting configurations, which are of zero measure. Therefore, the robots cannot gather even with constant probability. ∎

In the case of $N > 2$ robots on the line it is presumable that a similar construction can apply. However, in the plane, assuming exact angle measurements, there is a continuum of possible measurements, and therefore gathering may be possible, using some method to estimate the distance using the angles (and possibly memory of several angles). We make the conjecture that (possibly with some limitations on the model) this does not help to achieve gathering.

CONJECTURE 3.2. *$N > 2$ robots with inaccurate distance measurements and accurate angle measurements in the $\mathcal{FSYNC}$ model cannot achieve convergence in the plane.*

In the case of angle and distance errors in the plane, gathering is impossible for any number of robots.

THEOREM 3.3. *Gathering in the plane is impossible for any number of robots assuming inaccuracies in both the distance* and *angle measurements even in the strong setting outlined above.*

Proof. Consider N robots in the plane in the $\mathcal{FSYNC}$ model and a potential gathering algorithm `ALG`. At each round t, the algorithm `ALG` at robot j has only $N-1$ inputs $\bar{v}^j_{j'}(t)$, namely, the result of the measurement taken by j to any other robot $1 \ge j' \ne j \le N$. Assuming the adversary ensures that the robot j obtains the distance measurement results $m^j_{j'}[t]$ and angle measurements $\alpha^j_{j'}$ for the distance and angle to the j' robot, respectively, in each round t, the robot configuration $\bar{R}_j$ in round t becomes $\bar{R}_j[t+1] = \bar{R}_j[t] + \Delta\bar{R}_j[t]$, where $\Delta\bar{R}_j[t]$ is some function $\varphi(\{m^j_{j'}, \alpha^j_{j'}\})$ of $m^j_{j'}[t]$ and $\alpha^j_{j'}[t]$, and hence the total change $\bar{L}^j(n)$ in each robot j after n cycles is the sum of n outputs of the algorithm, $\bar{L}^j(n) = \sum_{t \le n} \Delta\bar{R}_j[t]$.

The adversary policy exploits this observation as follows. Partition the positive reals into infinitely many disjoint segments $(a_i, b_i]$, for $i \geq 1$, such that $b_i/a_i < (1+\varepsilon_{\mathrm{d}})/(1-\varepsilon_{\mathrm{d}})$ for each i. Partition the range $[0, 2\pi)$ into the $n > 2\pi/\varepsilon_\theta$ segments $[0, 2\pi/n), \ldots, [k\pi/n, (k+1)\pi/n), \ldots, [(n-1)\pi/n, 2\pi)$. The adversary initially selects for every i a single possible measurement result m_i in the range $(1-\varepsilon_{\mathrm{d}})b_i < m_i < (1+\varepsilon_{\mathrm{d}})a_i$, and for every k the adversary selects an angle θ_k such that $(k+1)\pi/n - \varepsilon_\theta < \theta_k < k\pi/n + \varepsilon_\theta$. During the execution of the algorithm, whenever a robot j takes a measurement where the true inter-robot distance $V^j_{j'}$ falls in the ith segment, $V^j_{j'} \in (a_i, b_i]$, the (inaccurate) measurement result will be $v^j_{j'} = m_i$. Whenever the angle between a robot i and a robot j falls in the range $[k\pi/n, (k+1)\pi/n)$ the angle will be measured as θ_k.

This policy allows the introduction of only countably many distinct measurement results. Hence each of the changes $\bar{L}^j(t)$ after n cycles is the sum of a finite number of real vectors from a countable basis. The set of possible robot location changes is therefore still countable. As the set of possible initial inter-distances between the robots in the starting configuration is uncountable, it follows that gathering is never achieved from almost every starting configuration.

Observe that adding memory (allowing robots to store their previous measurements) or allowing the use of randomness will not help, since now the output of the algorithm on round t is a function of a finite number of variables, $\bar{L}^j(t) = \varphi(\langle m^j_{j'}(t'), \theta^j_{j'}, \rho(j,t')\rangle_{t' \leq t})$, representing the entire history of the execution till this round, where $\rho(j,t')$ is the random number drawn by robot j on round t'. The results of each measurement are again taken from a countable set, and the values of ρ for each run form a countable set. It follows that, fixing the sequence of random number lists $\langle(\rho(j,t'))\rangle$ drawn in the execution and considering all uncountably many possible starting configurations, the robots will achieve gathering only from a countable number of starting configurations, which are of zero measure. Therefore, the robots cannot gather even with constant probability. □

It seems reasonable to conjecture that even convergence is impossible for robots with large measurement errors. The exact limits are not completely clear. The following theorem gives some rather weak limits on the possibility of convergence. We start with a technical lemma.

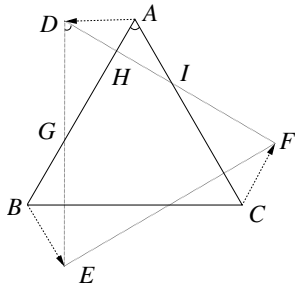


FIG. 2. *Illustration for Lemma* 3.4.

LEMMA 3.4. *If three robots are positioned in an equilateral triangular configuration and every robot moves the same distance counterclockwise parallel to the line formed by the locations of the other two robots, then the distances between the robots increase.*

Proof. A movement as in the lemma is illustrated in Figure 2, where the robots

move from the triangle $\triangle ABC$ to $\triangle DEF$, which by symmetry is also equilateral. Therefore $\angle BAC = 60°$ and $\angle EDF = 60°$. Now $\angle GHD = \angle FHA$. Thus, $\triangle HDG \sim \triangle HAI$. Since $DA \parallel BC$, also $\angle HAD = 60°$. Since $\angle HAI = 60°$, it follows that $\angle AHI < 120°$, and since it is external to $\triangle AHD$, $\angle HDA < 60° = \angle HAD$. This implies that $HD > HA$ and thus also $area(\triangle HDG) > area(\triangle HAI)$. By similar considerations on the other two corners, it follows that $area(\triangle DEF) > area(\triangle ABC)$. □

In the theorem we assume that the robot has no sense of direction in a strong way; i.e., at every cycle the adversary can choose each robot's axes independent of previous cycles.

THEOREM 3.5. *For a configuration of $N = 3$ robots having an error parameter $\varepsilon_\theta \geq \pi/3$ in angle measurement, there is no deterministic algorithm for convergence even assuming* exact *distance estimation, fully synchronous model, and unlimited memory.*

Proof. Start with a configuration with the robots positioned at the vertices of an equilateral triangle. The adversary can distort the measurements of each robot in such a way that it sees the locations of the other two robots as if its own location were exactly at the center of the line segment connecting them. This situation is symmetric under 180° rotation around the robot. Therefore, assuming a deterministic algorithm, the output vector $\bar{v}$ depends only on the robot's coordinate system. By rotating the coordinate system by 180°, the adversary can always ensure that the vector of movement is in the half-plane external to the triangle. By symmetry, the adversary can enforce a similar movement vector (rotated by 120°) for each of the three robots. In the resulting configuration, the robots will again be positioned at the vertices of an equilateral triangle (see Figure 3). Moreover, each of these movement vectors can be split into two movements: a movement away from the center, which will trivially increase the distances, and movement perpendicular to the line connecting the robot to the center, which by Lemma 3.4 will also increase the distances. Hence, the adversary can cause the algorithm to diverge. □

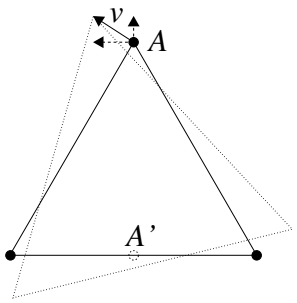


FIG. 3. *Illustration for Theorem* 3.5. *Robot A observes its environment and deduces that it is in location A'. It applies the algorithm and moves by vector v (whose projections are dashed). By symmetry the final configuration is the (larger) dotted triangle.*

3.2. Problems with existing algorithms. To illustrate the second point raised in the beginning of this section, consider the algorithm `3-Gather` presented in [1]. This algorithm achieves gathering of three robots using the following three simple rules:

(1) If two robots already reside in the same point, then the third should go to that point.

(2) If the robots form an obtuse triangle, they move towards the obtuse angled vertex.

(3) If the robots form an acute triangle, they move towards the intersection point of the angle bisectors.

As shown above, no algorithm can guarantee gathering when measurement errors occur. Furthermore, by Theorem 3.5, no algorithm can guarantee even convergence when angle measurement errors of $\varepsilon_\theta \geq 60^\circ$ might occur. We now show that even though Algorithm `3-Gather` is designed to robustness and achieves gathering even if one of the robots fails, we have the following.

OBSERVATION 3.6. *Algorithm* `3-Gather` *might fail to achieve convergence in the presence of angle measurement errors of at least* $\varepsilon_\theta \geq 15^\circ$.

Proof. Suppose the three robots form an equilateral triangle. Due to measurement errors of 15° to each of its neighbors, each robot may think its corner of the triangle forms a 90° angle and subsequently conclude that it need not move. Thus, a deadlock occurs. ∎

Likewise, for a group of $N > 3$ robots, the algorithm `N-Gather` is presented in [1]. In this algorithm (which is more complex and will not be described here), the smallest enclosing circle of the robot group is calculated, and in case there is a single robot inside this circle, it does not move. In the presence of measurement inaccuracies, this rule can potentially cause deadlock, implying the following.

OBSERVATION 3.7. *Algorithm* `N-Gather` *might fail to achieve even convergence in the presence of angle and distance measurement errors of* $\varepsilon_d > 0$.

Proof. Suppose the $N > 3$ robots form a regular N-gon. All robots are on the smallest enclosing circle. Any $\varepsilon_{\mathrm{d}} > 0$ error in the measurements taken by each of the robots may cause it to believe it is located inside the circle while all others are on the circle. Therefore, a deadlock can occur. ∎

4. The convergence algorithm.

4.1. Algorithm `Go_to_COG`. Arguably, the most natural algorithm for autonomous robot convergence is the gravitational algorithm, where each robot computes the average position (center of gravity) of the group as perceived by it,

$$\bar{v}^i_{cog} \;=\; \frac{1}{N}\sum_j \bar{v}^i_j \;,$$

and moves towards it. A formal definition of this algorithm follows.

Algorithm `Go_to_COG` (code for robot i):

1. Estimate the measured center of gravity, $\bar{v}^i_{cog} = \frac{1}{N}\sum_j \bar{v}^i_j$.
2. Move to the point $\bar{v}^i_{cog}$.

The properties of Algorithm `Go_to_COG` in a model with fully accurate measurements have been studied in [11]. In particular, the following theorem has been proved therein.

THEOREM 4.1 (see [11]). *A group of N robots executing Algorithm* `Go_to_COG` *will converge in the $\mathcal{ASYNC}$ model with no measurement errors.*

If distance measurements are not guaranteed to be accurate (but angle measurements are accurate, i.e., $\varepsilon_\theta = 0$), Algorithm `Go_to_COG` may not guarantee convergence. Nevertheless, as shown next, convergence *is* guaranteed in the fully synchronous model.

Denote the true center of gravity of the robots in the global coordinate system by

$$\bar{R}_{cog} \;=\; \frac{1}{N}\sum_j \bar{R}_j$$

and the vector from robot i to the center of gravity by

$$\overline{V}^i_{cog} \;=\; \bar{R}_{cog} - \bar{R}_i \;=\; \frac{1}{N}\sum_j \overline{V}^i_j \;,$$

recalling that $\overline{V}^i_j = \bar{R}_j - \bar{R}_i$. Denote the distance from the true center of gravity of the robots to the robot farthest from it by $D_{cog} = \max_i\{V^i_{cog}\}$. Also, denote the true distance from i to the robot farthest from it by $D^i_{max} = \max_j\{V^i_j\}$.

We have the following.

LEMMA 4.2. *In the $\langle \mathcal{FSYNC}, \mathcal{ERR}^- \rangle$ model with $\varepsilon_d < \frac{1}{2}$, a group of N robots performing Algorithm* `Go_to_COG` *converge.*

Proof. At every time step, each robot moves to its perceived center of gravity, $\bar{v}^i_{cog}$. For every robot, the perceived center of gravity satisfies $|\bar{v}^i_{cog} - \overline{V}^i_{cog}| \leq \varepsilon_{\mathrm{d}} D_{cog}$. Therefore, after step t, all robots are concentrated in a circle of radius $\varepsilon_{\mathrm{d}} D_{cog}[t]$ around $\bar{R}_{cog}[t]$. By convexity, the new center of gravity $\bar{R}_{cog}[t+1]$ also falls in this circle; hence the maximum distance between a robot and the new center of gravity is at most $D_{cog}[t+1] \leq 2\varepsilon_{\mathrm{d}} D_{cog}[t]$. Hence as long as $\varepsilon_{\mathrm{d}} < \frac{1}{2}$, $D_{cog}[t+1] \leq (1-\eta) D_{cog}[t]$ for some constant $\eta > 0$. Thus, the robots converge. □

The convergence of Algorithm `Go_to_COG` in the $\mathcal{SSYNC}$ model is not clear at the moment. However, as shown in section 6.2, in the $\mathcal{ASYNC}$ model there are scenarios where robots executing Algorithm `Go_to_COG` fail to converge. This leads us to propose the following slightly more involved algorithm.

4.2. Algorithm `RCG`. Our algorithm, named `Restricted_Go_to_COG`, or `RCG` for short, is based on calculating the center of gravity of the group of robots and also estimating the maximum possible error in the center-of-gravity calculation. The robot makes no movement if it is within the maximum possible error from the center of gravity. If it is outside the circle of error, it moves towards the center of gravity but only up to the bounds of the circle of error. We fix a conservative error estimate parameter, $\epsilon_0 > \varepsilon_{\mathrm{d}}$. We also fix a parameter $0 < \beta \leq 1$, controlling the rate of convergence. In subsection 4.4, dealing only with measurement inaccuracies, we assume $\beta = 1$. Later on, in section 5, this parameter needs to be adjusted.

Following is a more detailed explanation of the algorithm. In step 1, the measured center of gravity is estimated using the conducted measurements. In step 2 the distance to the farthest robot is found. Notice that this distance may not be accurate and that this might not even be the real farthest robot. The result of step 2 is used in step 3 to give an estimate of the possible error in the center-of-gravity calculation. In step 4 the robot decides to stay in place if it is within the circle of error or calculates its destination point, which is on the boundary of the error circle centered at the calculated center of gravity, as illustrated in Figure 4.

A formal description of the algorithm is given next. Note that Algorithm `Go_to_COG` is identical to Algorithm `RCG` with parameter $\epsilon_0 = 0$.

4.3. Analysis of `RCG` for measurement errors in the $\mathcal{FSYNC}$ model. We first prove the convergence of Algorithm `RCG` in the $\langle \mathcal{FSYNC}, \mathcal{ERR}^- \rangle$ model. We use the following two properties of D^i_{max}.

Algorithm RCG (code for robot i):

1. Estimate the measured center of gravity, $\bar{v}^i_{cog} = \frac{1}{N}\sum_j \bar{v}^i_j$.
2. Let $d^i_{max} = \max_j\{v^i_j\}$
   /* max distance measured to another robot */
3. Let $\rho^i = \dfrac{\epsilon_0}{1-\epsilon_0} \cdot d^i_{max}$
   /* estimate for max error between calculated and actual CoG */
   and $\mathcal{F} = 1 - \rho^i / v^i_{cog}$
   /* safe movement fraction */
4. If $\mathcal{F} > 0$, then move to the point $\bar{c}_i = \beta \cdot \mathcal{F} \cdot \bar{v}^i_{cog}$.
   Otherwise do not move.

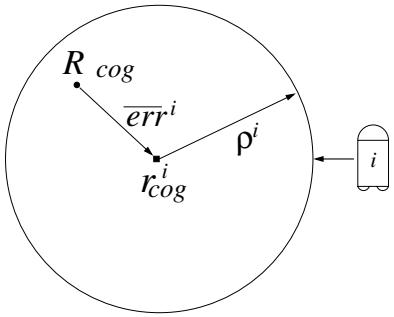


FIG. 4. *Illustration of the Algorithm* RCG.

FACT 4.3. *For every i, the true and perceived maximum distances satisfy*
(a) $(1-\epsilon_0)D^i_{max} < (1-\varepsilon_d)D^i_{max} \le d^i_{max} \le (1+\varepsilon_d)D^i_{max} < (1+\epsilon_0)D^i_{max}$,
(b) $D_{cog} \le D^i_{max} \le 2D_{cog}$.

Proof. The first property is immediate from the definition of D^i_{max}, the choice of d^i_{max}, and the assumption. For the lower bound in the second property notice that the center of gravity is in the convex hull of the robot group. For the upper bound in the second property, let $\ell = \arg\max_j\{V^i_j\}$. By the triangle inequality for each j, we have

$$D^i_{max} = V^i_\ell \le \frac{1}{N}\sum_j (V^i_j + V^\ell_j) = V^i_{cog} + V^\ell_{cog} \le 2D_{cog}. \qquad \square$$

For the synchronous model, we define the tth round to begin at time t and end at time $t+1$. The robots all perform their Look phase simultaneously. The robots' *moment of inertia* at time t is defined as

$$I[t] = \frac{1}{N}\sum_j \left(\overline{V}^j_{cog}[t]\right)^2 = \frac{1}{N}\sum_j \left(\bar{R}_j[t] - \bar{R}_{cog}[t]\right)^2.$$

Defining the function $I_{\bar{x}}[t] \equiv \frac{1}{N}\sum_j (\bar{R}_j[t] - \bar{x})^2$, we notice the following fact (cf. [16]).

FACT 4.4. *$I_{\bar{x}}[t]$ attains its minimum on $\bar{x} = \bar{R}_{cog}[t]$.*

We now identify a key property required to ensure convergence of Algorithm RCG. For some time t, denote the *error component* in the center-of-gravity calculation by robot i by

$$\overline{err}^i = \frac{1}{N}\sum_j \sigma^i_j \overline{V}^i_j \ .$$

We refer to the ratio err^i/D_{cog} at time t as the *error ratio* of the algorithm at t. Algorithm RCG is said to have *bounded error ratio* if

$$\frac{err^i}{D_{cog}} \;<\; 2\epsilon_0 \quad \text{at any time } t. \tag{1}$$

Our two main lemmas, presented next, relate the bounded error ratio property to convergence.

LEMMA 4.5. *In the $\langle \mathcal{FSYNC}, \mathcal{ERR} \rangle$ model, if $\epsilon_0 < 0.2$ and Algorithm* RCG *has bounded error ratio, then the algorithm guarantees that at every round t*

1. *at least one robot can move,*
2. *every robot i that makes a move decreases its distance from the true center of gravity at time t, i.e., $|\bar{R}_i[t+1] - \bar{R}_{cog}[t]| < |\bar{R}_i[t] - \bar{R}_{cog}[t]|$,*
3. *the robots' moment of inertia decreases, i.e., $I[t+1] < I[t]$.*

Proof. Consider some time t. By assumption, the algorithm satisfies inequality (1) at time t, and by the two parts of Fact 4.3, the calculated value ρ^i is bounded by

$$\rho^i \;\le\; \frac{\epsilon_0(1+\epsilon_0)}{1-\epsilon_0} \cdot D^i_{max} \;\le\; \frac{\epsilon_0(1+\epsilon_0)}{1-\epsilon_0} \cdot 2D_{cog}.$$

By assumption (1), we have for each i

$$err^i + \rho^i \;\le\; f(\epsilon_0) \cdot D_{cog} \;<\; D_{cog}\ , \tag{2}$$

where $f(\epsilon_0) = 4\epsilon_0/(1-\epsilon_0)$, and the last inequality follows from the assumption that $\epsilon_0 < 0.2$.

For $k = \arg\max_j \{V^j_{cog}\}$, the robot farthest from the center of gravity, we have $V^k_{cog} = D_{cog}$ and $\bar{v}^k_{cog} = \overline{V}^k_{cog} + \overline{err}^k$; hence by inequality (2) and the triangle inequality,

$$\rho^k \;<\; V^k_{cog} - err^k \;\le\; v^k_{cog}\ .$$

This implies that at round t, robot k is allowed to move in step 4 of the algorithm, proving part 1 of the Lemma.

To prove part 2, consider a round t and a robot i which moved in round t. Fix $\bar{x} = \bar{R}_{cog}[t]$ and take $\bar{a} = \overline{err}^i[t]/\rho^i[t]$ and $\bar{b} = \bar{v}^i_{cog}[t]/\rho^i[t]$. Note that by (1) and Fact 4.3(a),

$$err^i[t] \;\le\; \frac{\varepsilon_{\mathrm{d}}}{1-\varepsilon_{\mathrm{d}}} \cdot d^i_{max}[t] \;<\; \frac{\epsilon_0}{1-\epsilon_0} \cdot d^i_{max}[t] \;=\; \rho^i[t];$$

hence $a \le 1$. Also, at round $t+1$, robot i moves if and only if $v^i_{cog}[t] > \rho^i[t]$; hence if i moved, then $b = v^i_{cog}[t]/\rho^i[t] > 1$. Hence Lemma 2.1(1) can be applied. Noting that $\bar{b}/b = (\bar{R}_i[t+1] - \bar{R}_{cog}[t])/\rho^i[t]$, we get

$$|\bar{R}_i[t+1] - \bar{x}| < |\bar{R}_i[t] - \bar{x}|\ ,$$

yielding part 2 of the Lemma. It remains to prove part 3. Note that for a robot that did not move, $|\bar{R}_i[t+1] - \bar{x}| = |\bar{R}_i[t] - \bar{x}|$. Using this fact and part 2, we have that

$$I_{\bar{x}}[t+1] < I_{\bar{x}}[t] = I[t]\ .$$

Finally, by Fact 4.4 we get $I[t+1] \;\le\; I_{\bar{x}}[t+1]$, yielding part 3 of Lemma 4.5. ∎

LEMMA 4.6. *In the $\langle \mathcal{FSYNC}, \mathcal{ERR} \rangle$ model, if $\epsilon_0 < 0.2$ and Algorithm* RCG *has bounded error ratio, then in every execution of the algorithm, the robots converge.*

Proof. By part 1 of Lemma 4.5, the robot k most distant from the center of gravity can always move if Algorithm Go_to_COG is applied. By part 2 of Lemma 4.5, in round t every robot decreases its distance from the old center of gravity, $\bar{x} = \bar{R}_{cog}[t]$. Therefore, to bound from below the decrease in I, we are required only to examine the behavior of the most distant robot. By Lemma 2.1(1) with $a = \bar{R}_k[t+1] - \bar{R}_{cog}[t]$ and $b = \bar{R}_k[t] - \bar{R}_{cog}[t]$, for $\epsilon_0 < 0.2$ we have $v^k_{cog} > \rho^k$ and

$$(\bar{R}_k[t] - \bar{R}_{cog}[t])^2 - (\bar{R}_k[t+1] - \bar{R}_{cog}[t])^2 \;\geq\; (v^k_{cog} - \rho^k)^2 \;.$$

Since $\bar{v}^k_{cog} = \overline{V}^k_{cog} + \overline{err}^k$, and using the triangle inequality,

$$(\bar{R}_k[t] - \bar{R}_{cog}[t])^2 - (\bar{R}_k[t+1] - \bar{R}_{cog}[t])^2 \geq \left(V^k_{cog} - (\rho^k + err^k)\right)^2 \;. \tag{3}$$

Recall that since k is the most distant robot, $V^k_{cog} = D_{cog}$. Denoting $\gamma = 1 - f(\epsilon_0)$, we have by (2) that

$$V^k_{cog} - (\rho^k + err^k) \;\geq\; \gamma \cdot D_{cog} \;. \tag{4}$$

As mentioned above, if $\epsilon_0 < 0.2$, then $\gamma > 0$. We also use the fact that

$$I[t] = I_{\bar{x}}[t] \leq D^2_{cog}. \tag{5}$$

Using Fact 4.4 and inequalities (3), (4), and (5), we have that

$$\begin{aligned}
I[t+1] \leq I_{\bar{x}}[t+1] \;&=\; \frac{1}{N}\left((\bar{R}_k[t+1] - \bar{R}_{cog}[t])^2 + \sum_{j \neq k}(\bar{R}_j[t+1] - \bar{R}_{cog}[t])^2\right) \\
&\leq \frac{1}{N}(\bar{R}_k[t+1] - \bar{R}_{cog}[t])^2 + \frac{1}{N}\sum_{j \neq k}(\bar{R}_j[t] - \bar{R}_{cog}[t])^2 \\
&\leq \frac{1}{N}(\bar{R}_k[t+1] - \bar{R}_{cog}[t])^2 - \frac{1}{N}(\bar{R}_k[t] - \bar{R}_{cog}[t])^2 + I[t] \\
&\leq I[t] - \frac{1}{N}\left(V^k_{cog} - (\rho^k + err^k)\right)^2 \;\leq\; I[t] - \frac{\gamma^2}{N} \cdot D^2_{cog} \\
&\leq I[t]\left(1 - \frac{\gamma^2}{N}\right),
\end{aligned}$$

and therefore the system converges, proving the theorem. □

Next, our two main theorems, stating the convergence of Algorithm RCG in the $\mathcal{ERR}^-$ and $\mathcal{ERR}$ models, respectively, are established by showing that a suitable selection of ϵ_0 ensures that the algorithm has bounded error ratio.

THEOREM 4.7. *In the $\langle \mathcal{FSYNC}, \mathcal{ERR}^- \rangle$ model, if there exists a fixed ϵ_0 such that $\varepsilon_d < \epsilon_0 < 0.2$, then in every execution of Algorithm* RCG*, the robots converge.*

Proof. The center of gravity computed by robot i can be expressed as

$$\bar{v}^i_{cog} \;=\; \frac{1}{N}\sum_j \bar{r}^i_j \;=\; \frac{1}{N}\sum_j (\bar{R}_j + \sigma^i_j \overline{V}^i_j) \;=\; \overline{V}^i_{cog} + \overline{err}^i \;.$$

By the bounded imprecision assumption and Fact 4.3(b),

$$(6)\quad err^i \;=\; \frac{1}{N}\left|\sum_j \sigma_j^i \cdot V_j^i\right| \;\le\; \frac{1}{N}\sum_j |\sigma_j^i|\cdot V_j^i \;\le\; \varepsilon_{\rm d} D_{max}^i \;\le\; 2\varepsilon_{\rm d} D_{cog} \;<\; 2\epsilon_0 D_{cog}.$$

The claim now follows from Lemma 4.6. □

We now turn to the $\mathcal{ERR}$ model, also allowing inaccuracies in angle measurements.

THEOREM 4.8. *In the $\langle \mathcal{FSYNC}, \mathcal{ERR}\rangle$ model, if there exists a fixed ϵ_0 such that $\sqrt{2(1+\varepsilon_d)(1-\cos\varepsilon_\theta)+{\varepsilon_d}^2} < \epsilon_0 < 0.2$, then in every execution of Algorithm* RCG*, the robots converge.*

Proof. Recall that $\bar v_{cog}^i = \overline{V}^i_{cog} + \overline{err}^i$. By Lemma 2.2

$$\begin{aligned} err^i \;&=\; |\bar v_{cog}^i - \overline{V}^i_{cog}| < V_{cog}^i\sqrt{2(1+\varepsilon_{\rm d})(1-\cos\varepsilon_\theta)+{\varepsilon_{\rm d}}^2} \\ &\le \sqrt{2(1+\varepsilon_{\rm d})(1-\cos\varepsilon_\theta)+{\varepsilon_{\rm d}}^2}\, D_{max}^i \\ &\le 2\sqrt{2(1+\varepsilon_{\rm d})(1-\cos\varepsilon_\theta)+{\varepsilon_{\rm d}}^2}\, D_{cog} \;<\; 2\epsilon_0 D_{cog}\ . \end{aligned}$$

Lemma 4.6 can thus be applied, completing the proof. □

4.4. Analysis of RCG for measurement errors in the $\mathcal{SSYNC}$ model. Turning to the semisynchronous model, we observe that the results of Theorem 4.8 also hold true for the $\langle \mathcal{SSYNC}, \mathcal{ERR}\rangle$ model. We start with a lemma aiming to show that in any execution of Algorithm RCG, for any move of a group of robots that produces a shift of the center of gravity, the moment of inertia decreases.

LEMMA 4.9. *For a group of N robots performing Algorithm* RCG *in the $\mathcal{SSYNC}$ model, if at time step t the center of gravity has moved such that $|\bar R_{cog}[t+1]-\bar R_{cog}[t]| \ge \Delta x$ for some Δx, then $I[t]-I[t+1] \ge 2\epsilon_0(1-a)\Delta x D_{cog}[t]$ for some constant $a<1$.*

Proof. Consider a robot i moving along a vector $\bar y^i$ (of size y^i) at time t, and let $\rho^i[t]$ be the calculated maximum error for this robot at time t. Fix

$$\bar a \;=\; \overline{err}^i[t]/\rho^i[t] \qquad \text{and} \qquad \bar b \;=\; \bar v_{cog}^i/\rho^i[t]\ .$$

We therefore have that $a = err^i[t]/\rho^i[t]$ and, since we know that the motion of the robot is of size y^i, it follows that $|\bar b - \bar b/b| = y^i/\rho^i[t]$. Thus, $b = 1 + y^i/\rho^i[t]$. By (6), $err^i \le \varepsilon_{\rm d} D_{max}^i \le \varepsilon_{\rm d} d_{max}^i/(1-\varepsilon_{\rm d})$ (with $\varepsilon' \equiv \sqrt{2(1+\varepsilon_{\rm d})(1-\cos\varepsilon_\theta)+{\varepsilon_{\rm d}}^2}$ replacing $\varepsilon_{\rm d}$ for the $\mathcal{ERR}$ model), while $\rho^i = \epsilon_0 d_{max}^i/(1-\epsilon_0)$. This implies that $a \le \frac{\varepsilon_{\rm d}(1-\epsilon_0)}{\epsilon_0(1-\varepsilon_{\rm d})} < 1$ (where we use the fact that $\epsilon_0 > \varepsilon_{\rm d}$). Also, clearly $b \ge 1$. Hence Lemma 2.1(1) can be applied, yielding

$$(\bar a - \bar b)^2 - (\bar a - \bar b/b)^2 \ge (b-1)^2 + 2(1-a)(b-1) \ge 2(1-a)(b-1) = 2(1-a)y^i/\rho^i[t].$$

Now, $\bar a - \bar b = (\bar R_i[t] - \bar R_{cog}[t])/\rho^i[t]$, and $\bar a - \bar b/b = (\bar R_i[t+1] - \bar R_{cog}[t])/\rho^i[t]$. Thus,

$$(\bar R_i[t] - \bar R_{cog}[t])^2 - (\bar R_i[t+1] - \bar R_{cog}[t])^2 \;\ge\; 2(1-a)y^i\rho^i[t]\ .$$

Note that

$$(7)\qquad \rho^i[t] \;=\; \frac{\epsilon_0}{1-\epsilon_0} d_{max}^i[t] \;\ge\; \epsilon_0 D_{max}^i[t] \;\ge\; \epsilon_0 D_{cog}[t]\ ,$$

and therefore $(\bar{R}_i[t]-\bar{R}_{cog}[t])^2-(\bar{R}_i[t+1]-\bar{R}_{cog}[t])^2 \geq 2(1-a)y^i\epsilon_0 D_{cog}[t]$. The total decrease in I is therefore

$$I[t]-I[t+1] \geq I[t]-I_{\bar{x}}[t] \geq \frac{1}{N}\sum_i 2(1-a)y^i\epsilon_0 D_{cog}[t], \tag{8}$$

with $\bar{x}=\bar{R}_{cog}[t]$.

Now, we know that the center of gravity has moved by at least Δx, and, therefore,

$$\Delta x \leq |\bar{R}_{cog}[t+1]-\bar{R}_{cog}[t]| = \frac{1}{N}\left|\sum_i y^i\right| \leq \frac{1}{N}\sum_i |y^i| \ .$$

Thus, using (8), one obtains $I[t]-I[t+1] \geq 2(1-a)\Delta x\epsilon_0 D_{cog}[t]$. □

We now turn to prove the main theorem for the $\mathcal{SSYNC}$ model.

THEOREM 4.10. *In every execution of Algorithm* RCG *(with ϵ_0 as in Theorem* 4.8*) in the $\langle \mathcal{SSYNC}, \mathcal{ERR}\rangle$ model, the robots converge.*

Proof. The proof of Lemma 4.5 holds for any movement of any robot, and therefore also for any partial robot group making a move. Thus, it also applies to the semi-synchronous model. In Lemma 4.6, it may happen that the robot most distant from the center of gravity is inactive and does not make a move for a number of steps, during which the situation changes and the center of gravity approaches it due to movements taken by the other robots. Hence, we have to deal with the complication arising from the possibility that the most distant robot at some time t' is no longer the most distant when its turn to move arrives at some later time t.

Suppose at time t' this robot, k, was at distance $D_{cog}[t']$ from the center of gravity. Now take $t>t'$ as the time of its next activation. Take $\delta=\frac{1}{3}[1-f(\epsilon_0)]$. If $V^k_{cog}[t] > (1-\delta)D_{cog}[t']$ and $D_{cog}[t] < (1+\delta)D_{cog}[t']$, then $V^k_{cog}[t]/D_{cog}[t] > 1-2\delta$. This implies that robot k can still make a move at time t, since

$$\rho^k[t] \ = \ \frac{\epsilon_0}{1-\epsilon_0}\cdot d^k_{max}[t] \ \leq \ 2\frac{\epsilon_0}{1-\epsilon_0}(1+\varepsilon_{\mathrm{d}})D_{cog}[t] \ < \ \frac{f(\epsilon_0)}{1-2\delta}V^k_{cog}[t] \ < \ V^k_{cog}[t] \ , \tag{9}$$

where the last inequality holds as long as $f(\epsilon_0)<1$. We may now use Lemma 2.1(1), with $b=V^k_{cog}[t]/\rho^k[t] > (1-2\delta)/f(\epsilon_0)$, leading to

$$(\bar{R}_k[t]-\bar{R}_{cog}[t])^2-(\bar{R}_k[t+1]-\bar{R}_{cog}[t])^2 \ \geq \ (v^k_{cog}[t]-\rho^k)^2 \geq \left(\frac{1-2\delta}{f(\epsilon_0)}-1\right)^2\rho^k[t] \ .$$

Equation (7) implies that $\rho^k[t] \geq \epsilon_0 D_{cog}[t] \geq (1-2\delta)\epsilon_0 D^k_{cog}[t']$, and thus

$$I[t+1] \ \leq \ \left[1-\frac{1}{N}\left(\frac{1-2\delta}{f(\epsilon_0)}-1\right)^2[(1-2\delta)\epsilon_0]^{-2}\right]I[t'].$$

If one of the requirements is violated, this means that the center of gravity has moved at least a distance of $\delta D_{cog}[t']$ in some time interval $[t',t_1]\subset[t',t]$. Choose t_1 to be the first time at which the center of gravity has moved at least $\delta D_{cog}[t']$. Then, for every time $t^*\in[t',t_1-1]$, $D_{cog}[t^*]\geq(1-\delta)D_{cog}[t']$. Denote by $\overline{\Delta x}[t^*]$ the change in the center of gravity between cycles t^* and t^*+1. We have

$$\sum_{t^*\in[t',t_1-1]}\Delta x[t^*] \ \geq \ \left|\sum_{t^*\in[t',t_1-1]}\overline{\Delta x}[t^*]\right| \ \geq \ \delta D_{cog}[t'] \ .$$

By Lemma 4.9 we have that for all t^*,

$$I[t^*] - I[t^*+1] \geq 2(1-a)\Delta x[t^*]\epsilon_0 D_{cog}[t^*] \geq 2(1-a)\Delta x[t^*]\epsilon_0(1-\delta)D_{cog}[t'].$$

Therefore, the total improvement in I is

$$\begin{aligned} I[t'] - I[t] \;&\geq\; I[t'] - I[t_1] \;\geq \sum_{t^* \in [t', t_1 - 1]} 2(1-a)\Delta x[t^*]\epsilon_0(1-\delta)D_{cog}[t'] \\ &\geq\; 2(1-a)\delta D_{cog}[t']\epsilon_0(1-\delta)D_{cog}[t'] \;\geq\; 2(1-a)\delta\epsilon_0(1-\delta)I[t']; \end{aligned}$$

hence $I[t] \leq (1-\eta)I[t']$ for some constant $\eta > 0$. Thus, the robots converge. □

5. Handling movement and calculation errors. We now turn to treat the case of movement errors. Assume the robots' motion is inaccurate, where the robot aiming to move a distance d_0 at an angle α_0 will actually move a distance d satisfying $ad_0 < d < bd_0$ at an angle α satisfying $|\alpha - \alpha_0| < \delta$, for constants $0 < a < 1$, $b > 1$, and $\delta > 0$.

As mentioned earlier, another possible type of movement error discussed in the literature is that the robot might halt its movement prematurely (it is commonly assumed that the robot's movement is guaranteed to traverse at least some constant distance s).

Movement distance errors may be readily treated as measurement errors. In fact, movement distance errors are somewhat less severe, as they may not cause the robot to move in the wrong direction but only move an inaccurate distance in the correct direction. Therefore, no limit on the size of the error is needed (other than b being finite and $a > 0$), and the only requirement necessary for ensuring convergence of the algorithm is to set the calibration parameter β to $\beta = 1/b$. We have the following.

THEOREM 5.1. *Robots having movement errors performing Algorithm* `RCG` *will converge to a point.*

Proof. A robot having an angular movement error of $\alpha - \alpha_0 = \theta$ performing algorithm `RCG` will make exactly the same movement as a robot having accurate movement and viewing all other robots with an error of θ in the angle measurement (possibly in addition to any measurement errors included in the model). Therefore, a model with motion angle errors is reducible to a model with $\varepsilon_\theta = \varepsilon_\theta^{\mathrm{mv}}$. In general, if both measurement and movement errors occur, then the model is reducible (for Algorithm `RCG`) to a model with total angle measurement inaccuracy of $\varepsilon_\theta + \varepsilon_\theta^{\mathrm{mv}}$.

To adjust for the motion distance error, the parameter β is modified to $\beta = 1/b$ (or any lower value). Thus, every robot attempts to move to the point $\bar{c}_i = \frac{1}{b}(1 - \rho^i/v^i_{cog}) \cdot \bar{v}^i_{cog}$. This guarantees that the distance travelled is always less than $b\bar{c}_i = (1 - \rho^i/v^i_{cog}) \cdot \bar{v}^i_{cog}$ and at least $a\bar{c}_i = \frac{a}{b}(1 - \rho^i/v^i_{cog}) \cdot \bar{v}^i_{cog}$, a constant multiple of the distance travelled in the original algorithm. Furthermore, since the calculation of ρ^i is not influenced by β, it is guaranteed that any robot that can move in the original (accurate movement) model can also move in the inaccurate movement model. Therefore, using Lemma 2.3, one determines that whenever in the accurate movement model a robot decreases its squared distance from the center of gravity by some amount c, in the inaccurate movement model its squared distance from the center of gravity will decrease by at least ca/b, guaranteeing a constant factor improvement in I. The case presented above in the model definition is equivalent to the case $b \equiv 1 + \varepsilon_{\mathrm{d}}^{\mathrm{mv}}$, $a \equiv 1 - \varepsilon_{\mathrm{d}}^{\mathrm{mv}}$. □

The second possible form of movement inaccuracies is the "premature stopping" model, presented above, in which the robot may fail to move the full distance determined by the algorithm and stop prematurely. This model guarantees, however,

that the robot will complete a movement of at least some constant distance, s, unless the distance determined by the algorithm is less than s, in which case the robot is guaranteed to complete its movement.

THEOREM 5.2. *In the $\mathcal{FSYNC}$ and $\mathcal{SSYNC}$ models, a group of N robots performing algorithm* `RCG` *with premature stopping will converge to a point.*

Proof. We start with the $\mathcal{FSYNC}$ model. Take the robot k farthest from the center of gravity. There are two possibilities for this robot's movement.

1. Robot k completes its movement. This case is analyzed exactly the same as in Theorem 4.7 or Theorem 4.8 (depending on the error model), and I can be shown to decrease by a constant multiplicative factor.
2. Robot k travels some distance $s^* \geq s$ at time t. Assume the robot would have completed the move determined by the algorithm, of some distance $d \leq D_{cog}[t]$. Then, by Theorem 4.7 or Theorem 4.8, $I[t] - I[t+1] \geq \eta(N) I[t]$ for some $\eta(N)$. The ratio μ between the distance travelled by the robot and the distance determined by the algorithm satisfies

$$\mu \;=\; \frac{s^*}{d} \;\geq\; \frac{s}{d} \;\geq\; \frac{s}{D_{cog}[t]} \;.$$

By Lemma 2.3 the decrease in robot k's distance to the center of gravity by the actual movement is at least μ times the decrease due to the desired movement. Thus

$$I[t] - I[t-1] \;\geq\; \mu^2 \eta(N) I \;\geq\; \frac{s^2}{D_{cog}[t]^2} \eta(N) I[t] \;.$$

Since $I[t] \leq D_{cog}[t]^2$, it follows that $I[t] - I[t-1] \geq \eta(N) s^2$.

Thus, at every time step, I is decreased by either a multiplicative or an additive constant. The theorem follows.

In the case of the $\mathcal{SSYNC}$ model, as in the proof of Theorem 4.10, either the farthest robot k is at a distance at least $(1-\delta) D_{cog}[t']$ from the center of gravity when it performs its next move after time t', in which case it induces a multiplicative or an additive constant improvement in I (as in the $\mathcal{FSYNC}$ model above), or the center of gravity has moved a distance of at least $\delta D_{cog}[t']$, in which case I has improved by a multiplicative factor. ∎

Calculation errors and limited accuracy can be modeled in several ways. The most commonly used model assumes that each operand, x_i, in each operation can include an additional error term, ε_{ci}, which satisfies $|\varepsilon_{ci}| \leq \varepsilon_c x_i$ for some ε_c.

We begin with a technical lemma relating inaccuracies in a vector's component with angle inaccuracies.

LEMMA 5.3. *Let $\bar{a}$ be a vector with components (x, y), and $\bar{b}$ be a vector with components (x', y'), where $(1-\varepsilon)x < x' < (1+\varepsilon)x$ and $(1-\varepsilon)y < y' < (1+\varepsilon)y$ for some $0 < \varepsilon < 1$. The angle between $\bar{a}$ and $\bar{b}$ is at most $(\ln(1+\epsilon) - \ln(1-\epsilon))/2$ radians.*

Proof. If $x = 0$ or $y = 0$, then $x' = 0$ or $y' = 0$, respectively, so the angle between $\bar{a}$ and $\bar{b}$ is 0 and the theorem follows immediately. Assume now that $x \neq 0$ and $y \neq 0$ and denote $x' = (1+\alpha)x$ and $y' = (1+\beta)y$, where $-\epsilon < \alpha, \beta < \epsilon$. Notice that $\bar{a}$ and $\bar{b}$ are in the same quadrant. Now choose θ as the angle between $\bar{a}$ and the appropriate axis by the rule

$$\tan\theta = \begin{cases} |x|/|y|, & \alpha > \beta, \\ |y|/|x|, & \alpha \leq \beta, \end{cases}$$

and set θ' to be the angle between the same axis and $\bar{b}$ defined by

$$\tan\theta' = \begin{cases} |x'|/|y'|, & \alpha > \beta, \\ |y'|/|x'|, & \alpha \le \beta. \end{cases}$$

Thus, $\tan\theta' = \frac{1+\alpha}{1+\beta}\tan\theta$ if $\alpha > \beta$ and $\tan\theta' = \frac{1+\beta}{1+\alpha}\tan\theta$ otherwise. Either way, we have $\tan\theta \le \tan\theta' \le \frac{1+\epsilon}{1-\epsilon}\tan\theta$ or

$$\ln\tan\theta \le \ln\tan\theta' \le \ln\tan\theta + \ln(1+\epsilon) - \ln(1-\epsilon). \tag{10}$$

Denote $\theta' = \theta + \Delta\theta$. Since tan is a continuous function with continuous derivative inside each quadrant, we have

$$\begin{aligned} \ln\tan(\theta+\Delta\theta) &= \ln\tan\theta + \left(\frac{d\ln\tan\theta}{d\theta}\right)_{\theta=\theta^*} \cdot \Delta\theta = \ln\tan\theta + \frac{1}{|\cos^2\theta|\tan\theta}\ \Delta\theta \\ &= \ln\tan\theta + \frac{2}{|\sin 2\theta|}\Delta\theta \ \ge\ \ln\tan\theta + 2\Delta\theta, \end{aligned} \tag{11}$$

for some $\theta \le \theta^* \le \theta + \Delta\theta$. From (10) and (11) it follows that the angle $\Delta\theta$ between $\bar{a}$ and $\bar{b}$ satisfies $|\Delta\theta| \le (\ln(1+\epsilon) - \ln(1-\epsilon))/2$. □

THEOREM 5.4. *A group of robots having inaccurate or limited accuracy calculations with $\varepsilon_c < 0.125$ (assuming accurate measurement and movement) executing Algorithm* `RCG` *will converge to a point.*

Proof. We use uppercase letters to represent real quantities and lowercase to represent calculated quantities. For simplicity of notation we assume exact measurements and inaccurate calculations. A combination of the two with limited errors is expected to behave similarly using the same lines of argument.

In step 1 of the calculation in Algorithm `RCG`, the additional errors are equivalent to inaccurate measurements, since each $\bar{v}_j^i$ is augmented with an error term of at most $\varepsilon_c \bar{v}_j^i$. To bound the error, we notice that the calculated $\bar{v}_{cog}^i$ for each robot i is

$$\bar{v}_{cog}^i \ = \ \frac{1}{N}\sum_j (\bar{v}_j^i + \bar{\sigma}_j^i) \ = \ \frac{1}{N}\sum_j (\overline{V}_j^i + \overline{err}^i + \bar{\sigma}_j^i)\ .$$

It is known that $\sigma_j^i \le \varepsilon_c v_j^i \le (1+\varepsilon_d)\varepsilon_c V_j^i$. Defining $\varepsilon_t = (1+\varepsilon_c)(1+\varepsilon_d) - 1$ and assuming the calculation is done by components (so the error bound holds for each component and therefore also for the vector size), it follows that

$$\left|\bar{v}_{cog}^i - \overline{V}_{cog}^i\right| \ = \ \frac{1}{N}\left|\sum_j (\overline{err}^i + \bar{\sigma}_j^i)\right| \ \le \ \frac{1}{N}\sum_j (|\overline{err}^i| + |\bar{\sigma}_j^i|) \ \le \ \frac{1}{N}\sum_j \varepsilon_t V_j^i \ \le \ \varepsilon_t D_{max}\ .$$

This implies that the inaccuracy in the calculated center of gravity appears in exactly the same form as in inaccurate measurements.

Step 2 of Algorithm `RCG` involves a comparison between vectors, which may be considered accurate up to the limited accuracy in the calculation of the vector sizes. This may lead to the calculated d_{max}^i having a relative error of at most ε_c. In the calculation performed in step 3 of the algorithm, another relative error of ε_c may be introduced, and yet another may occur in the calculation of ρ^i/v_{cog}^i in step 4. Therefore, it can be concluded that

$$\frac{\epsilon_0}{1-\epsilon_0}(1-\varepsilon_c)^3 d_{max}^i \ \le \ \rho^i \ \le \ \frac{\epsilon_0}{1-\epsilon_0}(1+\varepsilon_c)^3 d_{max}^i\ .$$

The choice of ϵ_0 should be large enough to ensure that the value of ρ^i resulting from the calculation of step 3 will always be greater than the total error in the target location (discussed below) but small enough to ensure that at least one robot can move at every configuration, i.e., such that the calculated ρ^i satisfies $\rho^i < D_{cog}$. This imposes the conditions $\frac{\epsilon_0}{1-\epsilon_0}(1-\varepsilon_c)^3 > \frac{\varepsilon_c}{1-\varepsilon_c}$ and $\frac{\epsilon_0}{1-\epsilon_0}(1+\varepsilon_c)^3 d^i_{max} < D_{cog}$ for at least one i. Since for at least one robot $d^i_{max} \geq 2(1-\varepsilon_d)D_{cog}$, it follows that we require $\frac{\epsilon_0}{1-\epsilon_0}(1+\varepsilon_c)^3(1-\varepsilon_d) < [2(1-\varepsilon_d)]^{-1}$. That is, we must choose ϵ_0 such that this condition holds.

Errors in step 4 of the algorithm are relative errors in the motion vector and therefore are identical to movement errors, treated in Theorem 5.1. Since the calculated motion vector may contain a relative error of ε_c, leading to a factor of $1+\varepsilon_c$ in the motion vector size, β should be chosen such that $\beta < (1+\varepsilon_c)^{-1}$. By Lemma 5.3, an error of at most $\theta_c = (\ln(1+\varepsilon_c) - \ln(1-\varepsilon_c))/2$ may occur in the angle of motion, which is similar to the error in Theorem 5.1.

The total error in the aimed location of the center of gravity, using Lemma 2.2, is therefore

$$\sqrt{2(1+\varepsilon_t)(1-\cos(\theta_c+\varepsilon_\theta)) + {\varepsilon_t}^2}\; D_{max}\ .$$

In the case of exact measurements, $\varepsilon_d = 0$, ϵ_0 should satisfy

$$\frac{\sqrt{2(1+\varepsilon_c)(1-\cos\theta_c) + {\varepsilon_c}^2}}{(1-\varepsilon_c)^3} \;<\; \frac{\epsilon_0}{1-\epsilon_0} \;<\; \frac{1}{2(1+\varepsilon_c)^3}\ .$$

For these inequalities to allow a solution, the upper bound should be higher than the lower bound. Thus, ε_c should satisfy $\varepsilon_c < 0.126\ldots$. When $\varepsilon_d > 0$, the maximum value of ε_c should be lower accordingly. □

It should be noted that unlike the measurement and movement inaccuracies discussed earlier, calculation errors are rather negligible. In particular, commercially available processors, even low-end ones, can be assumed to introduce very small numerical errors; hence it is safe to assume that for any practical purpose $\varepsilon_c < 10^{-3}$ in any real system. Consequently, the limit above poses no problem in practical applications.

6. Analysis of RCG in the fully asynchronous model.

6.1. Convergence in the one-dimensional case. So far, we have not been able to establish the convergence of Algorithm RCG in the fully asynchronous model. In this section we prove its convergence in the restricted one-dimensional case and with no angle inaccuracies, i.e., in the $\langle \mathcal{ASYNC}, \mathcal{ERR}^- \rangle$ model.

Denote by $\bar{c}_i[t]$ the calculated destination of robot i at time t. If robot i has not gone through a Look yet, or has reached its previous destination, then, by definition, $\bar{c}_i[t] = \bar{R}_i[t]$. Notice that we set $\bar{c}_i[t]$ to be the destination of the robot's motion after the Look phase even if the robot has not yet completed its computation and is still unaware of this destination.

Following are some technical lemmas and definitions used to prove the result when the robots are on a straight line. Define $H[t]$ as the minimum segment containing all robots and destinations at time t.

LEMMA 6.1. *In the $\langle \mathcal{ASYNC}, \mathcal{ERR}^- \rangle$ model, $H[t] \subseteq H[t_0]$ for all times $t \geq t_0$.*

Proof. By the proof of part 2 of Lemma 4.5 it follows that every robot approaches the true center of gravity; i.e., in the one-dimensional case, it always moves in the correct direction towards the center of gravity. Furthermore, the size of its motion is

always an underestimation to the true distance. Therefore, the calculated destination resides between the current location of the robot and the real center of gravity, which is always in the segment. Thus no destination is calculated outside the segment. Therefore, no robot leaves the segment either. ∎

Next, we define the following quantities:

$$\begin{aligned}
\phi_1[t] &= \sum_{i=1}^{N} |\bar{R}_{cog}[t] - \bar{c}_i[t]| \ , \\
\phi_2[t] &= \sum_{i=1}^{N} |\bar{c}_i[t] - \bar{R}_i[t]| \ , \\
\phi[t] &= \phi_1[t] + \phi_2[t] \ , \\
h[t] &= |H[t]| \ , \\
\psi[t] &= \frac{\phi[t]}{2N} + h[t] \ .
\end{aligned}$$

We now claim that ϕ, h, and ψ are nonincreasing functions of time.

LEMMA 6.2. *For every $t_1 > t_0$, $\phi[t_1] \leq \phi[t_0]$.*

Proof. Examine the change in ϕ due to the various robot actions. If a Look operation is performed by robot i at time t, then the destination $\bar{c}_i[t]$ is between the robot and the real center of gravity. Therefore $|\bar{R}_{cog}[t] - \bar{c}_i[t]| + |\bar{c}_i[t] - \bar{c}_i[t]| = |\bar{c}_i[t^*] - \bar{R}_{cog}[t]|$ for any $t^* \in [t', t]$, where t' is the end of the last move performed by robot i. Therefore, ϕ is unchanged by the Look performed.

Now consider some time interval $[t'_0, t'_1] \subseteq [t_0, t_1]$, such that no Look operations were performed during $[t'_0, t'_1]$. Suppose that during this interval each robot i moved a distance Δ_i (where some of these distances may be 0). Then ϕ_2 decreased by $\sum_i \Delta_i$, the maximum change in the center of gravity is $|\bar{R}_{cog}[t_1] - \bar{R}_{cog}[t_0]| \leq \sum_i \Delta_i / N$, and the robots' calculated centers of gravity have not changed. Therefore, the change in ϕ_1 is at most $\phi_1[t_1] - \phi_1[t] \leq \sum_i \Delta_i$. Hence, the sum $\phi = \phi_1 + \phi_2$ cannot increase. ∎

LEMMA 6.3. *ψ is a nonincreasing function of time.*

Proof. By Lemma 6.2, ϕ is nonincreasing. By Corollary 6.1, h is nonincreasing. Therefore their sum is also nonincreasing. ∎

LEMMA 6.4. *$h \leq \psi \leq 2h$.*

Proof. The lower bound is trivial. For the upper bound, notice that ϕ is the sum of $2N$ summands, each of which is at most h (since they all reside in the segment). ∎

We now state a lemma which allows the analysis of the change in ϕ (and therefore also ψ) in terms of the contributions of individual robots.

LEMMA 6.5. *If by the individual action of a robot i alone, in the time interval $[t_0, t_1]$ its contribution to ϕ is δ_i, then $\phi[t_1] \leq \phi[t_0] + \delta_i$.*

Proof. Lemma 6.2 implies that Look actions have no effect on ϕ and therefore can be disregarded. A robot moving a distance Δ_i will always decrease its term in ϕ_2 by Δ_i, and the motions of other robots have no effect on this term. Denote by Δ_j the motions of the other robots. Notice that

$$\left| \bar{R}_{cog} + \frac{\Delta_i}{N} + \frac{1}{N} \sum_{j \neq i} \Delta_j - \bar{c}_k \right| \ \leq \ \left| \bar{R}_{cog} + \frac{\Delta_i}{N} - \bar{c}_k \right| + \frac{1}{N} \sum_{j \neq i} |\Delta_j|.$$

The function ϕ_1 contains N summands, each of which contains a contribution of at most $\frac{1}{N}|\Delta_j|$ from every robot $j \neq i$. Therefore, the total contribution of each robot to ϕ_1 is at most $|\Delta_j|$, which is canceled by the negative contribution of $|\Delta_j|$ to ϕ_2. ∎

LEMMA 6.6. *If at some time t_0, for some $0 < A < 1$, the convex hull of the robot group $H'[t]$ satisfies $|H'[t_0]| \leq (1-A)h[t_0]$, then there exists a time $t > t_0$ such that $\psi[t] \leq (1 - \frac{A}{4N^2})\psi[t_0]$.*

Proof. Take t^* to be the time after t_0 where all robots finished a complete Look-Compute-Move cycle. If for some time $t_1 \in [t_0, t]$ it held that $h[t_1] \leq (1 - \frac{A}{2})h[t_0]$, then $\psi[t_1] < (1 - \frac{A}{4N^2})\psi[t_0]$, and we are done. If no such time (t_1) existed, then at time t_0 at least one robot had its calculated destination at a distance $\frac{A}{4}h[t_0]$ from the convex hull of the robot group. Suppose robot k had the most distant destination from the group, $\Delta_k \geq \frac{A}{4}h[t_0]$. By the movement of robot k alone, the real center of gravity moves a distance $\frac{\Delta_k}{N}$ towards robot k's perceived center of gravity. Therefore, by its motion ψ decreases by $\frac{2\Delta_k}{N^2}$. By Lemma 6.5 the decrease of ψ by the motion of all robots is at least the same, thus proving the claim. ∎

We now proceed to prove the main theorem regarding convergence in the asynchronous model.

LEMMA 6.7. *For all times t_0 there exist a time $t_1 > t_0$ and a constant δ such that $\psi[t_1] \leq (1 - \frac{\delta}{16N^2})\psi[t_0]$.*

Proof. We have established that ψ is a nonincreasing function of time. Now assume that at time t_0, $H[t_0]$ is the convex hull of the locations of the robots and their destinations. Without loss of generality we assume $H[t_0] = [0, 1]$. Take $t^* > t_0$ to be the time when all robots completed at least one full cycle. If at some time $t \in (t_0, t^*]$ the size of the interval occupied by the robots is $h[t^*] \leq 1 - \delta$, we are done by Lemma 6.6.

Suppose now that at all times $t \in (t_0, t^*]$ the robots' interval never shrinks to $1-\delta$. This implies that there exist robots in the intervals $[0, \delta]$ and $[1-\delta, 1]$. Now, for each robot, i, there exists at least one robot, j, with distance at least $V_j^i \geq \frac{1-\delta}{2}$ from it, and this robot is viewed at a distance at least $v_j^i \geq (1-\varepsilon_c)\frac{1-\delta}{2} \geq (1-\epsilon_0)\frac{1-\delta}{2}$. Therefore, it follows that $\rho^i \geq \frac{\epsilon_0(1-\delta)}{2} \geq \frac{\epsilon_0}{2}$, and thus no robot calculates its destination to within $\frac{\epsilon_0}{2}$ of $\{0, 1\}$. If we take $\delta < \frac{\epsilon_0}{2}$, then no robot approaches a distance δ from the boundary.

Now take the leftmost and rightmost robots $k_l = \arg\min_i \bar{R}_i$ and $k_r = \arg\max_i \bar{R}_i$ and take $t_l > t^*$ and $t_r > t^*$ to be their next Look times. By assumption $\bar{R}_{k_l}[t^*] \in [0, \delta]$ and $\bar{R}_{k_r}[t^*] \in [1-\delta, 1]$. Assume, without loss of generality, that $t_l \leq t_r$. Then, for robot k_l, either $\bar{c}_{k_l}[t^*] \geq \delta$ or $\bar{c}_{k_l}[t_l] \geq \delta$, or otherwise robot k_l decided to maintain its position. This means that the real center of gravity at time t_l was within a distance $\rho_{k_l} + err_{k_l}$ of $\bar{R}_{k_l}$. For $\epsilon_0 < 0.2$, $\rho_{k_l} + err_{k_l} \leq (\epsilon_0 + \epsilon_0\frac{1+\epsilon_0}{1-\epsilon_0})D_{max} < \frac{1}{2}$. Take $2\delta < \frac{1}{2} - \epsilon_0 - \epsilon_0\frac{1+\epsilon_0}{1-\epsilon_0}$. Since $\bar{R}_{k_l} \in [0, \delta]$, it follows that the center of gravity at time t_l was in the interval $\bar{R}_{cog}[t_l] \in [0, \frac{1}{2} - \delta]$. Now again for k_r, either $\bar{c}_{k_r}[t^*] \leq 1 - \delta$ or $\bar{c}_{k_r}[t_r] \leq 1 - \delta$ or the center of gravity has moved to the interval $[\frac{1}{2} + \delta, 1]$, in which case the center of gravity moved by at least 2δ approaching $\bar{c}_{k_r}\bar{R}_{k_r}$, leading to the desired decrease in ψ. Thus, no robot enters the δ environment of the boundary, and either one of the robots in this environment has left or ψ decreased by the desired amount. Since the number of robots is finite, after a finite number of such steps it is guaranteed that either ψ decreased by the desired amount or the interval shrunk by at least δ, leading again to the desired decrease in ψ. ∎

Lemma 6.7 yields the following theorem.

THEOREM 6.8. *In the $\langle \mathcal{ASYNC}, \mathcal{ERR}^- \rangle$ model, N robots performing Algorithm* `RCG` *converge on the line.*

We make the following more general conjecture.

CONJECTURE 6.9. *Algorithm* `RCG` *converges in the $\langle \mathcal{ASYNC}, \mathcal{ERR} \rangle$ model for sufficiently small error in the angle and distance measurements.*

6.2. Separating Go_to_COG from RCG in the $\mathcal{ASYNC}$ model. This section establishes the advantage of Algorithm RCG over the basic Algorithm Go_to_COG. In the fully synchronous case there is no justification for using the more involved Algorithm RCG, since the simpler Algorithm Go_to_COG also guarantees convergence as shown above in Lemma 4.2.

However, a gap between the two algorithms can be established in the fully asynchronous model. Specifically, we now show that the ordinary center-of-gravity algorithm Go_to_COG does not converge in the $\langle \mathcal{ASYNC}, \mathcal{ERR}^- \rangle$ model, even when the robots are positioned on a straight line. Contrasting this result with Theorem 6.8 yields the claimed separation between the two algorithms.

THEOREM 6.10. *In the $\langle \mathcal{ASYNC}, \mathcal{ERR}^- \rangle$ model, for every ε_d and $N > 1/\varepsilon_d$ there exists an activation schedule for which Algorithm* Go_to_COG *does not converge, even when the robots are restricted to a line.*

Proof. Start with a configuration in which the first robot is at $\bar{R}_1 = 0$ and the other $N-1$ robots are located at $\bar{R}_i = 1$ for $i = 2, \ldots, N$. Robot 1 makes a Look, and sees the other robots at location $1 + \varepsilon_{\mathrm{d}}$. While robot 1 is at its Compute phase, the other robots go through a long sequence of cycles, leading them to a distance $\delta \ll \varepsilon_{\mathrm{d}}$ from robot 1. Robot 1 now finishes its Compute phase, concludes that the center of gravity is at location $\frac{N-1}{N}(1+\varepsilon_{\mathrm{d}})$, and moves to this location. The robots are now at a setup similar to the initial setup but with a distance of $1 - \frac{1}{N} + \frac{N-1}{N}\varepsilon_{\mathrm{d}} - \delta > 1$ between robot 1 and the rest of the robots. Repeating this process leads to the divergence of the algorithm. ∎

7. Conclusions. We have discussed the feasibility of robot swarm convergence under conditions of inaccuracy in the robots' sensors, calculations, and movements. We have presented several impossibility results under various conditions and have discussed the inadequacy of existing algorithms. We then presented an algorithm based on restricted movement to the center of gravity of the robot swarm and have proved its correctness under a range of the inaccuracy parameters.

We have also shown that in the case of the one-dimensional asynchronous model the restricted center-of-gravity algorithm guarantees convergence, while the standard center-of-gravity algorithm fails to converge under certain circumstances.

Establishing tight bounds for the convergence rate remains an open problem for future study. Our proofs provide only some trivial bounds stemming from the time required to traverse the distance between the two farthest robots. In the model allowing possible premature stopping of a robot after traversing a distance of at least S, this time (in terms of number of steps) is at least the distance between the two farthest robots divided by $2S$. Another trivial bound on the convergence rate of the center-of-gravity algorithm Go_to_COG follows from the ability of the adversary to increase or decrease the distance viewed from a robot to all other robots by a factor of $1 \pm \varepsilon_{\mathrm{d}}$. This implies that even in the fully synchronous model, instead of meeting at the center of gravity, after every step the robots may reach a configuration similar to their previous one, except with all distances multiplied by ε_{d}.

REFERENCES

[1] N. AGMON AND D. PELEG, *Fault-tolerant gathering algorithms for autonomous mobile robots*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 1063–1071.

[2] H. ANDO, Y. OASA, I. SUZUKI, AND M. YAMASHITA, *A distributed memoryless point convergence algorithm for mobile robots with limited visibility*, IEEE Trans. Robot. Autom., 15 (1999), pp. 818–828.

[3] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell, and M. Skutella, *The freeze-tag problem: How to wake up a swarm of robots*, Algorithmica, 46 (2006), pp. 193–221.

[4] T. Balch and R. Arkin, *Behavior-based formation control for multirobot teams*, IEEE Trans. Robot. Autom., 14 (1998), pp. 926–939.

[5] G. Beni and S. Hackwood, *Coherent swarm motion under distributed control*, in Proceedings of the First International Symposium on Distributed Robotic Systems (DARS'92), 1992, pp. 39–52.

[6] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, *Cooperative mobile robotics: Antecedents and directions,* Auton. Robots, 4 (1997), pp. 7–27.

[7] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro, *Solving the robots gathering problem*, in Proceedings of the 30th Annual International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, pp. 1181–1196.

[8] M. Cieliebak and G. Prencipe, *Gathering autonomous mobile robots*, in Proceedings of the 9th Annual International Colloquium on Structural Information and Communication Complexity, 2002, pp. 57–72.

[9] E. J. Cockayne and Z. A. Melzak, *Euclidean constructibility in graph minimization problems*, Math. Mag., 42 (1969), pp. 206–208.

[10] R. Cohen and D. Peleg, *Robot convergence via center-of-gravity algorithms*, in Proceedings of the 11th Annual Colloquium on Structural Information and Communication Complexity, Lecture Notes in Comput. Sci. 3104, Springer-Verlag, Berlin, 2004, pp. 79–88.

[11] R. Cohen and D. Peleg, *Convergence properties of the gravitational algorithm in asynchronous robot systems*, SIAM J. Comput., 34 (2005), pp. 1516–1528.

[12] X. Defago and A. Konagaya, *Circle formation for oblivious anonymous mobile robots with no common sense of orientation*, in Proceedings of the Second Annual ACM Workshop on Principles of Mobile Computing, ACM Press, New York, 2002, pp. 97–104.

[13] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots*, in Proceedings of the 10th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 1741, Springer-Verlag, London, 1999, p. 93–102.

[14] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, *Gathering of asynchronous mobile robots with limited visibility*, in Proceedings of the 18th Annual Symposium on Theoretial Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 2010, Springer-Verlag, Berlin, 2001, pp. 247–258.

[15] B. V. Gervasi and G. Prencipe, *Coordination without communication: The case of the flocking problem*, Discrete Appl. Math., 143 (2004), pp. 203–223.

[16] H. Goldstein, *Classical Mechanics*, Addison-Wesley, Reading, MA, 1980.

[17] D. Jung, G. Cheng, and A. Zelinsky, *Experiments in realising cooperation between autonomous mobile robots*, in Proceedings of the Fifth International Symposium on Experimental Robotics, Lecture Notes in Control and Inform. Sci. 232, Springer-Verlag, London, 1997, pp. 609–620.

[18] Y. Kawauchi, M. Inaba, and T. Fukuda, *A principle of decision making of cellular robotic system (CEBOT)*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Press, Piscataway, NJ, 1993, pp. 833–838.

[19] M. J. Mataric, *Interaction and Intelligent Behavior*, Ph.D. thesis, MIT, Cambridge, MA, 1994.

[20] S. Murata, H. Kurokawa, and S. Kokaji, *Self-assembling machine*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Press, Piscataway, NJ, 1994, pp. 441–448.

[21] L. E. Parker, *Designing control laws for cooperative agent teams*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Press, Piscataway, NJ, 1993, pp. 582–587.

[22] L. E. Parker, *On the design of behavior-based multi-robot teams*, J. Adv. Robotics, 10 (1996), pp. 547–578.

[23] L. E. Parker and C. Touzet, *Multi-robot learning in a cooperative observation task*, in Distributed Autonomous Robotic Systems, Vol. 4, Springer-Verlag, Berlin, 2000, pp. 391–401.

[24] L. E. Parker, C. Touzet, and F. Fernandez, *Techniques for learning in multi-robot teams*, in Robot Teams: From Diversity to Polymorphism, T. Balch and L. E. Parker, eds., A K Peters, Wellesley, MA, 2002, pp. 191–236.

[25] SensComp Inc., *Specification of* 6500 *Series Ranging Modules*, http://www.senscomp.com.

[26] K. Sugihara and I. Suzuki, *Distributed algorithms for formation of geometric patterns with many mobile robots*, J. Robotic Systems, 13 (1996), pp. 127–139.

[27] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots—Formation and agreement problems*, in Proceedings of the 3rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), 1996, pp. 313–330.

[28] I. Suzuki and M. Yamashita, *Distributed anonymous mobile robots: Formation of geometric patterns*, SIAM J. Comput., 28 (1999), pp. 1347–1363.

[29] M. Sztainberg, E. Arkin, M. Bender, and J. Mitchell, *Analysis of heuristics for the freeze-tag problem*, in Proceedings of the 8th Annual Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 2368, Springer-Verlag, Berlin, 2002, pp. 270–279.

[30] I. A. Wagner and A. M. Bruckstein, *From ants to a(ge)nts*: A special issue on ant-robotics, Ann. Math. Artif. Intell., 31 (2001), pp. 1–5.

[31] A. Weber, *Theory of the Location Industries*, University of Chicago Press, Chicago, 1937.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 303–328

PLOTTABLE REAL NUMBER FUNCTIONS AND THE COMPUTABLE GRAPH THEOREM*

VASCO BRATTKA†

Abstract. The Graph Theorem of classical recursion theory states that a total function on the natural numbers is computable if and only if its graph is recursive. It is known that this result can be generalized to real number functions where it has an important practical interpretation: the total computable real number functions are precisely those which can be effectively plotted with any given resolution. We generalize the Graph Theorem to appropriate partial real number functions and even further to functions defined on certain computable metric spaces. Besides the nonuniform version of the Graph Theorem which logically relates computability properties of the function and computability properties of its graph, we also discuss the uniform version: given a program of a function, can we algorithmically derive a description of its graph? And, vice versa, given a description of the graph, can we derive a program of the function? While the passage from functions to graphs is always computable, the inverse direction from graphs to functions is problematic, and it turns out that the answers to the uniform and the nonuniform questions do not coincide. We prove that in both cases certain topological and computational properties (such as compactness or effective local connectedness) are sufficient for a positive answer, and we provide counterexamples which show that the corresponding properties are not superfluous. Additionally, we briefly discuss the special situation of the linear case.

Key words. computable real number functions, recursive graphs

AMS subject classifications. 03F60, 03D45

DOI. 10.1137/060658023

1. Introduction.

The Graph Theorem of classical recursion theory states that a total function $f : \mathbb{N} \to \mathbb{N}$ on the natural numbers $\mathbb{N} := \{0, 1, 2, \ldots\}$ is computable if and only if its graph, i.e., the set $\text{graph}(f) := \{(n, f(n)) : n \in \mathbb{N}\} \subseteq \mathbb{N} \times \mathbb{N}$, is recursive [17]. In particular, this result shows that the notions of a computable function and of a recursive subset are logically equivalent (in the sense that one could be derived from the other).

We will investigate generalizations of the Graph Theorem from the point of view of computable analysis, which is the Turing machine based theory of computability on real numbers and other topological spaces. Pioneering work on this theory has been presented by Turing [19], Banach and Mazur [1], Lacombe [14, 15], and Grzegorczyk [12]. Recent monographs have been published by Pour-El and Richards [18], Ko [13], and Weihrauch [20].

Roughly speaking, a real number function $f : \mathbb{R}^n \to \mathbb{R}$ is computable in the sense of computable analysis if and only if there exists a Turing machine that transforms better and better rational approximations of x into corresponding approximations of $f(x)$. Based on the well-known physical Church–Turing thesis [17] one might derive a thesis that underlines that this notion has a direct relation to the abilities of physical computers.

*Received by the editors June 28, 2006; accepted for publication (in revised form) August 27, 2007; published electronically April 18, 2008. This work was supported by the National Research Foundation (NRF).

http://www.siam.org/journals/sicomp/38-1/65802.html

†Laboratory of Foundational Aspects of Computer Science, Department of Mathematics and Applied Mathematics, University of Cape Town, Rondebosch 7701, South Africa (Vasco.Brattka@uct.ac.za).

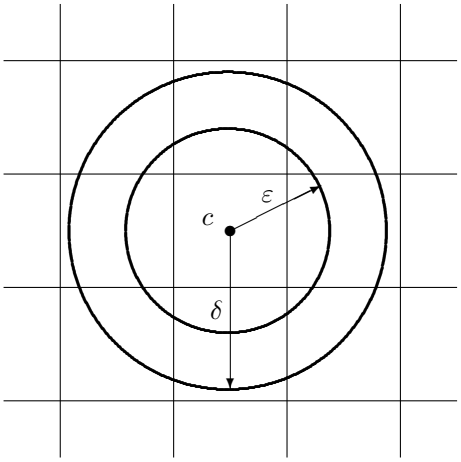


FIG. 1.1. *Determination of a pixel color.*

THESIS 1.1. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is computable if and only if it can be evaluated on a physical computer with arbitrary given precision.*

It is an important feature of computable analysis that the Graph Theorem can be generalized to continuous real number functions $f : \mathbb{R} \to \mathbb{R}$ (as discussed in [20], this fact was first proved for continuous functions $f : [0,1] \to \mathbb{R}$ by Zhou [23]).[1] Thus, a continuous function $f : \mathbb{R} \to \mathbb{R}$ is computable if and only if its graph $\text{graph}(f) := \{(x, f(x)) : x \in \mathbb{R}\} \subseteq \mathbb{R} \times \mathbb{R}$ is recursive.

Here, a closed subset $A \subseteq \mathbb{R} \times \mathbb{R}$ is called *recursive* if and only if its distance function $d_A : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, (x,y) \mapsto \inf_{(a,b)\in A} ||(x,y) - (a,b)||$ is computable (where $||(x,y)|| := \max\{|x|, |y|\}$ denotes the maximum norm). Similarly to the computable functions, the recursive subsets also have a very practical meaning: these are exactly those subsets that can be effectively displayed.

This is easy to see since, given a screen with pixels of a certain size, the color of any given pixel can be determined by computing the distance $d_A(c)$ from the center c of the pixel to the set A (cf. the discussions in [8, 20] or in [9]). If ε is the radius of a ball with center c that completely contains the pixel and if $\delta > \varepsilon$ is the radius of such a ball that is completely contained in the pixel and its immediate eight neighbor pixels, then $d_A(c) < \delta$ could lead to a black pixel and $d_A(c) > \varepsilon$ to a white pixel (depending on which inequality can be verified first; see Figure 1.1). Since in principle the decision cannot be made with infinite precision, there will always remain a certain area of indetermination (in our case the values in the interval $\varepsilon < d_A(c) < \delta$) where the pixel might be white as well as black. However, this procedure guarantees that, on the one hand, the pixel will be black if the set hits the pixel and that, if the pixel is black, then the set does hit at least one of the neighbor pixels or the pixel itself. Altogether, this justifies the derivation of the following thesis.

THESIS 1.2. *A closed subset $A \subseteq \mathbb{R}^2$ is recursive if and only if it can be displayed by a physical computer for an arbitrary given resolution.*

If we apply these considerations to functions $f : \mathbb{R} \to \mathbb{R}$ with a recursive graph, then we obtain that these functions are *plottable* in the sense that there exists an

[1] The Graph Theorem does not hold for real number functions in the so-called BSS model, where it holds only for algebraically closed fields [11].

algorithm that can display any part of the graph for any given resolution. Thus, in particular, any computable function f is plottable.

Unfortunately, this pleasant property does not hold true for partial functions. Topologically, $\mathrm{graph}(f) := \{(x, f(x)) : x \in \mathrm{dom}(f)\} \subseteq \mathbb{R} \times \mathbb{R}$ is not even closed in general if $f :\subseteq \mathbb{R} \to \mathbb{R}$ is a partial computable function. And even if the graph is closed, it is not necessarily recursive, as the following counterexample shows.

Example 1.3. There exists a partial computable function $f :\subseteq \mathbb{R} \to \mathbb{R}$ with $\mathrm{dom}(f) = \mathbb{R} \setminus \mathbb{N}$ such that $\mathrm{graph}(f) \subseteq \mathbb{R} \times \mathbb{R}$ is closed but not recursive closed.

Proof. Let $a : \mathbb{N} \to \mathbb{N}$ be some computable and injective enumeration of a non-recursive set $K = \mathrm{range}(a)$. Let the partial function $t :\subseteq \mathbb{R} \to \mathbb{R}$ be defined by $t(x) := \tan(\pi(x - \frac{1}{2}))$. Then t is a computable function with $\mathrm{dom}(t) = \mathbb{R} \setminus \mathbb{N}$. We will define $f :\subseteq \mathbb{R} \to \mathbb{R}$ by modifying t. Therefore, for any $n \in K = \mathrm{range}(a)$ let $p_n : [n + 2^{-a^{-1}(n)-4}, n + 2^{-a^{-1}(n)-2}] \to \mathbb{R}$ be the polygon defined by the vertices

$$\left(n + 2^{-a^{-1}(n)-4}, t(n + 2^{-a^{-1}(n)-4})\right),$$

$$\left(n + 2^{-a^{-1}(n)-3}, 0\right),$$

$$\left(n + 2^{-a^{-1}(n)-2}, t(n + 2^{-a^{-1}(n)-2})\right).$$

Now we define $f :\subseteq \mathbb{R} \to \mathbb{R}$ by

$$f(x) := \begin{cases} p_n(x) & \text{if } (\exists n \in K)\ x \in [n + 2^{-a^{-1}(n)-4}, n + 2^{-a^{-1}(n)-2}] \\ t(x) & \text{otherwise} \end{cases},$$

for all $x \in \mathbb{R}$. Figure 1.2 displays the graph of f on some interval $[n, n + 1]$ with the graph of t and the alternative part p_n as a dotted line. Then f with $\mathrm{dom}(f) = \mathbb{R} \setminus \mathbb{N}$ is computable since a is computable. On the other hand, $\mathrm{graph}(f) \subseteq \mathbb{R} \times \mathbb{R}$ is obviously closed but not recursive closed, since $K = \mathrm{range}(a)$ is not co–recursively enumerable (co-r.e.) and

$$n \notin K \iff d_{\mathrm{graph}(f)}\left(n + \frac{1}{8}, 0\right) > \frac{1}{8}. \qquad \square$$

The intuitive reason for this inconsistency between computability of partial functions and plottability of their graphs is the following: the computation of a function might take longer when we approach the singularities, while the decision for any pixel in a plot of its graph has to be made after finite time. In other words, the topology of the display does not take care of the singularities. It is worth mentioning that such pathologies are well known in practice. Figure 1.3 shows a standard plot of the tangent function (produced by gnuplot 3.7.3 with default options).

Of course, Example 1.3 exploits the fact that the domain of the function has countably many singularities. However, if one is prepared to accept a counterexample with a graph that is not closed (in $\mathbb{R} \times \mathbb{R}$), then there is such an example even for the domain $(0, 1)$.

Example 1.4. Let $X = (0, 1)$ and $Y = \mathbb{R}$. There exists a continuous and computable function $f : X \to Y$ such that $\overline{\mathrm{graph}(f)} \subseteq \mathbb{R} \times \mathbb{R}$ is not recursive closed. The same holds true for $Y = [0, 1]$.

Proof. Let $(a_i)_{i \in \mathbb{N}}$ be an increasing sequence of positive real numbers that converges to a left but not right computable real number $a < 1$. We define a function

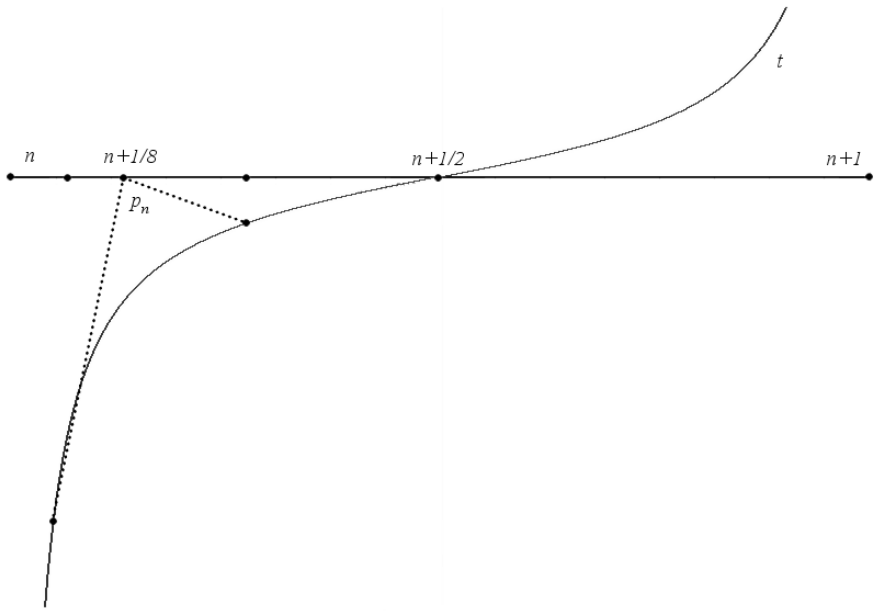


FIG. 1.2. *The function f on $[n, n+1]$ in case of $a^{-1}(n) = 0$.*

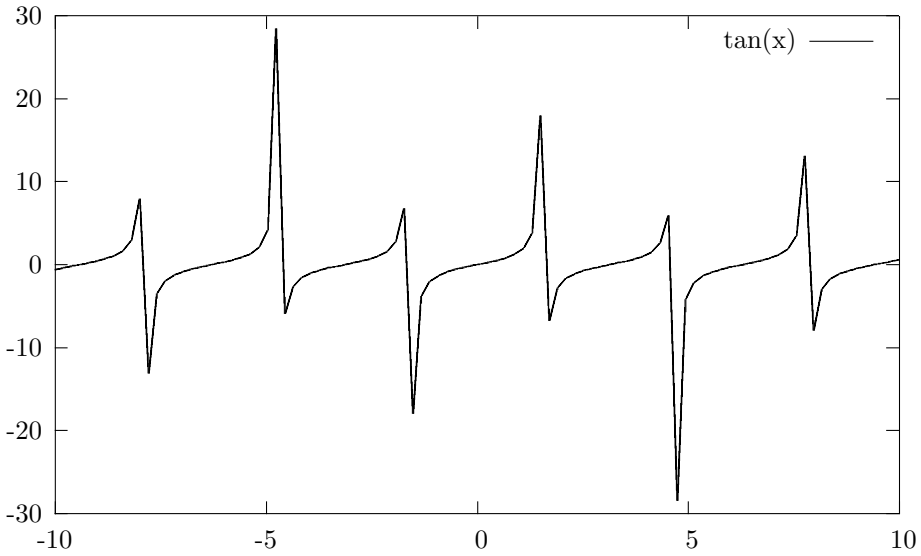


FIG. 1.3. *An unsatisfactory standard plot of the tangent function.*

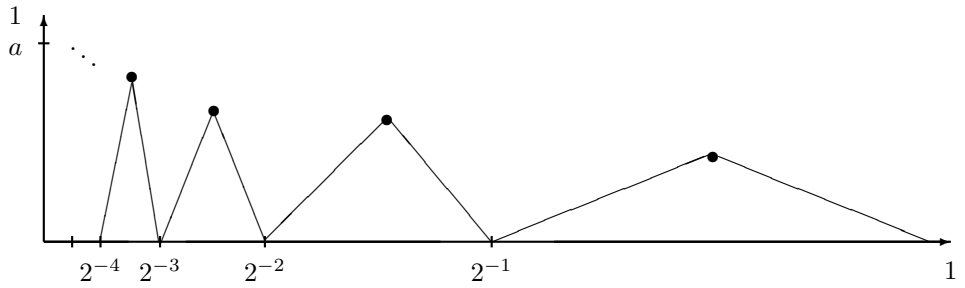


FIG. 1.4. *The function f.*

$f : (0,1) \to \mathbb{R}$ as follows. For any $n \in \mathbb{N}$ let $p_n : [2^{-n-1}, 2^{-n}] \to \mathbb{R}$ be the rational polygon with vertices

$$\left(2^{-n-1}, 0\right), \left(2^{-n-1} + 2^{-n-2}, a_n\right), \left(2^{-n}, 0\right),$$

and let $f(x) := p_n(x)$ for any $x \in [2^{-n-1}, 2^{-n})$ and $n \in \mathbb{N}$. The function f is displayed in Figure 1.4. The function $f : (0,1) \to \mathbb{R}$ defined in this way is obviously computable. However, $\overline{\text{graph}(f)} \cap (\{0\} \times \mathbb{R}) = \{0\} \times [0, a]$, and hence the distance function $\text{dist}_{\overline{\text{graph}(f)}} : \mathbb{R}^2 \to \mathbb{R}$ of $\overline{\text{graph}(f)} \subseteq \mathbb{R}^2$ is not computable. □

These examples show that even for very simple domains $X \subseteq \mathbb{R}$ it is no longer true that the graph of a computable function $f : X \to \mathbb{R}$ is recursive as a subset of $\mathbb{R} \times \mathbb{R}$, and thus it is not necessarily plottable. However, as we will see, it is always true and a property of interest that the graph is recursive, considered as a subset of $X \times \mathbb{R}$. This change of the topology has a practical interpretation in our examples above: when we approach the singularities the resolution has to be increased correspondingly.

Actually, we will even address a slightly more general question in the following (without much extra costs) where we consider as domain of our functions f certain metric spaces X (and in some cases we will consider metric spaces Y for the images as well). In this general case the intuitive notion of plottability is no longer meaningful. However, our formal notion of recursiveness is still available and has important applications in theory. Altogether, we arrive at the following problem.

Nonuniform graph problem. Which computable metric spaces X and Y have the property that for all continuous functions $f : X \to Y$ it holds that f is computable if and only if $\text{graph}(f) = \{(x, f(x)) : x \in X\} \subseteq X \times Y$ is recursive?

This nonuniform question relates a computability property of f just logically to a computability property of $\text{graph}(f)$. But given f, can we effectively compute $\text{graph}(f)$ from f? And, vice versa, given $\text{graph}(f)$, can we effectively compute f? This uniform question can be formulated as follows.

Uniform graph problem. Which computable metric spaces X and Y have the property that the mapping $\text{graph} : \mathcal{C}(X,Y) \to \mathcal{A}(X \times Y), f \mapsto \text{graph}(f)$ and its inverse are computable?

Here, by $\mathcal{C}(X,Y)$ we denote the set of continuous functions $f : X \to Y$ and by $\mathcal{A}(X \times Y)$ the set of closed subsets of $X \times Y$.

We are far from presenting a complete solution of the above questions, and we will mainly concentrate on the case $Y = \mathbb{R}^n$. Even in this special case it appears that the answers to these questions sensitively rely on topological and computational properties of the space X, and even in case of important and well-understood concrete spaces X we obtain a somewhat surprising variety of answers (see Figure 10.1).

We close this introduction with a short survey on the organization of this paper. In section 2 we will provide some basic concepts of computable analysis, and section 3 is devoted to computable metric spaces. To make all the above problems precise, we have, for instance, to define computability on the hyperspace $\mathcal{A}(X \times Y)$ of closed subsets, and we have to define an appropriate notion of recursiveness. The corresponding definitions will be presented in section 4. In section 5 we discuss one direction of the Graph Theorem that we will call the weak Graph Theorem: any total computable function $f : X \to Y$ has a recursive graph, and this holds even uniformly. In sections 6, 7, 8, and 9, we discuss different properties on computable metric spaces that guarantee that the inverse direction of the Graph Theorem holds as well. Such conditions are recursive compactness of the target space or different effective connectedness properties of the source space. Finally, in section 10 we give a survey of our results.

2. Preliminaries from computable analysis. In this section we briefly summarize some notions from computable analysis. For details the reader is referred to [20]. The basic idea of the representation based approach to computable analysis is to represent infinite objects like real numbers, functions, or sets by infinite strings over some alphabet Σ (which should contain at least the symbols 0 and 1). Thus, a *representation* of a set X is a surjective mapping $\delta :\subseteq \Sigma^\omega \to X$, and in this situation we will call (X, δ) a *represented space*. Here Σ^ω denotes the set of infinite sequences over Σ, and the inclusion symbol is used to indicate that the mapping might be partial. If we have two represented spaces, then we can define the notion of a computable function.

DEFINITION 2.1 (computable function). *Let (X, δ) and (Y, δ') be represented spaces. A function $f :\subseteq X \to Y$ is called* (δ, δ')-computable *if there exists some computable function $F :\subseteq \Sigma^\omega \to \Sigma^\omega$ such that $\delta' F(p) = f\delta(p)$ for all $p \in \mathrm{dom}(f\delta)$.*

Of course, we have to define computability of functions $F :\subseteq \Sigma^\omega \to \Sigma^\omega$ to make this definition complete, but this can be done via Turing machines: F is *computable* if there exists some Turing machine which computes for an infinitely long time and transforms each sequence p, written on the input tape, into the corresponding sequence $F(p)$, written on the one-way output tape. Later, we will also need computable multivalued operations $f :\subseteq X \rightrightarrows Y$, which are defined analogously to computable functions by substituting $\delta' F(p) \in f\delta(p)$ for the equation in Definition 2.1 above. The intuition behind the concept of computable multivalued operations f is that a computable realization F selects upon input of a name of x one of the possible values in $f(x)$. In some application $f(x)$ could be, for instance, the set of solutions of some equation with parameter x. This set could have continuum cardinality, but in practice we might just be interested in finding one arbitrary solution $y \in f(x)$. If the represented spaces are fixed or clear from the context, then we will simply call a function or operation f *computable*.

Analogously to the notion of computability we can define the notion of (δ, δ')-*continuity* for single- and multivalued operations by substituting a continuous function $F :\subseteq \Sigma^\omega \to \Sigma^\omega$ for the computable function F in the definitions above. On Σ^ω we use the *Cantor topology*, which is simply the product topology of the discrete topology on Σ. Again we will simply say that the corresponding function is *continuous* if the representations are fixed or clear from the context. If not mentioned otherwise, we will always assume that a represented space is endowed with the final topology induced by its representation.

This will lead to no confusion with the ordinary topological notion of continuity, as long as we are dealing with *admissible* representations. A representation δ of a topological space X is called *admissible* if δ is maximal among all continuous repre-

sentations δ' of X in the sense that the identity $\mathrm{id} : X \to X$ is (δ', δ)-continuous for all such δ'. If δ, δ' are admissible representations of T_0-spaces with countable bases, X, Y, then a function $f :\subseteq X \to Y$ is (δ, δ')-continuous if and only if it is continuous in the ordinary topological sense [20].

Given a represented space (X, δ), we will occasionally use the notions of a *computable sequence* and a *computable point*. A *computable sequence* is a computable function $f : \mathbb{N} \to X$, where we assume that $\mathbb{N} = \{0, 1, 2, \ldots\}$ is represented by $\delta_{\mathbb{N}}(1^n 0^\omega) := n$, and a point $x \in X$ is called *computable* if there is a constant computable function with value x.

Given two represented spaces (X, δ) and (Y, δ'), there is a canonical representation $[\delta, \delta']$ of $X \times Y$ and a representation $[\delta \to \delta']$ of certain functions $f : X \to Y$. If δ, δ' are *admissible* representations of T_0-spaces with countable bases, then $[\delta \to \delta']$ is actually a representation of the set $\mathcal{C}(X, Y)$ of continuous functions $f : X \to Y$. If $Y = \mathbb{R}$, then we write for short $\mathcal{C}(X) := \mathcal{C}(X, \mathbb{R})$. The function space representation can be characterized by the fact that it admits evaluation and type conversion.

PROPOSITION 2.2 (evaluation and type conversion). *Let $(X, \delta), (Y, \delta')$ be admissibly represented T_0-spaces with countable bases, and let (Z, δ'') be a represented space. Then the following hold:*

1. (Evaluation) $\mathrm{ev} : \mathcal{C}(X, Y) \times X \to Y, (f, x) \mapsto f(x)$ *is* $([[\delta \to \delta'], \delta], \delta')$*-computable.*
2. (Type conversion) $f : Z \times X \to Y$, *is* $([\delta'', \delta], \delta')$*-computable, if and only if the function $\check{f} : Z \to \mathcal{C}(X, Y)$, defined by $\check{f}(z)(x) := f(z, x)$, is $(\delta'', [\delta \to \delta'])$-computable.*

The proof of this proposition is based on a version of smn and utm theorems and can be found in [20]. If (X, δ), (Y, δ') are admissibly represented T_0-spaces with countable bases, then in the following we will always assume that $\mathcal{C}(X, Y)$ is represented by $[\delta \to \delta']$. It is known that the computable points in $(\mathcal{C}(X, Y), [\delta \to \delta'])$ are just the (δ, δ')-computable functions $f : X \to Y$ [20]. If (X, δ) is a represented space, then we will always assume that the set of sequences $X^{\mathbb{N}}$ is represented by $\delta^{\mathbb{N}} := [\delta_{\mathbb{N}} \to \delta]$. The computable points in $(X^{\mathbb{N}}, \delta^{\mathbb{N}})$ are just the computable sequences in (X, δ). Moreover, we assume that X^n is always represented by δ^n, which can be defined inductively by $\delta^1 := \delta$ and $\delta^{n+1} := [\delta^n, \delta]$.

3. Computable metric spaces. In this section we will briefly discuss computable metric spaces; see also [4] for further details. First, we just mention that we will denote the *open balls* of a metric space (X, d) by $B(x, \varepsilon) := \{y \in X : d(x, y) < \varepsilon\}$ for all $x \in X$, $\varepsilon > 0$ and correspondingly the *closed balls* by $\overline{B}(x, \varepsilon) := \{y \in X : d(x, y) \leq \varepsilon\}$. Occasionally, we denote complements of sets $A \subseteq X$ by $A^c := X \setminus A$.

DEFINITION 3.1 (computable metric space). *A tuple (X, d, α) is called a* computable metric space *if*

1. *$d : X \times X \to \mathbb{R}$ is a metric on X,*
2. *$\alpha : \mathbb{N} \to X$ is a sequence that is dense in X, and*
3. *$d \circ (\alpha \times \alpha) : \mathbb{N}^2 \to \mathbb{R}$ is a computable (double) sequence in $\mathbb{R}$.*

Here, we tacitly assume that the reader is familiar with the notion of a computable sequence of reals, but we will come back to that point below. Occasionally, we will say for short that X is a *computable metric space*. Obviously, a computable metric space is in particular separable. Given a computable metric space (X, d, α), its *Cauchy representation* $\delta_X :\subseteq \Sigma^\omega \to X$ can be defined by

$$\delta_X(01^{n_0+1}01^{n_1+1}01^{n_2+1}\ldots) := \lim_{i \to \infty} \alpha(n_i)$$

for all n_i such that $(\alpha(n_i))_{i\in\mathbb{N}}$ converges and $d(\alpha(n_i),\alpha(n_j)) \leq 2^{-i}$ for all $j > i$ (and undefined for all other input sequences). In the following we tacitly assume that computable metric spaces are represented by their Cauchy representations. If X is a computable metric space, then it is easy to see that $d : X \times X \to \mathbb{R}$ is computable (see Proposition 5.3 in [4]). All Cauchy representations are admissible with respect to the corresponding metric topology.

An important computable metric space is $(\mathbb{R}, d_{\mathbb{R}}, \alpha_{\mathbb{R}})$ with the Euclidean metric $d_{\mathbb{R}}(x,y) := |x-y|$ and some numbering $\alpha_{\mathbb{R}}\langle i,j,k\rangle := (i-j)/(k+1)$ of the rational numbers $\mathbb{Q}$. Here, $\langle i,j\rangle := \frac{1}{2}(i+j)(i+j+1)+j$ denotes *Cantor pairs* and this definition is extended inductively to finite tuples. For short we will occasionally write $\overline{k} := \alpha_{\mathbb{R}}(k)$. In the following we assume that $\mathbb{R}$ is endowed with the Cauchy representation $\delta_{\mathbb{R}}$ induced by the computable metric space given above. This representation of $\mathbb{R}$ can also be defined if $(\mathbb{R}, d_{\mathbb{R}}, \alpha_{\mathbb{R}})$ just fulfills (1) and (2) of the definition above, and this leads to a definition of computable real number sequences without circularity. Occasionally, we will also use the represented space $(\mathbb{Q}, \delta_{\mathbb{Q}})$ of rational numbers with $\delta_{\mathbb{Q}}(1^n0^\omega) := \alpha_{\mathbb{R}}(n) = \overline{n}$. We close this section with a brief discussion of metric product spaces.

PROPOSITION 3.2 (product spaces). *If (X,d,α), (Y,d',α') are computable metric spaces, then the product space $(X\times Y, d'', \alpha'')$, defined by*

$$d''((x,y),(x',y')) := \max\{d(x,x'),d'(y,y')\} \text{ and } \alpha''\langle i,j\rangle := (\alpha(i),\alpha'(j)),$$

is a computable metric space, too, and the canonical projections of the product space $\mathrm{pr}_1 : X\times Y \to X$ and $\mathrm{pr}_2 : X\times Y \to Y$ are computable.

Particularly, the corresponding Cauchy representations $\delta_{X\times Y}$ and $[\delta_X,\delta_Y]$ are computably equivalent. In the following we will tacitly use the computable metric spaces $\mathbb{R}^n$, $\{0,1\}^{\mathbb{N}}$, and $\mathbb{N}^{\mathbb{N}}$, defined in the standard way.

4. Hyperspaces of closed subsets. In this section we want to introduce computability on the hyperspace of closed subsets of computable metric spaces. At this point we have to carefully consider the fact that different equivalent characterizations of the notion of a recursive closed subset $A \subseteq \mathbb{R}^n$ split into inequivalent notions if we generalize them to computable metric spaces. The logical relation between several notions of effectivity of closed subsets of metric spaces is illustrated in Figure 4.1.

The displayed results have been obtained in [7] from a very uniform point of view and each arrow in the diagram indicates not only an implication but also an effective reducibility. Below, we will precisely define the notions that are relevant for the present paper. In case of Euclidean space (and certain other computable metric spaces) the vertical arrows can be reversed, and thus the three horizontal layers of the diagram collapse [8]. However, some examples prove that these notions have to be distinguished in the general case of computable metric spaces [7, 3]. If we consider a subset $A \subseteq \mathbb{N}$ as a closed subset of $\mathbb{R}$ (embedded in the canonical way), then the set is recursively enumerable (r.e.), co-r.e., and recursive in the classical sense if and

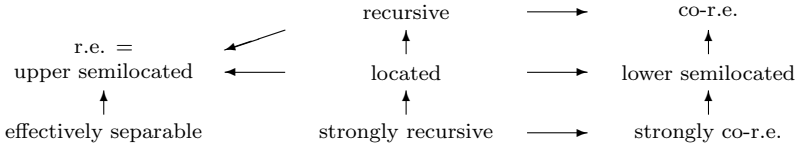


FIG. 4.1. *Notions of effectivity for closed subsets of computable metric spaces.*

only if it is r.e., co-r.e., and recursive in the metric sense, respectively. Thus, all the displayed notions can be considered as formal generalizations of the corresponding classical notions of r.e., co-r.e., and recursive sets.

Now the following question appears: Which of these notions fit together with the Graph Theorem? It could happen, for instance, that the graph of a computable function is recursive (but not strongly recursive in general), while only continuous functions with a strongly recursive graph are computable in general. However, our results show that this does not happen and actually the notion of "recursive" is the one which optimally fits the Graph Theorem. In the following definition we start to define corresponding hyperspace representations. Such representations have been studied in the Euclidean case in [8, 20] and for the metric case in [7].

DEFINITION 4.1 (hyperspace of closed subsets). *Let (X, d, α) be a computable metric space. We endow the hyperspace $\mathcal{A}(X) := \{A \subseteq X : A \text{ closed}\}$ of closed subsets with the representation $\delta^<_{\mathcal{A}(X)}$, defined by*

$$\delta^<_{\mathcal{A}(X)}(01^{\langle n_0,k_0\rangle+1}01^{\langle n_1,k_1\rangle+1}01^{\langle n_2,k_2\rangle+1}\ldots) = A$$

$$:\iff \{(n,k) : A \cap B(\alpha(n),\overline{k}) \neq \emptyset\} = \{(n_i,k_i) \in \mathbb{N}^2 : i \in \mathbb{N}\},$$

and with the representation $\delta^>_{\mathcal{A}(X)}$, defined by

$$\delta^>_{\mathcal{A}(X)}(01^{\langle n_0,k_0\rangle+1}01^{\langle n_1,k_1\rangle+1}01^{\langle n_2,k_2\rangle+1}\ldots) := X \setminus \bigcup_{i=0}^{\infty} B(\alpha(n_i),\overline{k_i}).$$

The intuition behind the representation $\delta^<_{\mathcal{A}(X)}$ is that a closed set A is represented by a list of positive information on the set A. Such positive pieces of information are just open rational balls. Analogously, $\delta^>_{\mathcal{A}(X)}$ represents closed sets A by negative information. A name of A is just a list of open rational balls whose union exhausts the complement of A. Whenever we have two representations δ, δ' of some set, we can define the *infimum* $\delta \sqcap \delta'$ of δ and δ' by $(\delta \sqcap \delta')\langle p,q\rangle = x :\iff \delta(p) = x$ and $\delta'(q) = x$. We use the short notation $\mathcal{A}_< = \mathcal{A}_<(X) = (\mathcal{A}_<(X), \delta^<_{\mathcal{A}(X)})$, $\mathcal{A}_> = \mathcal{A}_>(X) = (\mathcal{A}_>(X), \delta^>_{\mathcal{A}(X)})$, and $\mathcal{A} = \mathcal{A}(X) = (\mathcal{A}(X), \delta^<_{\mathcal{A}(X)} \sqcap \delta^>_{\mathcal{A}(X)})$ for the corresponding represented spaces. For the computable points of these spaces special names are used.

DEFINITION 4.2 (r.e. and recursive sets). *Let X be a computable metric space and let $A \subseteq X$ be a closed subset.*

(1) *A is called* r.e. closed *if A is a computable point in $\mathcal{A}_<(X)$.*
(2) *A is called* co-r.e. closed *if A is a computable point in $\mathcal{A}_>(X)$.*
(3) *A is called* recursive closed *if A is a computable point in $\mathcal{A}(X)$.*

Intuitively, a set A is r.e. closed if we can effectively enumerate all rational open balls that intersect the set, and A is called co-r.e. closed if we can enumerate sufficiently many open rational balls whose union is the complement of A. Finally, A is called recursive closed if it has both properties. We continue this section with some helpful results on hyperspaces, which follow directly from results in [7]. The first result states that we can represent closed subsets by preimages of continuous functions. It is an effective version of the statement that closed subsets of metric spaces coincide with the functional closed subsets.

PROPOSITION 4.3 (functional closed subsets). *Let X be a computable metric space. The map $Z : \mathcal{C}(X,\mathbb{R}) \to \mathcal{A}_>(X), f \mapsto f^{-1}\{0\}$ is computable and admits a computable right-inverse $\mathcal{A}_>(X) \rightrightarrows \mathcal{C}(X,\mathbb{R})$.*

The second result can be considered as an effective version of the statement that closed subsets of separable metric spaces are separable. Here and in the following $\overline{A}$ denotes the topological closure of a subset $A \subseteq X$ of some topological space X.

PROPOSITION 4.4 (separable closed subsets). *Let X be a computable metric space. The mapping $\overline{\text{range}} : X^{\mathbb{N}} \to \mathcal{A}_<(X), (x_n)_{n\in\mathbb{N}} \mapsto \overline{\{x_n : n \in \mathbb{N}\}}$ is computable and if X is complete, then it admits a computable multivalued partial right-inverse $\subseteq \mathcal{A}_<(X) \rightrightarrows X^{\mathbb{N}}$, defined for all nonempty closed subsets.*

We close this section with the definition of the remaining notions of effectivity for closed subsets that we are going to use in this paper.

DEFINITION 4.5 (strongly recursive and located subsets). *Let (X, d, α) be a computable metric space and let $A \subseteq X$ be a closed subset. Then the following hold:*

1. *A is called* strongly co-r.e. closed *if $\{\langle n, k\rangle \in \mathbb{N} : A \cap \overline{B}(\alpha(n), \overline{k}) = \emptyset\}$ is r.e.*
2. *A is called* strongly recursive closed *if A is r.e. closed and strongly co-r.e. closed.*
3. *A is called* located *if $A = \emptyset$ or the distance function $d_A : X \to \mathbb{R}, x \mapsto \inf_{y\in X} d(x, y)$ of A is computable.*

The definition of locatedness can even be applied to sets that are not closed.

5. The weak Graph Theorem. In this section we will study the easy direction of the Graph Theorem, and we will discuss a number of counterexamples that limit the possibilities for the problematic direction. We immediately obtain the following positive result, which is based on Propositions 4.4 and 4.3.

THEOREM 5.1 (weak Graph Theorem). *Let X and Y be computable metric spaces. The mapping $\text{graph} : \mathcal{C}(X, Y) \to \mathcal{A}(X \times Y), f \mapsto \text{graph}(f)$ is computable.*

Proof. We consider the computable metric spaces (X, d, α), (Y, d', β). Then, $\alpha : \mathbb{N} \to X$ is a computable function such that $\overline{\text{range}(\alpha)} = X$. If $f : X \to Y$ is continuous, then $S : X \to X \times Y, x \mapsto (x, f(x))$ is continuous, too, and because of continuity of S we obtain that $\text{range}(S\alpha)$ is dense in $\text{range}(S) = \text{graph}(f)$. Using evaluation and type conversion as well as Proposition 4.4, it follows that $\text{graph} : \mathcal{C}(X, Y) \to \mathcal{A}_<(X \times Y)$ is computable.

Now $U : X \times Y \to \mathbb{R}, (x, y) \mapsto d'(f(x), y)$ is continuous, since $f : X \to Y$ and the metric $d' : Y \times Y \to \mathbb{R}$ are continuous, and we obtain

$$U(x, y) = 0 \iff d'(f(x), y) = 0 \iff f(x) = y \iff (x, y) \in \text{graph}(f).$$

Thus $U^{-1}\{0\} = \text{graph}(f)$ and evaluation together with type conversion and Proposition 4.3 allow us to conclude that $\text{graph} : \mathcal{C}(X, Y) \to \mathcal{A}_>(X \times Y)$ is computable. Altogether, this shows that $\text{graph} : \mathcal{C}(X, Y) \to \mathcal{A}(X \times Y)$ is computable. □

Unfortunately, the partial inverse graph^{-1} is not continuous in general, as the following result shows. Here and in the following we write $w \sqsubseteq p$ if w is a finite *prefix* of p.

PROPOSITION 5.2. *The function $\text{graph}^{-1} :\subseteq \mathcal{A}(X \times \mathbb{R}) \to \mathcal{C}(X)$ is discontinuous in case of Cantor space $X = \{0, 1\}^{\mathbb{N}}$.*

Proof. Let $f_n : \{0, 1\}^{\mathbb{N}} \to \mathbb{R}$ be defined by

$$f_n(p) := \begin{cases} n + 1 & \text{if } 0^n \sqsubseteq p, \\ 0 & \text{otherwise} \end{cases}$$

for all $n \in \mathbb{N}$ and $p \in \{0, 1\}^{\mathbb{N}}$. Then $(f_n)_{n\in\mathbb{N}}$ is a sequence in $\mathcal{C}(\{0, 1\}^{\mathbb{N}})$ that does not converge, but $(\text{graph}(f_n))_{n\in\mathbb{N}}$ converges in $\mathcal{A}(\{0, 1\}^{\mathbb{N}} \times \mathbb{R})$ to $A := \{0, 1\}^{\mathbb{N}} \times \{0\}$ (recall that $\mathcal{A}(\{0, 1\}^{\mathbb{N}} \times \mathbb{R})$ is endowed with the final topology induced by the representation

$\delta^{<}_{\mathcal{A}(\{0,1\}^{\mathbb{N}}\times\mathbb{R})} \sqcap \delta^{>}_{\mathcal{A}(\{0,1\}^{\mathbb{N}}\times\mathbb{R})}$, which means that a sequence $(A_n)_{n\in\mathbb{N}}$ converges in this space to A if and only if for any rational ball $B(x,r)$ with $A \cap B(x,r) \neq \emptyset$ it follows that $A_n \cap B(x,r) \neq \emptyset$ for almost all n and for any closed rational ball $\overline{B}(x,r)$ with $A \cap \overline{B}(x,r) = \emptyset$ it follows that $A_n \cap \overline{B}(x,r) = \emptyset$ for almost all n; in other words, $(A_n)_{n\in\mathbb{N}}$ converges to A if any positive or negative information on A is shared by almost all sets in the sequence). □

Thus, the uniform version of the Graph Theorem does not hold true in general, not even for compact computable metric spaces X. However, as Corollary 7.3 will show, this fact cannot be used to construct a computable counterexample for the nonuniform case. Such a nonuniform counterexample can be obtained if we omit compactness, as the following example shows.

Example 5.3. Let $X = \mathbb{N}^{\mathbb{N}}$ be the Baire space. There exists a continuous but noncomputable function $f : X \to \mathbb{R}$ such that $\mathrm{graph}(f) \subseteq X \times \mathbb{R}$ is strongly recursive closed. The same holds in case of $X = \mathbb{N}\{0,1\}^{\mathbb{N}}$.

Proof. Let $a : \mathbb{N} \to \mathbb{N}$ be some computable and injective enumeration of a nonrecursive set $K = \mathrm{range}(a)$. Define $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{R}$ by

$$f(np) := \begin{cases} k+1 & \text{if there exists some } k \text{ with } a(k) = n \text{ and } 0^k \sqsubseteq p, \\ 0 & \text{otherwise} \end{cases}$$

for all $p \in \mathbb{N}^{\mathbb{N}}$ and $n \in \mathbb{N}$. Then f is continuous but noncomputable since $K = \mathrm{range}(a)$ is not co-r.e. and

$$n \notin K \iff f(n0^\omega) < 1.$$

Moreover,

$$\begin{aligned} &\Big(w\mathbb{N}^{\mathbb{N}} \times (r,s)\Big) \cap \mathrm{graph}(f) \neq \emptyset \\ \iff &(\exists n,k \geq 0, j \geq 1)\Big(\Big((w \sqsubseteq n0^k \text{ or } n0^k \sqsubseteq w) \text{ and } a(k) = n \text{ and } k+1 \in (r,s)\Big) \text{ or} \\ &\qquad \Big((w = n0^k \text{ or } n0^k j \sqsubseteq w) \text{ and } n \notin \{a(0),\ldots,a(k)\} \text{ and } 0 \in (r,s)\Big)\Big) \end{aligned}$$

for all $w \in \mathbb{N}^*$ and $r,s \in \mathbb{Q}$ with $r \leq s$. Since a is computable and only finitely many values n,k,j have to be checked, it follows that the stated property is recursive in r,s, and w. Particularly, $\mathrm{graph}(f)$ is r.e. closed. Analogously, the equivalence also holds with $[r,s]$ substituted for (r,s) in all of its three incidences. As the property is recursive, its negation is in particular r.e. Thus, $\mathrm{graph}(f)$ is also strongly co-r.e. closed. Altogether, $\mathrm{graph}(f)$ is strongly recursive closed. The proof for the case $X = \mathbb{N}\{0,1\}^{\mathbb{N}}$ is the same. □

Altogether, this shows that the problematic direction of the Graph Theorem does not hold in general—neither in the uniform case nor in the nonuniform case. The previous example additionally shows that in general it does not help to use the stronger notion of "strongly recursive closed subsets" for the problematic direction (not even in the nonuniform case).

Finally, we ask whether the positive result of this section could be improved by obtaining a stronger computability property of the graph. On the one hand, results of Cenzer and Remmel show that there is a computable function $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ such that $\mathrm{graph}(f)$ is not strongly recursive (see Corollary 4.6 in [10]). On the other hand, a reinterpretation of Examples 1.3 and 1.4 shows that the graph of a computable

function is not located in general. For the reformulation we consider $X := \mathbb{R} \setminus \mathbb{N}$ or $X = (0,1)$ as a computable metric subspace of $\mathbb{R}$, endowed with the restriction of the Euclidean metric to $X \times X$.

Example 5.4. Consider $X = \mathbb{R} \setminus \mathbb{N}$ or $X = (0,1)$ as a computable metric subspace of $\mathbb{R}$ endowed with the restriction of the Euclidean metric. There exists a computable function $f : X \to \mathbb{R}$ such that $\mathrm{graph}(f) \subseteq X \times \mathbb{R}$ is not located.

Since $\mathrm{graph}(f)$ need not be closed as a subset of $\mathbb{R} \times \mathbb{R}$, we could even obtain the same result with $[0,1]$ as the target space instead of $\mathbb{R}$ (as Example 1.4 shows). The reader should notice the fine distinctions: $\mathrm{graph}(f)$ is recursive as a closed subset of $X \times \mathbb{R}$ by Theorem 5.1, but it is not recursive as a closed subset of $\mathbb{R} \times \mathbb{R}$ in Example 1.3. Moreover, it is not located as a subset of $X \times \mathbb{R}$ and thus, in particular, not strongly recursive in the same sense (however, there is a different but equivalent metric on X that yields the aforementioned properties). Therefore, Theorem 5.1 formulates the best possible general result (with respect to the computability properties of subsets given in Figure 4.1).

6. Compact target spaces. In this section we want to prove that the inverse graph^{-1} of the graph mapping becomes computable if the target space is effectively compact in a certain sense. We start with the definition of some effectiveness notions for compact subsets (see [7]).

Definition 6.1 (recursive compact sets). *Let (X, d, α) be a computable metric space. A subset $K \subseteq X$ is called* co-r.e. compact *if it is compact and the set*

$$\left\{ \langle \langle n_0, k_0 \rangle, \langle n_1, k_1 \rangle, \ldots, \langle n_j, k_j \rangle, j \rangle \in \mathbb{N} : K \subseteq \bigcup_{i=0}^{j} B(\alpha(n_i), \overline{k_i}) \right\}$$

is r.e. Moreover, a subset $K \subseteq X$ is called recursive compact *if it is both r.e. closed and co-r.e. compact.*

One can prove that, for computable metric spaces that fulfill certain additional properties, a compact subset $K \subseteq X$ is co-r.e. compact if and only if it is co-r.e. closed [7]. We will call a computable metric space Y *recursive compact* if it is recursive compact as a subset of itself. Any computable metric space Y that is co-r.e. compact as a subset of itself is automatically recursive compact, since any computable metric space is r.e. closed as a subset of itself. To prepare for the main result of this section, we prove two lemmas.

Lemma 6.2. *Let X, Y be computable metric spaces. Then the section map*

$$\mathrm{sec} : \mathcal{A}_>(X \times Y) \times X \to \mathcal{A}_>(Y), (A, x) \mapsto A_x := \{y \in Y : (x, y) \in A\}$$

is computable.

Proof. Let $A \subseteq X \times Y$ be a closed subset, and let $(x_i, y_i) \in X \times Y$ and $r_i \in \mathbb{Q}$ for all $i \in \mathbb{N}$ be points such that $\bigcup_{i=0}^{\infty} B((x_i, y_i), r_i) = (X \times Y) \setminus A$. Then one directly obtains $\bigcup \{B(y_i, r_i) : i \in \mathbb{N} \text{ and } x \in B(x_i, r_i)\} = Y \setminus A_x$. Since the property "$x \in B(x_i, r_i)$" is r.e., the claim follows. □

The following lemma will be formulated slightly more generally than necessary for the proof of Theorem 6.4 (the case $Z \neq Y$ will be used in order to prove Proposition 7.2 in the next section).

Lemma 6.3. *Let Y be a computable metric space, and let $Z \subseteq Y$ be recursive compact. The injection map $\mathrm{in}_Z : Z \to \mathcal{A}_>(Y), y \mapsto \{y\}$ and its partial inverse $\mathrm{in}_Z^{-1} :\subseteq \mathcal{A}_>(Y) \to Z$ are computable.*

Proof. Consider the computable metric space (Y, d, α). Since Z is r.e. closed we can consider it as a computable metric subspace $(Z, d|_{Z\times Z}, \beta)$ of Y. Without loss of generality we can assume $\text{range}(\beta) \subseteq \text{range}(\alpha)$. Computability of the injection map in_Z follows directly since

$$Y \setminus \{y\} = \bigcup \{B(\alpha(n), \overline{k}) : n, k \in \mathbb{N}, \text{ and } d(y, \alpha(n)) > \overline{k}\}$$

for all $y \in Y$. Now let $y \in Z$ and assume that $x_i \in \text{range}(\alpha)$ and $r_i \in \mathbb{Q}$ for all $i \in \mathbb{N}$ are points such that $Y \setminus \{y\} = \bigcup_{i=0}^{\infty} B(x_i, r_i)$. Then

$$y \in B(x, r) \iff Z \subseteq B(x, r) \cup \bigcup_{i=0}^{\infty} B(x_i, r_i)$$

$$\iff (\exists j) Z \subseteq B(x, r) \cup \bigcup_{i=0}^{j} B(x_i, r_i)$$

for all $x \in \text{range}(\beta)$ and $r \in \mathbb{Q}$ with $r > 0$, where the last equivalence holds since Z is compact. Since Z is even co-r.e. compact, it follows that in_Z^{-1} is computable. □

Using the previous two lemmas we can now prove the main result of this section.

THEOREM 6.4 (compact Graph Theorem). *Let X and Y be computable metric spaces and let Y be co-r.e. compact. The graph map* $\text{graph} : \mathcal{C}(X, Y) \to \mathcal{A}_>(X \times Y)$, $f \mapsto \text{graph}(f)$ *as well as its partial inverse*

$$\text{graph}^{-1} :\subseteq \mathcal{A}_>(X \times Y) \to \mathcal{C}(X, Y)$$

are computable.

Proof. The first part of the claim follows from the weak Graph Theorem, Theorem 5.1. Now any continuous function $f : X \to Y$ can be represented as

$$f(x) = \text{in}_Y^{-1} \circ \text{sec}(\text{graph}(f), x)$$

for any $x \in X$, where $\text{sec} : \mathcal{A}_>(X \times Y) \times X \to \mathcal{A}_>(Y)$ and $\text{in}_Y : Y \to \mathcal{A}_>(Y)$ are the mappings defined in Lemmas 6.2 and 6.3. Since sec and in_Y^{-1} are computable, it follows by type conversion that graph^{-1} is computable. □

As a corollary of Theorems 5.1 and 6.4 we immediately obtain the following nonuniform version of the Graph Theorem.

COROLLARY 6.5. *Let X, Y be computable metric spaces, let Y be co-r.e. compact, and let $f : X \to Y$ be a (continuous) function. Then f is computable if and only if* $\text{graph}(f)$ *is co-r.e. closed and if and only if* $\text{graph}(f)$ *is recursive closed.*

The formula in the proof of Theorem 6.4 additionally shows that f is automatically continuous whenever $\text{graph}(f)$ is closed (under the assumptions of the previous corollary). Hence, the corollary remains true without the word "continuous." The previous corollary has been proved directly in case of $X = Y = [0, 1]$ and $X = Y = \{0, 1\}^{\mathbb{N}}$ by Cenzer and Remmel (see Theorem 4.7 in [10]). In these cases one can even replace co-r.e. by strongly co-r.e. Proposition 5.2 and Example 5.3 show that the compactness conditions are not superfluous in the previous results. The following example shows that co-r.e. closed graphs cannot be replaced by r.e. closed graphs in the previous corollary.

Example 6.6. Let $X = Y = [0, 1]$. There exists a continuous but noncomputable function $f : X \to Y$ such that $\text{graph}(f) \subseteq X \times Y$ is r.e. closed. The same holds in case of $X = Y = \mathbb{R}$.

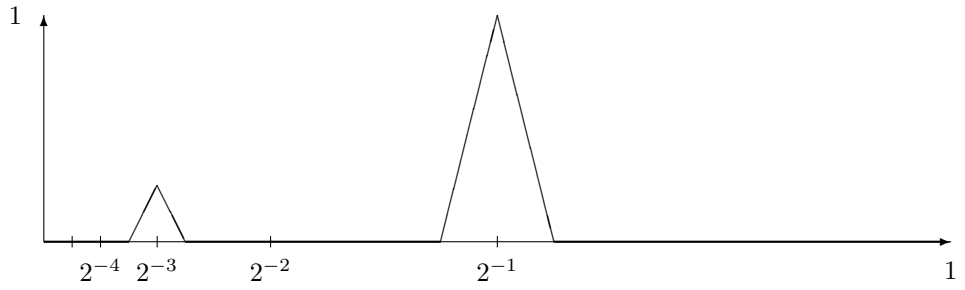


FIG. 6.1. *The function f in case of $a(0) = 2$, $a(1) = 0$ and $1, 3, 4, 5, 6, \ldots \notin K$.*

Proof. Let $a : \mathbb{N} \to \mathbb{N}$ be some computable and injective enumeration of a non-recursive set $K = \text{range}(a)$. We define a function $f : [0,1] \to [0,1]$ as follows. For any $n \in K = \text{range}(a)$ let $p_n : [2^{-n-1} - 2^{-n-3}, 2^{-n-1} + 2^{-n-3}] \to \mathbb{R}$ be the rational polygon with vertices

$$\left(2^{-n-1} - 2^{-n-3}, 0\right), \left(2^{-n-1} - 2^{-n-3-a^{-1}(n)}, 0\right), \left(2^{-n-1}, 2^{-n}\right),$$
$$\left(2^{-n-1} + 2^{-n-3-a^{-1}(n)}, 0\right), \left(2^{-n-1} + 2^{-n-3}, 0\right),$$

and let

$$f(x) := \begin{cases} p_n(x) & \text{if there exists some } n \in \text{range}(a) = K \text{ with} \\ & x \in [2^{-n-1} - 2^{-n-3}, 2^{-n-1} + 2^{-n-3}], \\ 0 & \text{otherwise} \end{cases}$$

for all $x \in [0,1]$; see Figure 6.1.

It is easy to see that f is continuous and $\text{graph}(f) \subseteq [0,1] \times [0,1]$ is r.e. closed since a is computable. On the other hand, f is not computable, since $K = \text{range}(a)$ is not co-r.e. and

$$n \notin K \iff f\left(2^{-n-1}\right) < 2^{-n}. \qquad \square$$

Similar counterexamples have been presented by Cenzer and Remmel (also for the case $X = Y = \mathbb{N}^{\mathbb{N}}$; see Theorem 4.8 in [10]). As a further corollary of Theorems 5.1 and 6.4, we can formulate a uniform version of the statement that negative information on the graph of a continuous function with compact target can be translated into positive information.

COROLLARY 6.7. *Let X, Y be computable metric spaces and let Y be co-r.e. compact. Then the identity* $\text{id} : \mathcal{A}_>(X \times Y) \to \mathcal{A}(X \times Y), A \mapsto A$, *restricted to such subsets $A \subseteq X \times Y$ that are graphs of (continuous) functions $f : X \to Y$, is computable.*

The following proposition shows that this does not hold in the general case of arbitrary computable metric spaces Y.

PROPOSITION 6.8. *The function* $\text{graph}^{-1} :\subseteq \mathcal{A}_>(X \times Y) \to \mathcal{C}(X, Y)$ *is discontinuous in case of the unit interval $X = [0,1]$ and $Y = \mathbb{R}$. The same holds true with $\mathcal{A}_<$ instead of $\mathcal{A}_>$ (and in this case one can even choose $Y = [0,1]$).*

Proof. Now let $f_n : [0,1] \to \mathbb{R}$ be defined by $f_n(x) := n$ for all $x \in [0,1]$ and $n \in \mathbb{N}$. Then $(f_n)_{n \in \mathbb{N}}$ is a sequence in $\mathcal{C}[0,1]$ that does not converge, but $(\text{graph}(f_n))_{n \in \mathbb{N}}$

converges in $\mathcal{A}_>([0,1]\times\mathbb{R})$ to $A := [0,1]\times\{0\}$ (since any negative information on A is shared by almost all sets in the sequence; we mention that $\mathcal{A}_>([0,1]\times\mathbb{R})$ is not a Hausdorff space, and hence limits of converging sequences are not uniquely determined, and the given limit A is just one of many possible limits). This proves the claim for $\mathcal{A}_>$.

Let $f_n : [0,1] \to \mathbb{R}$ be defined by

$$\operatorname{graph}(f_n) := \Big\{(x, 2^{n+1}x) : x \in [0, 2^{-n-1}]\Big\}$$
$$\cup\ \Big\{(x, 2 - 2^{n+1}x) : x \in [2^{-n-1}, 2^{-n}]\Big\} \cup \Big([2^{-n}, 1]\times\{0\}\Big).$$

Then $(f_n)_{n\in\mathbb{N}}$ is a sequence of continuous functions in $\mathcal{C}[0,1]$ that does not converge, but the sequence $(\operatorname{graph}(f_n))_{n\in\mathbb{N}}$ converges in $\mathcal{A}_<([0,1]\times\mathbb{R})$ to $A := [0,1]\times\{0\}$ (since any positive information on A is shared by almost all sets in the sequence). This proves the claim for $\mathcal{A}_<$. Since $\operatorname{range}(f_n) \subseteq [0,1]$, the same holds true in case of $Y = [0,1]$. □

The second part of the previous proposition shows that the previous corollary does not hold true if we replace $\mathcal{A}_>$ by $\mathcal{A}_<$—not even if both X and Y are recursive compact.

7. Locally compact target spaces. In this section we will show that the nonuniform version of the compact Graph Theorem, i.e., Corollary 6.5, can be transferred to the case that the source space X is compact and the target space Y is effectively locally compact. Up to now different notions of "effective local compactness" have been used (see, for instance, [22, 7]), and the terminology is not stable yet. In the next definition we do not want to propose a new version, but we just formulate a very weak property that should be met by most of the reasonable definitions of effective local compactness and which is sufficient for our proofs.

DEFINITION 7.1 (locally computable). *A computable metric space X is called* locally computable *if for any compact subset $A \subseteq X$ there exists a recursive compact subset $K \subseteq X$ such that $A \subseteq K$.*

It is obvious that, for instance, the Euclidean space $\mathbb{R}^n$ is locally computable in this sense (since any closed ball is recursive compact). Next we will show that if the target space is locally computable, then we can deduce that any continuous function with co-r.e. graph-maps computable points to computable points. Furthermore, it is computable restricted to any r.e. compact subset. If $f : X \to Y$ is a total function and $A \subseteq X$, then we denote by $f|_A :\subseteq X \to Y$ the *restriction* of f to A, defined by $f|_A(x) := f(x)$ and $\operatorname{dom}(f) := A$.

PROPOSITION 7.2. *Let X, Y be computable metric spaces, let Y be locally computable, and let $f : X \to Y$ be a continuous function. If* $\operatorname{graph}(f)$ *is co-r.e. closed, then $f|_K$ is computable for any compact subset $K \subseteq X$. Particularly, f maps computable points to computable points.*

Proof. Since f is continuous, it follows that $f(K)$ is compact for any compact set $K \subseteq X$. Since Y is locally computable, there is a recursive compact set $L \subseteq Y$ such that $f(K) \subseteq L$. Particularly, L can be considered as a computable metric subspace of Y, and thus the injection $L \hookrightarrow Y$ is computable. Now f can be represented as

$$f(x) = \mathrm{in}_L^{-1} \circ \operatorname{sec}(\operatorname{graph}(f), x)$$

for any $x \in K$, where $\operatorname{sec} : \mathcal{A}_>(X\times Y)\times X \to \mathcal{A}_>(Y)$ and $\mathrm{in}_L : L \to \mathcal{A}_>(Y)$ are the functions defined in Lemmas 6.2 and 6.3. Since sec, in_L^{-1} and $L \hookrightarrow Y$ are computable,

the partial function $f|_K :\subseteq X \to Y$ is computable. The last part of the claim holds since $K := \{x\}$ is a compact subset for any $x \in X$. □

We obtain the following corollary of the previous result and the weak Graph Theorem (Theorem 5.1) for compact source spaces and locally computable target spaces.

COROLLARY 7.3. *Let X, Y be computable metric spaces, let X be compact, let Y be locally computable, and let $f : X \to Y$ be a continuous function. Then f is computable if and only if* graph(f) *is co-r.e. closed and if and only if* graph(f) *is recursive closed.*

This corollary in particular applies to continuous functions $f : [0,1] \to \mathbb{R}$. Proposition 6.8 shows that the uniform version of the first equivalence of the previous corollary does not hold true in general. Here, continuity of f does not automatically follow if graph(f) is closed (see Example 4.3 in [10]). Finally, Example 5.3 shows that the previous corollary does not hold true for locally compact source spaces X. The next example proves that local computability of the target space is not a superfluous property in the previous corollary. The example is a modified version of Example 6.6, and as target space we will use the computable metric space (see, e.g., [6] for details)

$$\ell_2 := \{x \in \mathbb{R}^{\mathbb{N}} : ||x||_2 < \infty\} \text{ with } ||(x_k)_{k\in\mathbb{N}}||_2 := \sqrt{\sum_{k=0}^{\infty} |x_k|^2}.$$

Example 7.4. Let $X = [0,1]$ and $Y = \ell_2$. There exists a continuous but noncomputable function $f : X \to Y$ such that graph$(f) \subseteq X \times Y$ is recursive closed.

Proof. Let $a : \mathbb{N} \to \mathbb{N}$ be some computable and injective enumeration of a nonrecursive set $K = \text{range}(a)$. We define a function $f : [0,1] \to \ell_2$ using the polygons p_n as defined in Example 6.6 as follows:

$$f(x)(k) := \begin{cases} p_n(x) & \text{if } a(k) = n \text{ and } x \in [2^{-n-1} - 2^{-n-3}, 2^{-n-1} + 2^{-n-3}] =: I_n, \\ 0 & \text{otherwise} \end{cases}$$

for all $x \in [0,1]$ and $k \in \mathbb{N}$.

It is easy to see that f is continuous and graph$(f) \subseteq [0,1] \times \ell_2$ is r.e. closed since a is computable. One can also show that graph(f) is co-r.e. In case that $x \in I_n$ and we already know that $n \notin a(\{0,\ldots,k\})$, we obtain $B((x,y),r) \subseteq \text{graph}(f)^c$ for any $y = (y_0, y_1, \ldots, y_k, 0, 0, \ldots) \in \ell_2$ with $||y|| > 0$ and $0 < r < \min\{||y||, 2^{-n-4}\}$. Exploiting this idea, one can exhaust graph$(f)^c$ effectively by open balls, and thus graph(f) is co-r.e.

On the other hand, f is not computable, since $K = \text{range}(a)$ is not co-r.e. and

$$n \notin K \iff ||f\left(2^{-n-1}\right)||_2 < 2^{-n}. \qquad \square$$

In the previous example $f : [0,1] \to \ell_2$ is continuous and thus $K := f[0,1]$ is compact. However, K cannot be co-r.e. compact by Corollary 6.5, and by Corollary 7.3 ℓ_2 is not locally computable. In particular, there is no recursive compact set $K' \subseteq \ell_2$ such that $K \subseteq K'$. This observation in particular proves that pure compactness of Y is not sufficient in Theorem 6.4 (and correspondingly for Corollary 6.5). Moreover, local computability of Y cannot be replaced by compactness in Corollary 7.3.

8. Locally connected source spaces. In this section we will discuss another property of the source space that guarantees that the uniform inverse direction of

the Graph Theorem becomes computable. We will call a computable metric space X effectively locally connected (cf. [16]) if for any point $x \in X$ and any open ball $B(x,r)$ we can effectively find some connected neighborhood U of x that is contained in $B(x,r)$. For the precise formulation we employ the hyperspace $\mathcal{O}(X)$ of open subsets of X, represented by $\delta_{\mathcal{O}(X)}(p) = X \setminus \delta_{\mathcal{A}_>(X)}(p)$.

DEFINITION 8.1 (effective local connectedness). *A computable metric space X is called* effectively locally connected *if there exists a computable multivalued function $C :\subseteq X \times \mathbb{R} \rightrightarrows \mathcal{O}(X)$ such that for any $x \in X$ and $r > 0$ there exists some $U \in C(x,r)$ and any such U is connected and fulfills $x \in U \subseteq B(x,r)$.*

One should mention that it would not suffice to consider balls $U = B(x,s)$ with some $s \leq r$ because it might happen that any such ball is not connected although there exists some connected (more complicated) U (consider an appropriate spiral line in $\mathbb{R}^2$ such that any box with the same center as the spiral cuts the spiral into several distinct connectedness components).

The main result of this section will be based on the following lemma, which employs the hyperspace $\mathcal{K}(X) := \{K \subseteq X : K \text{ nonempty and compact}\}$ of nonempty compact subsets of a metric space X, endowed with the *Hausdorff metric* $d_{\mathcal{K}} : \mathcal{K}(X) \times \mathcal{K}(X) \to \mathbb{R}$, defined by

$$d_{\mathcal{K}}(A,B) := \max\left\{\sup_{a\in A} d_B(a), \sup_{b\in B} d_A(b)\right\}.$$

If (X,d,α) is a computable metric space, then a standard numbering $\alpha_{\mathcal{K}}$ of the set $\mathcal{Q}$ of finite subsets of range(α) can be defined by $\alpha_{\mathcal{K}}\langle k, \langle n_0, \ldots, n_k\rangle\rangle := \{\alpha(n_0), \ldots, \alpha(n_k)\}$. It is easy to see that under these assumptions $(\mathcal{K}(X), d_{\mathcal{K}}, \alpha_{\mathcal{K}})$ is a computable metric space, too. In the following we assume that $\mathcal{K}(X)$ is endowed with the corresponding Cauchy representation $\delta_{\mathcal{K}}$. It is easy to prove that the canonical injection $\mathcal{K}(X) \hookrightarrow \mathcal{A}(X)$ is computable [7]. To prepare for our main result, we formulate a lemma. Here, ∂A denotes the *boundary* of a subset $A \subseteq X$ of a metric space X.

LEMMA 8.2. *Let $n \geq 1$ a natural number. The function*

$$I :\subseteq \mathbb{R}^n \times \mathbb{R} \to \mathcal{K}(\mathbb{R}^n), (y,\varepsilon) \mapsto \partial B(y,\varepsilon),$$

defined for all $(y,\varepsilon) \in \mathbb{R}^n \times \mathbb{R}$ with $\varepsilon > 0$, is computable.

The proof is straightforward. The fact that the target space $\mathbb{R}^n$ is locally compact and has balls whose boundaries are compact is essential for the following result. Proposition 8.6 will show that the same result does not hold true in case of the Hilbert space ℓ_2 as target space. Now we are prepared to prove the following theorem.

THEOREM 8.3 (locally connected Graph Theorem). *Let X be an effectively locally connected computable metric space and let $n \geq 1$ be a natural number. The graph mapping* $\mathrm{graph} : \mathcal{C}(X, \mathbb{R}^n) \to \mathcal{A}(X \times \mathbb{R}^n), f \mapsto \mathrm{graph}(f)$ *as well as its partial inverse*

$$\mathrm{graph}^{-1} :\subseteq \mathcal{A}(X \times \mathbb{R}^n) \to \mathcal{C}(X, \mathbb{R}^n)$$

are computable.

Proof. We consider the computable metric space (X,d,α). Computability of the map graph follows from Theorem 5.1. We investigate the mapping graph^{-1}. Thus, let $f : X \to \mathbb{R}^n$ be a continuous function. We can assume that $\mathrm{graph}(f) \in \mathcal{A}(X \times \mathbb{R}^n)$ is given. Furthermore, let $x \in X$ and a precision $k \in \mathbb{N}$ be given. We prove that we can effectively evaluate $f(x)$ up to precision 2^{-k}, which implies by type conversion the desired result. Since $\mathrm{graph}(f)$ is given as a point of $\mathcal{A}(X \times \mathbb{R}^n)$, we can, on the

one hand, find points $(x_i, y_i) \in \text{range}(\alpha) \times \mathbb{Q}^n$ and rational numbers $r_i > 0$ such that

$$\text{graph}(f)^c = (X \times \mathbb{R}^n) \setminus \text{graph}(f) = \bigcup_{i=0}^{\infty} B((x_i, y_i), r_i) = \bigcup_{i=0}^{\infty} (B(x_i, r_i) \times B(y_i, r_i)),$$

and, on the other hand, given some open set $U \in \mathcal{O}(X \times \mathbb{R}^n)$, we can effectively recognize that $U \cap \text{graph}(f) \neq \emptyset$ if this is the case. Given a compact subset $K \in \mathcal{K}(\mathbb{R}^n)$ such that $K \subseteq \bigcup_{i=0}^{\infty} B(y_i, r_i)$, the computable Heine–Borel theorem (cf. Theorems 4.6 and 4.10 in [8]) ensures that we can effectively find a natural number $m \in \mathbb{N}$ such that $K \subseteq \bigcup_{i=0}^{m} B(y_i, r_i)$. Now the evaluation of $f(x)$ up to precision 2^{-k} works as follows: we systematically search for some $y \in \mathbb{Q}^n$, some rational numbers $r > 0$, $t \geq 1$, and natural numbers $n_0, \ldots, n_m \in \mathbb{N}$ such that for some arbitrary determined $W \in C(x, r)$ the following conditions are satisfied (where (1) holds automatically):

(1) $x \in W \subseteq B(x, r)$ and W is connected and open,
(2) $U \cap \text{graph}(f) \neq \emptyset$ with $U := W \times B(y, r)$,
(3) $d(x_{n_i}, x) + r < r_{n_i}$ for all $i = 0, \ldots, m$,
(4) $\partial B(y, tr) \subseteq \bigcup_{i=0}^{m} B(y_{n_i}, r_{n_i})$, and
(5) $tr < 2^{-k}$.

Figure 8.1 illustrates the situation. If for certain values $y, r, t, n_0, \ldots, n_m$ and $W \in C(x, r)$ the conditions (2)–(5) are fulfilled, then we can eventually recognize this by the previous considerations and the previous lemma and since X is effectively locally connected. Now we claim that such values always exist and $||f(x) - y|| < 2^{-k}$ if (1)–(5) are fulfilled.

Thus, let us first assume that (1)–(5) hold. It is obvious that (3) implies $B(x, r) \subseteq \bigcap_{i=0}^{m} B(x_{n_i}, r_{n_i})$ and thus by (4) it follows that

$$B(x, r) \times \partial B(y, tr) \subseteq \bigcup_{i=0}^{m} (B(x_{n_i}, r_{n_i}) \times B(y_{n_i}, r_{n_i})) \subseteq \text{graph}(f)^c.$$

Thus, by (1) we obtain $f(W) \subseteq B(y, tr) \cup (\mathbb{R}^n \setminus \overline{B}(y, tr))$, and hence $f(W)$ is covered by a disjoint union of open subsets. Since W is connected and f is continuous, it follows that $f(W)$ is connected, too, and thus $f(W)$ has empty intersection with either $B(y, tr)$ or $\mathbb{R}^n \setminus \overline{B}(y, tr)$. By (2) there is some $z \in W$ such that $f(z) \in B(y, r) \subseteq B(y, tr)$ and hence $f(W) \cap \overline{B}(y, tr)^c = \emptyset$. This implies $f(x) \in B(y, tr)$ by (1) and hence $||f(x) - y|| < tr < 2^{-k}$ by (5).

Now we still have to prove that there are always values $y, r, t, n_0, \ldots, n_m$ such that (1)–(5) are fulfilled with arbitrary $W \in C(x, r)$. Therefore, let $0 < s < 2^{-k}$ be a

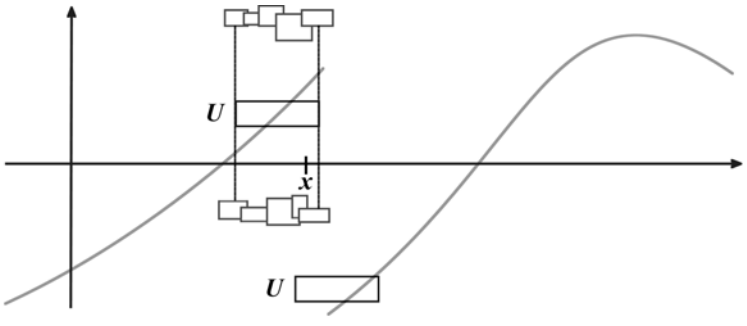


FIG. 8.1. *A graph of some function f with two potential sets U.*

rational number. Then $\partial B(f(x),s) \subseteq \bigcup\{B(y_i,r_i) : i \in \mathbb{N} \text{ and } x \in B(x_i,r_i)\}$ and by compactness there exist $n_0,\dots,n_m \in \mathbb{N}$ such that $\partial B(f(x),s) \subseteq \bigcup_{i=0}^m B(y_{n_i},r_{n_i})$ and $x \in \bigcap_{i=0}^m B(x_{n_i},r_{n_i}) =: V$. Thus, V is an open neighborhood of x and we can choose some rational r with $0 < r < s$ such that $d(x_{n_i},x) + r < r_{n_i}$ for all $i = 0,\dots,m$, in particular, $B(x,r) \subseteq V$. Now there exists some $y \in \mathbb{Q}^n$ such that $||f(x) - y|| < r$ and still $\partial B(y,s) \subseteq \bigcup_{i=0}^m B(y_{n_i},r_{n_i})$. For this choice of $y, r, n_0,\dots,n_m$, $t := \frac{s}{r}$, and $W \in C(x,r)$, conditions (1)–(5) are fulfilled: (1) holds since $W \in C(x,r)$. (2) holds since by definition $x \in W$ and $f(x) \in B(y,r)$, and thus $(x,f(x)) \in \text{graph}(f) \cap U$. (4) holds by choice of y and t and (3) by choice of r. (5) holds since $tr = s < 2^{-k}$ by choice of s and definition of t. □

This result in particular applies to computable normed spaces X and other spaces with connected balls (such as locally convex spaces). We directly obtain the following nonuniform corollary.

COROLLARY 8.4. *Let X be an effectively locally connected computable metric space and let $f : X \to \mathbb{R}^n$ be a continuous function. Then f is computable if and only if* graph(f) *is recursive closed.*

Example 5.3 already showed that "effective local connectedness" is not a superfluous condition—neither in the previous result nor in its uniform version. The following proposition shows that in the uniform case the target space $Y = \mathbb{R}^n$ cannot be replaced by an arbitrary computable metric space (not even if we restrict ourselves to linear operators). The result is based on the following easy lemma and results from [6].

LEMMA 8.5. *Let X, Y be computable metric spaces. The "swap map"*

$$S : \mathcal{A}(X \times Y) \to \mathcal{A}(Y \times X), A \mapsto \{(y,x) \in Y \times X : (x,y) \in A\}$$

is computable.

The proof follows from the fact that the pointwise "swap function" $s : X \times Y \to Y \times X, (x,y) \mapsto (y,x)$ operates on centers of balls, i.e.,

$$sB((x,y),r) = B((y,x),r) = B(s(x,y),r)$$

for all $(x,y) \in X \times Y$ and $r > 0$ provided that $X \times Y$ and $Y \times X$ are endowed with the product space structure according to Proposition 3.2.

PROPOSITION 8.6. *The mapping* $\text{graph}^{-1} :\subseteq \mathcal{A}(\ell_2 \times \ell_2) \to \mathcal{C}(\ell_2,\ell_2)$*, defined for all closed subsets $A \subseteq \ell_2 \times \ell_2$ such that $A =$* graph(T) *for some linear bounded and bijective operator $T : \ell_2 \to \ell_2$ with $||T|| = 1$, is discontinuous.*

Proof. Let us assume that $\text{graph}^{-1} :\subseteq \mathcal{A}(\ell_2 \times \ell_2) \to \mathcal{C}(\ell_2,\ell_2)$ would be continuous in the stated sense. By the previous lemma and the weak Graph Theorem (Theorem 5.1) it follows that the inversion $I :\subseteq \mathcal{C}(\ell_2,\ell_2) \to \mathcal{C}(\ell_2,\ell_2), T \mapsto T^{-1}$, restricted to linear bounded and bijective operators $T : \ell_2 \to \ell_2$ with $||T|| = 1$, would be continuous, too, since $I = \text{graph}^{-1} \circ S \circ \text{graph}$. This contradicts Example 17 in [6]. □

Finally, we want to prove that with the same assumptions on the spaces Theorem 8.3 cannot be strengthened in the way that only negative information on the graph is required in order to determine the function. In the following example we consider $\mathbb{R} \setminus \mathbb{N}$ as a computable metric subspace of $\mathbb{R}$, endowed with the restriction of the Euclidean metric.

Example 8.7. Consider $X = \mathbb{R} \setminus \mathbb{N}$ as a computable metric subspace of $\mathbb{R}$ endowed with the restriction of the Euclidean metric. There exists a continuous but noncomputable function $f : X \to \mathbb{R}$ such that graph$(f) \subseteq X \times \mathbb{R}$ is strongly co-r.e. closed.

Proof. Let $a : \mathbb{N} \to \mathbb{N}$ be some computable and injective enumeration of a nonrecursive set $K = \text{range}(a)$. Define $f : \mathbb{R} \setminus \mathbb{N} \to \mathbb{R}$ by

$$f(x) := \begin{cases} n+1 & \text{if } (\exists n, k)\ x \in (k, k+1) \text{ and } a(n) = k, \\ 0 & \text{otherwise} \end{cases}$$

for all $x \in \mathbb{R} \setminus \mathbb{N}$. Then f is continuous but noncomputable since $K = \text{range}(a)$ is not co-r.e. and

$$k \notin K \iff f\left(k + \frac{1}{2}\right) < 1.$$

Moreover, $\text{graph}(f)$ is strongly co-r.e. closed since a is computable and

$$\begin{aligned} & ([c,d] \times [r,s]) \cap \text{graph}(f) = \emptyset \\ \iff & (\exists k)\Big(\Big(k < c \le d < k+1 \text{ and } (\forall n+1 \in [r,s])(a(n) \ne k)\Big) \\ & \text{and } \Big(0 \in [r,s] \implies (\exists n)(a(n) = k)\Big)\Big) \end{aligned}$$

for all $c, d, r, s \in \mathbb{Q}$ such that $r \le s$ and there exists some k with $k < c \le d < k+1$. The general case of those $c \le d$ that are not subject to the previous inequality can easily be reduced to the presented special case. □

9. Pathwise connected source spaces. In this section we will discuss another connectedness property of the source space which guarantees that the uniform inverse direction of the Graph Theorem becomes computable, even if essentially only negative information on the graph is available. The only positive information that we will require is a single point $(a, b) \in \text{graph}(f)$.

We will call a computable metric space X effectively pathwise connected if for all points $x, y \in X$ we can effectively find some continuous function $p : [0,1] \to X$ such that $p(0) = x$ and $p(1) = y$. More precisely, we have the following definition.

DEFINITION 9.1 (effective pathwise connectedness). *A computable metric space X is called* effectively pathwise connected *if there exists a computable multivalued function $P : X \times X \rightrightarrows \mathcal{C}([0,1], X)$ such that for all $x, y \in X$ and $p \in P(x, y)$ it holds that $p(0) = x$ and $p(1) = y$.*

It is easy to see that the Euclidean space $\mathbb{R}^n$ and computable normed spaces X are effectively pathwise connected (a path $p : [0,1] \to X$ can simply be obtained by $p(r) := x + r(y - x)$ in these cases). Now we are prepared to prove the following theorem. The proof is a modified version of the proof of Theorem 8.3.

THEOREM 9.2 (pathwise connected Graph Theorem). *Let X be an effectively pathwise connected computable metric space and let $n \ge 1$ be a natural number. The mapping*

$$F :\subseteq \mathcal{A}_>(X \times \mathbb{R}^n) \times X \times \mathbb{R}^n \to \mathcal{C}(X, \mathbb{R}^n), (A, a, b) \mapsto \text{graph}^{-1}(A),$$

which is defined for all (A, a, b) such that $A = \text{graph}(f)$ and $f(a) = b$ for some $f \in \mathcal{C}(X, \mathbb{R}^n)$, is computable.

Proof. We consider the computable metric space (X, d, α). Let $f : X \to \mathbb{R}^n$ be a continuous function and let $(a, b) \in X \times \mathbb{R}^n$ with $f(a) = b$. Furthermore, let $x \in X$ and a precision $k \in \mathbb{N}$ be given. We prove that we can effectively evaluate $f(x)$ up

to precision 2^{-k}, which implies by type conversion the desired result. We can assume that graph(f) is given as point graph$(f) \in \mathcal{A}_>(X \times \mathbb{R}^n)$, and thus we can find points $(x_i, y_i) \in \text{range}(\alpha) \times \mathbb{Q}^n$ and rational numbers $r_i > 0$ such that

$$\text{graph}(f)^c = (X \times \mathbb{R}^n) \setminus \text{graph}(f) = \bigcup_{i=0}^{\infty} (B(x_i, r_i) \times B(y_i, r_i)) .$$

Since X is effectively pathwise connected, we can effectively find a continuous function $p : [0,1] \to X$ such that $p(0) = a$ and $p(1) = x$. The strategy of the proof is to follow the graph along the path p from a to x in order to evaluate f at x. Therefore, we systematically search for values $m \in \mathbb{N}$ and a partition $0 = a_0 < a_1 < a_2 < \cdots < a_m = 1$ with $a_j \in \mathbb{Q}$, and we guess possible function values $b_0 := b$, $b_{j+1} \in B(b_j, 2^{-k}) \cap \mathbb{Q}^n$ for $j = 0, \ldots, m-1$ such that we obtain

$$p[a_j, a_{j+1}] \times \partial B(b_j, 2^{-k}) \subseteq \bigcup_{i=0}^{\infty} (B(x_i, r_i) \times B(y_i, r_i)) \tag{9.1}$$

and such that we obtain for all $j = 0, \ldots, m-2$

$$\{p(a_{j+1})\} \times \overline{(\overline{B}(b_j, 2^{-k}) \Delta \overline{B}(b_{j+1}, 2^{-k}))} \subseteq \bigcup_{i=0}^{\infty} (B(x_i, r_i) \times B(y_i, r_i)) . \tag{9.2}$$

Here $C \Delta D := C \setminus D \cup D \setminus C$ denotes the symmetric difference. Figure 9.1 illustrates the situation. We first mention that the inclusions in (9.1) and (9.2) are r.e. in all parameters by the computable Heine–Borel theorem (cf. Theorems 4.6 and 4.10 in [8]). This follows from Lemma 8.2 together with the fact that the mapping $C([0,1], X) \times \mathcal{K}[0,1] \to \mathcal{K}(X), (p, A) \mapsto p(A)$ is computable (see Theorem 3.3 in [21] and Theorem 4.12 in [7]) and since $x \mapsto \{x\}$ and $(x, y, k) \mapsto \overline{(\overline{B}(x, 2^{-k}) \Delta \overline{B}(y, 2^{-k}))}$ are both computable with respect to $\delta_{\mathcal{K}}$ on the output.

Now we claim that always appropriate values m, a_j, b_j exist such that (9.1) and (9.2) are fulfilled, and we claim that $||f(x) - y|| < 2^{-k}$ holds for $y := b_m$ in this case.

Thus, let us first assume that (9.1) and (9.2) hold. Since the composition $fp : [0,1] \to \mathbb{R}^n$ is continuous, it follows that graph(fp) is connected, and since $fp(a_0) = fp(0) = f(a) = b = b_0$, it follows that

$$\text{graph}(fp) \subseteq \bigcup_{j=0}^{m-1} \left([a_j, a_{j+1}] \times B(b_j, 2^{-k}) \right) .$$

In particular, $f(x) = fp(1) = fp(a_m) \in B(b_m, 2^{-k})$, and thus $||f(x) - y|| < 2^{-k}$, as we have claimed.

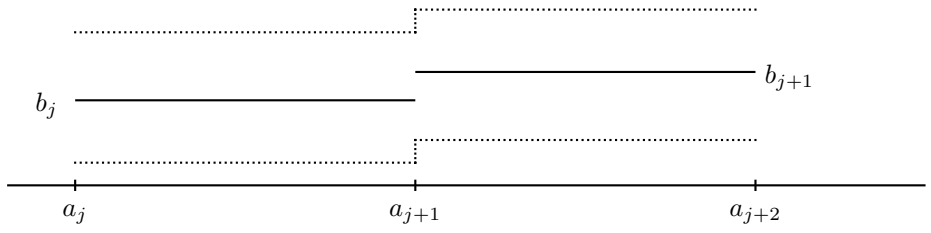


FIG. 9.1. *The tube constructed around the path p.*

Now we still have to prove that there always exist appropriate values m, a_j, b_j such that (9.1) and (9.2) are fulfilled. Since $fp : [0,1] \to \mathbb{R}^n$ is continuous and $[0,1]$ is compact, it follows that fp is even uniformly continuous and there exists some value $l \in \mathbb{N}$ such that $fp[a_j, a_{j+1}] \subseteq B(fp(a_j), 2^{-k-2})$ for all $j = 0, \ldots, m-1$ with $m := 2^l$ and $a_j := j \cdot 2^{-l}$. Thus, if we choose $b_j \in \mathbb{Q}^n$ such that $||fp(a_j) - b_j|| \leq 2^{-k-2}$, then (9.1) and (9.2) are fulfilled. □

Since in case of complete spaces X positive information on the graph allows us to find some x and y with $f(x) = y$ (which follows from Proposition 4.4), we obtain the following corollary.

COROLLARY 9.3. *Let X be an effectively pathwise connected and complete computable metric space. The mapping* $\text{graph} : \mathcal{C}(X, \mathbb{R}^n) \to \mathcal{A}(X \times \mathbb{R}^n), f \mapsto \text{graph}(f)$ *as well as its inverse* $\text{graph}^{-1} :\subseteq \mathcal{A}(X \times \mathbb{R}^n) \to \mathcal{C}(X, \mathbb{R}^n)$ *are computable.*

We do not know whether completeness is a superfluous property in this case. In the nonuniform case we have learned by Proposition 7.2 that a continuous function $f : X \to \mathbb{R}^n$ with a co-r.e. graph always maps computable inputs to computable outputs. Therefore, we obtain the following corollary of Theorem 9.2.

COROLLARY 9.4. *Let X be an effectively pathwise connected computable metric space and let $f : X \to \mathbb{R}^n$ be a continuous function. Then f is computable if and only if* $\text{graph}(f)$ *is recursive closed and if and only if* $\text{graph}(f)$ *is co-r.e. closed.*

Moreover, we can easily conclude a linear version of the previous theorem (where we consider only negative information on the graph) since in the linear case $f(0) = 0$ and thus the additional input information (a pair from the graph of f) is automatically available. Here, a *computable normed* space is a computable metric space with an additional vector space structure such that the algebraic operations are computable (see, e.g., [6] for details).

COROLLARY 9.5. *Let X be a computable normed space and let $n \geq 1$ be a natural number. The mapping* $\text{graph} :\subseteq \mathcal{C}(X, \mathbb{R}^n) \to \mathcal{A}_>(X \times \mathbb{R}^n), f \mapsto \text{graph}(f)$*, defined for linear mappings $f : X \to \mathbb{R}^n$, as well as its partial inverse* $\text{graph}^{-1} :\subseteq \mathcal{A}_>(X \times \mathbb{R}^n) \to \mathcal{C}(X, \mathbb{R}^n)$ *are computable.*

We close this section with an example that shows that in general a linear function $f : X \to Y$ with a co-r.e. closed graph need not be computable. For any $x = (x_i)_{i \in \mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ we denote by $x[j] := (x_0, x_1, \ldots, x_j, 0, 0, \ldots) \in \mathbb{R}^{\mathbb{N}}$ the sequence up to element j and filled up with zeros.

Example 9.6. There exists a linear and continuous but noncomputable map $f : \ell_2 \to \ell_2$ with a co-r.e. closed $\text{graph}(f) \subseteq \ell_2 \times \ell_2$.

Proof. Let $a : \mathbb{N} \to \mathbb{N}$ be an injective computable function such that $\text{range}(a) = K$ is r.e. but not recursive. We define a linear map $f : \ell_2 \to \ell_2$ by $f(e_j) := a_{ij} e_i$, where

$$a_{ij} := \begin{cases} 1 & \text{if } a(i) = j, \\ 0 & \text{otherwise} \end{cases}$$

for all $i, j \in \mathbb{N}$. Then $||f|| \leq 1$ and f is obviously not computable, as

$$||f(e_j)||_2 = \begin{cases} 1 & \text{if } j \in K, \\ 0 & \text{otherwise} \end{cases}$$

for all $j \in \mathbb{N}$. However, $\text{graph}(f)$ is co-r.e. closed, as we will prove now. Therefore, let $(x, y) = ((x_i)_{i \in \mathbb{N}}, (y_i)_{i \in \mathbb{N}}) \in (\ell_2 \times \ell_2) \setminus \text{graph}(f)$. Then there is some $m \in \mathbb{N}$ such that $B((x, y), 2^{-m}) \subseteq (\ell_2 \times \ell_2) \setminus \text{graph}(f)$ and, in particular, $||f(x) - y||_2 \geq 2^{-m}$. For this

m there exist $n, k \in \mathbb{N}$ and $q = (q_0, \ldots, q_n, 0, 0, \ldots)$, $r = (r_0, \ldots, r_k, 0, 0, \ldots) \in \mathbb{Q}^{\mathbb{N}}$ such that

$$||q - x||_2 < 2^{-m-2} \text{ and } ||r - y||_2 < 2^{-m-2}. \tag{9.3}$$

Due to the definition of f, k can even be chosen large enough such that $f(q) = f(q)[k]$. Equation (9.3) implies $||f(q) - f(x)||_2 < ||f|| \cdot 2^{-m-2} \leq 2^{-m-2}$, and we obtain

$$\begin{aligned} ||f(q)[k] - r||_2 &\geq ||f(x) - y||_2 - ||f(q) - f(x)||_2 - ||r - y||_2 \\ &> 2^{-m} - 2^{-m-2} - 2^{-m-2} = 2^{-m-1}. \end{aligned}$$

Thus, given some arbitrary $(x, y) \in \ell_2 \times \ell_2$, we can systematically search for $n, k, m \in \mathbb{N}$ such that (9.3) and

$$||f(q)[k] - r||_2 > 2^{-m-1} \tag{9.4}$$

hold. Due to the definition of f this condition is decidable for q, r of the particular form. In the case that both conditions are satisfied, it follows that

$$\begin{aligned} ||f(x) - y||_2 &\geq ||f(q) - r||_2 - ||f(q) - f(x)||_2 - ||y - r||_2 \\ &\geq ||f(q)[k] - r||_2 - ||f(q) - f(x)||_2 - ||y - r||_2 \\ &> 2^{-m-1} - 2^{-m-2} - 2^{-m-2} = 0 \end{aligned}$$

and, in particular, $(x, y) \notin \text{graph}(f)$. Altogether, the proof shows that the algorithm finds suitable n, k, m for given (x, y) if and only if $(x, y) \notin \text{graph}(f)$. But this means that $\text{graph}(f)$ is co-r.e. closed. □

10. Conclusions. The table in Figure 10.1 gives a survey of our results. We assume that X, Y are computable metric spaces and $f : X \to Y$ is a continuous function. Each double row lists a certain version of the Graph Theorem (where the first row of any double row contains a nonuniform version and the second row the corresponding uniform version). The first double row treats the easy direction of the Graph Theorem, the remaining double rows of the table consider the problematic direction, and they are split into three cases where positive, negative, and both types of information on graphs are considered (in reverse order). The columns correspond to different additional conditions on the spaces X and Y (and on the functions f in the second half, which is devoted to the linear case). Any symbol "+" indicates that the statement of the corresponding row holds in general under the assumptions of the corresponding column. Any "−" indicates that there are counterexamples of spaces (and functions) that fulfill the assumptions of the corresponding column, but the statement of the corresponding row does not hold in case of the counterexample (in the uniform case it typically indicates that the corresponding mapping is not even continuous). Finally, the bottom of the columns contains examples of functionalities $X \to Y$ that simultaneously meet all indicated properties of the corresponding column.

The results of the first double row follow directly from Theorem 5.1, the weak version of the Graph Theorem. The positive results in the second column follow from Theorem 8.3, the Graph Theorem for locally connected source spaces. The additional positive results in the third column follow from Corollary 7.3, the Graph Theorem

	X, Y computable metric spaces	X eff. locally connected, $Y = \mathbb{R}^n$	X compact, Y locally computable	X eff. pathwise connected and complete, $Y = \mathbb{R}^n$	Y recursive compact	X, Y computable Banach spaces, f linear	X computable Banach space, $Y = \mathbb{R}^n$, f linear	$X = \mathbb{R}^m$, $Y = \mathbb{R}^n$, f linear
$f : X \to Y$ computable $\Longrightarrow$ graph(f) recursive	+	+	+	+	+	+	+	+
graph : $\mathcal{C}(X, Y) \to \mathcal{A}(X \times Y)$ computable	+	+	+	+	+	+	+	+
graph(f) recursive closed $\Longrightarrow f : X \to Y$ computable	−	+	+	+	+	+	+	+
graph$^{-1} :\subseteq \mathcal{A}(X \times Y) \to \mathcal{C}(X, Y)$ computable	−	+	−	+	+	−	+	+
graph(f) co-r.e. closed $\Longrightarrow f : X \to Y$ computable	−	−	+	+	+	−	+	+
graph$^{-1} :\subseteq \mathcal{A}_>(X \times Y) \to \mathcal{C}(X, Y)$ computable	−	−	−	−	+	−	+	+
graph(f) r.e. closed $\Longrightarrow f : X \to Y$ computable	−	−	−	−	−	+	+	+
graph$^{-1} :\subseteq \mathcal{A}_<(X \times Y) \to \mathcal{C}(X, Y)$ computable	−	−	−	−	−	−	−	+
	$\mathbb{N}^{\mathbb{N}} \to \mathbb{R}$	$\mathbb{R} \setminus \mathbb{N} \to \mathbb{R}$	$\{0,1\}^{\mathbb{N}} \to \mathbb{R}$	$\mathbb{R} \to \mathbb{R}$	$\mathbb{R} \to [0,1]$	$\ell_2 \to \ell_2$	$\ell_2 \to \mathbb{R}^n$	$\mathbb{R}^m \to \mathbb{R}^n$

FIG. 10.1. *Survey of different versions of the Graph Theorem.*

for compact source spaces and locally computable target spaces. The positive results in the fourth column follow from Corollaries 9.3 and 9.4, the Graph Theorem for pathwise connected source spaces. The positive results in the fifth column follow from Theorem 6.4, the Graph Theorem for compact target spaces. All positive results in the first row of the second and fourth double row concerning the linear case follow from the effective version of the closed Graph Theorem [2]. The remaining positive results in the second double row are implied by the corresponding positive results of the second column. The remaining positive results in the third double row follow from Corollary 9.5. The remaining positive result of the fourth double row is a consequence of Theorem 4.7 in [5]. All negative results in the first column follow from Example 5.3 for the space $X = \mathbb{N}^{\mathbb{N}}$ and Proposition 5.2. The negative results in the second column of the third double row follow from Example 8.7 for the space $X = \mathbb{R} \setminus \mathbb{N}$. The negative results in the third column and second and third double rows follow from Proposition 5.2 for the space $X = \{0,1\}^{\mathbb{N}}$. The negative results in the second, third, fourth, and fifth columns of the fourth double row follow from Example 6.6 for the spaces $X = Y = [0,1]$ or $X = Y = \mathbb{R}$. This example can easily be modified in order to obtain the negative results concerning the third column for the space $X = \{0,1\}^{\mathbb{N}}$. The remaining negative result in the fourth column is a consequence of Proposition 6.8 for the spaces $X = Y = \mathbb{R}$. The negative results in the sixth column are consequences of Proposition 8.6 for the spaces $X = Y = \ell_2$ and of Example 9.6. The negative result in the seventh column is a consequence of Theorem 4.5 in [5]. It should be mentioned that completeness in the fourth column has been used only for the positive result of the third row.

Of course the table leaves open many further questions. We just formulate one example.

Open problem. Is there an effectively pathwise connected computable metric space X such that $\text{graph}^{-1} :\subseteq \mathcal{A}(X \times \mathbb{R}^n) \to \mathcal{C}(X, \mathbb{R}^n)$ is not computable?

By the known results it follows that such a space X can be neither effectively locally connected nor complete. The space $X = (\mathbb{R} \times \{0\}) \cup (\mathbb{Q} \times \mathbb{R})$, endowed with the subtopology of the Euclidean topology, is an example of an effectively pathwise connected computable metric space that is neither locally connected nor complete and thus a potential candidate.

One interesting consequence of the Graph Theorem is that it allows certain conclusions with respect to the inversion problem: spaces X, Y such that the mapping from $\text{graph}(f)$ to $f : X \to Y$ is computable also have the property that the mapping from any homeomorphism $g : Y \to X$ to its inverse $g^{-1} : X \to Y$ is computable (in the uniform case as well as in the nonuniform case). This follows since information on the graph is symmetric (this is made precise by Lemma 8.5). Thus, in all those cases where the second double row in Figure 10.1 carries a plus, it follows that inversion is effective. However, the negative results cannot be transferred that easily (since it might happen that inversion can be established without using the graph as an intermediate step). This interesting problem deserves further investigation.

Acknowledgments. The author would like to express his gratitude to the anonymous referees who made a number of very valuable comments that helped to clarify and improve the presentation. One referee pointed out an omission in the proof of Theorem 9.2 that led to the introduction of (9.2).

REFERENCES

[1] S. Banach and S. Mazur, *Sur les fonctions calculables*, Ann. Soc. Pol. de Math., 16 (1937), p. 223.

[2] V. Brattka, *Computability of Banach Space Principles*, Informatik Berichte 286, FernUniversität Hagen, Fachbereich Informatik, Hagen, Germany, 2001.

[3] V. Brattka, *Random numbers and an incomplete immune recursive set*, in Automata, Languages and Programming, P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, eds., Lecture Notes in Comput. Sci. 2380, Springer-Verlag, Berlin, 2002, pp. 950–961.

[4] V. Brattka, *Computability over topological structures*, in Computability and Models, S. Barry Cooper and Sergey S. Goncharov, eds., Kluwer/Plenum, New York, 2003, pp. 93–136.

[5] V. Brattka, *Effective representations of the space of linear bounded operators*, Appl. Gen. Topol., 4 (2003), pp. 115–131.

[6] V. Brattka, *The inversion problem for computable linear operators*, in STACS 2003, H. Alt and M. Habib, eds., Lecture Notes in Comput. Sci. 2607, Springer-Verlag, Berlin, 2003, pp. 391–402.

[7] V. Brattka and G. Presser, *Computability on subsets of metric spaces*, Theoret. Comput. Sci., 305 (2003), pp. 43–76.

[8] V. Brattka and K. Weihrauch, *Computability on subsets of Euclidean space* I: *Closed and compact subsets*, Theoret. Comput. Sci., 219 (1999), pp. 65–93.

[9] M. Braverman and S. Cook, *Computing over the reals: Foundations for scientific computing*, Notices Amer. Math. Soc., 53 (2006), pp. 318–329.

[10] D. Cenzer and J. B. Remmel, *Effectively closed sets and graphs of computable real functions*, Theoret. Comput. Sci., 284 (2002), pp. 279–318.

[11] C. Ceola and P. B. A. Lecomte, *Computability of a map and decidability of its graph in the model of Blum, Shub and Smale*, Theoret. Comput. Sci., 194 (1998), pp. 219–223.

[12] A. Grzegorczyk, *On the definitions of computable real continuous functions*, Fund. Math., 44 (1957), pp. 61–71.

[13] K.-I. Ko, *Complexity Theory of Real Functions*, Progress in Theoretical Computer Science, Birkhäuser Boston, Boston, MA, 1991.

[14] D. Lacombe, *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles* I, C. R. Acad. Sci. Paris, 240 (1955), pp. 2478–2480.
[15] D. Lacombe, *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles* II, III, C. R. Acad. Sci. Paris, 241 (1955), pp. 13–14, 151–153.
[16] J. S. Miller, *Effectiveness for embedded spheres and balls*, in CCA 2002 Computability and Complexity in Analysis, V. Brattka, M. Schröder, and K. Weihrauch, eds., Electron. Notes Theor. Comput. Sci. 66, Elsevier, Amsterdam, 2002.
[17] P. Odifreddi, *Classical Recursion Theory*, Studies in Logic and the Foundations of Mathematics 125, North–Holland, Amsterdam, 1989.
[18] M. B. Pour-El and J. I. Richards, *Computability in Analysis and Physics*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1989.
[19] A. M. Turing, *On computable numbers, with an application to the "Entscheidungsproblem,"* Proc. London Math. Soc., 42 (1936), pp. 230–265.
[20] K. Weihrauch, *Computable Analysis*, Springer-Verlag, Berlin, 2000.
[21] K. Weihrauch, *Computational complexity on computable metric spaces*, Math. Log. Q., 49 (2003), pp. 3–21.
[22] M. Yasugi, T. Mori, and Y. Tsujii, *Effective properties of sets and functions in metric spaces with computability structure*, Theoret. Comput. Sci., 219 (1999), pp. 467–486.
[23] Q. Zhou, *Computable real-valued functions on recursive open and closed subsets of Euclidean space*, Math. Logic Q., 42 (1996), pp. 379–409.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 329–365

MAX ONES GENERALIZED TO LARGER DOMAINS*

PETER JONSSON†, FREDRIK KUIVINEN†, AND GUSTAV NORDH‡

Abstract. We study a family of problems, called MAXIMUM SOLUTION, where the objective is to maximize a linear goal function over the feasible integer assignments to a set of variables subject to a set of constraints. When the domain is Boolean (i.e., restricted to $\{0,1\}$), the maximum solution problem is identical to the well-studied MAX ONES problem, and the approximability is completely understood for all restrictions on the underlying constraints [S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson, *SIAM J. Comput.*, 30 (2001), pp. 1863–1920]. We continue this line of research by considering domains containing more than two elements. We present two main results: a complete classification for the approximability of all maximal constraint languages over domains of cardinality at most 4, and a complete classification of the approximability of the problem when the set of allowed constraints contains all permutation constraints. Under the assumption that a conjecture due to Szczepara [*Minimal Clones Generated by Groupoids*, Ph.D. thesis, Université de Móntreal, Montreal, QC, 1996] holds, we give a complete classification for all maximal constraint languages. These classes of languages are well studied in universal algebra and computer science; they have, for instance, been considered in connection with machine learning and constraint satisfaction. Our results are proved by using algebraic results from clone theory, and the results indicate that this approach is very powerful for classifying the approximability of certain optimization problems.

Key words. combinatorial optimization, constraint satisfaction, approximability, universal algebra

AMS subject classifications. 08A70, 90C27, 68Q17, 68Q25

DOI. 10.1137/060669231

1. Introduction.

Our starting point is the general combinatorial optimization problem MAX ONES(Γ), where Γ (known as the *constraint language*) is a finite set of finitary relations over $\{0,1\}$. An instance of this problem consists of constraints from Γ applied to a number of Boolean variables, and the goal is to find an assignment that satisfies all constraints while maximizing the number of variables set to 1. It is easy to see that by choosing the constraint language appropriately, MAX ONES(Γ) captures a number of well-known problems, for instance, MAX INDEPENDENT SET (problem GT23 in [2]) and certain variants of MAX 0/1 PROGRAMMING (problem MP2 in [2]). Many other problems are equivalent to MAX ONES under different reductions: for instance, MAX SET PACKING (also known as MAX HYPERGRAPH MATCHING) and MAX ONES are equivalent under PTAS-reductions [3].

The approximability (and thus the computational complexity) is known for all choices of Γ [37]. For any Boolean constraint language Γ, MAX ONES(Γ) is either in **PO** or is **APX**-complete or **poly-APX**-complete, or finding a solution of nonzero value is **NP**-hard or finding any solution is **NP**-hard. The exact borderlines between

*Received by the editors September 7, 2006; accepted for publication (in revised form) January 7, 2008; published electronically April 18, 2008. Preliminary versions of parts of this article appeared in *Proceedings of the* 12*th International Conference on Principles and Practice of Constraint Programming*, Nantes, France, 2006.

http://www.siam.org/journals/sicomp/38-1/66923.html

†Department of Computer and Information Science, Linköpings Universitet, SE-581 83 Linköping, Sweden (petej@ida.liu.se, freku@ida.liu.se). The first author's research was supported by the Center for Industrial Information Technology (CENIIT) under grant 04.01 and the Swedish Research Council (VR) under grants 621-2003-3421 and 2006-4532. The second author's research was supported by VR under grant 2006-4532 and the National Graduate School in Computer Science (CUGS).

‡LIX, École Polytechnique, 91128 Palaiseau, France (nordh@lix.polytechnique.fr). This author's research was supported by CUGS Sweden and Svensk-Franska Stiftelsen.

the different cases are given in [37]. Actually, two different problems are studied in [37]: the *weighted* problem (where each variable is assigned a nonnegative weight and the objective is to find a solution of maximum total weight) and the *unweighted* problem (where each variable is assigned the weight 1). They prove that the approximabilities for the weighted and unweighted versions of the problem coincide.

We will study a generalization of MAX ONES where variable domains are different from $\{0,1\}$: this allows us to capture more problems than with MAX ONES. For instance, this enables the study of certain problems in integer linear programming [28], problems in multiple-valued logic [36], and in equation solving over Abelian groups [38]. For larger domains, it seems significantly harder to obtain an exact characterization of approximability than in the Boolean case. Such a characterization would, for instance, show whether the dichotomy conjecture for constraint satisfaction problems is true or not—a famous open question which is believed to be difficult [25]. Hence, we exhibit restricted (but still fairly general) families of constraint languages where the approximability can be determined.

Let us now formally define the problem that we will study: let $D \subset \mathbb{N}$ (*the domain*) be a finite set. The set of all n-tuples of elements from D is denoted by D^n. Any subset of D^n is called an n-ary relation on D. The set of all finitary relations over D is denoted by R_D. A *constraint language* over a finite set, D, is a finite set $\Gamma \subseteq R_D$. Constraint languages are the way in which we specify restrictions on our problems. The constraint satisfaction problem over the constraint language Γ, denoted $\textsc{Csp}(\Gamma)$, is defined to be the decision problem with instance (V, D, C), where

- V is a set of variables,
- D is a finite set of values (sometimes called a domain), and
- C is a set of constraints $\{C_1, \ldots, C_q\}$, in which each constraint C_i is a pair (s_i, R_i), where s_i is a list of variables of length m_i, called the constraint scope, and R_i is an m_i-ary relation over the set D, belonging to Γ, called the constraint relation.

The question is whether there exists a solution to (V, D, C) or not, that is, a function from V to D such that, for each constraint in C, the image of the constraint scope is a member of the constraint relation. To exemplify this definition, let NAE be the following ternary relation on $\{0,1\}$: $NAE = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$. It is easy to see that the well-known **NP**-complete problem NOT-ALL-EQUAL SAT can be expressed as $\textsc{Csp}(\{NAE\})$.

The optimization problem that we are going to study, WEIGHTED MAXIMUM SOLUTION(Γ) (which we abbreviate W-MAX SOL(Γ)), is defined as follows:

Instance: Tuple (V, D, C, w), where D is a finite subset of $\mathbb{N}$, (V, D, C) is a $\textsc{Csp}(\Gamma)$ instance, and $w : V \to \mathbb{N}$ is a weight function.

Solution: An assignment $f : V \to D$ to the variables such that all constraints are satisfied.

Measure: $\sum_{v \in V} w(v) \cdot f(v)$.

EXAMPLE 1.1. *Consider the domain $D = \{0,1\}$ and the binary relation $R = \{(0,0),(1,0),(0,1)\}$. Then,* W-MAX SOL($\{R\}$) *is exactly the weighted* MAXIMUM INDEPENDENT SET *problem.*

Although the W-MAX SOL(Γ) problem is defined only for finite constraint languages, we will, in order to simplify the presentation, sometimes deal with sets of relations which are infinite. For a (possible infinite) set of relations X we will say that W-MAX SOL(X) is tractable if W-MAX SOL(Y) is tractable for every finite subset Y of X. Here "tractable" may be containment in one of **PO**, **APX**, or **poly-**

APX. Similarly, we say that W-MAX SOL(X) is hard if there is a finite subset Y of X such that W-MAX SOL(Y) is hard. Here "hard" will be one of **APX**-hard, **poly-APX**-hard, or that it is **NP**-hard to find feasible solutions.

Note that our choice of measure function in the definition of W-MAX SOL(Γ) is just one of several reasonable choices. Another reasonable alternative, used in [38], would be to let the domain D be any finite set and introduce an additional function $g : D \to \mathbb{N}$ mapping elements from the domain to natural numbers. The measure could then be defined as $\sum_{v \in V} w(v) \cdot g(f(v))$. This would result in a parameterized problem W-MAX SOL(Γ, g) where the goal is to classify the complexity of W-MAX SOL(Γ, g) for all combinations of constraint languages Γ and functions g. Note that our definition of W-MAX SOL(Γ) is equivalent to the definition of W-MAX SOL(Γ, g) if, in addition, g is required to be injective. One of our motivations for the choice of measure function in the definition of W-MAX SOL(Γ) is to stay closer to the definition of integer programming.

Considering only finite constraint language is in many cases not very restrictive. Consider, for instance, integer programming over the bounded domain $\{0, \ldots, d-1\}$. Each row in the constraint matrix can be viewed as an inequality

$$a_1 x_1 + a_2 x_2 + \cdots + a_k x_k \geq b.$$

Obviously, such an inequality is equivalent to the following three inequalities:

$$\begin{gathered} a_1 x_1 + a_2 x_2 + \cdots + a_{\lfloor k/2 \rfloor} x_{\lfloor k/2 \rfloor} - z \geq 0, \\ -a_1 x_1 - a_2 x_2 - \cdots - a_{\lfloor k/2 \rfloor} x_{\lfloor k/2 \rfloor} + z \geq 0, \\ z + a_{\lfloor k/2 \rfloor + 1} + \cdots + a_k x_k \geq b, \end{gathered}$$

where z denotes a fresh variable that is given the weight 0 in the objective function. By repeating this process, one ends up with a set of inequalities where each inequality contains at most three variables, and the optimal solution to this instance has the same measure as the original instance. There are at most $2^d + 2^{d^2} + 2^{d^3}$ different inequalities of length ≤ 3 since the domain contains d elements; that is, we have reduced the problem to one with a finite constraint language. Finally, this reduction is polynomial-time: each inequality of length k in the original instance gives rise to at most $3^{\lceil \log_2 k \rceil} = O(k^2)$ inequalities and at most $O(k^2)$ new variables.

While the approximability of W-MAX SOL is well understood for the Boolean domain, this is not the case for larger domains. For larger domains we are aware of three results, the first one being a tight (in)approximability result for W-MAX SOL(Γ) when Γ is the set of relations that can be expressed as linear equations over $\mathbb{Z}_p$ [38] (see also section 6.3.1 where we define the problem formally). The second result is due to Hochbaum and Naor [28] who study integer programming with monotone constraints, i.e., every constraint is of the form $ax - by \leq c$, where x and y are variables and $a, b \in \mathbb{N}$ and $c \in \mathbb{Z}$. In our setting, their result is a polynomial-time algorithm for certain constraint languages. The third result is a study of the approximability of certain logically defined constraint languages [36]. The main goal of this article is to gain a better understanding of non-Boolean W-MAX SOL—for doing so, we will adapt the algebraic approach for CSPs [13, 32] for studying the approximability of W-MAX SOL.

When the algebraic approach is applicable to a certain problem, there is an equivalence relation on the constraint languages such that two constraint languages which are equivalent under this relation have the same complexity. More specifically, two constraint languages are in the same equivalence class if they generate the same *relational clone*. The relational clone generated by Γ captures the expressive power of Γ

and is denoted by $\langle\Gamma\rangle$. Hence, instead of studying every possible finite set of relations it is enough to study the relational clones. Thus, given two constraint languages Γ_1 and Γ_2 such that $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle$, it follows that W-MAX SOL(Γ_1) and W-MAX SOL(Γ_2) are equivalent under polynomial-time reductions.

The clone-theoretic approach for studying the complexity of CSPs has been very successful: it has, for instance, made it possible to design new efficient algorithms and to clarify the borderline between tractability and intractability in many important cases. In particular the complexity of the CSP problem over three-element domains is now completely understood [10]. In addition to the CSP problem it is possible to use the tools from universal algebra to prove complexity results in many other CSP-like problems. One example of such a problem is the quantified constraint satisfaction problem (QCSP), where variables can not only be existentially quantified but also universally quantified. The complexity of QCSP has been successfully attacked with the clone-theoretic approach [7, 17]. Furthermore, the #CSP problem [11] (where the number of solutions to a CSP is counted) has also benefitted from this approach. However, it seems that this technique cannot be used for some other CSP-like problems: notable exceptions are MAX CSP [34] and the problem of enumerating all solutions to a CSP instance [46]. For some problems it is the case that the relational clones are a useable tool in the Boolean domain but not in larger domains. The enumeration problem is one such case [46].

We begin by proving that the algebraic approach is applicable to W-MAX SOL, and this result can be found in Theorem 3.3.[1] In fact, we show that, given two constraint languages Γ_1 and Γ_2 such that $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle$, W-MAX SOL($\Gamma_1$) S-reduces to W-MAX SOL(Γ_2), and vice versa. An *S-reduction* is a certain strong approximation-preserving reduction: if $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle$, then Γ_1 and Γ_2 are very similar with respect to approximability. For instance, if W-MAX SOL(Γ_1) is **NP**-hard to approximate within some constant c, then W-MAX SOL(Γ_2) is **NP**-hard to approximate within c, too. The proof is accompanied by an example of how the approach can be used "in practice" for proving approximability results. We note that the clone-theoretic approach was not used in the original classification of MAX ONES and, consequently, the technique we use differs substantially from those used in [37]. The results that we prove with the aid of Theorem 3.3 are the following.

Result 1. Our first result concerns the complexity of W-MAX SOL for *maximal* constraint languages. A constraint language Γ is maximal if, for any $R \notin \langle\Gamma\rangle$, $\Gamma \cup \{R\}$ has the ability to express (in a sense to be formally defined later) every relation in R_D. Such languages have attracted much attention lately: for instance, the complexity of the corresponding CSP problems has been completely classified. In [14] the complexity was classified for domains $|D| \leq 3$, and necessary conditions for tractability were proved for the general case. More recently, in [8], it was proved that those necessary conditions also are sufficient for tractability. Maximal constraint languages have also been studied in the context of machine learning [24] and quantified CSPs [18], and they attract a great deal of attention in universal algebra; cf. the survey by Quackenbush [44].

Our results show that if Γ is maximal and $|D| \leq 4$, then W-MAX SOL(Γ) is tractable, or **APX**-complete, or **poly-APX**-complete, or finding any solution with nonzero measure is **NP**-hard, or CSP(Γ) is not tractable. Moreover, we prove that

[1] The proof is easy to adapt to other problems such as W-MIN SOL (the minimization version of W-MAX SOL) and AW-MAX SOL (where both positive and negative weights are allowed).

under a conjecture by Szczepara [47] our classification of maximal constraint languages extends to arbitrary finite domains. In the conference version [35] of this article we claimed that we had characterized the complexity of MAX SOL for all maximal constraint languages. Unfortunately, there was a flaw in one of the proofs. We have managed to repair some of it by proving the weaker results as stated above, but the general case, when $|D| > 4$ and Szczepara's conjecture is not assumed to hold, remains open. We also note that the different cases can be efficiently recognized; i.e., the approximability of a maximal constraint language Γ can be decided in polynomial time (in the size of Γ).

When proving this result, we identified a new large tractable class of W-MAX SOL(Γ): *generalized* max-*closed* constraints. This class (which may be of independent interest) significantly extends some of the tractable classes of MAX ONES that were identified by Khanna et al. [37]. It is also related to *monotone* constraints which have been studied in mathematical programming and computer science [27, 28, 51]. In fact, generalized max-closed constraints generalize monotone constraints over finite domains. A certain kind of generalized max-closed constraints is relevant in constraint programming languages such as CHIP [50], as is pointed out in [33]. It may thus be possible to extend such languages with optimization capabilities by using the techniques presented in this article.

Result 2. We completely characterize the approximability of W-MAX SOL(Γ) when Γ contains all permutation constraints. Such languages are known as *homogeneous* languages, and Dalmau [23] has determined the complexity of CSP(Γ) for all such languages, while the complexity of the corresponding quantified CSPs has been studied by Börner et al. [6]. Szendrei [48] provides a compact presentation of algebraic results on homogeneous algebras and languages.

We show that W-MAX SOL(Γ) is either tractable, **APX**-complete, or **poly-APX**-complete, or that CSP(Γ) is not tractable. The four different cases can, just as in Result 1, be efficiently recognized. The proof is based on the characterization of homogeneous algebras by Marczewski [40] and Marchenkov [39]. For each domain D, there exists a set of relations Q_D such that every tractable homogeneous constraint language on D is a subset of Q_D. The relations in Q_D are invariant under a certain operation $t : D^3 \to D$ (known as the *discriminator* on D), and the algebra $(D; t)$ is an example of a *quasi-primal* algebra in the sense of Pixley [41]. We note that the tractable homogeneous constraint languages have been considered earlier in connection with *soft constraints* [21], i.e., constraints which allow different levels of "desirability" to be associated with different value assignments [5]. In the terminology of [21], these languages are invariant under a $\langle \mathsf{Mjrty}_1, \mathsf{Mjrty}_2, \mathsf{Mnrty}_3 \rangle$ multimorphism. We also note that the tractable homogeneous languages extend the *width*-2 *affine* class of MAX ONES that was identified by Khanna et al. [37].

We remark that we do not deal explicitly with the unweighted version of the problem (denoted MAX SOL(Γ)), where all variables have weight 1. The reason for this is that the approximability classifications for MAX SOL(Γ) can be deduced from the classifications for W-MAX SOL(Γ) (for all constraint languages Γ considered in this paper). In fact, as we explain in section 8, MAX SOL(Γ) has the same approximability as W-MAX SOL(Γ) when Γ is one of the constraint languages considered in this article.

The article is structured as follows: we begin by presenting some basics on approximability in section 2. The algebraic approach for studying W-MAX SOL is presented in section 3, section 4 identifies certain hard constraint languages, and section 5 contains some tractability results. We continue with section 6 that contains

Result 1 and section 7 that contains Result 2. Finally, section 8 contains some final remarks.

2. Approximability, reductions, and completeness. A *combinatorial optimization problem* is defined over a set of *instances* (admissible input data); each instance I has a finite set $\mathsf{sol}(I)$ of *feasible solutions* associated with it. Given an instance I and a feasible solution s of I, $m(I,s)$ denotes the positive integer *measure* of s. The objective is, given an instance I, to find a feasible solution of *optimum* value with respect to the measure m. The optimal value is the largest one for *maximization* problems and the smallest one for *minimization* problems. A combinatorial optimization problem is said to be an **NPO** problem if its instances and solutions can be recognized in polynomial time, the solutions are polynomially bounded in the input size, and the objective function can be computed in polynomial time (see, e.g., [2]).

We say that a solution $s \in \mathsf{sol}(I)$ to an instance I of an **NPO** problem Π is *r-approximate* if it satisfies

$$\max\left\{\frac{m(I,s)}{\text{OPT}(I)}, \frac{\text{OPT}(I)}{m(I,s)}\right\} \leq r,$$

where $\text{OPT}(I)$ is the optimal value for a solution to I. An approximation algorithm for an **NPO** problem Π has *performance ratio* $\mathcal{R}(n)$ if, given any instance I of Π with $|I| = n$, it outputs an $\mathcal{R}(n)$-approximate solution.

We define **PO** to be the class of **NPO** problems that can be solved (to optimality) in polynomial time. An **NPO** problem Π is in the class **APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a constant. Similarly, Π is in the class **poly-APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a polynomial in the size of the input. Completeness in **APX** and **poly-APX** is defined using appropriate reductions, called AP-reductions and A-reductions, respectively [22, 37]. AP-reductions are more sensitive than A-reductions, and every AP-reduction is also an A-reduction [37]. In this paper we will not need the added flexibility of A-reductions for proving our **poly-APX**-completeness results. Hence, we need only the definition of AP-reductions.

DEFINITION 2.1. *An* **NPO** *problem* Π_1 *is said to be* AP-reducible *to an* **NPO** *problem* Π_2 *if two polynomial-time computable functions* F *and* G *and a constant* α *exist such that*

(a) *for any instance* I *of* Π_1, $F(I)$ *is an instance of* Π_2*;*
(b) *for any instance* I *of* Π_1 *and any feasible solution* s' *of* $F(I)$, $G(I,s')$ *is a feasible solution of* I*;*
(c) *for any instance* I *of* Π_1 *and any* $r \geq 1$, *if* s' *is an* r*-approximate solution of* $F(I)$, *then* $G(I,s')$ *is an* $(1+(r-1)\alpha+o(1))$*-approximate solution of* I, *where the o-notation is with respect to* $|I|$.

An **NPO** *problem* Π *is* **APX**-hard *(***poly-APX**-hard*) if every problem in* **APX** *(***poly-APX***) is AP-reducible (A-reducible) to it. If, in addition,* Π *is in* **APX** *(***poly-APX***), then* Π *is called* **APX**-complete *(***poly-APX**-complete*).* It is a well-known fact (see, e.g., section 8.2.1 in [2]) that AP-reductions compose. In some proofs we will use another kind of reduction, S-reductions. They are defined as follows.

DEFINITION 2.2. *An* **NPO** *problem* Π_1 *is said to be* S-reducible *to an* **NPO** *problem* Π_2 *if two polynomial-time computable functions* F *and* G *exist such that the following hold:*

(a) *Given any instance* I *of* Π_1, *algorithm* F *produces an instance* $I' = F(I)$ *of*

Π_2, *such that the measure of an optimal solution for* I', $\text{OPT}(I')$, *is exactly* $\text{OPT}(I)$.

(b) *Given* $I' = F(I)$ *and any solution* s' *to* I', *algorithm* G *produces a solution* s *to* I *such that* $m_1(I, G(s')) = m_2(I', s')$, *where* m_1 *is the measure for* Π_1 *and* m_2 *is the measure for* Π_2.

Obviously, the existence of an S-reduction from Π_1 to Π_2 implies the existence of an AP-reduction from Π_1 to Π_2. The reason why we need S-reductions is that AP-reductions do not (generally) preserve membership in **PO** [37]. We also note that S-reductions preserve approximation thresholds exactly for problems in **APX**: letting Π_1, Π_2 be problems in **APX**, assume that it is **NP**-hard to approximate Π_1 within c, and that there exists an S-reduction from Π_1 to Π_2. Then, it is **NP**-hard to approximate Π_2 within c, too.

In some of our hardness proofs, it will be convenient for us to use a type of approximation-preserving reduction called L-reduction [2].

DEFINITION 2.3. *An* **NPO** *maximization problem* Π_1 *is said to be* L-reducible *to an* **NPO** *maximization problem* Π_2 *if two polynomial-time computable functions* F *and* G *and positive constants* β *and* γ *exist such that*

(a) *given any instance* I *of* Π_1, *algorithm* F *produces an instance* $I' = F(I)$ *of* Π_2 *such that the measure of an optimal solution for* I', $\text{OPT}(I')$, *is at most* $\beta \cdot \text{OPT}(I)$;

(b) *given* $I' = F(I)$ *and any solution* s' *to* I', *algorithm* G *produces a solution* s *to* I *such that* $|m_1(I, s) - \text{OPT}(I)| \leq \gamma \cdot |m_2(I', s') - \text{OPT}(I')|$, *where* m_1 *is the measure for* Π_1 *and* m_2 *is the measure for* Π_2.

It is well known (see, e.g., Lemma 8.2 in [2]) that, if Π_1 is L-reducible to Π_2 and $\Pi_1 \in$ **APX**, then there is an AP-reduction from Π_1 to Π_2.

3. Algebraic approach. We sometimes need to define relations in terms of other relations, using certain logical formulas. In these definitions we use the standard correspondence between constraints and relations: a relation consists of all tuples of values satisfying the corresponding constraint. Although we sometimes use the same symbol for a constraint and its corresponding relation, the meaning will always be clear from the context. More specifically, for a relation R with arity a we will sometimes write $R(x_1, \ldots, x_a)$ with the meaning $(x_1, \ldots, x_a) \in R$, and the constraint $((x_1, \ldots, x_a), R)$ will sometimes be written as $R(x_1, \ldots, x_a)$.

An *operation* on a finite set D (the domain) is an arbitrary function $f : D^k \to D$. Any operation on D can be extended in a standard way to an operation on tuples over D as follows: let f be a k-ary operation on D and let R be an n-ary relation over D. For any collection of k tuples, $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R$, the n-tuple $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k})$ is defined as follows: $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) = (f(\boldsymbol{t_1}[1], \boldsymbol{t_2}[1], \ldots, \boldsymbol{t_k}[1]),\ f(\boldsymbol{t_1}[2], \boldsymbol{t_2}[2], \ldots, \boldsymbol{t_k}[2]), \ldots, f(\boldsymbol{t_1}[n], \boldsymbol{t_2}[n], \ldots, \boldsymbol{t_k}[n]))$, where $\boldsymbol{t_j}[i]$ is the ith component in tuple $\boldsymbol{t_j}$. A technique that has shown to be useful in determining the computational complexity of $\text{CSP}(\Gamma)$ is that of investigating whether the constraint language Γ is invariant under certain families of operations [32].

Now, let $R_i \in \Gamma$. If f is an operation such that for all $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R_i$, $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) \in R_i$, then R_i is *invariant* (or, in other words, *closed*) under f. If all constraint relations in Γ are invariant under f, then Γ is invariant under f. An operation f such that Γ is invariant under f is called a *polymorphism* of Γ. The set of all polymorphisms of Γ is denoted $Pol(\Gamma)$. Given a set of operations F, the set of all relations that are invariant under all the operations in F is denoted $Inv(F)$. Whenever there is only one operation under consideration, we write $Inv(f)$ instead of $Inv(\{f\})$.

We will need a number of operations in what follows: an operation f over D is said to be

- a *constant* operation if f is unary and $f(a) = c$ for all $a \in D$ and some $c \in D$;
- a *majority* operation if f is ternary and $f(a,a,b) = f(a,b,a) = f(b,a,a) = a$ for all $a, b \in D$;
- a *binary commutative idempotent* operation if f is binary, $f(a,a) = a$ for all $a \in D$, and $f(a,b) = f(b,a)$ for all $a, b \in D$;
- an *affine* operation if f is ternary and $f(a,b,c) = a - b + c$ for all $a, b, c \in D$, where $+$ and $-$ are the binary operations of an Abelian group $(D, +, -)$.

EXAMPLE 3.1. *Let $D = \{0,1,2\}$ and let f be the majority operation on D where $f(a,b,c) = a$ if a, b, and c are all distinct. Furthermore, let*

$$R = \{(0,0,1), (1,0,0), (2,1,1), (2,0,1), (1,0,1)\}.$$

It is then easy to verify that for every triple of tuples, $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in R$, we have $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \in R$. For example, if $\boldsymbol{x} = (0,0,1)$, $\boldsymbol{y} = (2,1,1)$, and $\boldsymbol{z} = (1,0,1)$, then

$$\begin{aligned} f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) &= (f(\boldsymbol{x}[1], \boldsymbol{y}[1], \boldsymbol{z}[1]), f(\boldsymbol{x}[2], \boldsymbol{y}[2], \boldsymbol{z}[2]), f(\boldsymbol{x}[3], \boldsymbol{y}[3], \boldsymbol{z}[3])) \\ &= (f(0,2,1), f(0,1,0), f(1,1,1)) = (0,0,1) \in R. \end{aligned}$$

We can conclude that R is invariant under f or, equivalently, that f is a polymorphism of R.

We continue by defining a closure operation $\langle \cdot \rangle$ on sets of relations: for any set $\Gamma \subseteq R_D$ the set $\langle \Gamma \rangle$ consists of all relations that can be expressed using relations from $\Gamma \cup \{=_D\}$ ($=_D$ is the equality relation on D), conjunction, and existential quantification. Intuitively, constraints using relations from $\langle \Gamma \rangle$ are exactly those which can be simulated by constraints using relations from Γ. The sets of relations of the form $\langle \Gamma \rangle$ are referred to as *relational clones*. An alternative characterization of relational clones is given in the following theorem.

THEOREM 3.2 (see [43]). *For every set $\Gamma \subseteq R_D$, $\langle \Gamma \rangle = Inv(Pol(\Gamma))$.*

The following theorem states that when we are studying the approximability of W-MAX SOL(Γ), it is sufficient to consider constraint languages that are relational clones.

THEOREM 3.3. *Let Γ be a constraint language and let $\Gamma' \subseteq \langle \Gamma \rangle$ be finite. Then* W-MAX SOL(Γ') *is S-reducible to* W-MAX SOL(Γ).

Proof. Consider an instance $I = (V, D, C, w)$ of W-MAX SOL(Γ'). We transform I into an instance $F(I) = (V', D, C', w')$ of W-MAX SOL(Γ).

For every constraint $C = ((v_1, \ldots, v_m), R)$ in I, R can be represented as

$$\exists_{v_{m+1}}, \ldots, \exists_{v_n} R_1(v_{11}, \ldots, v_{1n_1}) \wedge \cdots \wedge R_k(v_{k1}, \ldots, v_{kn_k}),$$

where $R_1, \ldots, R_k \in \Gamma \cup \{=_D\}$, $v_{m+1}, \ldots, v_n$ are fresh variables, and $v_{11}, \ldots, v_{1n_1}$, $v_{21}, \ldots, v_{kn_k} \in \{v_1, \ldots, v_n\}$. Replace the constraint C with the constraints

$$((v_{11}, \ldots, v_{1n_1}), R_1), \ldots, ((v_{k1}, \ldots, v_{kn_k}), R_k),$$

add $v_{m+1}, \ldots, v_n$ to V, and extend w so that $v_{m+1}, \ldots, v_n$ are given weight 0. If we repeat the same reduction for every constraint in C, then it results in an equivalent instance of W-MAX SOL($\Gamma_1 \cup \{=_D\}$).

For each equality constraint $((v_i, v_j), =_D)$, we do the following:

- replace all occurrences of v_j with v_i, update w' so that the weight of v_j is added to the weight of v_i, remove v_j from V, and remove the weight corresponding to v_j from w'; and

- remove $((v_i, v_j), =_D)$ from C.

The resulting instance $F(I) = (V', D, C', w')$ of W-MAX SOL(Γ) has the same optimum as I (i.e., $\text{OPT}(I) = \text{OPT}(F(I))$) and has been obtained in polynomial time.

Now, given a feasible solution S' for $F(I)$, let $G(I, S')$ be the feasible solution for I where the following hold:

- The variables in I assigned by S' inherit their value from S'.
- The variables in I which are still unassigned all occur in equality constraints, and their values can be found by simply propagating the values of the variables which have already been assigned.

It should be clear that $m(I, G(I, S')) = m(F(I), S')$ for any feasible solution S' for $F(I)$. Hence, the functions F and G, as described above, are the two parts of an S-reduction from W-MAX SOL(Γ') to W-MAX SOL(Γ). □

To exemplify the use of the results in this section, we prove the following tight approximability result.

LEMMA 3.4. *Let Γ be a finite constraint language over the domain $\{0, 1\}$. If* MAX ONES(Γ) *is in* **APX** *and not in* **PO**, *then there is a polynomial-time approximation algorithm for* MAX ONES(Γ) *with performance ratio 2, and it is* **NP***-hard to approximate* MAX ONES(Γ) *within $2 - \epsilon$, for any $\epsilon > 0$.*

Proof sketch. It follows from the classification results in [37] that if MAX ONES(Γ) is in **APX** and not in **PO**, then Γ is closed under the affine function $f(x, y, z) = x - y + z \pmod 2$. It also follows from [37, Lemma 6.6] that MAX ONES(Γ) is approximable within 2.

In the Boolean domain, the structure of all relational clones is known. This classification was made by Post in [42] and is often referred to as Post's lattice. A gentle introduction to Boolean relations and Post's lattice can be found in [15, 16].

By Theorem 3.3, it is enough to study the relational clones. By studying Post's lattice and the results for MAX ONES in [37], one can conclude that there are three relational clones which are interesting in our case (i.e., there are three relational clones such that MAX ONES(Γ) is in **APX** but not in **PO**). Those relational clones are called IL_0, IL_2, and IL_3 and can be defined as follows [16]:

$$IL_0 = \{x_1 + \cdots + x_k = 0 \pmod 2 \mid k \in \mathbb{N}\},$$
$$IL_2 = \{x_1 + \cdots + x_k = c \pmod 2 \mid k \in \mathbb{N},\ c \in \{0, 1\}\},$$
$$IL_3 = \{x_1 + \cdots + x_k = c \pmod 2 \mid k \text{ even},\ c \in \{0, 1\}\}.$$

We get the following inclusions from Post's lattice: $IL_0 \subset IL_2$ and $IL_3 \subset IL_2$.

It is proved in [38] that for a certain finite subset Γ of IL_3, MAX ONES(Γ) is **NP**-hard to approximate within $2 - \epsilon$ for all $\epsilon > 0$. As $IL_3 \subset IL_2$ we get that MAX ONES(Γ) is **NP**-hard to approximate within $2 - \epsilon$ for all $\epsilon > 0$ if $\langle\Gamma\rangle = IL_2$.

What remains to be done is to prove **NP**-hardness for approximating MAX ONES(Γ) within $2 - \epsilon$ if $\langle\Gamma\rangle = IL_0$. We do this with a reduction from MAX-E3-LIN-2 which is the following problem: given a set of equations over $\mathbb{Z}_2$ with exactly three variables per equation, satisfy as many equations as possible. It is proved in [29] that it is **NP**-hard to approximate MAX-E3-LIN-2 within $2 - \epsilon$ for any $\epsilon > 0$.

Let I be an instance of MAX-E3-LIN-2. We will construct an instance I' of MAX ONES(Γ) for a subset Γ of IL_0. Given an equation $x_1 + x_2 + x_3 = 1 \pmod 2$ in I (we can assume that all equations have 1 on the right-hand side [29]), we add the equation $x_1 + x_2 + x_3 = z$ (where z is a fresh variable that occurs only in one equation) to I'. Furthermore, we assign the weight 0 to x_1, x_2, and x_3 and the weight 1 to z. It is

not hard to see that a solution with measure m to I can easily be transformed into a solution with measure m for I'. It is also the case that a solution of measure m for I' can be seen as a solution with measure m for I. ∎

4. Hardness and membership results. In this section, we first prove some general **APX** and **poly-APX** membership results for W-MAX SOL(Γ). We also prove **APX**-completeness and **poly-APX**-completeness for some particular constraint languages. Most of our hardness results in subsequent sections are based on these results.

We begin by making the following easy but interesting observation: we know from the classification of W-MAX SOL(Γ) over the Boolean domain $\{0,1\}$ that there exist many constraint languages Γ for which W-MAX SOL(Γ) is **poly-APX**-complete. However, if 0 is not in the domain, then there are no constraint languages Γ such that W-MAX SOL(Γ) is **poly-APX**-complete.

PROPOSITION 4.1. *If* CSP(Γ) *is in* **P** *and* $0 \notin D$, *then* W-MAX SOL(Γ) *is in* **APX**.

Proof. It is proved in [19] that if CSP(Γ) is in **P**, then we can also find a solution in polynomial time. It should be clear that this solution is a $\frac{\max(D)}{\min(D)}$-approximate solution. Hence, we have a trivial approximation algorithm with performance ratio $\frac{\max(D)}{\min(D)}$. ∎

Next, we present a general membership result for W-MAX SOL(Γ). The proof is similar to the proof of the corresponding result for the Boolean domain in [37, Lemma 6.2] so we omit the proof.

LEMMA 4.2. *Let* $\Gamma^c = \Gamma \cup \{\{(d_1)\}, \ldots, \{(d_n)\}\}$, *where* $D = \{d_1, \ldots, d_n\}$ *(i.e.,* Γ^c *is the constraint language corresponding to* Γ *where we can force variables to take any given value in the domain). If* CSP(Γ^c) *is in* **P**, *then* W-MAX SOL(Γ) *is in* **poly-APX**.

We continue by proving the **APX**-completeness of some constraint languages.

LEMMA 4.3. *Let* $R = \{(a,a),(a,b),(b,a)\}$ *and* $a, b \in D$ *such that* $0 < a < b$. *Then,* W-MAX SOL($\{R\}$) *is* **APX**-*complete.*

Proof. Containment in **APX** follows from Proposition 4.1. To prove the hardness result we give an L-reduction (with parameters $\beta = 4b$ and $\gamma = \frac{1}{b-a}$) from the **APX**-complete problem INDEPENDENT SET restricted to degree 3 graphs [1] to MAX SOL($\{R\}$). Given an instance $I = (V, E)$ of INDEPENDENT SET (restricted to graphs of degree at most 3 and containing no isolated vertices), let $F(I) = (V, D, C)$ be the instance of MAX SOL($\{R\}$) where, for each edge $(v_i, v_j) \in E$, we add the constraint $R(x_i, x_j)$ to C. For any feasible solution S' for $F(I)$, let $G(I, S')$ be the solution for I where all vertices corresponding to variables assigned b in S' form the independent set. We have $|V|/4 \leq \text{OPT}(I)$ and $\text{OPT}(F(I)) \leq b|V|$, so $\text{OPT}(F(I)) \leq 4b\text{OPT}(I)$. Thus, $\beta = 4b$ is an appropriate parameter.

Let K be the number of variables being set to b in an arbitrary solution S' for $F(I)$. Then,

$$|\text{OPT}(I) - m(I, G(I, S'))| = \text{OPT}(I) - K \quad \text{and}$$
$$|\text{OPT}(F(I)) - m(F(I), S')| = (b-a)(\text{OPT}(I) - K).$$

Hence,

$$|\text{OPT}(I) - m(I, G(I, S'))| = \frac{1}{b-a}|\text{OPT}(F(I)) - m(F(I), S')|,$$

and $\gamma = \frac{1}{b-a}$ is an appropriate parameter. ∎

The generic **poly-APX**-complete constraint languages are presented in the following lemma.

LEMMA 4.4. *Let* $R = \{(0,0),(0,b),(b,0)\}$ *and* $b \in D$ *such that* $0 < b$. *Then,* W-MAX SOL($\{R\}$) *is* **poly-APX**-*complete.*

Proof. It is proved in [37, Lemma 6.15] that for $Q = \{(0,0),(0,1),(1,0)\}$, it is the case that W-MAX SOL($\{Q\}$) is **poly-APX**-complete. To prove the **poly-APX**-hardness we give an *AP*-reduction from W-MAX SOL($\{Q\}$) to W-MAX SOL($\{R\}$). Given an instance I of W-MAX SOL($\{Q\}$), let $F(I)$ be the instance of W-MAX SOL($\{R\}$) where all occurrences of Q have been replaced by R. For any feasible solution S' for $F(I)$, let $G(I,S')$ be the solution for I where all variables assigned b in S' are instead assigned 1. It should be clear that this is an *AP*-reduction, since if S' is an α-approximate solution to $F(I)$, then $G(I,S')$ is an α-approximate solution for I.

To see that W-MAX SOL($\{R\}$) is in **poly-APX**, let $D = \{d_1,\ldots,d_n\}$ and note that $\Gamma^c = \{R,\{(d_1)\},\ldots,\{(d_n)\}\}$ is invariant under the min function. As the min function is associative, commutative, and idempotent, CSP(Γ^c) is solvable in polynomial time [32]. Hence, W-MAX SOL($\{R\}$) is in **poly-APX** due to Lemma 4.2. ∎

5. Tractable constraint languages. In this section, we present tractability results for two classes of constraint languages: *injective* constraint languages and *generalized* max-*closed* constraint languages. The tractability of injective constraints follows from Cohen et al. [21, sect. 4.4], but we present a simple proof for increased readability. The tractability result for generalized max-closed constraints is new, and its proof constitutes the main part of this section.

These two classes can be seen as substantial and nontrivial generalizations of the tractable classes known for the corresponding (WEIGHTED) MAX ONES problem over the Boolean domain. There are only three tractable classes of constraint languages over the Boolean domain, namely, width-2 affine, 1-valid, and weakly positive [37]. Width-2 affine constraint languages are examples of injective constraint languages, and the classes of 1-valid and weakly positive constraint languages are examples of generalized max-closed constraint languages. The monotone constraints which are, for instance, studied by Hochbaum et al. [27] and Hochbaum and Naor [28] (in relation to integer programming) and Woeginger [51] (in relation to constraint satisfaction) are also related to generalized max-closed constraints. Hochbaum and Naor [28] show that monotone constraints can be characterized as those constraints that are simultaneously invariant under the max and min operators. Hence, monotone constraints are also generalized max-closed constraints as long as the underlying domain is finite.

5.1. Injective relations. We begin by formally defining *injective relations.*

DEFINITION 5.1. *A relation* $R \in R_D$ *is called* injective *if there exists a subset* $D' \subseteq D$ *and an injective function* $\pi : D' \to D$ *such that*

$$R = \{(x,\pi(x)) \mid x \in D'\}.$$

It is important to note that the function π is *not* assumed to be total on D. Let I^D denote the set of all injective relations on the domain D and let $\Gamma_I^D = \langle I^D \rangle$.

EXAMPLE 5.2. *Let* $D = \{0,1\}$ *and let* $R = \{(x,y) \mid x,y \in D,\ x+y \equiv 1 \pmod 2\}$. *The relation* R *is injective because the function* $f : D \to D$ *defined as* $f(0) = 1$ *and* $f(1) = 0$ *is injective. More generally, let* $G = (D',+,-)$ *be an arbitrary Abelian group and let* $c \in D'$ *be an arbitrary group element. It is easy to see that the relation* $\{(x,y) \mid x,y \in D',\ x+y = c\}$ *is injective.*

R is an example of a relation which is invariant under an affine operation. Such relations have previously been studied in relation to the MAX ONES problem in [37, 38]. We will give some additional results for such constraints in section 6.3. With the terminology used in [37, 38], R is said to be *width-2 affine*. The relations which can be expressed as the set of solutions to an equation with two variables over an Abelian group are exactly the width-2 affine relations, so the injective relations are a superset of the width-2 affine relations.

To see that W-MAX SOL(Γ) is in **PO** for every finite constraint language $\Gamma \subseteq \langle I^D \rangle$, it is sufficient to prove that W-MAX SOL(I^D) is in **PO** by Theorem 3.3. Given an instance of W-MAX SOL(I^D), consider the graph having the variables as vertices and edges between the vertices/variables occurring together in the same constraint. Each connected component of this graph represents an independent subproblem that can be solved separately. If a value is assigned to a variable/vertex, all variables/vertices in the same component will be forced to take a value by propagating this assignment. Hence, each connected component has at most $|D|$ different solutions (that can be easily enumerated), and an optimal one can be found in polynomial time.

5.2. Generalized max-closed relations. We begin by giving the following basic definition.

DEFINITION 5.3. *A constraint language Γ over a domain $D \subset \mathbb{N}$ is* generalized max-closed *if and only if there exists a binary operation $f \in Pol(\Gamma)$ such that for all $a, b \in D$,*

1. *if $a \neq b$ and $f(a,b) \leq \min(a,b)$, then $f(b,a) > \max(a,b)$; and*
2. *$f(a,a) \geq a$.*

In the conference version of this article [35], the definition of generalized max-closed constraint languages was slightly more restrictive. The following two examples will clarify the definition above.

EXAMPLE 5.4. *Assume that the domain D is $\{0,1,2,3\}$. As an example of a generalized* max*-closed relation consider*

$$R = \{(0,0),(1,0),(0,2),(1,2)\}.$$

R is invariant under max *and is therefore generalized* max*-closed since* max *satisfies the properties of Definition* 5.3*. Now, consider the relation Q defined as*

$$Q = \{(0,1),(1,0),(2,1),(2,2),(2,3)\}.$$

Q is not invariant under max *because*

$$\max((0,1),(1,0)) = (\max(0,1),\max(1,0)) = (1,1) \notin Q.$$

Let the operation $\circ : D^2 \to D$ be defined by the following Cayley table:[2]

$\circ$	0	1	2	3
0	0	2	2	3
1	2	1	2	2
2	2	2	2	3
3	3	2	3	3

Now, it is easy to verify that $Inv(\circ)$ is a set of generalized max*-closed relations and that $Q \in Inv(\circ)$.*

[2]Note that we write $x \circ y$ instead of $\circ(x,y)$.

EXAMPLE 5.5. *Consider the relations R_1 and R_2 defined as*

$$R_1 = \{(1,1,1),(1,0,0),(0,0,1),(1,0,1)\}$$

and $R_2 = R_1 \setminus \{(1,1,1)\}$. The relation R_1 is 1*-valid because the all-*1 *tuple is in R_1, i.e., $(1,1,1) \in R_1$. R_2, on the other hand, is not* 1*-valid but is weakly positive*[3] *because it is invariant under* max*. Note that both R_1 and R_2 are generalized* max*-closed since R_1 is invariant under $f(x,y) = 1$ and R_2 is invariant under $f(x,y) = \max(x,y)$. It is in fact the case that every weakly positive relation is invariant under* max *(more is true in the Boolean domain: a relation is weakly positive if and only if it is invariant under* max*), so the* 1*-valid and weakly positive relations are subsets of the generalized* max*-closed relations.*

The tractability of generalized max-closed constraint languages crucially depends on the following lemma.

LEMMA 5.6. *If Γ is generalized* max*-closed, then all relations*

$$R = \{(d_{11}, d_{12}, \ldots, d_{1m}), \ldots, (d_{t1}, d_{t2}, \ldots, d_{tm})\}$$

in Γ have the property that the tuple

$$\boldsymbol{t_{\max}} = (\max\{d_{11}, \ldots, d_{t1}\}, \ldots, \max\{d_{1m}, \ldots, d_{tm}\})$$

is in R, too.

Proof. Assume that there is a relation R in Γ such that the tuple

$$\boldsymbol{t_{\max}} = (\max\{d_{11}, \ldots, d_{t1}\}, \ldots, \max\{d_{1m}, \ldots, d_{tm}\})$$

is not in R. Define the distance between two tuples to be the number of coordinates where they disagree (i.e., the Hamming distance). Let $\boldsymbol{a}$ be a tuple in R with minimal distance from $\boldsymbol{t_{\max}}$ and let I denote the set of coordinates where $\boldsymbol{a}$ agrees with $\boldsymbol{t_{\max}}$. By the assumption that $\boldsymbol{t_{\max}}$ is not in R, we know that the distance between $\boldsymbol{a}$ and $\boldsymbol{t_{\max}}$ is at least 1. Hence, without loss of generality, assume that $\boldsymbol{a}[1] \neq \boldsymbol{t_{\max}}[1]$ and that $\boldsymbol{a}[1]$ is maximal for all tuples in R agreeing with $\boldsymbol{t_{\max}}$ on the coordinates in I. Let $\boldsymbol{b}$ be a tuple in R such that $\boldsymbol{b}[1] = \boldsymbol{t_{\max}}[1]$.

Since Γ is generalized max-closed, there exists an operation $f \in Pol(\Gamma)$ such that for all $a, b \in D$ $(a \neq b)$, it holds that $f(a,b) > \max(a,b)$ whenever $f(b,a) \leq \min(a,b)$. Furthermore, for all $a \in D$ it holds that $f(a,a) \geq a$. Now consider the tuple $\boldsymbol{x^n}$ $(n = |D|)$ defined as follows: $\boldsymbol{x^1} = f(\boldsymbol{a}, \boldsymbol{b})$ and

$$\boldsymbol{x^{i+1}} = \begin{cases} f(\boldsymbol{x^i}, \boldsymbol{a}) & \text{if } f(\boldsymbol{x^i}[1], \boldsymbol{a}[1]) > \boldsymbol{a}[1], \\ f(\boldsymbol{a}, \boldsymbol{x^i}) & \text{otherwise.} \end{cases}$$

We begin by proving that $\boldsymbol{x^n}$ agrees with $\boldsymbol{a}$ on all coordinates in I. Let $\boldsymbol{z}$ be an arbitrary tuple in R. Note that for each $i \in I$ such that $\boldsymbol{z}[i] \neq \boldsymbol{a}[i]$, it is the case that $f(\boldsymbol{a}[i], \boldsymbol{z}[i]) \leq \min(\boldsymbol{a}[i], \boldsymbol{z}[i])$ implies that $f(\boldsymbol{z}[i], \boldsymbol{a}[i]) > \max(\boldsymbol{a}[i], \boldsymbol{z}[i])$. Hence, as $\boldsymbol{a}[i] = \boldsymbol{t_{\max}}[i]$, we cannot have that $f(\boldsymbol{a}[i], \boldsymbol{z}[i]) \leq \min(\boldsymbol{a}[i], \boldsymbol{z}[i])$. So, for each $\boldsymbol{z} \in R$ and $i \in I$, we must have $f(\boldsymbol{a}[i], \boldsymbol{z}[i]) > \min(\boldsymbol{a}[i], \boldsymbol{z}[i])$ whenever $\boldsymbol{a}[i] \neq \boldsymbol{z}[i]$. By an analogous argument, it follows that for each $\boldsymbol{z} \in R$ and $i \in I$ we must have $f(\boldsymbol{z}[i], \boldsymbol{a}[i]) > \min(\boldsymbol{a}[i], \boldsymbol{z}[i])$ whenever $\boldsymbol{a}[i] \neq \boldsymbol{z}[i]$.

[3] A relation is weakly positive if it can be expressed as a formula in conjunctive normal form having at most one negated variable in each clause.

This together with the fact that $f(d,d) \geq d$ for all $d \in D$ and that $\boldsymbol{a}$ agrees with $\boldsymbol{t_{\max}}$ on I implies that $f(\boldsymbol{a}, \boldsymbol{x^n})$ agrees with $\boldsymbol{a}$ on I.

We now show that $\boldsymbol{x^n}[1] > \boldsymbol{a}[1]$. This follows from essentially the same argument as above. First note that $f(\boldsymbol{a}[1], \boldsymbol{b}[1]) = \boldsymbol{x^1}[1] > \boldsymbol{a}[1]$. If $f(\boldsymbol{a}[1], \boldsymbol{b}[1]) \leq \min(\boldsymbol{a}[1], \boldsymbol{b}[1])$, then $f(\boldsymbol{b}[1], \boldsymbol{a}[1]) > \boldsymbol{b}[1]$ which is not possible since $\boldsymbol{b}[1] = \boldsymbol{t_{\max}}[1]$. Hence, we must have $f(\boldsymbol{a}[1], \boldsymbol{b}[1]) = \boldsymbol{x^1}[1] > \min(\boldsymbol{a}[1], \boldsymbol{b}[1])$. Now, by the definition of $\boldsymbol{x}^{i+1}$, it follows that if $\boldsymbol{x^i}[1] > \boldsymbol{a}[1]$, then $\boldsymbol{x}^{i+1}[1] > \min(\boldsymbol{x^i}[1], \boldsymbol{a}[1]) = \boldsymbol{a}[1]$ (just note that at least one of $f(\boldsymbol{x^i}[1], \boldsymbol{a}[1])$ and $f(\boldsymbol{a}[1], \boldsymbol{x^i}[1])$ is strictly larger than $\min(\boldsymbol{x^i}[1], \boldsymbol{a}[1]) = \boldsymbol{a}[1]$). Hence, it follows by induction that $\boldsymbol{x^n}[1] > \boldsymbol{a}[1]$.

Thus, we have a contradiction with the fact that $a[1]$ is maximal for all tuples in R agreeing with $\boldsymbol{t_{\max}}$ on the coordinates in I. Hence, our assumption was wrong and $\boldsymbol{t_{\max}}$ is in R. □

The algorithm for solving W-Max Sol(Γ) when Γ is generalized max-closed is a simple consistency-based algorithm. The algorithm, which is based on pairwise consistency, closely follows the algorithm for Csps over max-closed constraint languages from [33].

We first need to introduce some terminology.

Definition 5.7. *Given a constraint $C_i = (s_i, R_i)$ and an (ordered) subset s_i' of the variables in s_i, where $(i_1, i_2, \ldots, i_k)$ are the indices in s_i of the elements in s_i', the* projection *of C_i onto the variables in s_i' is denoted by $\pi_{s_i'} C_i$ and defined as $\pi_{s_i'} C_i = C_i' = (s_i', R_i')$, where R_i' is the relation $\{(\boldsymbol{a}[i_1], \boldsymbol{a}[i_2], \ldots, \boldsymbol{a}[i_k]) \mid \boldsymbol{a} \in R_i\}$.*

Definition 5.8. *For any pair of constraints $C_i = (s_i, R_i)$, $C_j = (s_j, R_j)$, the* join *of C_i and C_j, denoted $C_i \bowtie C_j$, is the constraint on $s_i \cup s_j$ containing all tuples $\boldsymbol{t}$ such that $\pi_{s_i}\{\boldsymbol{t}\} \in R_i$ and $\pi_{s_j}\{\boldsymbol{t}\} \in R_j$.*

Definition 5.9 (see [30]). *An instance of a constraint satisfaction problem $I = (V, D, C)$ is* pairwise consistent *if and only if for any pair of constraints $C_i = (s_i, R_i)$, $C_j = (s_j, R_j)$ in C, it holds that the constraint resulting from projecting C_i onto the variables in $s_i \cap s_j$ equals the constraint resulting from projecting C_j onto the variables in $s_i \cap s_j$; i.e., $\pi_{s_i \cap s_j} C_i = \pi_{s_i \cap s_j} C_j$.*

We are now ready to prove the tractability of generalized max-closed constraint languages.

Theorem 5.10. *If Γ is generalized* max*-closed, then* W-Max Sol(Γ) *is in* **PO**.

Proof. Since $Inv(f)$ is a relational clone, constraints built over $Inv(f)$ are invariant when taking joins and projections [31, Lemma 2.8] (i.e., the underlying relations are still invariant under f). It was proved in [30] that any set of constraints can be reduced to an equivalent set of pairwise consistent constraints in polynomial time. Since the set of pairwise consistent constraints can be obtained by repeated application of the join and projection operations, the underlying relations in the resulting constraints are still in $Inv(f)$.

Hence, given an instance $I = (V, D, C, w)$ of W-Max Sol($Inv(f)$), we can assume that the constraints in C are pairwise consistent. We prove that for pairwise consistent C, either C has a constraint with a constraint relation that does not contain any tuples (i.e., no assignment satisfies the constraint and there is no solution) or we can find the optimal solution in polynomial time.

Assume that C has no empty constraints. For each variable x_i, let d_i be the maximum value allowed for that variable by some constraint C_j (where x_i is in the constraint scope of C_j). We will prove that $(d_1, \ldots, d_n)$ is an optimal solution to I. Obviously, if $(d_1, \ldots, d_n)$ is a solution to I, then it is the optimal solution. Hence, it is sufficient to prove that $(d_1, \ldots, d_n)$ is a solution to I.

Assume, with the aim of reaching a contradiction, that $(d_1, \ldots, d_n)$ is not a solution to I. Then, there exists a constraint C_j in C not satisfied by $(d_1, \ldots, d_n)$. Since the constraint relation corresponding to C_j is generalized max-closed, there exists a variable x_i in the constraint scope of C_j such that C_j has no solution where d_i is assigned to x_i. Note that it is essential that C_j is generalized max-closed to rule out the possibility that there exist two variables x_i and x_j in the constraint scope of C_j such that C_j has two solutions t, u where $t(x_i) = d_i$ and $u(x_j) = d_j$, but C_j has no solution s where $s(x_i) = d_i$ and $s(x_j) = d_j$. We know that there exists a constraint C_i in C having x_i in its constraint scope and d_i an allowed value for x_i. This contradicts the fact that C is pairwise consistent. Thus, $(d_1, \ldots, d_n)$ is a solution to I. □

6. Maximal constraint languages. A *maximal constraint language* Γ is a constraint language such that $\langle\Gamma\rangle \subset R_D$, and if $R \notin \langle\Gamma\rangle$, then $\langle\Gamma \cup \{R\}\rangle = R_D$. That is, the maximal constraint languages are the largest constraint languages that are not able to express all finitary relations over D. This implies, among other things, that there exists an operation f such that $\langle\Gamma\rangle = Inv(f)$ whenever Γ is a maximal constraint language [45]. Relational clones $\langle\Gamma\rangle$ such that Γ is a maximal constraint language are called maximal relational clones. The complexity of the Csp(Γ) problem for all maximal constraint languages on domains $|D| \leq 3$ was determined in [14]. Moreover, it was shown in [14] that the only case that remained to be classified in order to extend the classification to all maximal constraint languages over a finite domain was the case where $\langle\Gamma\rangle = Inv(f)$ for binary commutative idempotent operations f. These constraint languages were finally classified by Bulatov in [8].

Theorem 6.1 (see [8, 14]). *Let Γ be a maximal constraint language on an arbitrary finite domain D. Then,* Csp(Γ) *is in* **P** *if $\langle\Gamma\rangle = Inv(f)$ where f is a constant operation, a majority operation, a binary commutative idempotent operation, or an affine operation. Otherwise,* Csp(Γ) *is* **NP***-complete.*

In this section, we classify the approximability of W-Max Sol(Γ) for all maximal constraint languages Γ over $|D| \leq 4$. Moreover, we prove that the only cases that remain to be classified, in order to extend the classification to all maximal constraint languages over finite domains, are constraint languages Γ such that $\langle\Gamma\rangle$ is invariant under a binary commutative idempotent operation. We also prove that if a certain conjecture regarding minimal clones generated by binary operations, due to Szczepara [47], holds, then our classification can be extended to also capture these last cases.

Theorem 6.2. *Let Γ be maximal constraint language on a finite domain D, with $|D| \leq 4$, and $\langle\Gamma\rangle = Inv(f)$.*

1. *If Γ is generalized* max*-closed or an injective constraint language, then* W-Max Sol(Γ) *is in* **PO***;*
2. *else if f is an affine operation, a constant operation different from the constant 0 operation, or a binary commutative idempotent operation satisfying $f(0, b) > 0$ for all $b \in D \setminus \{0\}$ (assuming $0 \in D$), or if $0 \notin D$ and f is a binary commutative idempotent operation or a majority operation, then* W-Max Sol(Γ) *is* **APX***-complete;*
3. *else if f is a binary commutative idempotent operation or a majority operation, then* W-Max Sol(Γ) *is* **poly-APX***-complete;*
4. *else if f is the constant 0 operation, then finding a solution with nonzero measure is* **NP***-hard;*
5. *otherwise, finding a feasible solution is* **NP***-hard.*

Moreover, if Conjecture 131 *from* [47] *holds, then the results above hold for arbitrary finite domains D.*

The proof of the preceding theorem consists of a careful analysis of the approximability of W-MAX SOL(Γ) for all maximal constraint languages Γ such that $\langle\Gamma\rangle = Inv(f)$, where f is one of the types of operations in Theorem 6.1. These results are presented below.

6.1. Constant operation. We begin by considering maximal constraint languages that are invariant under constant operations. Given an instance $I = (V, D, C)$ of a CSP problem, we define the *constraint graph* of I to be $G = (V, E)$, where $\{v, v'\} \in E$ if there is at least one constraint $c \in C$ which has both v and v' in its constraint scope.

LEMMA 6.3. *Let $d^* = \max(D)$ and let C_d be a constraint language such that $\langle C_d\rangle = Inv(f_d)$, where $f_d : D \to D$ satisfies $f_d(x) = d$ for all $x \in D$. Then,* W-MAX SOL(C_{d^*}) *is in* **PO**, W-MAX SOL(C_d) *is* **APX**-*complete if $d \in D\backslash\{d^*, 0\}$, and it is* **NP**-*hard to find a solution with nonzero measure for* W-MAX SOL(C_0).

Proof. The tractability of W-MAX SOL(C_{d^*}) is trivial, since the optimum solution is obtained by assigning d^* to all variables.

For the **APX**-hardness of W-MAX SOL(C_d) ($d \in D\backslash\{d^*, 0\}$), it is sufficient to note that $\{(d, d), (d, d^*), (d^*, d)\}$ is in $\langle C_d\rangle$, and since $0 < d < d^*$ it follows from Lemma 4.3 that W-MAX SOL(C_d) is **APX**-hard. It is easy to realize that W-MAX SOL(C_d) is in **APX**, since we can obtain a $\frac{d^*}{d}$-approximate solution by assigning the value d to all variables.

The fact that it is **NP**-hard to find a solution with nonzero measure for W-MAX SOL(C_0) over the Boolean domain $\{0, 1\}$ is proved in [37, Lemma 6.23]. To prove that it is **NP**-hard to find a solution with nonzero measure for W-MAX SOL(C_0) over a domain D of size ≥ 3, we give a reduction from the well-known **NP**-complete problem POSITIVE-1-IN-3-SAT [26], i.e., CSP($\{R\}$) with $R = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. It is easy to see that POSITIVE-1-IN-3-SAT restricted to instances where the constraint graph is connected is still **NP**-complete.

Now, let $R' = \{(b, a, a), (a, b, a), (a, a, b), (0, 0, 0)\}$, where $0 < a < b$ and $a, b, 0 \in D$. For an instance $I = (V, D, C)$ of CSP($\{R\}$) where the constraint graph of I is connected, create an instance I' of W-MAX SOL($\{R'\}$) by replacing all occurrences of R by R' and giving all variables weight 1. Since the constraint graph is connected, I has a solution if and only if I' has a solution with nonzero measure, and since $R' \in C_0$, it follows that it is **NP**-hard to find a solution with nonzero measure for W-MAX SOL(C_0). □

6.2. Majority operation. Maximal constraint languages based on majority operations are fairly easy to analyze due to the results in section 4.

LEMMA 6.4. *Let m be an arbitrary majority operation on D. Then,* W-MAX SOL($Inv(m)$) *is* **APX**-*complete if $0 \notin D$ and* **poly-APX**-*complete if $0 \in D$.*

Proof. Arbitrarily choose elements $a, b \in D$ such that $a < b$. Then, it is easy to see that $\{(a, a), (a, b), (b, a)\}$ is in $Inv(m)$. Thus, by Proposition 4.1 and Lemmas 4.3 and 4.4, it follows that W-MAX SOL($Inv(m)$) is **APX**-complete or **poly-APX**-complete depending on whether 0 is in D or not. □

6.3. Affine operation. We split the proof of this result into two parts. The first part, section 6.3.1, contains the hardness result: for every affine operation $a : D^3 \to D$, W-MAX SOL($Inv(a)$) is **APX**-hard. The proof is based on a reduction from MAX-p-CUT which is a well-known **APX**-complete problem [2]. Membership in **APX** is proved in section 6.3.2 by presenting an approximation algorithm with constant performance ratio.

We will denote the affine operation on the group G by a_G; i.e., if $G = (D, +_G, -_G)$, then $a_G(x, y, z) = x -_G y +_G z$.

6.3.1. APX-hardness. In this section we will prove Theorem 6.11, which states that relations invariant under an affine operation give rise to **APX**-hard W-Max Sol-problems. We need a number of lemmas before we can prove this result. We begin by giving an L-reduction from Max-p-Cut to W-Max Sol Eqn$(\mathbb{Z}_p, g)$, where p is prime. Max-p-Cut and W-Max Sol Eqn are defined as follows.

Definition 6.5 (see [2]). Max-p-Cut *is an optimization problem with the following:*

Instance: *A graph $G = (V, E)$.*
Solution: *A partition of V into p disjoint sets $C_1, C_2, \ldots, C_p$.*
Measure: *The number of edges between the disjoint sets, i.e.,*

$$\sum_{i=1}^{p-1} \sum_{j=i+1}^{p} |\{\{v, v'\} \in E \mid v \in C_i \text{ and } v' \in C_j\}|.$$

Definition 6.6 (see [38]). *Let $G = (D, +_G, -_G)$ be a group and $g : D \to \mathbb{N}$ a function.* W-Max Sol Eqn(G, g) *is an optimization problem with the following:*

Instance: *A triple (V, E, w), where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of variables, E is a set of equations of the form $u_1 +_G \ldots +_G u_k = 0_G$, where each u_i is either a variable (e.g., "v_4"), an inverted variable (e.g., "$-_G v_7$"), or a group constant, and w is a weight function $w : V \to \mathbb{N}$.*
Solution: *An assignment $f : V \to D$ to the variables such that all equations are satisfied.*
Measure: $\sum_{v \in V} w(v) g(f(v))$.

We do not require the group G to be Abelian in the definition of W-Max Sol Eqn, but this will always be the case in this article. Note that the function g and the group G are not parts of the input so W-Max Sol Eqn(G, g) is a problem parameterized by G and g. We refer the reader to [38] for more information on the problem W-Max Sol Eqn$(\mathbb{Z}_p, g)$.

The following lemma follows from the proof of Proposition 2.3 in [20].

Lemma 6.7. *For any instance $I = (V, E)$ of* Max-p-Cut, *we have* opt$(I) \geq |E|(1 - 1/p)$.

We can now prove the **APX**-hardness of W-Max Sol Eqn.

Lemma 6.8. *For every prime p and every nonconstant function $g : \mathbb{Z}_p \to \mathbb{N}$,* W-Max Sol Eqn$(\mathbb{Z}_p, g)$ *is* **APX**-*hard.*

Proof. Given an instance $I = (V, E)$ of Max-p-Cut, we construct an instance $F(I)$ of W-Max Sol Eqn$(\mathbb{Z}_p, g)$, where, for every vertex $v_i \in V$, we create a variable x_i and give it weight 0 and, for every edge $\{v_i, v_j\} \in E$, we create p variables $z_{ij}^{(k)}$ for $k = 0, \ldots, p-1$ and give them weight 1. Let $g_{\min}$ denote an element in $\mathbb{Z}_p$ that minimizes g, i.e.,

$$\min_{x \in \mathbb{Z}_p} g(x) = g(g_{\min}),$$

and let g_s denote the sum

$$\sum_{k=0}^{p-1} g(k).$$

For every edge $\{v_i, v_j\} \in E$, we introduce the equations

$$k(x_i - x_j) + g_{\min} = z_{ij}^{(k)}$$

for $k = 0, \ldots, p-1$. If $x_i = x_j$, then the p equations for the edge $\{v_i, v_j\}$ will contribute $pg(g_{\min})$ to the measure of the solution. On the other hand, if $x_i \neq x_j$, then the p equations will contribute g_s to the measure.

Given a solution s' to $F(I)$, we can construct a solution s to I in the following way: let $s(v_i) = s'(x_i)$; i.e., for every vertex v_i, place this vertex in partition $s'(x_i)$. The measures of the solutions s and s' are related to each other by the equality

$$m'(F(I), s') = |E| \cdot p \cdot g(g_{\min}) + (g_s - p \cdot g(g_{\min})) \cdot m(I, s). \tag{6.1}$$

From (6.1), we get

$$\textsc{opt}(F(I)) = |E| \cdot p \cdot g(g_{\min}) + (g_s - p \cdot g(g_{\min})) \cdot \textsc{opt}(I) \tag{6.2}$$

and from Lemma 6.7, we have that $\textsc{opt}(I) \geq |E| \cdot (1 - 1/p)$, which implies $\textsc{opt}(I) \geq |E|/p$. By combining this with (6.2), we can conclude that

$$\begin{aligned}\textsc{opt}(F(I)) &= \textsc{opt}(I)\left(\frac{|E| \cdot p \cdot g(g_{\min})}{\textsc{opt}(I)} + g_s - p \cdot g(g_{\min})\right) \\ &\leq \textsc{opt}(I)\Big(p^2 \cdot g(g_{\min}) + g_s - p \cdot g(g_{\min})\Big).\end{aligned}$$

Hence, $\beta = p(p-1) \cdot g(g_{min}) + g_s$ is an appropriate parameter for the L-reduction.

We will now deduce an appropriate γ-parameter for the L-reduction: from (6.1) and (6.2) we get

$$|\textsc{opt}(F(I)) - m'(F(I), s')| = (g_s - p \cdot g(g_{\min})) \cdot |\textsc{opt}(I) - m(I, s)|;$$

thus, $\gamma = 1/(g_s - p \cdot g(g_{\min}))$ is sufficient (γ is well defined because a nonconstant g implies $g_s > p \cdot g_{\min}$). □

We need two lemmas before we can prove the **APX**-hardness of affine relations. Let $v_1, v_2, \ldots, v_k$ be a collection of variables, $G = (D, +_G, -_G)$ an Abelian group, and E an equation of the form $x_1 +_G x_2 +_G \ldots +_G x_n = c$, where each x_i is a (possibly inverted) variable and $c \in D$. Note that each variable may occur several times in E. The set of all solutions to E may be seen as a k-ary relation R_E on D^k. The following two lemmas are well known [32].

LEMMA 6.9. *The relation R_E is invariant under a_G.*

LEMMA 6.10. *If P is a coset of G, then P is invariant under a_G.*

We now have all results needed to prove the main theorem of this section.

THEOREM 6.11. W-MAX SOL($Inv(a_G)$) *is* **APX***-hard for every affine operation a_G.*

Proof. We show that there exists a prime p and a nonconstant function $h : \mathbb{Z}_p \to \mathbb{N}$ such that W-MAX SOL EQN($\mathbb{Z}_p, h$) can be S-reduced to W-MAX SOL($Inv(a_G)$). The result will then follow from Lemma 6.8.

Let p be a prime such that $\mathbb{Z}_p$ is isomorphic to a subgroup H of G. We know that such a p always exists by the fundamental theorem of finitely generated Abelian groups. Let α be the isomorphism which maps elements of $\mathbb{Z}_p$ to elements of H and let $h = \alpha$. (Note that $H \subset \mathbb{N}$ since the domain is a subset of $\mathbb{N}$. Consequently, h may be viewed as a function from $\mathbb{Z}_p$ to $\mathbb{N}$.)

Let $I = (V, E, w)$ be an instance of W-MAX SOL EQN$(\mathbb{Z}_p, h)$ with variables $V = \{v_1, \ldots, v_n\}$ and equations $E = \{e_1, \ldots, e_m\}$. We will construct an instance $I' = (V, D, C, w)$ of W-MAX SOL$(Inv(a_G))$.

Let U be the unary relation for which $x \in U \iff x \in H$; this relation is in $Inv(a_G)$ by Lemma 6.10. For every equation $E_i \in E$, there is a corresponding pair (s_i, R_i), where s_i is a list of variables and R_i is a relation in $Inv(a_G)$ such that the set of solutions to E_i contains exactly the tuples which satisfy (s_i, R_i) by Lemma 6.9. We can now construct C:

$$C = \{(v_i, U) \mid 1 \leq i \leq n\} \cup \{(s_i, R_i) \mid 1 \leq i \leq m\}.$$

It is easy to see that I and I' are essentially the same in the sense that every feasible solution to I is also a feasible solution to I', and that they have the same measure. The converse is also true: every feasible solution to I' is also a feasible solution to I. Hence, we have given an S-reduction from W-MAX SOL EQN$(\mathbb{Z}_p, h)$ to W-MAX SOL$(Inv(a_G))$. As h is not constant (it is in fact injective), it follows from Lemma 6.8 that W-MAX SOL EQN$(\mathbb{Z}_p, h)$ is **APX**-hard. This S-reduction implies that W-MAX SOL$(Inv(a_G))$ is **APX**-hard. □

6.3.2. Membership in APX. We will now prove that relations that are invariant under an affine operation give rise to problems which are in **APX**. It has been proved that a relation which is invariant under an affine operation is a coset of a subgroup of some Abelian group [32]. We will give an approximation algorithm for the more general problem when the relations are cosets of subgroups of a finite group.

Our algorithm is based on an algorithm by Bulatov and Dalmau [12] for deciding the satisfiability of *Mal'tsev constraints*. A *Mal'tsev operation* is a ternary operation m such that $m(x, y, y) = m(y, y, x) = x$ for all $x, y \in D$. If a constraint language Γ is invariant under a Mal'tsev operation, then Bulatov and Dalmau have proved that CSP(Γ) is solvable in polynomial time. We note that every affine operation is a Mal'tsev operation since $x -_G y +_G y = x$ and $y -_G y +_G x = x$.

Let G^k denote the direct product of k copies of G. We are now ready to prove containment in **APX**.

THEOREM 6.12. *Let $G = (D; +_G, -_G)$ be a finite group and let Γ be a constraint language such that for each $R \in \Gamma$ there is an integer k such that R is a coset of some subgroup of G^k. Then* W-MAX SOL(Γ) *is in* **APX**.

Proof. Let $I = (V, D, C, w)$ be an arbitrary instance of W-MAX SOL(Γ), where $V = \{v_1, \ldots, v_n\}$. Feasible solutions to our optimization problem can be viewed as certain elements in $H = G^n$. Each constraint $C_i \in C$ defines a coset $a_i +_G J_i$ of H with representative $a_i \in H$, for some subgroup J_i of H. The set of solutions to the problem is the intersection of all those cosets. Thus, $S = \bigcap_{i=1}^{|C|} a_i +_G J_i$ denotes the set of all solutions.

Since Γ is invariant under the affine operation $a_G(x, y, z) = x -_G y +_G z$ and a_G is a Mal'tsev operation, we can decide if there are any solutions to I in polynomial time [12]. Clearly, S is empty if and only if there are no solutions. It is well known that an intersection of a set of cosets is either empty or a coset so if $S \neq \emptyset$, then S is a coset.

We will represent the elements of G^n by vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$ where each x_i is an element of G. For any instance I, we define $\mathcal{R}(I)$ to be the random variable which is uniformly distributed over the set of solutions to I. Let V_i denote the random variable which corresponds to the value which will be assigned to v_i by $\mathcal{R}(I)$. We claim that V_i is uniformly distributed over some subset of G. As S is a coset, there

are a subgroup S' of G^n and an element $\boldsymbol{s} \in S$ such that $S = \boldsymbol{s} +_G S'$. Assume, for the sake of contradiction, that V_i is not uniformly distributed. Then, there are group elements $a, b \in G$ such that the sets

$$X_a = \{\boldsymbol{x} \in S' \mid x_i = a\} \quad \text{and} \quad X_b = \{\boldsymbol{x} \in S' \mid x_i = b\}$$

have different cardinality. Assume that $|X_a| > |X_b|$. Arbitrarily pick $\boldsymbol{y} \in X_a, \boldsymbol{z} \in X_b$ and construct the set $Z = \{\boldsymbol{x} -_G \boldsymbol{y} +_G \boldsymbol{z} \mid \boldsymbol{x} \in X_a\}$. From the definition of Z and the fact that S' is invariant under a_G, it follows that $Z \subseteq S'$. For each $\boldsymbol{x} \in Z$ we have $x_1 = b$; hence $Z \subseteq X_b$. However, we also have $|Z| = |X_a|$, which contradicts the assumption that $|X_a| > |X_b|$. We conclude that this cannot hold, and thus V_i is uniformly distributed. Hence, for each $1 \leq i \leq n$, V_i is uniformly distributed.

Now, let A denote the set of indices such that for every $i \in A$, $\Pr[V_i = c_i] = 1$ for some $c_i \in G$. That is, A contains the indices of the variables V_i which are constant in every feasible solution. Let B contain the indices for the variables which are not constant in every solution, i.e., $B = [n] \setminus A$.

Let $S^* = \sum_{i \in B} w(v_i) \max(D) + \sum_{i \in A} w(v_i) c_i$ and note that $S^* \geq \text{OPT}$. Furthermore, let

$$E_{\min} = \min_{X \subseteq G, |X| > 1} \frac{1}{|X|} \cdot \sum_{x \in X} x$$

and note that $\max(D) > E_{\min} > 0$.

The expected value of the measure of $\mathcal{R}(I)$ can now be estimated as

$$\begin{aligned} \text{(6.3)} \quad \mathrm{E}\left[\sum_{i=1}^{n} w(v_i) V_i\right] &= \sum_{i \in A} w(v_i) \mathrm{E}[V_i] + \sum_{i \in B} w(v_i) \mathrm{E}[V_i] \\ &\geq \sum_{i \in A} w(v_i) c_i + E_{\min} \sum_{i \in B} w(v_i) \geq \frac{E_{\min}}{\max(D)} S^* \geq \frac{E_{\min}}{\max(D)} \text{OPT}. \end{aligned}$$

Since $E_{\min} / \max(D) > 0$, it follows that the measure of $\mathcal{R}(I)$ has, in expectation, a constant performance ratio. We will denote $\frac{E_{\min}}{\max(D)} \cdot \text{OPT}$ by E.

To get a deterministic polynomial-time algorithm, note that for any instance I we can use the algorithm by Bulatov and Dalmau [12] to compute the two sums in (6.3) in polynomial time. Hence, we can compute the expected measure of $\mathcal{R}(I)$ in polynomial time. Our algorithm is presented in Figure 6.1.

We claim that the following loop invariant holds in the algorithm: before line 4 is executed it is always the case that the expected measure of $\mathcal{R}(I_i)$ is at least E.

We first prove the correctness of the algorithm assuming that the loop invariant holds. From the loop invariant it follows that the expected measure of $\mathcal{R}(I_{|V|+1})$ is at least E. In $I_{|V|+1}$ there is, for each variable $v_i \in V$, a constraint of the form $v_i = x_i$; therefore there is only one solution to $I_{|V|+1}$. This solution will be returned by the algorithm.

We now prove that the loop invariant holds. The first time line 4 is reached the expected performance ratio of $\mathcal{R}(I_1)$ is at least E, per the calculations above. Now assume that the loop invariant holds in iteration $i = k \leq |V|$; we will prove that it also holds in iteration $i = k + 1$. Since the performance ratio of $\mathcal{R}(I_k)$ is at least E, there must be some value $x \in D$ such that when v_i is fixed to x, the performance ratio of $\mathcal{R}(I_{k+1})$ is at least E. This element will be found by the algorithm as it maximizes the expected performance ratio $\mathcal{R}(I_{k+1})$. Hence, the loop invariant holds for $i = k + 1$. ∎

Input: An instance $I = (V, D, C, w)$ of W-Max Sol(Γ).
Output: A solution with performance ratio at least $E_{\min}/\max(D)$, or "no solution" if there are no solutions.

1. Return "no solution" if there are no solutions (use Bulatov and Dalmau's algorithm to check this).
2. Let $I_1 = I$.
3. For each i from 1 to $|V|$:
4. For each $x \in D$:
5. Let $I_i = I$ and add the constraint $v_i = x$ to I_i.
6. If there is no solution to I_i, then go to 8.
7. Compute the expected measure of $\mathcal{R}(I_i)$.
8. Remove the constraint $v_i = x$ from I_i.
9. Let $x_i \in D$ be the value which maximizes the expected measure of $\mathcal{R}(I_i)$ in the computations in 4–8. Create a new instance, I_{i+1}, which is identical to I_i except for the addition of the constraint $v_i = x_i$.
10. Return the unique solution to $I_{|V|+1}$.

Fig. 6.1. *The algorithm in Theorem* 6.12.

6.4. Binary commutative idempotent operation. We now investigate the complexity of W-Max Sol(Γ) for maximal constraint languages Γ satisfying $\langle\Gamma\rangle = Inv(f)$ where f is a binary commutative idempotent operation.

Let $(F; +_F, -_F, \cdot_F, 1_F)$ be a finite field of prime order p, where $+_F$, $-_F$, $\cdot_F$, and 1_F denote addition, subtraction, multiplication, and multiplicative identity, respectively (we refrain from defining a notation for multiplicative inverses, as we do not need it). Furthermore, let z_F be the unique element in F such that $z_F + z_F = 1_F$. Note that for $F = \mathbb{Z}_p$ we get $1_F = 1$ and $z_F = \frac{p+1}{2}$.

Let $\mathcal{A}$ denote the set of operations $f(x, y) = z_F \cdot_F (x +_F y)$, where F is a finite field of prime order $p = |D|$ and $p > 2$. The proof will be partitioned into two main cases due to the following result.

Lemma 6.13 (see [14, 49]). *If $Inv(f)$ is a maximal relational clone and f is a binary idempotent operation, then either*

1. *$Inv(f) = Inv(g)$, where $g \in \mathcal{A}$, or*
2. *$B \in Inv(f)$ for some two-element $B \subseteq D$.*

The classification result is given in the next lemma together with a proof outline. Full proofs concerning the case when $Inv(f) = Inv(g)$ and $g \in \mathcal{A}$ can be found in section 6.4.1. In section 6.4.2 we give a complete characterization of the complexity for the second case for domains D such that $|D| \leq 4$. Finally, in section 6.4.3 we extend the classification to general domains under the assumption of a conjecture due to Szczepara (Conjecture 6.18).

Lemma 6.14. *Let f be a binary commutative idempotent operation on D such that $Inv(f)$ is a maximal relational clone, and let Γ be a constraint language such that $\langle\Gamma\rangle = Inv(f)$.*

- *If $Inv(f) = Inv(g)$ for some $g \in \mathcal{A}$, then* W-Max Sol(Γ) *is* **APX***-complete.*
- *Else if $|D| \leq 4$ and there exist $a, b \in D$ such that $a < b$ and $f(a, b) = a$, then let a_* be the minimal such element (according to $<$). Then*
  - W-Max Sol(Γ) *is* **poly-APX***-complete if $a_* = 0$, and*
  - **APX***-complete if $a_* > 0$.*
- *Otherwise, if $|D| \leq 4$, then* W-Max Sol(Γ) *is in* **PO**.

Proof. If $Inv(f) = Inv(g)$ and $g \in \mathcal{A}$, then the result follows from section 6.4.1.

If there exist $a, b \in D$ such that $a < b$ and $f(a, b) = a$, then we need to consider two cases depending on a_*. If $a_* = 0$, then W-MAX SOL(Γ) is **poly-APX**-hard by Lemma 4.4 and a member of **poly-APX** by Lemma 4.2 since CSP is in **P** [14]. If $a_* > 0$, then W-MAX SOL(Γ) is **APX**-complete by Lemma 6.25 in section 6.4.2.

Finally, if there do not exist any $a, b \in D$ such that $a < b$ and $f(a, b) = a$, then f acts as the max operation on every two-element $B \subseteq D$ such that $B \in Inv(f)$. Lemma 6.26 shows that f is a generalized max operation in this case, and W-MAX SOL(Γ) is in **PO** by Theorem 5.10. □

6.4.1. f is contained in $\mathcal{A}$. We will now prove that W-MAX SOL(Γ) is **APX**-complete whenever $f \in \mathcal{A}$ and $\langle\Gamma\rangle = Inv(f)$.

LEMMA 6.15. *Let $f(x, y) = z_F \cdot_F (x +_F y)$, where F is a finite field of prime order $p = |D| > 2$ and $Inv(f)$ is a maximal relational clone. Then,* W-MAX SOL(Γ) *is* **APX***-complete if $\langle\Gamma\rangle = Inv(f)$.*

Proof. We will give the proof for $F = \mathbb{Z}_p$ and after that we will argue that the proof can easily be adapted to the general case.

Let $q = \frac{p+1}{2}$ and f be the function $f(x, y) = q(x+y) \pmod p$. We will show that we can express $x - y + z$ through f.

Note that

$$\sum_{i=1}^{p-1} q^i = \frac{1 - q^p}{1 - q} - 1 = 0 \pmod p. \tag{6.4}$$

(The second equality follows from Fermat's little theorem: $a^{p-1} = 1 \pmod p$ for any prime p and integer a not divisible by p.) By using (6.4) and Fermat's little theorem again, we get

$$\sum_{i=1}^{p-2} q^i = -1 \pmod p. \tag{6.5}$$

We can now express $x - y + z$ as follows:

$$\begin{aligned}
&\underbrace{f(f(f(\dots f(f(f(}_{p-1 \text{ times}} x, z), y), y) \dots), y), y) \\
&\quad = q(q(q(\dots q(q(q(x + z) + y) + y) + \cdots) + y) + y) \\
&\quad = q^{p-1}x + q^{p-1}z + \sum_{i=1}^{p-2} q^i y \\
&\quad = x - y + z \pmod p,
\end{aligned}$$

where the final equality follows from (6.4), (6.5), and Fermat's little theorem.

As any finite field F of prime order is isomorphic to $\mathbb{Z}_p$, it is not hard to see that $x -_F y +_F z$ can be expressed through f for any such field. Since $Inv(f)$ is a maximal relational clone, $x -_F y +_F z$ can be expressed through f, and $x -_F y +_F z$ is not a projection, it follows that $Inv(f) = Inv(x -_F y +_F z)$. We now get containment in **APX** from Theorem 6.12 and **APX**-hardness from Theorem 6.11. □

6.4.2. f is not contained in $\mathcal{A}$. In the first part of this section we classify the complexity of W-MAX SOL(Γ) when $\langle\Gamma\rangle = Inv(f)$ for all 2-semilattice operations f. Recall that a 2-semilattice operation f is an operation satisfying the conditions $f(x,x) = x$, $f(x,y) = f(y,x)$, and $f(x, f(x,y)) = f(x,y)$. It is noted in [9] that binary operations f such that $Inv(f)$ is a maximal constraint language on $|D| \leq 4$ are either 2-semilattices or otherwise CSP(Γ) is **NP**-complete. Hence, we get a classification of the complexity of W-MAX SOL(Γ) when $\langle\Gamma\rangle = Inv(f)$ is a maximal constraint language over $|D| \leq 4$ and f is a binary operation.

The second result in this section is a complete complexity classification of W-MAX SOL(Γ) for maximal constraint languages Γ, such that $\langle\Gamma\rangle = Inv(f)$ where f is a binary operation, under the condition that Conjecture 131 from [47] holds.

LEMMA 6.16. *Let f be a 2-semilattice operation on D and let $\langle\Gamma\rangle = Inv(f)$. If there exist $a, b \in D$ such that $a < b$, $f(a,b) = a$, and $a^* > 0$, where a^* is the minimal element such that there is b^* with $f(a^*, b^*) = a^*$, then* W-MAX SOL(Γ) *is* **APX**-*complete.*

Proof. The **APX**-hardness part is clear. What remains is to show that the problem is in **APX**. We can assume, without loss of generality, that $a = a^*$ and $b = b^*$. We begin by proving that $U = D \setminus \{0\}$ is in $Inv(f)$. Assume that $f(a,b) = 0$ and $a, b > 0$; then $f(a, f(a,b)) = f(a,b) = 0$ and, consequently, $f(a,0) = 0$, contradicting the assumption that $a > 0$ was the minimal such element. Hence, $f(a,b) = 0$ if and only if $a = b = 0$. In particular U is in $Inv(f)$.

We continue with the actual proof of the lemma. Let $I = (V, D, C, w)$ be an arbitrary instance of W-MAX SOL(Γ). Define $V' \subseteq V$ such that

$$V' = \{v \in V \mid S(v) = 0 \text{ for every solution } S \text{ of } I\}.$$

We see that V' can be computed in polynomial time: a variable v is in V' if and only if the CSP instance $(V, D, C \cup \{((v), U)\})$ is not satisfiable.

Given two assignments $A, B : V \to D$, we define the assignment $f(A,B)$ such that $f(A,B)(v) = f(A(v), B(v))$. We note that if A and B are solutions of I, then $f(A,B)$ is a solution to I, too: indeed, arbitrarily choose one constraint $((x_1, \ldots, x_k), r) \in C$. Then, $(A(x_1), \ldots, A(x_k)) \in r$ and $(B(x_1), \ldots, B(x_k)) \in r$, which implies that $(f(A(x_1), B(x_1)), \ldots, f(A(x_k), B(x_k))) \in r$, too.

Let $S_1, \ldots, S_m$ be an enumeration of all solutions of I and define

$$S^+ = f(S_1, f(S_2, f(S_3 \ldots f(S_{m-1}, S_m) \ldots))).$$

By the choice of V' and the fact that $f(c,d) = 0$ if and only if $c = d = 0$, we see that the solution S^+ has the following property: $S^+(v) = 0$ if and only if $v \in V'$. Let p denote the second least element in D, and note that $\text{OPT}(I) \geq \sum_{v \in V \setminus V'} w(v)p = c$. Thus, by finding a solution with measure $\geq c$, we have approximated I within $(\max D)/p$, and W-MAX SOL(Γ) is in **APX**. To find such a solution, we consider the instance $I' = (V, D, C', w)$, where $C' = C \cup \{((v), u) \mid v \in V \setminus V'\}$. This instance has feasible solutions (since S^+ is a solution), and every solution has measure $\geq c$. Finally, a concrete solution can be found in polynomial time by the result in [19]. □

LEMMA 6.17. *If f is a 2-semilattice operation such that $f \notin \mathcal{A}$, Γ is a maximal constraint language satisfying $\langle\Gamma\rangle = Inv(f)$, and for all two-element $B \in Inv(f)$ the operation f acts as the* max *operation on B, then* W-MAX SOL(Γ) *is in* **PO**.

Proof. What we will prove is that if f acts as max on all two-element $B \in Inv(f)$, then f is a generalized max operation and consequently W-MAX SOL(Γ) is in **PO**.

First note that if $a \neq b$ and $f(a,b) = a$, then by assumption $a > b$ and $f(a,b) > \min\{a,b\}$. Now, if $f(a,b) \neq a$, then $f(a, f(a,b)) = f(a,b)$ and by assumption f

is max on $\{a, f(a,b)\}$. As a consequence of this we get $f(a,b) > \min\{a,b\}$. Now, $f(a,b) > \min\{a,b\}$ for all $a \neq b$. Moreover, f is idempotent, so f is a generalized max operation and tractability follows from Theorem 5.10. □

We have now completely classified the complexity of W-MAX SOL(Γ) for all constraint languages Γ such that $\langle\Gamma\rangle$ is maximal and $|D| \leq 4$.

6.4.3. Complete classification under a conjecture. In this section we will prove that the validity of Conjecture 131 from [47] implies a complete complexity classification of W-MAX SOL(Γ) for all constraint languages Γ such that $\langle\Gamma\rangle$ is maximal. Given a binary operation f on D, the *fixity* of f is denoted $\mathcal{F}(f)$ and is defined by

$$\mathcal{F}(f) = \{(x,y) \in D^2 \mid f(x,y) \in \{x,y\}\}.$$

The *fixity-count* of f is defined to be the cardinality of $\mathcal{F}(f)$ and is denoted $|\mathcal{F}(f)|$.

CONJECTURE 6.18 (see Conjecture 131). *If $Inv(f)$ is a maximal relational clone and $Inv(f') = Inv(f)$, then $|\mathcal{F}(f)| = |\mathcal{F}(f')|$.*

Although Conjecture 6.18 is not known to hold in the general case, it has been verified for small domains. In particular, it was shown in [47] that for domains D such that $|D| \leq 4$ the conjecture holds. Our proof builds on a construction that facilitates the study of operation f—the details are collected in Lemma 6.19. The underlying idea and the proof of Lemma 6.19 are inspired by Lemma 3 in [14].

Let f be a binary operation on D and define operations $f_1, f_2, \ldots : D^2 \to D$ inductively:

$$f_1(x,y) = f(x,y),$$

$$f_{n+1}(x,y) = f(x, f_n(x,y)).$$

LEMMA 6.19. *Assume f to be a binary commutative idempotent operation on D such that $Inv(f)$ is a maximal relational clone and $Inv(f) \neq Inv(g)$ for every $g \in \mathcal{A}$. The following hold:*

1. *$f|_B = f_n|_B$ for every $n \geq 1$ and every two-element $B \subseteq D$ in $Inv(f)$; and*
2. *$Inv(f) = Inv(f_n)$, $n \geq 1$.*

Proof. 1. Arbitrarily choose a two-element $\{a,b\} = B \subseteq D$ in $Inv(f)$. There are two possible binary commutative idempotent operations on B, namely, max and min. We assume without loss of generality that $f|_B = \max$ and prove the result by induction over n. Since $f_1 = f$, the claim holds for $n = 1$. Assume it holds for $n = k$ and consider f_{k+1}. We see that $f_{k+1}(a,b) = f(a, f_k(a,b))$ and, by the induction hypothesis, $f_k(a,b) = \max(a,b)$. Hence, $f_{k+1}(a,b) = \max(a, \max(a,b)) = \max(a,b)$.

2. Obviously, $f_n \in Pol(Inv(f))$, and, thus, $Inv(f) \subseteq Inv(f_n) \subseteq R_D$. Since $Inv(f) \neq Inv(g)$ for every $g \in \mathcal{A}$, we know from Lemma 6.13 that there is some two-element $B \in Inv(f)$. By the proof above, we also know that $f|_B = f_n|_B$ so $f_n|_B$ (and consequently f_n) is not a projection. Thus, $Inv(f_n) \neq R_D$, since $Inv(f') = R_D$ if and only if f' is a projection. By the assumption that $Inv(f)$ is a maximal relational clone and the fact that $Inv(f) \subseteq Inv(f_n) \subsetneq R_D$, we can draw the conclusion that $Inv(f) = Inv(f_n)$. □

We will now present some technical machinery that is needed for proving Lemmas 6.25 and 6.26.

LEMMA 6.20 (see [47, Lemma 28]). *Let f be an idempotent binary operation and let $n \in \mathbb{N}$. Then, $\mathcal{F}(f) \subseteq \mathcal{F}(f_n)$.*

Proof. Let $(x, y) \in \mathcal{F}(f)$. Then, either $f(x, y) = x$ or $f(x, y) = y$. Now

$$f(x,y) = x \Longrightarrow f_n(x,y) = \underbrace{f(x, f(x, \ldots, f(x,y)\ldots))}_{n \text{ times}}$$

$$= \underbrace{f(x, f(x, \ldots, f(x,x)\ldots))}_{n-1 \text{ times}} = x \Longrightarrow f_n(x,y) = x,$$

while

$$f(x,y) = y \Longrightarrow f_n(x,y) = \underbrace{f(x, f(x, \ldots, f(x,y)\ldots))}_{n \text{ times}}$$

$$= \underbrace{f(x, f(x, \ldots, f(x,y)\ldots))}_{n-1 \text{ times}} \Longrightarrow f_n(x,y) = y. \qquad \square$$

Assuming that Conjecture 6.18 holds, we get the following corollary as a consequence of Lemma 6.20.

COROLLARY 6.21. *If $Inv(f)$ is a maximal relational clone such that f is commutative and idempotent, and $(x, y) \in \mathcal{F}(f_k)$ (i.e., $f_k(x, y) \in \{x, y\}$), then $\{x, y\} \in Inv(f)$.*

Proof. By Lemma 6.20, we have $\mathcal{F}(f) \subseteq \mathcal{F}(f_k)$, and if Conjecture 6.18 holds, then $|\mathcal{F}(f)| = |\mathcal{F}(f_k)|$, which implies that $\mathcal{F}(f) = \mathcal{F}(f_k)$. Now, if $f_k(x, y) \in \{x, y\}$, then $f(x, y) \in \{x, y\}$, and by the commutativity of f we have $f(y, x) \in \{x, y\}$. Since f is idempotent, it is clear that $\{x, y\} \in Inv(f)$. $\square$

We continue by introducing a digraph associated with the binary operation f. This digraph enables us to make efficient use of Lemma 6.19. Given a binary operation $f : D^2 \to D$, we define $G_f = (V, E)$ such that $V = D \times D$ and $E = \{((a, b), (a, f(a, b))) \mid a, b \in D\}$. We make the following observations about G_f:

(1) an edge $((a, b), (a, c))$ implies that $f(a, b) = c$;
(2) every vertex has out-degree 1; and
(3) there is no edge $((a, b), (c, d))$ with $a \neq c$.

We extract some more information about G_f in the next three lemmas.

LEMMA 6.22. *The digraph G_f contains no directed cycle.*

Proof. Assume G_f contains a directed cycle. Fact (3) allows us to assume (without loss of generality) that the cycle is $(0, 1), (0, 2), \ldots, (0, k), (0, 1)$ for some $k \geq 2$. Fact (1) tells us that $f(0, 1) = 2$, $f(0, 2) = 3, \ldots$, $f(0, k - 1) = k$, and $f(0, k) = 1$. Furthermore, one can see that $f_2(0, 1) = 3$, $f_2(0, 2) = 4, \ldots$, and inductively $f_p(0, i) = i + p \pmod{k}$. This implies that $f_k(0, 1) = 1 + k \pmod{k} = 1$. By Corollary 6.21, $\{0, 1\}$ is a subalgebra of $Inv(f)$, which contradicts the fact that $f(0, 1) = 2$. $\square$

LEMMA 6.23. *Every path in G_f of length $n \geq |D|$ ends in a reflexive vertex; i.e., $f_n(a, b) = c$ implies that (a, c) is a reflexive vertex.*

Proof. Assume that G_f contains a path P of length $n \geq |D|$. This path can contain at most $|D|$ distinct vertices by fact (3). Fact (2) together with the acyclicity of G_f implies that at least one vertex v on P is reflexive; by using fact (2) once again, we see that there exists exactly one reflexive vertex on P and that it must be the last vertex. $\square$

LEMMA 6.24. *If $f_n(a, b) = c$, then (a, c) is a reflexive vertex in G_f.*

Proof. By Lemma 6.23, every path in G_f of length $n \geq |D|$ ends in a reflexive vertex. Hence, $(a, f_n(a, b)) = (a, c)$ is a reflexive vertex. $\square$

LEMMA 6.25. *Let f be a binary commutative idempotent operation on D such that Γ is a maximal constraint language satisfying $\langle\Gamma\rangle = Inv(f)$. If there exist $a, b \in D$ such that $a < b$, $f(a,b) = a$, and $a^* > 0$, where a^* is the minimal element such that there is b^* with $f(a^*, b^*) = a^*$, then (assuming Conjecture* 6.18*)* W-MAX SOL(Γ) *is* **APX***-complete.*

Proof. We can assume, without loss of generality, that $a = a^*$ and $b = b^*$. The **APX**-hardness part follows from Lemma 4.3. What remains is to show that the problem is in **APX**. We begin the proof by proving that the unary relation $U = D\backslash\{0\}$ is a member of $Inv(f)$. Consider the digraph G_f. As we have already observed in Lemma 6.22, there are no cycles $(a,b_1),\ldots,(a,b_k),(a,b_1)$, $k \geq 2$, in G_f, and every path of length $n \geq |D|$ ends in a reflexive vertex (by Lemma 6.23). Obviously, no vertex $(a,0)$ $(a > 0)$ in G_f is reflexive since this implies that $f(a,0) = 0$ which is a contradiction. In particular, there exists no path in G_f of length $n \geq |D|$ starting in a vertex (a,b) $(a > 0)$ and ending in a vertex $(a,0)$, since this implies that $(a,0)$ is reflexive by Lemma 6.23.

We can now conclude that $f_n(a,b) > 0$ when $a > 0$: if $f_n(a,b) = 0$, then $(a,0)$ is reflexive by Lemma 6.24, which would lead to a contradiction. Hence, $f_n(a,b) > 0$ whenever $a, b \in D \setminus \{0\} = U$ so U is in $Inv(f_n)$, and, by Lemma 6.19(2), U is in $Inv(f)$, too. We also note that $U \in Inv(f)$ together with the assumption that $f(0,b) > 0$ for all $b > 0$ implies that $f(c,d) = 0$ if and only if $c = d = 0$. The rest of the proof is identical to the second part of the proof of Lemma 6.16. ∎

LEMMA 6.26. *If f is a binary commutative idempotent operation such that $f \notin \mathcal{A}$, Γ is a maximal constraint language satisfying $\langle\Gamma\rangle = Inv(f)$, and for all two-element $B \in Inv(f)$ the operation f acts as the* max *operation on B, then (assuming Conjecture* 6.18*)* W-MAX SOL(Γ) *is in* **PO***.*

Proof. What we will prove is that if $f \notin \mathcal{A}$ and f acts as max on all two-element $B \in Inv(f)$, then there exists a generalized max function f' such that $Inv(f') = Inv(f)$ and, hence, W-MAX SOL(Γ) is in **PO**.

Recall that there are no cycles in G_f and every path of length $n = |D|$ must end in a reflexive vertex by Lemmas 6.22 and 6.23. We also note that if G_f contains a reflexive vertex (a,c) with $a \neq c$, then $f(a,c) = c$ and $c > a$ since f is assumed to act as the max operation on all two-element $B \in Inv(f)$.

We now claim that f_n is a generalized max operation. Arbitrarily choose $a, b \in D$. If $f_n(a,b) = c$ $(a \neq c)$, then there is a path in G_f from (a,b) to a reflexive vertex (a,c), and $c > a$ as explained above. If $f_n(a,b) = a$, then $\{a,b\} \in Inv(f)$ by Corollary 6.21. Since $f(a,b) = \max(a,b)$, Lemma 6.19(1) implies that $f_n(a,b) = \max(a,b)$. Thus, f_n is a generalized max-operation. ∎

7. Homogeneous constraint languages. In this section, we will classify the complexity of MAX SOL when the constraint language is *homogeneous*. A constraint language is called homogeneous if every *permutation relation* is contained in the language.

DEFINITION 7.1. *A relation R is a* permutation relation *if there is a permutation $\pi : D \to D$ such that*

$$R = \{(x, \pi(x)) \mid x \in D\}.$$

Let Q denote the set of all permutation relations on D. The main result of this section is Theorem 7.16 which gives a complete classification of the complexity of W-MAX SOL(Γ) when $Q \subseteq \Gamma$. The theorem provides the exact borderlines between tractability, **APX**-completeness, **poly-APX**-completeness, and **NP**-hardness of finding a feasible solution.

As a direct consequence of Theorem 7.16, we get that the class of injective relations is a maximal tractable class for W-MAX SOL(Γ). That is, if we add a single relation which is not an injective relation to the class of all injective relations, then the problem is no longer in **PO** (unless **P** = **NP**).

Dalmau has completely classified the complexity of CSP(Γ) when Γ is a homogeneous constraint language [23], and this classification relies heavily on the structure of homogeneous algebras. An algebra is called *homogeneous* if and only if every permutation on its universe is an automorphism of the algebra. We will not need a formal definition of homogeneous algebras and refer the reader to [39, 40, 48] for further information on their properties. All homogeneous algebras have been completely classified by Marczewski [40] and Marchenkov [39].

Our classification for the approximability of W-MAX SOL(Γ) when Γ is a homogeneous constraint language uses the same approach as in [23], namely, we exploit the inclusion structure of homogeneous algebras (as proved in [39, 40]). By Theorem 3.3, it is sufficient to consider constraint languages Γ that are relational clones. This is where the homogeneous algebras comes in: the classification of homogeneous algebras gives us a classification of all homogeneous relational clones and, in particular, their inclusion structure (lattice) under set inclusion. We refer the reader to [48] for a deeper treatment of homogeneous algebras and their inclusion structure.

The lattice of all homogeneous relational clones on a domain D having n (≥ 5) elements is given in Figure 7.1. The lattices for the corresponding relational clones over smaller domains (i.e., $2 \leq |D| \leq 4$) contain some exceptional relational clones and are presented separately in Figures 7.2–7.4. Note that the corresponding lattices presented in [23] and [48] are the duals of ours, since they instead consider the inclusion structure among the corresponding clones of operations (but the two approaches are in fact equivalent as shown in [43, Satz 3.1.2]). To understand the lattices we first need some definitions.

Throughout this section n denotes the size of the domain D, i.e., $n = |D|$.

DEFINITION 7.2.

- *The* switching operation s *is defined by*

$$s(a,b,c) = \begin{cases} c & \text{if } a = b, \\ b & \text{if } a = c, \\ a & \text{otherwise.} \end{cases}$$

- *The* discriminator operation t *is defined by*

$$t(a,b,c) = \begin{cases} c & \text{if } a = b, \\ a & \text{otherwise.} \end{cases}$$

- *The* dual discriminator operation d *is defined by*

$$d(a,b,c) = \begin{cases} a & \text{if } a = b, \\ c & \text{otherwise.} \end{cases}$$

- *The* k*-ary* near projection operation l_k $(3 \leq k \leq n)$ *is defined by*

$$l_k(a_1,\ldots,a_k) = \begin{cases} a_1 & \text{if } |\{a_1,\ldots,a_k\}| < k, \\ a_k & \text{otherwise.} \end{cases}$$

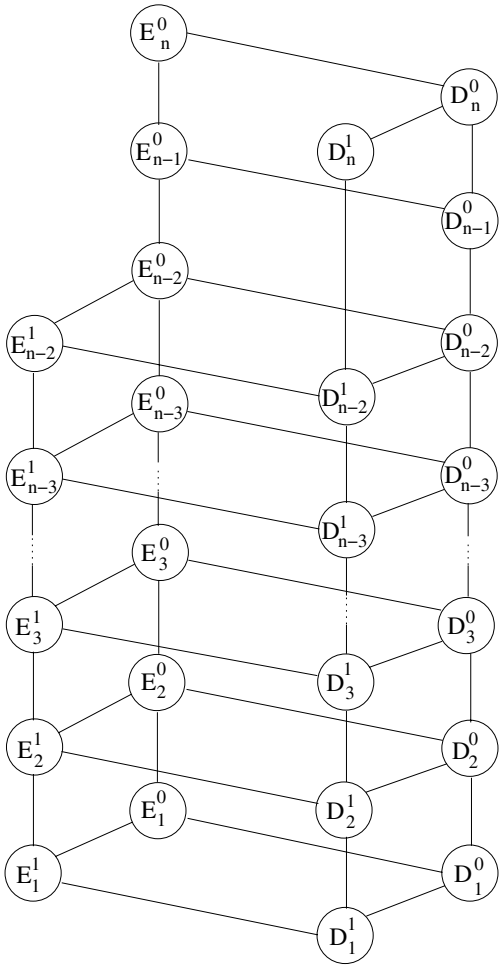


FIG. 7.1. *Lattice of all homogeneous relational clones over domain size $n \geq 5$.*

- *The $(n-1)$-ary operation r_n is defined by*

$$r_n(a_1, \ldots, a_{n-1}) = \begin{cases} a_1 & \text{if } |\{a_1, \ldots, a_{n-1}\}| < n-1, \\ a_n & \text{otherwise.} \end{cases}$$

  In the second case we have $\{a_n\} = D \setminus \{a_1, \ldots, a_{n-1}\}$.
- *The $(n-1)$-ary operation d_n (where $n \geq 4$) is defined by*

$$d_n(a_1, \ldots, a_{n-1}) = \begin{cases} d(a_1, a_2, a_3) & \text{if } |\{a_1, \ldots, a_{n-1}\}| < n-1, \\ a_n & \text{otherwise.} \end{cases}$$

  In the second case we have $\{a_n\} = D \setminus \{a_1, \ldots, a_{n-1}\}$.
- *The operation $[x+y+z]$ is defined by $x+y+z$ where $(D, +, -)$ is a four-element group of exponent 2.*

The notation in the lattices in Figures 7.1–7.4 is explained below.

- $D^0_1 = Inv(t)$;
- $D^0_i = Inv(\{d, l_{i+1}\})$ for $2 \leq i \leq n-1$;

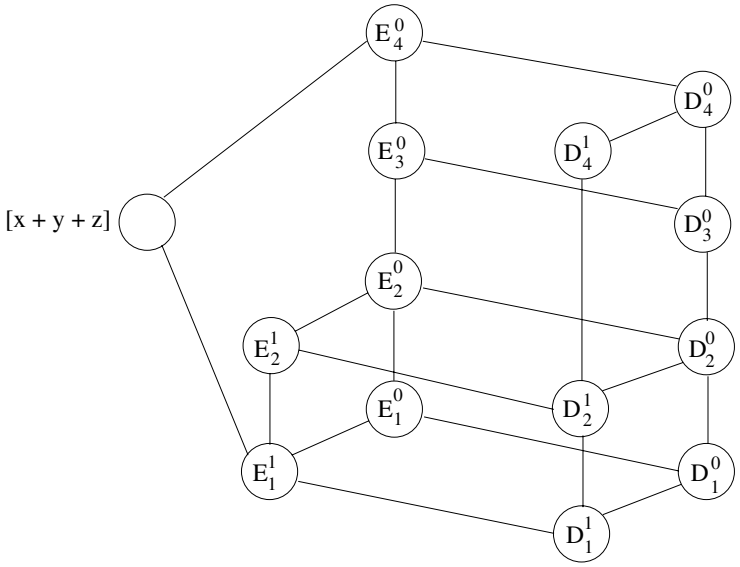


FIG. 7.2. *Lattice of all homogeneous relational clones over domain size* $n = 4$.

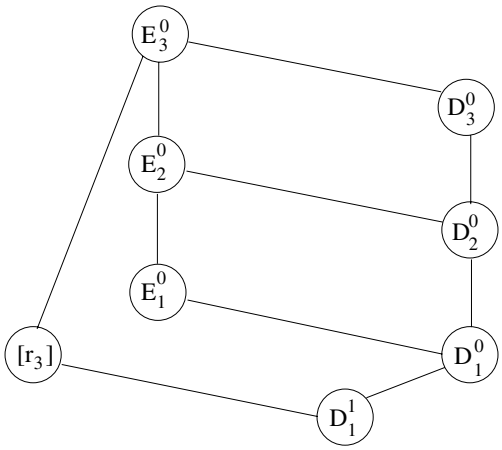


FIG. 7.3. *Lattice of all homogeneous relational clones over domain size* $n = 3$.

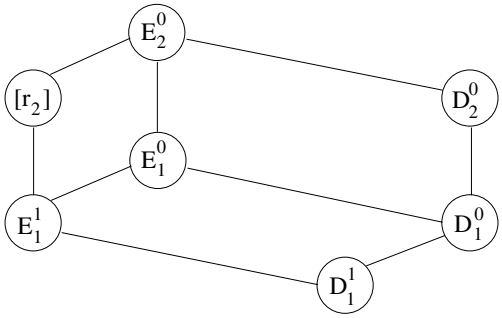


FIG. 7.4. *Lattice of all homogeneous relational clones over domain size* $n = 2$.

- $D_n^0 = Inv(d)$;
- $D_1^1 = Inv(\{t, r_n\})$;
- $D_i^1 = Inv(\{d, l_{i+1}, r_n\})$ for $2 \leq i \leq n-2$;
- $D_n^1 = Inv(d_n)$;
- $E_1^0 = Inv(s)$;
- $E_i^0 = Inv(l_{i+1})$ for $2 \leq i \leq n-1$;
- $E_n^0 = R_D$, i.e., all finitary relations over D;
- $E_1^1 = Inv(\{s, r_n\})$ for $n \neq 3$;
- $E_i^1 = Inv(\{l_{i+1}, r_n\})$ for $2 \leq i \leq n-3$;
- $E_{n-2}^1 = Inv(r_n)$ for $n \geq 4$.

Note that the relational clones described above depend on the size of the domain $|D| = n$. Hence, E_2^0 in Figure 7.3 ($|D| = 3$) is not the same relational clone as E_2^0 ($|D| = 4$) in Figure 7.2. Also note that when we state our (in)approximability results by saying, for example, that W-MAX SOL(E_1^1) is **APX**-complete, we mean that W-MAX SOL(E_1^1) is **APX**-complete for all sizes of the domain where E_1^1 is defined (e.g., E_1^1 is not defined for $n = 3$). We always assume that $n = |D| \geq 2$.

We now state Dalmau's classification for the complexity of CSP(Γ) for homogeneous constraint languages Γ.

THEOREM 7.3 (see [23]). *Let Γ be a homogeneous constraint language. Then,* CSP(Γ) *is in* **P** *if $Pol(\Gamma)$ contains the dual discriminator operation d, the switching operation s, or an affine operation. Otherwise,* CSP(Γ) *is* **NP***-complete.*

We have the following corollary of Dalmau's classification.

COROLLARY 7.4. *Let Γ be a homogeneous constraint language. Then,* W-MAX SOL(Γ) *is in* **poly-APX** *if $Pol(\Gamma)$ contains the dual discriminator operation d, the switching operation s, or an affine operation. Otherwise, it is* **NP***-hard to find a feasible solution to* W-MAX SOL(Γ).

Proof. All dual discriminator operations, switching operations, and affine operations are idempotent (i.e., $f(x, x, x) = x$ for all $x \in D$). Hence, Γ is invariant under a dual discriminator operation, switching operation, or an affine operation if and only if $\Gamma^c = \{\Gamma \cup \{\{(d_1)\}, \ldots, \{(d_n)\}\}$ is invariant under the corresponding operation. Thus, it follows from Theorem 7.3 that CSP(Γ^c) is in **P** if $Pol(\Gamma)$ contains the dual discriminator operation d, the switching operation s, or an affine operation. This together with Lemma 4.2 gives us that W-MAX SOL(Γ) is in **poly-APX** if $Pol(\Gamma)$ contains the dual discriminator operation d, the switching operation s, or an affine operation.

The **NP**-hardness part follows immediately from Theorem 7.3. □

We begin by investigating the approximability of W-MAX SOL(Γ) for some particular homogeneous constraint languages Γ.

LEMMA 7.5. W-MAX SOL(D_n^0) *is in* **APX** *if $0 \notin D$ and in* **poly-APX** *otherwise.*

Proof. Remember that $D_n^0 = Inv(d)$. Hence, membership in **poly-APX** follows directly from Corollary 7.4. It is known from Dalmau's classification that CSP(D_n^0) is in **P**. Thus, by Proposition 4.1, it follows that W-MAX SOL(D_n^0) is in **APX** when $0 \notin D$. □

LEMMA 7.6. W-MAX SOL(D_2^1) *is* **APX***-complete if $0 \notin D$ and* **poly-APX***-complete if $0 \in D$.*

Proof. Choose any $a, b \in D$ such that $a < b$. The relation $r = \{(a, a), (a, b), (b, a)\}$ is in $D_i^1 = Inv(\{d, l_{i+1}, r_n\})$ for $2 \leq i \leq n-2$. Hence, by Lemmas 4.3 and 4.4, it

follows that W-MAX SOL(D_2^1) is **APX**-hard if $0 \notin D$ and **poly-APX**-hard if $0 \in D$. This together with Lemma 7.5 and the fact that $D_i^1 \subseteq D_n^0$ gives us that W-MAX SOL(D_2^1) is **APX**-complete if $0 \notin D$ and **poly-APX**-complete if $0 \in D$. □

LEMMA 7.7. *Finding a feasible solution to* W-MAX SOL(E_2^1) *is* **NP***-hard.*

Proof. Remember that $E_2^1 = Inv(\{l_3, r_n\})$ when $n > 4$ and $E_2^1 = Inv(r_n)$ when $n = 4$, so it follows from Dalmau's classification that CSP(E_2^1) is **NP**-complete. □

LEMMA 7.8. W-MAX SOL(D_1^0) *is in* **PO**.

Proof. It is well known that $D_1^0 = Inv(t) = \langle I^D \rangle$; for instance, it is a direct consequence of Theorem 4.2 in [48]. Hence, W-MAX SOL(D_1^0) is in **PO** by the results in section 5.1. □

LEMMA 7.9. W-MAX SOL(E_1^0) *is in* **APX**.

Proof. Remember that $E_1^0 = Inv(s)$. Dalmau gives a polynomial-time algorithm for CSP($Inv(s)$) in [23] (he actually gives a polynomial-time algorithm for the more general class of paraprimal problems). Dalmau's algorithm exploits in a clever way the internal structure of paraprimal algebras to show that any instance I of CSP($Inv(s)$) can be split into independent subproblems $I_1, \ldots, I_j$, such that

- the set of solutions is preserved (i.e., any solution to I is also a solution to each of the independent subproblems, and any solution to all of the independent subproblems is also a solution to I); and
- each I_i $(1 \leq i \leq j)$ is either an instance of CSP($Inv(t)$) or an instance of CSP($Inv(a)$), where t is the discriminator operation and a is an affine operation.

Hence, to show that W-MAX SOL(E_1^0) is in **APX**, we first use Dalmau's algorithm to reduce the problem (in a solution preserving manner) to a set of independent W-MAX SOL(Γ) problems where Γ is either invariant under an affine operation or the discriminator operation. We know from Lemma 7.8 that W-MAX SOL(Γ) is in **PO** when Γ is invariant under the discriminator operation, and from Theorem 6.12 we know that W-MAX SOL(Γ) is in **APX** when Γ is invariant under an affine operation. Since all the independent subproblems are in **APX**, we get that the original W-MAX SOL(E_1^0) problem is also in **APX**. □

LEMMA 7.10. W-MAX SOL(E_1^1) *is* **APX***-complete.*

Proof. Remember that $E_1^1 = Inv(\{s, r_n\})$. Note that $E_1^1 \subseteq E_1^0$; thus membership in **APX** follows from Lemma 7.9.

For the hardness part, we begin by considering the general case when $n = |D| \geq 4$. Choose an arbitrary two-element subset $\{a, b\}$ of D (without loss of generality assume that $a < b$) and let $(G, +, -)$ be the two-element group on $G = \{a, b\}$ defined by $a + a = a$, $a + b = b + a = b$, and $b + b = a$. Let Γ be the set of all relations expressible as the set of solutions to equations over $(G, +, -)$. It is easy to realize that Γ is invariant under r_n (since r_n $(n \geq 4)$ acts as a projection on $\{a, b\}$). Furthermore Γ is invariant under s since $s(x, y, z)$ acts as the affine operation $x + y + z$ on $\{a, b\}$. It is proved in Lemma 6.8 that W-MAX SOL EQN($\mathbb{Z}_2, g$) is **APX**-hard, so W-MAX SOL(Γ) is **APX**-complete. For $n = |D| = 3$ there is no relational clone of the type $E_1^1 = Inv(\{s, r_n\})$, so the only case that remains to be dealt with is the case where $n = |D| = 2$.

Without loss of generality assume that $D = \{a, b\}$, where $a < b$. Again consider the group $(G, +, -)$ on $\{a, b\}$. Let $\Gamma = \{R_1, R_2\}$, where R_1 is the (4-ary) relation on D which is the set of solutions to the equation $x_1 + x_2 + x_3 + x_4 = a$ and R_2 is the (binary) relation representing the set of solutions to the equation $y_1 + y_2 = b$. Furthermore, let $\Gamma_{0,1} = \{R_1, R_2\}$ denote the special case where $a = 0, b = 1$ (i.e.,

$D = \{0, 1\}$). It has been proved in [37, Lemma 6.9] that MAX SOL($\Gamma_{0,1}$) is **APX**-complete. Let I be an instance of MAX SOL($\Gamma_{0,1}$) containing k variables. It is easy to realize that if I has a solution, then $\text{OPT}(I) \geq k/2$ (just note that the complement of any solution to I is also a solution to I). We give an L-reduction from MAX SOL($\Gamma_{0,1}$) to MAX SOL($\Gamma_{a,b}$). Let $F(I)$ be the instance of MAX SOL($\Gamma_{a,b}$) where all occurrences of 0 have been replaced by a and all occurrences of 1 have been replaced by b. Since $\text{OPT}(I) \geq k/2$, we get that $\text{OPT}(F(I)) \leq bk \leq 2b \cdot \text{OPT}(I)$ and $\beta = 2b$ is a valid parameter in the L-reduction. Let s be an arbitrary solution to $F(I)$ and define $G(F(I), s)$ to be the corresponding solution to I, where a is replaced by 0 and b is replaced by 1. Then,

$$|m(I, G(F(I), s)) - \text{OPT}(I)| \leq \frac{1}{b-a} |m(F(I), s) - \text{OPT}(F(I))|,$$

and $\gamma = \frac{1}{b-a}$ is a valid parameter in the L-reduction. This completes the **APX**-hardness proof for MAX SOL($\Gamma_{a,b}$). Now, the unary operation r_2 acts as the non-identity permutation on $\{a, b\}$ (i.e., $r_2(a) = b$, and $r_2(b) = a$) so both R_1 and R_2 are invariant under r_2. As we have already observed, s acts as the affine operation on $\{a, b\}$ and R_1 and R_2 are invariant under s, too. Hence, $\Gamma_{a,b} \subseteq Inv(\{s, r_2\})$, which concludes the proof. □

We have now proved all the results needed to give a complete classification for the approximability of W-MAX SOL(Γ) for all homogeneous relational clones Γ over domains D of size at least 5. In order to complete the classification also for domains of sizes 2, 3, and 4, we need to consider some exceptional homogeneous relational clones. For domains of size 4, we need to consider the homogeneous relational clone $Inv(x+y+z)$, where $+$ is the operation of a four-element group $(D, +, -)$ of exponent 2 (i.e., a four-element group such that, for all $a \in D$, $a + a = e$, where e is the identity element in $(D, +, -)$).

LEMMA 7.11. *Let $f(x, y, z) = x + y + z$, where $(D, +, -)$ is a group of exponent 2. Then,* W-MAX SOL($Inv(f)$) *is* **APX**-*complete.*

Proof. The operation $x + y + x$ is the affine operation on $(D, +, -)$ (since $-y = y$ in $(D, +, -)$), and it follows directly from Theorems 6.11 and 6.12 that W-MAX SOL($Inv(x + y + z)$) is **APX**-complete. □

For three-element domains, it remains to classify the approximability of W-MAX SOL($Inv(r_3)$), where r_3 is the binary operation defined as follows:

$$r(a_1, a_2) = \begin{cases} a_1 & \text{if } a_1 = a_2, \\ a_3 & \text{where } \{a_3\} = D \setminus \{a_1, a_2\} \text{ otherwise.} \end{cases}$$

LEMMA 7.12. W-MAX SOL($Inv(r_3)$) *is* **APX**-*complete.*

Proof. The operation $r_3(x, y)$ is actually an example of an operation of the type $\frac{p+1}{2}(x + y)$ (where $+$ is the operation of an Abelian group of order $|D| = p$) from Lemma 6.15. In our case $p = 3$ and $r_3(x, y) = 2x + 2y$, where $+$ is the operation of the Abelian group $(D, +, -)$ isomorphic to $\mathbb{Z}_3$. Hence, it follows from Lemma 6.15 that W-MAX SOL($Inv(r_3)$) is **APX**-complete. □

For three-element domains we also need to classify the approximability of W-MAX SOL(E_2^0) since hardness no longer follows from the hardness of W-MAX SOL(E_2^1) (there is no relational clone E_2^1 over three-element domains).

LEMMA 7.13. *It is* **NP**-*hard to find a feasible solution to* W-MAX SOL(E_2^0).

Proof. The proof is an immediate consequence of Theorem 7.3. □

Similarly, we also need to classify the approximability of W-MAX SOL(D_2^0) since hardness no longer follows from the hardness of W-MAX SOL(D_2^1) (there is no relational clone D_2^1 over three-element domains).

LEMMA 7.14. W-MAX SOL(D_2^0) *is* **APX**-*complete if* $0 \notin D$ *and* **poly-APX**-*complete if* $0 \in D$.

Proof. Remember that $D_2^0 = Inv(\{d, l_3\})$. It follows from the proof of Lemma 7.6 that W-MAX SOL(D_2^0) is **APX**-complete if $0 \notin D$ and **poly-APX**-complete if $0 \in D$. □

For two-element domains $\{a, b\}$, we need to classify the unary operation r_2 which acts as the nonidentity permutation (i.e., $r_2(a) = b$ and $r_2(b) = a$).

LEMMA 7.15. *Finding a feasible solution to* W-MAX SOL($Inv(r_2)$) *is* **NP**-*hard.*

Proof. The proof follows from Dalmau's classification. □

Finally, we are in a position to present the complete classification for the approximability of all homogeneous constraint languages.

THEOREM 7.16. *Let* Γ *be a homogeneous constraint language.*

1. *If* $\langle\Gamma\rangle \in \{E_j^i \mid i \in \{0,1\}, j \geq 2\}$ *or* $\langle\Gamma\rangle = Inv(r_2)$*, then it is* **NP**-*hard to find a feasible solution to* W-MAX SOL(Γ)*;*
2. *else, if* $0 \in D$ *and* $\langle\Gamma\rangle \in \{D_j^i \mid i \in \{0,1\}, j \geq 2\}$*, then* W-MAX SOL($\Gamma$) *is* **poly-APX**-*complete;*
3. *else, if*

$$\langle\Gamma\rangle \in \{E_1^1, E_1^0, Inv(x+y+z), Inv(r_3)\} \quad \text{or}$$
$$\langle\Gamma\rangle \in \{D_j^i \mid i \in \{0,1\}, j \geq 2\} \quad \text{and} \quad 0 \notin D,$$

   then W-MAX SOL(Γ) *is* **APX**-*complete;*
4. *otherwise,* $\langle\Gamma\rangle \in \{D_1^1, D_1^0\}$ *and* W-MAX SOL(Γ) *is in* **PO**.

Proof. We know from Theorem 3.3 that it is sufficient to consider constraint languages that are relational clones.

1. It is proved in Lemmas 7.7 and 7.13 that it is **NP**-hard to find a feasible solution to W-MAX SOL(E_2^1) and W-MAX SOL(E_2^0). **NP**-hardness of finding a feasible solution to W-MAX SOL($Inv(r_2)$) was proved in Lemma 7.15. The result follows from the fact that $E_2^1 \subseteq E_j^1$ and $E_2^0 \subseteq E_j^0$ $(2 \leq j \leq n)$.
2. Membership in **poly-APX** is proved in Lemma 7.5, and **poly-APX**-hardness of W-MAX SOL(D_2^1) and W-MAX SOL(D_2^0) when $0 \in D$ is proved in Lemmas 7.6 and 7.14. The result follows from the fact that $D_2^1 \subseteq D_j^1 \subseteq D_n^0$ and $D_2^0 \subseteq D_j^0 \subseteq D_n^0$ $(2 \leq j \leq n)$.
3. It is proved in Lemmas 7.11 and 7.12 that W-MAX SOL($Inv(x+y+z)$) and W-MAX SOL($Inv(r_3)$) are **APX**-complete. It is proved in Lemma 7.9 that W-MAX SOL(E_1^0) is in **APX**. **APX**-hardness for W-MAX SOL(E_1^1) is proved in Lemma 7.10. For $n \geq 4$ or $n = 2$ we have that $E_1^1 \subseteq E_1^0$, and it follows that W-MAX SOL(E_1^0) and W-MAX SOL(E_1^1) are **APX**-complete. For $n = 3$, there exists no E_1^1 so **APX**-hardness for W-MAX SOL(E_1^0) must be proved separately. Since $E_1^0 = Inv(s)$ it is easy to see that the proof of Lemma 7.10 gives **APX**-hardness for W-MAX SOL(E_1^0).
   Membership in **APX** for W-MAX SOL(D_n^0) when $0 \notin D$ is proved in the first part of Lemma 7.5. **APX**-hardness of W-MAX SOL(D_2^1) and W-MAX SOL(D_2^0) is proved in Lemmas 7.6 and 7.14, respectively. Hence, **APX**-completeness of W-MAX SOL(D_j^0) and W-MAX SOL(D_j^1) $(2 \leq j \leq n)$ when $0 \notin D$ follows from the fact that $D_2^1 \subseteq D_j^1 \subseteq D_n^0$, $D_2^0 \subseteq D_j^0 \subseteq D_n^0$ $(2 \leq j \leq n)$.
4. It is proved in Lemma 7.8 that W-MAX SOL(D_1^0) is in **PO**, and the result follows from the fact that $D_1^1 \subseteq D_1^0$. □

As a direct consequence of the preceding theorem, we get that the class of injective relations is a maximal tractable class for W-MAX SOL(Γ). That is, if we add a single relation which is not an injective relation to the class of all injective relations, then the problem is no longer in **PO** (unless **P** = **NP**).

COROLLARY 7.17. *Let Γ_I^D be the class of injective relations and let R be an arbitrary relation which is not in Γ_I^D. Then,* W-MAX SOL($\Gamma_I^D \cup \{R\}$) *is not in* **PO** *(unless* **P** = **NP***).*

Proof. We know from the proof of Lemma 7.8 that $D_1^0 = Inv(t) = \Gamma_I^D = \langle I^D \rangle$. It follows from the lattices of homogeneous relational clones that either $E_1^0 \subseteq \langle D_1^0 \cup \{R\} \rangle$ or $D_2^0 \subseteq \langle D_1^0 \cup \{R\} \rangle$. We know from Lemmas 7.10 and 7.6 that both W-MAX SOL(E_1^0) and W-MAX SOL(D_2^0) are **APX**-hard. Thus, W-MAX SOL($D_1^0 \cup \{R\}$) is **APX**-hard, and it follows that W-MAX SOL($\Gamma_I^D \cup \{R\}$) is not in **PO** (unless **P** = **NP**). □

8. Conclusions. We view this article as a first step toward a better understanding of the approximability of non-Boolean W-MAX SOL. The ultimate long-term goal for this research is, of course, to completely classify the approximability for all finite constraint languages. However, we expect this to be a hard problem since not even a complete classification for the corresponding decision problem CSP is known. A more manageable task would be to completely classify W-MAX SOL for constraint languages over small domains (say, of size 3 or 4). For size 3, this has already been accomplished for CSP [10] and MAX CSP [34]. Another obvious way to extend the results of this paper would be to complete the classification of maximal constraint languages over arbitrary finite domains, perhaps by proving Conjecture 131 from [47].

Our results combined with the results of Khanna et al. [37] for Boolean domains suggest the following conjecture.

CONJECTURE 8.1. *For every constraint language Γ, one of the following holds:*

1. W-MAX SOL(Γ) *is in* **PO**;
2. W-MAX SOL(Γ) *is* **APX***-complete;*
3. W-MAX SOL(Γ) **poly-APX***-complete;*
4. *it is* **NP***-hard to find a nonzero solution to* W-MAX SOL(Γ); *or*
5. *it is* **NP***-hard to find any solution to* W-MAX SOL(Γ).

If this conjecture is true, then there does not exist any constraint language Γ_1 such that W-MAX SOL(Γ_1) has a polynomial-time approximation scheme (PTAS) but W-MAX SOL(Γ_1) is not in **PO**. Such natural classes exist, however, if one restricts the way constraints are applied to variables (instead of restricting the allowed constraint types). MAXIMUM INDEPENDENT SET (and, equivalently, MAX ONES($\{(0,0),(1,0),(0,1)\}$)) is one example: the unrestricted problem is **poly-APX**-complete and not approximable within $O(n^{1-\epsilon})$, $\epsilon > 0$ (unless **P**=**NP**) [52], but the problem restricted to planar instances admits a PTAS [4]. One may ask several questions in connection with this: is there a constraint language with the properties of Γ_1 above? For which constraint languages does W-MAX SOL admit a PTAS on planar instances? Or more generally: under which restrictions on variable scopes does W-MAX SOL(Γ) admit a PTAS?

It is interesting to note that the applicability of the algebraic approach to W-MAX SOL demonstrated in this article also holds for the corresponding minimization problem W-MIN SOL; that is, Theorem 3.3 still holds. The question of whether the algebraic approach can shed some new light on the intriguing approximability of minimization problems (as manifested, e.g., in [37]) is an interesting open question.

As mentioned in the introduction, it is known from [37] that the approximability of the weighted and unweighted versions of (W)-MAX SOL coincide for all Boolean

constraint languages. We remark that the same result holds for all constraint languages considered in this article. This can be readily verified by observing that the AP-reduction from W-Max Ones to Max Ones in the proof of Lemma 3.11 in [37] easily generalizes to arbitrary finite domains. Hence, we get an AP-reduction from W-Max Sol to Max Sol. Furthermore, our tractability proofs are given for the weighted version of the problem. In general, it is still an open problem if tractability (i.e., membership in **PO**) of Max Sol(Γ) implies tractability of W-Max Sol(Γ) for every constraint language Γ (the AP-reduction used above does not give us this result, as AP-reductions do not, in general, preserve membership in **PO**).

Acknowledgment. The authors would like to thank the anonymous referees for many valuable comments and in particular for pointing out a serious flaw in the (previous) proof of Lemma 6.14.

REFERENCES

[1] P. Alimonti and V. Kann, *Some APX-completeness results for cubic graphs*, Theoret. Comput. Sci., 237 (2000), pp. 123–134.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. M. Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin, 1999.

[3] G. Ausiello, A. D'Atri, and M. Protasi, *Structure preserving reductions among convex optimization problems*, J. Comput. System Sci., 21 (1980), pp. 136–153.

[4] B. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. ACM, 41 (1994), pp. 153–180.

[5] S. Bistarelli, U. Montanari, and F. Rossi, *Semiring-based constraint satisfaction and optimization*, J. ACM, 44 (1997), pp. 201–236.

[6] F. Börner, A. Bulatov, P. Jeavons, and A. Krokhin, *Quantified constraints: Algorithms and complexity*, in Proceedings of the 17th International Workshop on Computer Science Logic (CSL-2003), Springer, Berlin, Heidelberg, 2003, pp. 58–70.

[7] F. Börner, A. Krokhin, A. Bulatov, and P. Jeavons, *Quantified Constraints and Surjective Polymorphisms*, Technical report RR-02-11, Computing Laboratory, University of Oxford, Oxford, UK, 2002.

[8] A. Bulatov, *A graph of a relational structure and constraint satisfaction problems*, in Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS-2004), 2004, pp. 448–457.

[9] A. Bulatov, *Combinatorial problems raised from* 2*-semilattices*, J. Algebra, 298 (2006), pp. 321–339.

[10] A. Bulatov, *A dichotomy theorem for constraint satisfaction problems on a* 3*-element set*, J. ACM, 53 (2006), pp. 66–120.

[11] A. Bulatov and V. Dalmau, *Towards a dichotomy for the counting constraint satisfaction problem*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS-2003), 2003, pp. 562–571.

[12] A. Bulatov and V. Dalmau, *A simple algorithm for Mal'tsev constraints*, SIAM J. Comput., 36 (2006), pp. 16–27.

[13] A. Bulatov, P. Jeavons, and A. Krokhin, *Classifying the complexity of constraints using finite algebras*, SIAM J. Comput., 34 (2005), pp. 720–742.

[14] A. Bulatov, A. Krokhin, and P. Jeavons, *The complexity of maximal constraint languages*, in Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC-2001), 2001, pp. 667–674.

[15] E. Böhler, N. Creignou, S. Reith, and H. Vollmer, *Playing with Boolean blocks, part* I: *Post's lattice with applications to complexity theory*, ACM-SIGACT Newsletter, 34 (4) (2003), pp. 38–52.

[16] E. Böhler, N. Creignou, S. Reith, and H. Vollmer, *Playing with Boolean blocks, part* II: *Constraint satisfaction problems*, ACM-SIGACT Newsletter, 35 (1) (2004), pp. 22–35.

[17] H. Chen, *The complexity of quantified constraint satisfaction: Collapsibility, sink algebras, and the three-element case*, SIAM J. Comput., 37 (2008), pp. 1674–1701.

[18] H. Chen, *Quantified constraint satisfaction, maximal constraint languages, and symmetric polymorphisms*, in Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS-2005), Springer, Berlin, Heidelberg, 2005, pp. 315–326.

[19] D. Cohen, *Tractable decision for a constraint language implies tractable search*, Constraints, 9 (2004), pp. 219–229.
[20] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin, *Supermodular functions and the complexity of Max CSP*, Discrete Appl. Math., 149 (2005), pp. 53–72.
[21] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin, *The complexity of soft constraint satisfaction*, Artificial Intelligence, 170 (2006), pp. 909–1030.
[22] N. Creignou, S. Khanna, and M. Sudan, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, SIAM, Philadelphia, 2001.
[23] V. Dalmau, *A new tractable class of constraint satisfaction problems*, Ann. Math. Artif. Intell., 44 (2005), pp. 61–85.
[24] V. Dalmau and P. Jeavons, *Learnability of quantified formulas*, Theoret. Comput. Sci., 306 (2003), pp. 485–511.
[25] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.
[26] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
[27] D. Hochbaum, N. Megiddo, J. Naor, and A. Tamir, *Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality*, Math. Programming, 62 (1993), pp. 69–84.
[28] D. S. Hochbaum and J. Naor, *Simple and fast algorithms for linear and integer programs with two variables per inequality*, SIAM J. Comput., 23 (1994), pp. 1179–1192.
[29] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
[30] P. Janssen, P. Jegou, B. Nouguier, and M. Vilarem, *A filtering process for general constraint satisfaction problems: Achieving pair-wise consistency using an associated binary representation*, in Proceedings of the IEEE Workshop on Tools for Artificial Intelligence, 1989, pp. 420–427.
[31] P. Jeavons, *On the algebraic structure of combinatorial problems*, Theoret. Comput. Sci., 200 (1998), pp. 185–204.
[32] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.
[33] P. Jeavons and M. Cooper, *Tractable constraints on ordered domains*, Artificial Intelligence, 79 (1996), pp. 327–339.
[34] P. Jonsson, M. Klasson, and A. Krokhin, *The approximability of three-valued* Max *CSP*, SIAM J. Comput., 35 (2006), pp. 1329–1349.
[35] P. Jonsson, F. Kuivinen, and G. Nordh, *Approximability of integer programming with generalised constraints*, in Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP-2006), Springer, Berlin, Heidelberg, 2006, pp. 256–270.
[36] P. Jonsson and G. Nordh, *Generalised integer programming based on logically defined relations*, in Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS-2006), Springer, Berlin, Heidelberg, 2006, pp. 549–560.
[37] S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson, *The approximability of constraint satisfaction problems*, SIAM J. Comput., 30 (2001), pp. 1863–1920.
[38] F. Kuivinen, *Tight approximability results for the maximum solution equation problem over Z_p*, in Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS-2005), Springer, Berlin, 2005, pp. 628–639.
[39] S. Marchenkov, *Homogeneous algebras*, Problemy Kibernet., 39 (1982), pp. 85–106 (in Russian).
[40] E. Marczewski, *Homogeneous algebras and homogeneous operations*, Fund. Math., 56 (1964), pp. 81–103.
[41] A. Pixley, *Functionally complete algebras generating distributive and permutable classes*, Math. Z., 114 (1970), pp. 361–372.
[42] E. Post, *The two-valued iterative systems of mathematical logic*, Ann. of Math. Stud., 5 (1941), pp. 1–122.
[43] R. Pöschel and L. Kaluznin, *Funktionen- und Relationenalgebren*, DVW, Berlin, 1979.
[44] R. Quackenbush, *A survey of minimal clones*, Aequationes Math., 50 (1995), pp. 3–16.
[45] I. Rosenberg, *Minimal clones* I: *The five types*, in Lectures in Universal Algebra, L. Szabó and Á. Szendrei, eds., North-Holland, Amsterdam, 1986, pp. 405–427.
[46] H. Schnoor and I. Schnoor, *Enumerating all solutions for constraint satisfaction problems*, in Proceedings of the 24th Annual Symposium on Theoretical Aspects (STACS-2007), Springer, Berlin, Heidelberg, 2007, pp. 694–705.

[47] B. Szczepara, *Minimal Clones Generated by Groupoids*, Ph.D. thesis, Université de Móntreal, Montreal, QC, 1996.
[48] Á. Szendrei, *Clones in Universal Algebra*, Seminaries de Mathématiques Supéreiores 99, University of Montreal, Montreal, QC, 1986.
[49] Á. Szendrei, *Idempotent algebras with restrictions on subalgebras*, Acta Sci. Math. (Szeged), 51 (1987), pp. 57–65.
[50] P. van Hentenryck, Y. Deville, and C.-M. Teng, *A generic arc-consistency algorithm and its specializations*, Artificial Intelligence, 57 (1992), pp. 291–321.
[51] G. Woeginger, *An efficient algorithm for a class of constraint satisfaction problems*, Oper. Res. Lett., 30 (2002), pp. 9–16.
[52] D. Zuckerman, *Linear degree extractors and the inapproximability of max clique and chromatic number*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC-2006), 2006, pp. 681–690.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 366–384

EXPONENTIAL SEPARATION OF QUANTUM AND CLASSICAL ONE-WAY COMMUNICATION COMPLEXITY*

ZIV BAR-YOSSEF†, T. S. JAYRAM‡, AND IORDANIS KERENIDIS§

Abstract. We give the first exponential separation between quantum and bounded-error randomized one-way communication complexity. Specifically, we define the Hidden Matching Problem HM_n: Alice gets as input a string $\mathbf{x} \in \{0,1\}^n$, and Bob gets a perfect matching M on the n coordinates. Bob's goal is to output a tuple $\langle i, j, b\rangle$ such that the edge (i, j) belongs to the matching M and $b = x_i \oplus x_j$. We prove that the quantum one-way communication complexity of HM_n is $O(\log n)$, yet any randomized one-way protocol with bounded error must use $\Omega(\sqrt{n})$ bits of communication. No asymptotic gap for one-way communication was previously known. Our bounds also hold in the model of Simultaneous Messages (SM), and hence we provide the first exponential separation between quantum SM and randomized SM with public coins. For a Boolean decision version of HM_n, we show that the quantum one-way communication complexity remains $O(\log n)$ and that the 0-error randomized one-way communication complexity is $\Omega(n)$. We prove that any randomized *linear* one-way protocol with bounded error for this problem requires $\Omega(\sqrt[3]{n \log n})$ bits of communication.

Key words. communication complexity, quantum computation, separation, hidden matching

AMS subject classifications. 68P30, 68Q15, 68Q17, 81P68

DOI. 10.1137/060651835

1. Introduction.

The investigation of the strength and limitations of quantum computing has become an important field of study in theoretical computer science. The celebrated algorithm of Shor [22] for factoring numbers in polynomial time on a quantum computer gives strong evidence that quantum computers are more powerful than classical ones. The further study of the relationship between quantum and classical computing in models like black-box computation, communication complexity, and interactive proof systems contributes to a better understanding of quantum and classical computing.

In this paper we answer an open question about the relative power of quantum *one-way communication protocols*. We describe a problem which can be solved by a quantum one-way communication protocol exponentially faster than any classical one. No asymptotic gap was previously known. We prove a similar result in the model of Simultaneous Messages.

Communication complexity, defined by Yao [23] in 1979, is a central model of computation with numerous applications. It has been used for proving lower bounds

*Received by the editors February 10, 2006; accepted for publication (in revised form) July 10, 2007; published electronically April 23, 2008.

http://www.siam.org/journals/sicomp/38-1/65183.html

†Department of Electrical Engineering, Technion, Haifa 32000, Israel (zivby@ee.technion.ac.il). This work was done in part while the author was at the IBM Almaden Research Center. This author's research was supported by the European Commission Marie Curie International Re-integration Grant.

‡IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (jayram@almaden.ibm.com).

§CNRS - LRI, Université Paris-Sud, 91405 Orsay, France (jkeren@lri.fr). Part of this work was done while the author was visiting the IBM Almaden Research Center. This author's research was supported by ARO grant DAAD19-03-1-0082 and by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as contract 015848.

in many areas including Boolean circuits, time-space tradeoffs, data structures, automata, and formula size. Examples of these applications can be found in the textbook of Kushilevitz and Nisan [15]. A communication complexity problem is defined by three sets X, Y, Z and a relation $\mathcal{R} \subseteq X \times Y \times Z$. Two computationally all-powerful players, Alice and Bob, are given inputs $x \in X$ and $y \in Y$, respectively. Neither of the players has any information about the other player's input. Alice and Bob exchange messages according to a shared *protocol*, until Bob has sufficient information to announce an output $z \in Z$ such that $(x, y, z) \in \mathcal{R}$. The *communication cost* of a protocol is the sum of the lengths of messages (in bits) that Alice and Bob exchange on the worst-case choice of inputs x and y. The *communication complexity* of the problem $\mathcal{R}$ is the cost of the best protocol that computes $\mathcal{R}$ correctly.

One important special case of the above model is *one-way communication complexity* [19, 2, 14], where Alice is allowed to send only one message to Bob, after which Bob announces the output. *Simultaneous Messages (SM)* is a variant in which Alice and Bob cannot communicate directly with each other; instead, each of them sends a single message to a third party, the "referee," who announces the output based on the two messages.

Depending on the kind of allowed protocols, we can define different measures of communication complexity for a problem $\mathcal{R}$. The classical *deterministic* communication complexity of $\mathcal{R}$ is the one described above. In a *bounded-error randomized* protocol with error probability $\delta > 0$, both players have access to *public* random coins, and for any inputs x, y, the output z announced should be correct (i.e., should satisfy $(x, y, z) \in \mathcal{R}$) with probability at least $1 - \delta$ (the probability is over the public random coins). The cost of such a protocol is the number of bits that Alice and Bob exchange on the worst-case choice of inputs and values for the random coins. The randomized communication complexity of $\mathcal{R}$ (w.r.t. δ) is the cost of the optimal randomized protocol for $\mathcal{R}$. In a 0-*error randomized* protocol (also known as Las Vegas protocol) the output announced should be correct with probability 1. The cost of such a protocol is the *expected* number of bits that Alice and Bob exchange on the worst-case choice of inputs. These complexity measures can also be specialized by restricting the communication model to be SM or one-way communication. An interesting variant for randomized protocols in the SM model is when the random coins are restricted to be *private* (i.e., each player has access to his/her own private random coins, and these coins are independent of each other).[1]

Quantum communication complexity, also introduced by Yao [25], apart from being of interest in itself, has been used to prove bounds on quantum formula size, automata, data structures, etc. (e.g., [25, 12, 21]). In this setting, Alice and Bob hold qubits, some of which are initialized to the input. In a communication round, each player can perform some arbitrary unitary operation on his/her part of the qubits and send some of them to the other player. At the end of the protocol they perform a measurement and decide on an outcome. The output of the protocol is required to be correct with probability $1 - \delta$ for some $\delta > 0$. The quantum communication complexity of $\mathcal{R}$ is the number of qubits exchanged in the optimal bounded-error quantum protocol for $\mathcal{R}$. It can be shown that the quantum communication is as powerful as bounded-error randomized communication with private coins,[2] even when

[1] The difference in complexity between public and private coins for the other models is only $O(\log\log(|X||Y|) + \log(1/\delta))$ [16].

[2] As noted earlier, the distinction between public and private coins is significant only for the SM model.

restricted to variants such as one-way communication and SM. It is a natural and important question to ask whether quantum channels can significantly reduce the amount of communication necessary to solve certain problems.

It is known that randomized one-way communication protocols can be much more efficient than deterministic protocols. For example, the equality function on bitstrings of length n can be solved by an $O(1)$ randomized one-way protocol, though its deterministic one-way communication complexity is $\Omega(n)$ (cf. [15]). However, the question of whether quantum one-way communication protocols could be exponentially more efficient than randomized one-way communication protocols remained open. We resolve this in the affirmative, by exhibiting a problem for which the quantum complexity is exponentially smaller than the randomized one.

1.1. Related work. The area of quantum communication complexity was introduced by Yao [25]. Since then, a series of papers have investigated the power and limitations of quantum communication complexity. Buhrman, Cleve, and Wigderson [7] described a relation $\mathcal{R}$ with deterministic communication complexity of $\Theta(n)$ and 0-error quantum communication complexity of $\Theta(\log n)$. However, the bounded-error randomized communication complexity of this problem is $O(1)$. An exponential separation w.r.t. bounded-error randomized protocols was given by Ambainis et al. [4] in the so-called sampling model. However, the separation does not hold in the presence of public coins. Buhrman et al. [6] were able to solve the equality problem in the SM model with a quantum protocol of complexity $O(\log n)$ rather than the $\Theta(\sqrt{n})$ bits necessary in any bounded-error randomized SM protocol with private coins [17, 5]. Again, if we allow the players to share random coins, then equality can be solved classically with $O(1)$ communication.

Raz [20] was the first to show an exponential gap between the quantum and the bounded-error public-coin randomized communication complexity models. He described a relation $\mathcal{P}_1$ with an efficient quantum protocol of complexity $O(\log n)$. He then proved a lower bound of $\Omega(n^{1/4})$ on the classical randomized communication complexity of $\mathcal{P}_1$. Since the quantum protocol given for $\mathcal{P}_1$ uses two rounds, the separation holds only for protocols that use two rounds or more. The definition of $\mathcal{P}_1$ was motivated, in part, by another relation $\mathcal{P}_0$. The latter was first introduced by Kremer [13], who showed that $\mathcal{P}_0$ is a complete problem for quantum one-way communication complexity (in particular, it has an $O(\log n)$ quantum one-way protocol). However, no lower bound is given for the one-way randomized communication complexity of $\mathcal{P}_0$. Proving an exponential separation of classical and quantum one-way communication complexity has been an open question ever since.

Klauck [12] proved that the 0-error quantum one-way communication complexity of *total* functions (i.e., problems $\mathcal{R} \subseteq X \times Y \times Z$, for which *every* $x \in X$ and $y \in Y$ have exactly *one* $z \in Z$ with $(x, y, z) \in \mathcal{R}$) is equal to the classical deterministic one. It is still an open question whether, for total functions, quantum and bounded-error randomized one-way communication complexity are polynomially related.

Subsequent to our work, Aaronson [1] showed that for any Boolean function f, the deterministic one-way communication complexity of f is $O(\log |Y| \cdot Q^1(f) \cdot \log Q^1(f))$, where $Q^1(f)$ is the bounded-error quantum one-way communication complexity of f; namely, if the given communication problem is a Boolean function in which Bob's domain is small, then deterministic one-way communication complexity is almost as efficient as bounded-error quantum one-way communication complexity. Moreover, Gavinsky et al. [10] described a relation which has randomized communication complexity $O(\log n)$ in the SM with public coins model, though its quantum SM commu-

nication complexity is $\Omega(\sqrt{n})$. This separation in the opposite direction from ours shows that the two models are incomparable.

1.2. Our results. Our main result is the definition and analysis of the communication complexity of the Hidden Matching Problem. This provides the first exponential separation between quantum and classical one-way communication complexity.

The HIDDEN MATCHING PROBLEM. Let n be a positive even integer. In the Hidden Matching Problem, denoted HM_n, Alice is given $\mathbf{x} \in \{0,1\}^n$ and Bob is given $M \in \mathcal{M}_n$ ($\mathcal{M}_n$ denotes the family of all possible perfect matchings on n nodes). Their goal is to output a tuple $\langle i, j, b\rangle$ such that the edge (i, j) belongs to the matching M and $b = x_i \oplus x_j$.

This problem is new, and we believe that its definition plays the major role in obtaining our result. The inspiration comes from the work by Kerenidis and de Wolf on Locally Decodable Codes [11]. Let us give the intuition for why this problem is hard for classical communication complexity protocols. Suppose (to make the problem even easier) that Bob's matching M is restricted to be one of n fixed disjoint matchings on $\mathbf{x}$. Bob's goal is to find the value of $x_i \oplus x_j$ for some $(i, j) \in M$. However, since Alice has no information about which matching Bob has, her message needs to contain information about the parity of at least one pair from each matching. Hence, she needs to communicate parities of $\Omega(n)$ different pairs to Bob. It can be shown that such message must be of size $\Omega(\sqrt{n})$. In section 4 we turn this intuition into a proof for the randomized one-way communication complexity of HM_n. We also show that our lower bound is tight by describing a randomized one-way protocol with communication $O(\sqrt{n})$. In this protocol, Alice just sends $O(\sqrt{n})$ random bits of her input. By the birthday paradox, with high probability, Bob can recover the value of at least one of his matching pairs from Alice's message.

Remarkably, this problem remains easy for quantum one-way communication. Alice only needs to send a uniform superposition of her string $\mathbf{x}$, hence communicating only $O(\log n)$ qubits. Bob can perform a measurement on this superposition which depends on the matching M and then output the parity of some pair in M. In section 3 we describe the quantum protocol in more detail.

In section 5 we show that HM_n also provides the first exponential separation between quantum SM and randomized SM with public coins. Previously such a bound was known only in the private-coin model. This result, together with the exponential separation in the opposite direction proved by Gavinsky et al. [10], shows that, in fact, the models of quantum SM and public-coin randomized SM are incomparable.

Our main result exhibits a separation between quantum and classical one-way communication complexity for a relation. Ideally, one would like to prove such a separation for the most basic type of problems—total Boolean functions. The best-known separation between quantum and classical communication complexity (even for an arbitrary number of rounds) for such functions is only quadratic [7]. It is still conceivable that for total functions, the two models are polynomially related. Raz's result [20] shows an exponential gap for a partial Boolean function (i.e., a Boolean function that is defined only on a subset of the domain $\mathcal{X} \times \mathcal{Y}$) and for two-way communication protocols.

We consider a partial Boolean function induced by the Hidden Matching Problem, defined below. In the definition we view each matching $M \in \mathcal{M}_n$ as an $\frac{n}{2} \times n$ edge-vertex incidence matrix. For two Boolean vectors $\mathbf{v}, \mathbf{w}$, we denote by $\mathbf{v} \oplus \mathbf{w}$ the vector obtained by taking the XOR $\mathbf{v}$ and $\mathbf{w}$ coordinatewise. For a bit $b \in \{0,1\}$, we denote by $\mathbf{b}$ the vector all of whose entries are b.

The Boolean Hidden Matching Problem. Let n be a positive integer multiple of 4. In the Boolean Hidden Matching Problem, denoted BHM_n, Alice is given $\mathbf{x} \in \{0,1\}^n$ and Bob is given $M \in \mathcal{M}_n$ and $\mathbf{w} \in \{0,1\}^{n/2}$, which satisfy the following promise: either $M\mathbf{x} \oplus \mathbf{w} = \mathbf{1}$ (a Yes instance) or $M\mathbf{x} \oplus \mathbf{w} = \mathbf{0}$ (a No instance). Their goal is to decide which of the two cases holds.

With a slight modification, the quantum protocol for HM_n also solves BHM_n using $O(\log n)$ qubits. We believe that BHM_n should also exhibit an exponential gap in its quantum and classical one-way communication complexity.

Conjecture 1. *The one-way randomized communication complexity of the Boolean Hidden Matching Problem is* $n^{\Omega(1)}$.

Although we were unable to resolve the above conjecture, we give a strong indication for our belief with two lower bounds. First, we prove an $\Omega(n)$ lower bound on the 0-error randomized one-way communication complexity of BHM_n. We then show that a natural class of randomized bounded-error protocols require $\widetilde{\Omega}(\sqrt[3]{n})$ bits of communication to compute BHM_n. The protocols we refer to are *linear*; that is, Alice and Bob use the public coins to choose a random matrix $\mathbf{A}$, and Alice's message on input $\mathbf{x}$ is simply $\mathbf{Ax}$. These protocols are natural for our problem, because Bob needs to compute a linear transformation of Alice's input. In particular, the $O(\sqrt{n})$ communication protocol that we described earlier is trivially a linear protocol. We note that the study of linear protocols is not unique to our problem. For example, one-way linear protocols for the indexing function can be viewed as linear *random access codes* [3]. Thus, lower bounds for linear protocols preclude the feasibility of a natural class of protocols. Generalizing this lower bound to the case of nonlinear randomized protocols still remains an open problem. These results are described in section 6.

2. Preliminaries.

2.1. Information theory. Throughout the paper we use basic notions and facts from information theory, which we briefly review next. We refer the reader to the textbook of Cover and Thomas [8] for details and proofs.

We deal only with finite discrete probability spaces. The distribution of a random variable X is denoted by μ_X, and we let $\mu_X(x) \stackrel{\text{def}}{=} \Pr[X = x]$. The *entropy* of X (or, equivalently, of μ_X) is $\mathrm{H}(X) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} \mu_X(x) \log \frac{1}{\mu_X(x)}$, where $\mathcal{X}$ is the domain of X. The entropy of a Bernoulli random variable with probability of success p is called the *binary entropy function* of p and is denoted $H_2(p)$. The *joint entropy* of X and Y is the entropy of the joint distribution μ_{XY} of X and Y. The *conditional entropy* of X given an event A, denoted $\mathrm{H}(X|A)$, is the entropy of the conditional distribution of μ_X given A. The *conditional entropy* of X given Y is $\mathrm{H}(X|Y) \stackrel{\text{def}}{=} \sum_{y \in \mathcal{Y}} \mu_Y(y)\, \mathrm{H}(X|Y = y)$, where $\mathcal{Y}$ is the domain of Y. The *mutual information* between X and Y is $\mathrm{I}(X;Y) \stackrel{\text{def}}{=} \mathrm{H}(X) - \mathrm{H}(X|Y) = \mathrm{H}(Y) - \mathrm{H}(Y|X)$. The *conditional mutual information* between X and Y given Z is $\mathrm{I}(X;Y|Z) = \mathrm{H}(X|Z) - \mathrm{H}(X|Y,Z) = \mathrm{H}(Y|Z) - \mathrm{H}(Y|X,Z)$.

Some basic properties of entropy and mutual information we are using in this paper are given in the following theorem.

Theorem 2.1. *Let X, Y, Z be random variables.*

1. $\mathrm{H}(X) \le \log |\mathit{supp}(X)|$, *where $\mathit{supp}(X)$ is the support of X. Equality holds if and only if X is uniform on $\mathit{supp}(X)$.*
2. *Conditioning reduces entropy:* $\mathrm{H}(X|Y) \le \mathrm{H}(X)$. *Equality holds if and only if X, Y are independent.*

3. *Entropy subadditivity:* $\mathrm{H}(X,Y) \le \mathrm{H}(X) + \mathrm{H}(Y)$. *Equality holds if and only if* X, Y *are independent.*
4. *Data processing inequality* I: *for any* k*-to-*1 *function* f, $\mathrm{H}(X) \le \mathrm{H}(f(X)) + \log k$.
5. *Data processing inequality* II: *for any function* f, $\mathrm{I}(X; f(Y)) \le \mathrm{I}(X;Y)$. *Equality holds if* f *is* 1-1.
6. *Chain rule for mutual information:* $\mathrm{I}(X; Y, Z) = \mathrm{I}(X;Y) + \mathrm{I}(X; Z|Y)$.
7. $\mathrm{I}(X;Y) = 0$ *if and only if* X, Y *are independent.*
8. *If* X, Y *are jointly independent of* Z, *then* $\mathrm{I}(X; Y|Z) = \mathrm{I}(X;Y)$.
9. *For any positive integers* n *and* $m \le n/2$, $\sum_{i=0}^{m} \binom{n}{i} \le 2^{nH_2(m/n)}$.

We will also use the following theorems.

THEOREM 2.2 (Fano's inequality). *Let* X *be a binary random variable, and let* Y *be any random variable on a domain* $\mathcal{Y}$. *Let* $f : \mathcal{Y} \to \{0,1\}$ *be a prediction function, which tries to predict the value of* X *based on an observation of* Y. *Let* $\delta \stackrel{\text{def}}{=} \Pr(f(Y) \neq X)$ *be the error probability of the prediction function. Then,* $H_2(\delta) \ge \mathrm{H}(X|Y)$.

THEOREM 2.3. *Let* $C \subseteq \{0,1\}^*$ *be a finite prefix-free code (i.e., no codeword in* C *is a prefix of any other codeword in* C*). Let* X *be a random variable corresponding to a uniformly chosen codeword in* C. *Then,* $\mathrm{H}(X) \le E(|X|)$.

2.2. Quantum computation. We explain the standard notation of quantum computing and describe the basic notions that will be useful in this paper. For more details we refer the reader to the textbook of Nielsen and Chuang [18].

Let H denote a 2-dimensional complex vector space, equipped with the standard inner product. We pick an orthonormal basis for this space, label the two basis vectors $|0\rangle$ and $|1\rangle$, and for simplicity identify them with the vectors $\binom{1}{0}$ and $\binom{0}{1}$, respectively. A *qubit* is a unit length vector in this space and thus can be expressed as a linear combination of the basis states:

$$\alpha_0|0\rangle + \alpha_1|1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}.$$

Here α_0, α_1 are complex *amplitudes*, and $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

An m*-qubit system* is a unit vector in the m-fold tensor space $H \otimes \cdots \otimes H$. The 2^m basis states of this space are the m-fold tensor products of the states $|0\rangle$ and $|1\rangle$. For example, the basis states of a 2-qubit system are the four 4-dimensional unit vectors $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$, $|1\rangle \otimes |0\rangle$, and $|1\rangle \otimes |1\rangle$. We abbreviate, e.g., $|1\rangle \otimes |0\rangle$ to $|1\rangle|0\rangle$, or $|1,0\rangle$, or $|10\rangle$, or even $|2\rangle$ (since 2 is 10 in binary). With these basis states, an m-qubit state $|\phi\rangle$ is a 2^m-dimensional complex unit vector

$$|\phi\rangle = \sum_{i \in \{0,1\}^m} \alpha_i |i\rangle.$$

We use $\langle\phi| = |\phi\rangle^*$ to denote the conjugate transpose of the vector $|\phi\rangle$, and $\langle\phi|\psi\rangle = \langle\phi| \cdot |\psi\rangle$ for the inner product between states $|\phi\rangle$ and $|\psi\rangle$. These two states are *orthogonal* if $\langle\phi|\psi\rangle = 0$. The *norm* of $|\phi\rangle$ is $\|\phi\| = \sqrt{\langle\phi|\phi\rangle}$.

Let $|\phi\rangle$ be an m-qubit state and $B = \{|b_1\rangle, \ldots, |b_{2^m}\rangle\}$ an orthonormal basis of the m-qubit space. A measurement of the state $|\phi\rangle$ in the B basis means that we apply the projection operators $P_i = |b_i\rangle\langle b_i|$ to $|\phi\rangle$. The resulting quantum state is $|b_i\rangle$ with probability $p_i = |\langle\phi|b_i\rangle|^2$.

3. The quantum upper bound. We present a quantum protocol for the Hidden Matching Problem with communication complexity of $O(\log n)$ qubits. Let $\mathbf{x} = x_1 \ldots x_n$ be Alice's input and $M \in \mathcal{M}_n$ be Bob's input.

QUANTUM PROTOCOL FOR HM_n.

1. Alice sends the state $|\psi\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} (-1)^{x_i} |i\rangle$.
2. Bob performs a measurement on the state $|\psi\rangle$ in the orthonormal basis $B = \{\frac{1}{\sqrt{2}}(|k\rangle \pm |\ell\rangle) \mid (k,\ell) \in M\}$.

The probability that the outcome of the measurement is a basis state $\frac{1}{\sqrt{2}}(|k\rangle + |\ell\rangle)$ is

$$\left| \langle \psi | \frac{1}{\sqrt{2}} (|k\rangle + |\ell\rangle) \rangle \right|^2 = \frac{1}{2n} ((-1)^{x_k} + (-1)^{x_\ell})^2.$$

This equals $2/n$ if $x_k \oplus x_\ell = 0$, and 0 otherwise. Similarly for the states $\frac{1}{\sqrt{2}}(|k\rangle - |\ell\rangle)$ we have that $|\langle \psi | \frac{1}{\sqrt{2}}(|k\rangle - |\ell\rangle)\rangle|^2$ equals 0 if $x_k \oplus x_\ell = 0$ and $2/n$ if $x_k \oplus x_\ell = 1$. Hence, if the outcome of the measurement is a state $\frac{1}{\sqrt{2}}(|k\rangle + |\ell\rangle)$, then Bob knows with certainty that $x_k \oplus x_\ell = 0$ and outputs $\langle k, \ell, 0 \rangle$. If the outcome is a state $\frac{1}{\sqrt{2}}(|k\rangle - |\ell\rangle)$, then Bob knows with certainty that $x_k \oplus x_\ell = 1$ and hence outputs $\langle k, \ell, 1 \rangle$. Note that the measurement depends only on Bob's input and that the algorithm is 0-error.

This protocol can be tweaked to also solve BHM_n: after obtaining the value $\langle k, \ell, c \rangle$ from that protocol, where (k, ℓ) is the ith pair in Bob's input matching M, Bob outputs $w_i \oplus c$. Note that if $c = x_k \oplus x_\ell$, then $w_i \oplus c$ equals the desired bit b.

Remark. As mentioned above, the inspiration for this problem comes from Locally Decodable Codes. We can think of a 2-query Locally Decodable Code as a code $C : \{0,1\}^n \to \{0,1\}^m$ with the property that for every index $k \in [n]$ there exists a matching M_k on the coordinates of $C(\mathbf{x})$, such that for every pair $(i,j) \in M_k$, $x_k = C(\mathbf{x})_i \oplus C(\mathbf{x})_j$. We can cast this problem as a communication problem, by letting Alice have the codeword $C(\mathbf{x})$, letting Bob have the index k and the corresponding matching M_k, and setting the goal for Bob to output $C(\mathbf{x})_i \oplus C(\mathbf{x})_j$ for some $(i,j) \in M_k$. This gives rise to our Hidden Matching Problem. The fact that a uniform superposition of $C(\mathbf{x})$ is sufficient to compute the parity of some pair in each matching was used by Kerenidis and de Wolf [11] to prove a lower bound on the length of classical 2-query Locally Decodable Codes.

4. The randomized lower bound. We prove an $\Omega(\sqrt{n})$ lower bound on the one-way communication complexity of the Hidden Matching Problem.

THEOREM 4.1. *Any one-way randomized protocol for computing* HM_n *with error probability less than* $1/8$ *requires* $\Omega(\sqrt{n})$ *bits of communication.*

Proof. Using Yao's lemma [24], in order to prove the lower bound, it suffices to construct a "hard" distribution μ over instances of HM_n and prove a lower bound for deterministic one-way protocols whose distributional error w.r.t. μ is at most δ, where $\delta < 1/8$. This is accomplished as follows.

For a deterministic one-way protocol Π and for inputs $\mathbf{x}$ and M to Alice and Bob, respectively, we denote by $\Pi(\mathbf{x}, M)$ the output of the protocol on $\mathbf{x}$ and M. This output is a triple (i,j,b), where $i,j \in [n]$ and b is a bit. The protocol is correct on $(\mathbf{x}, M)$ if $(i,j,b) \in \mathrm{HM}_n(\mathbf{x}, M)$. That is, (i,j) is an edge in M and $b = \mathbf{x}_i \oplus \mathbf{x}_j$. Since Bob knows the matching M and also is the one who announces the output $\Pi(\mathbf{x}, M)$, we can assume without loss of generality that the pair (i,j) is always an edge in M. Therefore, an error can occur only if $b \neq \mathbf{x}_i \oplus \mathbf{x}_j$.

The following notation will be used in the proof. For a pair of random inputs U and V to Alice and Bob, respectively, let $\mathcal{E}rr_\Pi(U,V)$ denote the distributional error of Π when the inputs are drawn according to the joint distribution μ of U and V.

That is,

$$\mathcal{E}rr_{\Pi}(U,V) = \Pr_{(U,V)\sim\mu}(\Pi(U,V) \notin \mathrm{HM}_n(U,V)).$$

We define the hard distribution μ as follows: let $\mathbf{X}$ be a uniformly chosen bitstring in $\{0,1\}^n$; let $\mathbf{M}$ be an independent and uniformly chosen perfect matching in $\mathcal{M}$, where $\mathcal{M}$ is any set of $m = \Omega(n)$ pairwise edge-disjoint matchings. (For example, let $m = n/2$ and $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, where M_i is defined by the edge set $\{(j, m + (j + i) \bmod m) \mid j = 0, \ldots, m-1\}$.) Fix any deterministic protocol Π whose distributional error on μ is at most δ, i.e., $\mathcal{E}rr_{\Pi}(\mathbf{X}, \mathbf{M}) \leq \delta$.

For each $\mathbf{x}$, let $\hat{e}_{\mathbf{x}} = \mathcal{E}rr_{\Pi}(\mathbf{x}, \mathbf{M})$ denote the distributional error of Π on the fixed input $\mathbf{x}$ to Alice and a random input $\mathbf{M}$ to Bob. Note that $\mathbb{E}[\hat{e}_{\mathbf{X}}] = \mathcal{E}rr_{\Pi}(\mathbf{X}, \mathbf{M}) \leq \delta$. By Markov's inequality, $\Pr[\hat{e}_{\mathbf{X}} \geq 2\delta] \leq 1/2$. Since $\mathbf{X}$ is uniformly distributed, this means that $\hat{e}_{\mathbf{x}} \leq 2\delta$ for at least half of the individual $\mathbf{x}$'s.

Let $A(\mathbf{x})$ denote the message that Alice sends on input $\mathbf{x}$ in the protocol Π. For every message τ, define

$$S_\tau = \{\mathbf{x} \mid A(\mathbf{x}) = \tau \text{ and } \hat{e}_{\mathbf{x}} \leq 2\delta\}.$$

Since $\hat{e}_{\mathbf{x}} \leq 2\delta$ for at least half of the $\mathbf{x}$'s, we have

$$\sum_{\tau} |S_\tau| \geq 2^{n-1}. \tag{1}$$

On the other hand, we will prove the following upper bound on the size of S_τ.

LEMMA 4.2. *For every message τ,*

$$|S_\tau| \leq 2^{n - \Omega(\sqrt{n})}.$$

Before we prove the lemma observe that (1) and Lemma 4.2 imply that the number of distinct τ's is at least $2^{\Omega(\sqrt{n})}$. Thus, the communication cost of Π is $\Omega(\sqrt{n})$, proving the theorem. □

Proof of Lemma 4.2. Fix a message τ of Alice, and let $\mathbf{X}_\tau$ be uniformly distributed in S_τ. Define $e_\tau = \mathcal{E}rr_{\Pi}(\mathbf{X}_\tau, \mathbf{M})$ to be the distributional error of Π on the random inputs $(\mathbf{X}_\tau, \mathbf{M})$. By the definition of S_τ, $\hat{e}_{\mathbf{x}} \leq 2\delta$ for every input $\mathbf{x}$ in S_τ. It follows that $e_\tau \leq 2\delta$ as well.

For each matching $M \in \mathcal{M}$, let $e_{\tau,M} = \mathcal{E}rr_{\Pi}(\mathbf{X}_\tau, M)$. Note that $\mathbb{E}[e_{\tau,\mathbf{M}}] = e_\tau$. By Markov's inequality,

$$\Pr(e_{\tau,\mathbf{M}} \geq 2e_\tau) \leq \frac{1}{2}.$$

Therefore, for at least half of the matchings M, $e_{\tau,M} \leq 2e_\tau \leq 4\delta$ (recall that $\mathbf{M}$ is uniform on the matchings). Let $\mathcal{M}'$ be the set of matchings M for which $e_{\tau,M} \leq 2e_\tau$. By the above, $|\mathcal{M}'| \geq |\mathcal{M}|/2 = m/2 = \Omega(n)$.

The output of Π depends only on τ and on the input to Bob. Therefore, for each fixed matching M, the output of the protocol on inputs $\mathbf{X}_\tau$ and M is a constant triple (i_M, j_M, b_M), where (i_M, j_M) is an edge in M and b_M is a bit. Let G_τ be the graph defined by the set of edges $\{(i_M, j_M) | M \in \mathcal{M}'\}$. The graph has n nodes and $|\mathcal{M}'| \geq \Omega(n)$ edges. (Note that here we use the fact the matchings are pairwise disjoint; hence edges corresponding to different matchings must be distinct.) Let $\mathbf{u} \in \{0,1\}^{|\mathcal{M}'|}$ be the sequence of bits b_M for $M \in \mathcal{M}'$.

We will use the following proposition to extract an acyclic subgraph of G_τ.

PROPOSITION 4.3. *Every graph G with t edges has an acyclic subgraph with $\Omega(\sqrt{t})$ edges.*

The proof of the proposition is given below. Then let F be an acyclic subgraph of G_τ with $|F| = \Omega(\sqrt{n})$ (here, $|F|$ denotes the number of edges in F). Let N denote the $n \times |F|$ vertex-edge incidence matrix of F, and let $\mathbf{v} \in \{0,1\}^{|F|}$ denote the projection of $\mathbf{u}$ on F. For any $\mathbf{x} \in \{0,1\}^n$, the vector $\mathbf{x}N$ taken over $GF[2]$ is of length $|F|$.

Since F is a subgraph of G_τ and each edge of G_τ is associated with a unique matching $M \in \mathcal{M}'$, we can label the coordinates of the vectors $\mathbf{x}N$ and $\mathbf{v}$ by matchings. Let $\mathcal{M}_F$ be the set of matchings $M \in \mathcal{M}'$, for which $(i_M, j_M) \in F$. For a matching $M \in \mathcal{M}_F$, we know $\mathbf{v}_M = b_M$. On the other hand, $(\mathbf{x}N)_M = \mathbf{x}_{i_M} \oplus \mathbf{x}_{j_M}$. Thus, disagreements of the vectors $\mathbf{v}$ and $\mathbf{x}N$ correspond to errors of Π on the input $\mathbf{x}$. In fact, the number of errors of Π on inputs of the form $(\mathbf{x}, M)$, where M varies over $\mathcal{M}_F$, is exactly the *Hamming distance* between the vectors $\mathbf{x}N$ and $\mathbf{v}$.

Let $\mathbf{F}$ be a uniformly chosen matching from $\mathcal{M}_F$ and consider $\mathcal{E}rr_\Pi(\mathbf{X}_\tau, \mathbf{F})$—the distributional error of Π on inputs drawn from $\mathbf{X}_\tau$ and $\mathbf{F}$ independently. Let $h(\mathbf{x}N, \mathbf{v})$ denote the *relative Hamming distance* between the vectors $\mathbf{x}N$ and $\mathbf{v}$. (That is, $h(\mathbf{x}N, \mathbf{v}) = (1/|F|) \cdot |\{M \in \mathcal{M}_F | (\mathbf{x}N)_M \neq \mathbf{v}_M\}|$.) By the above observation, $\mathcal{E}rr_\Pi(\mathbf{X}_\tau, \mathbf{F}) = \mathbb{E}[h(\mathbf{X}_\tau N, \mathbf{v})]$.

There is another way to look at $\mathcal{E}rr_\Pi(\mathbf{X}_\tau, \mathbf{F})$. Recall that for a fixed matching M, $e_{\tau,M} = \mathcal{E}rr_\Pi(\mathbf{X}_\tau, M)$. Recall also that for every matching $M \in \mathcal{M}'$, $e_{\tau,M} \leq 2e_\tau$. Hence, $\mathcal{E}rr_\Pi(\mathbf{X}_\tau, \mathbf{F}) = \mathbb{E}[e_{\tau,\mathbf{F}}] \leq 2e_\tau$. We conclude that $\mathbb{E}[h(\mathbf{X}_\tau N, \mathbf{v})] \leq 2e_\tau$.

Next, we resort to another proposition, whose proof appears below.

PROPOSITION 4.4. *Let Z be a random variable on $\{0,1\}^k$ and suppose there exists a vector $\mathbf{v} \in \{0,1\}^k$ such that $\mathbb{E}[h(Z, \mathbf{v})] \leq \epsilon$ with $0 \leq \epsilon \leq 1/2$. Then,* $\mathrm{H}(Z) \leq k \cdot H_2(\epsilon)$.

It follows from the above proposition and the fact $\mathbb{E}[h(\mathbf{X}_\tau N, \mathbf{v})] \leq 2e_\tau$ that $\mathrm{H}(\mathbf{X}_\tau N) \leq |F| \cdot H_2(2e_\tau)$. Note that $2e_\tau \leq 4\delta < 1/2$ as required by the proposition.

The matrix N has full rank, because it is the vertex-edge incidence matrix of an acyclic graph. It follows that the dimension of the null space of N is $n - \mathrm{rank}(N) = n - |F|$. Hence, the mapping $\mathbf{x} \mapsto \mathbf{x}N$ is a $2^{n-|F|}$-to-1 mapping. By the first data processing inequality (Theorem 2.1, part 4), since $\mathrm{H}(\mathbf{X}_\tau N) \leq |F| \cdot H_2(2e_\tau)$,

$$\mathrm{H}(\mathbf{X}_\tau) \leq |F| \cdot H_2(2e_\tau) + \log(2^{n-|F|}) = n - |F| \cdot (1 - H_2(2e_\tau)) = n - \Omega(\sqrt{n}),$$

since $H_2(2e_\tau) \leq H_2(4\delta) < 1$ when $2e_\tau \leq 4\delta < 1/2$. This completes the proof of the lemma. □

Proof of Proposition 4.3. Let $C_1, C_2, \ldots, C_s$ be the connected components of G, and let $b_1, b_2, \ldots, b_s$ be the number of edges they have ($b_1 + b_2 + \cdots + b_s = t$). C_i has $\Omega(\sqrt{b_i})$ nodes and thus has a spanning tree with $\Omega(\sqrt{b_i})$ edges. Therefore, G contains a forest F with at least $\sum_i \Omega(\sqrt{b_i}) = \Omega(\sqrt{t})$ edges, using the fact that $\sqrt{u} + \sqrt{v} \geq \sqrt{u+v}$. F is the desired acyclic subgraph. □

Proof of Proposition 4.4. We will prove the proposition for the case $\mathbf{v} = 0^k$. The generalization to other $\mathbf{v}$'s is straightforward.

Let $(Z_1, \ldots, Z_k)$ be the coordinates of Z. Each Z_i is a Bernoulli random variable, and we let $\alpha_i = \Pr[Z_i = 1]$. We have

$$\mathbb{E}[h(Z, 0^k)] = \frac{1}{k} \sum_{i=1}^{k} \mathbb{E}[Z_i] = \frac{1}{k} \sum_{i=1}^{k} \Pr(Z_i = 1) = \frac{1}{k} \sum_{i=1}^{k} \alpha_i. \tag{2}$$

Applying the subadditivity of entropy (Theorem 2.1, part 3) and the concavity of the binary entropy function,

$$\mathrm{H}(Z) \le \sum_{i=1}^{k} \mathrm{H}(Z_i) = \sum_{i=1}^{k} H_2(\alpha_i) \le k \cdot H_2\left(\frac{1}{k}\sum_{i=1}^{k}\alpha_i\right). \tag{3}$$

Combining (2) and (3), we obtain

$$\mathrm{H}(Z) \le k \cdot H_2(\mathbb{E}[h(Z, 0^k)]) \le k \cdot H_2(\epsilon),$$

since $\mathbb{E}[h(Z, 0^k)] \le \epsilon \le 1/2$ and $H_2(\cdot)$ is nondecreasing in $[0, 1/2]$. □

Next, we describe a public-coin randomized protocol of complexity $O(\sqrt{n})$ for HM_n. Alice uses the shared random string to pick $O(\sqrt{n})$ locations in $[n]$ and sends the corresponding bits to Bob. A standard birthday paradox argument shows that these bits include the end-points of at least one edge of the matching with constant probability. This shows that our lower bound is tight, and thus we have the following theorem.

THEOREM 4.5. *The randomized one-way communication complexity of* HM_n *is* $\Theta(\sqrt{n})$.

5. An exponential separation for Simultaneous Messages. Recall that in the model of SM, Alice and Bob both send a single message to a referee, after which he computes the output. We prove an exponential separation in this model between quantum and public-coin randomized communication complexity. To this end, we use a restricted version of the Hidden Matching Problem.

The RESTRICTED HIDDEN MATCHING PROBLEM (RHM_n).

Let n be a positive even integer. In the Restricted Hidden Matching Problem, fix $\mathcal{M}$ to be any set of $m = \Omega(n)$ pairwise edge-disjoint matchings. Alice is given $\mathbf{x} \in \{0,1\}^n$, and Bob is given $M \in \mathcal{M}$. Their goal is to output a tuple $\langle i, j, b\rangle$ such that the edge (i, j) belongs to the matching M and $b = x_i \oplus x_j$.

The lower bound we proved for HM_n in the model of one-way communication (Theorem 4.1) is in fact a lower bound for RHM_n. This lower bound holds also in the SM model since this model is no more powerful than one-way communication (cf. [14]). Hence, we have the following theorem.

THEOREM 5.1. *Any public-coin SM protocol for computing* RHM_n *with error probability at most* $1/8$ *requires* $\Omega(\sqrt{n})$ *bits of communication.*

On the other hand, the Restricted Hidden Matching Problem can be solved by a quantum protocol with only $O(\log n)$ communication. Bob sends the index of his matching to the referee using $O(\log n)$ bits, and Alice sends a superposition of her input string using $O(\log n)$ qubits, similarly to the one-way protocol. Since the referee knows Bob's input matching M, he can perform the same measurement Bob performed in the one-way protocol and compute the XOR of some pair in the matching.

6. The complexity of Boolean Hidden Matching.

6.1. Lower bound for 0-error protocols. In order to prove the lower bound for 0-error randomized one-way protocols, we note the following characterization of such protocols for partial functions. Let $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1, *\}$ be a partial Boolean function. We say that the input (x, y) is *legal* if $f(x, y) \ne *$. A protocol for f is required to be correct only on legal inputs; it is allowed to output arbitrary answers on illegal inputs. The *confusion graph* G_f of f is a graph whose vertex set is $\mathcal{X}$; (x, x') is an edge in G_f if and only if there exists a y such that both (x, y) and (x', y) are legal inputs and $f(x, y) \ne f(x', y)$.

It is known [15] that the deterministic one-way communication complexity of f is $\log \chi(G_f) + O(1)$, where $\chi(G_f)$ is the chromatic number of the graph G_f. We will obtain a lower bound on the 0-error randomized one-way communication complexity via another measure on G_f. For any graph $G = (V, E)$, let

$$\theta(G) = \max_{W \subseteq V} \frac{|W|}{\alpha(G_W)},$$

where G_W is the subgraph of G induced on W and $\alpha(G_W)$ is the independence number of G_W. It is easy to see that $\chi(G) \geq \theta(G)$. The following theorem gives a lower bound on the 0-error communication complexity of f in terms of $\theta(G_f)$.

THEOREM 6.1. *The 0-error randomized one-way communication complexity of any partial Boolean function f is at least $\Omega(\log \theta(G_f))$.*

Proof. Let $G_f = (V, E)$ and let $W \subseteq V$ achieve the maximum for $\theta(G_f)$. Define μ to be the uniform distribution on W.

Suppose Π is a randomized 0-error one-way protocol for f with public randomness R, and whose cost is $c+1$ (Bob just outputs a bit which is the last bit of the transcript). Let $A(\mathbf{x}, R)$ be the message sent by Alice on input $\mathbf{x}$, and let $B(\tau, y, R)$ be the output of the protocol given by Bob on input y when the message sent by Alice is τ. For any legal input (x, y), we have $\mathbb{E}[|A(x, R)|] \leq c$, and $\Pr[B(A(x, R), y, R) = f(x, y)] = 1$.

Let X be a random input for Alice whose distribution is μ. Then $\mathbb{E}[|A(X, R)|] \leq c$, where the randomness is now over both X and R. Therefore, there exists a choice r^* for R such that $\mathbb{E}[|A(X, r^*)|] \leq c$. Define a deterministic protocol where $A'(x) = A(x, r^*)$ and $B'(\tau, y) = B(\tau, y, r^*)$. Note that this protocol correctly computes f and $\mathbb{E}[|A'(X)|] \leq c$. Let T be the set of messages sent by Alice in this new protocol. For any message $\tau \in T$, define $S_\tau = \{x \in W : A'(x) = \tau\}$. By the definition of G_f, it follows that S_τ is an independent set, so $|S_\tau| \leq \alpha(G_W)$. Therefore, the entropy of the random variable $A'(X)$ satisfies

$$\begin{aligned} \mathrm{H}(A'(X)) &= \sum_{\tau \in T} \frac{|S_\tau|}{|W|} \log\left(\frac{|W|}{|S_\tau|}\right) \\ &\geq \sum_{\tau \in T} \frac{|S_\tau|}{|W|} \log\left(\frac{|W|}{\alpha(G_W)}\right) = \log \theta(G_f), \end{aligned} \tag{4}$$

because the S_τ's partition W.

Finally, if we assume that the messages are prefix-free (which can be achieved with a constant factor blow-up in the communication cost), then $\mathbb{E}[|A'(X)|] \geq \mathrm{H}(A'(X))$ (Theorem 2.3). It follows from (4) that $c = \Omega(\log \theta(G_f))$. ∎

We use this characterization to prove the lower bound for BHM_n.

THEOREM 6.2. *Let $n = 4p$, where p is prime. Then, the 0-error randomized one-way communication complexity of* BHM_n *is $\Omega(n)$.*

Proof. Let f denote the partial function BHM_n. The vertex set of the confusion graph G_f is $\{0,1\}^n$. We next show that $(\mathbf{x}, \mathbf{x}')$ is an edge in G_f if and only if the Hamming distance between $\mathbf{x}$ and $\mathbf{x}'$ is exactly $n/2$.

Suppose $(\mathbf{x}, \mathbf{x}')$ is an edge in G_f. Therefore, there exist a matching M and a vector $\mathbf{w}$, so that $M\mathbf{x} \oplus \mathbf{w} = \mathbf{0}$ and $M\mathbf{x}' \oplus \mathbf{w} = \mathbf{1}$, or vice versa. That means that for every edge $(i, j) \in M$, $x_i \oplus x_j \neq x'_i \oplus x'_j$, and thus $\mathbf{x}, \mathbf{x}'$ agree on one of the position i, j and disagree on the other. Hence, the Hamming distance between $\mathbf{x}$ and $\mathbf{x}'$ is exactly $n/2$. Conversely, given two strings $\mathbf{x}, \mathbf{x}'$ of Hamming distance $n/2$, let M be

any matching between the positions on which $\mathbf{x}, \mathbf{x}'$ agree and the positions on which they disagree. Let $\mathbf{w} = M\mathbf{x}$. Clearly, $M\mathbf{x} \oplus \mathbf{w} = \mathbf{0}$. For each edge (i, j) in M we have $x_i \oplus x_j \neq x'_i \oplus x'_j$, and therefore $M\mathbf{x}' \oplus \mathbf{w} = \mathbf{1}$, implying that $(\mathbf{x}, \mathbf{x}')$ is an edge in G_f.

If $n/2$ is odd, G_f is the bipartite graph between the even and odd parity vertices. Therefore, G_f is 2-colorable, implying that f has a $O(1)$ protocol (Alice just sends the parity of her input). We will show that the situation changes dramatically when n is a prime multiple of 4.

PROPOSITION 6.3 (Frankl and Wilson [9]). *Let $m = 4p - 1$, where p is prime. Define the graph $G = (V, E)$ where $V = \{A \subseteq [m] : |A| = 2p - 1\}$, and $(A, B) \in E$ if and only if $|A \cap B| = p - 1$. Then,*

$$\alpha(G) \leq \sum_{i=0}^{p-1} \binom{m}{i}.$$

Let $m = 4p - 1 = n - 1$, and let G be the graph defined by Proposition 6.3. We claim that G is isomorphic to a vertex-induced subgraph of the confusion graph G_f: for every vertex A in G, the corresponding vertex in G_f is the characteristic vector of the set $A \cup \{4p\}$. Let V denote the vertex set of G; it follows that $\theta(G_f) \geq |V|/\alpha(G)$.

We have $|V| = \binom{m}{2p-1} \approx 2^m/\sqrt{m}$, and by Proposition 6.3 and Theorem 2.1, part 9, $\alpha(G) \leq 2^{mH_2(\gamma)}$, where H_2 is the binary entropy function and $\gamma = (p-1)/(4p-1) \leq 1/4$. The result now follows from Theorem 6.1. □

6.2. Lower bound for linear randomized protocols. In this section, we study a natural class of randomized bounded-error protocols for BHM_n and show a $\tilde{\Omega}(\sqrt[3]{n})$ communication lower bound for them.

DEFINITION 6.4 (linear protocols). *A deterministic one-way communication complexity protocol is called* linear *if for any input $\mathbf{x} \in \{0,1\}^n$, Alice's message on $\mathbf{x}$ is of the form $A\mathbf{x}$, where A is a fixed $c \times n$ Boolean matrix.*

A public-coin one-way protocol is linear *if for any input $\mathbf{x} \in \{0,1\}^n$, Alice's message on $\mathbf{x}$ is of the form $\mathbf{Ax}$, where $\mathbf{A}$ is a random $c \times n$ Boolean matrix chosen using the public random bits.*

THEOREM 6.5. *Let n be a positive integer multiple of 4 and let $0 < \delta < 1/2$ be a constant bounded away from $1/2$. Then, any δ-error public-coin one-way linear protocol for BHM_n requires $\Omega(\sqrt[3]{n \log n})$ bits of communication.*

Proof. Using Yao's lemma [24], to prove the lower bound, it suffices to construct a "hard" distribution μ over instances of BHM_n and prove a distributional lower bound w.r.t. deterministic one-way linear protocols. We define μ as follows: let $\mathbf{X}$ be a uniformly chosen bitstring in $\{0,1\}^n$; let $\mathbf{M}$ be a uniformly chosen perfect matching in $\mathcal{M}_n$; and let B be a uniformly chosen bit. $\mathbf{W}$ is a random bitstring in $\{0,1\}^{n/2}$, defined as $\mathbf{W} \stackrel{\text{def}}{=} \mathbf{MX} \oplus \mathbf{B}$ (recall that $\mathbf{B}$ is the vector all of whose entries are B).

Let Π be any deterministic one-way linear protocol that has error probability of at most δ when solving BHM_n on inputs drawn according to μ. Let c be the communication cost of Π.

Since Π is deterministic, one-way, and linear, there exists a fixed $c \times n$ Boolean matrix A, such that the message of Π on any input $\mathbf{x}$ is $A\mathbf{x}$. By adding at most one bit to the communication cost of Π, we can assume $\mathbf{1}$ is one of the rows of A. We also assume, without loss of generality, that A has a full row rank, because otherwise Alice sends redundant information, which Bob can figure out by himself.

We assume c satisfies $c^3/\log c \leq 3n/4$, since, otherwise, $c \geq \Omega(\sqrt[3]{n \log n})$, and we are done.

For a matrix T, we denote by $\mathrm{sp}(T)$ the span of the row vectors of T over the field $GF(2)$. Clearly, for any matrix T, $\mathbf{0} \in \mathrm{sp}(T)$. In particular, $\mathbf{0} \in \mathrm{sp}(M) \cap \mathrm{sp}(A)$ for any matching $M \in \mathcal{M}_n$ (recall that we view a matching M as an $\frac{n}{2} \times n$ edge-vertex incidence matrix). By our assumption about A, $\mathbf{1} \in \mathrm{sp}(A)$. Since M is a perfect matching, the sum of its rows is $\mathbf{1}$; thus $\mathbf{1} \in \mathrm{sp}(M)$. We conclude that for any M, $\{\mathbf{0}, \mathbf{1}\} \subseteq \mathrm{sp}(M) \cap \mathrm{sp}(A)$. Let Z be an indicator random variable of the event $\{\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}(A) = \{\mathbf{0}, \mathbf{1}\}\}$, meaning that $\mathbf{0}$ and $\mathbf{1}$ are the only vectors in the intersection of the spans.

At a high level, our plan for the rest of the proof is as follows. Loosely speaking, for any matching M belonging to the above event (i.e., $\mathrm{sp}\, M \cap \mathrm{sp}(A) = \{\mathbf{0}, \mathbf{1}\}$) and for any possible input $\mathbf{x} \neq \mathbf{0}, \mathbf{1}$, the vectors $M\mathbf{x}$ and $A\mathbf{x}$ are linearly independent. This linear independence also implies *statistical* independence of the random variables $M\mathbf{X}$ and $A\mathbf{X}$. In particular, it means that Alice's message, $A\mathbf{X}$, has no information about $M\mathbf{X}$, and thus Bob cannot determine whether $M\mathbf{X} \oplus \mathbf{W} = \mathbf{1}$ or $M\mathbf{X} \oplus \mathbf{W} = \mathbf{0}$. We conclude that the protocol can succeed only when the event does not happen. We will prove that the probability of this event not happening is $\tilde{O}(c^3/n)$, and thus only when $c \geq \tilde{\Omega}(\sqrt[3]{n})$, is the success probability of the protocol sufficiently high. The formal argument follows.

In the protocol Π, Bob observes values of the random variables $A\mathbf{X}, \mathbf{M}$, and $\mathbf{W}$ and uses them to predict the random variable B with error probability δ. Therefore, by Fano's inequality (Theorem 2.2),

$$H_2(\delta) \geq \mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}). \tag{5}$$

Since conditioning reduces entropy,

$$\begin{aligned} \mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}) &\geq \mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}, Z) \\ &= \mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 1) \cdot \Pr(Z = 1) \\ &\quad + \mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 0) \cdot \Pr(Z = 0) \\ &\geq \mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 1) \cdot \Pr(Z = 1). \qquad (6) \end{aligned}$$

The following two lemmas bound the two factors in the last expression.

LEMMA 6.6. $\mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 1) = 1$.

LEMMA 6.7. $\Pr(Z = 1) \geq 1 - O(\frac{c^3}{n \log c})$.

The proofs of Lemmas 6.6 and 6.7 are provided below. Let us first show how the two lemmas derive the theorem. By combining (5) and (6), and Lemmas 6.6 and 6.7, we have

$$H_2(\delta) \geq 1 - O\left(\frac{c^3}{n \log c}\right).$$

Therefore,

$$\begin{aligned} c &\geq \Omega(\sqrt[3]{n(1 - H_2(\delta)) \cdot \log(n(1 - H_2(\delta)))}) \\ &= \Omega(\sqrt[3]{n \log n}), \end{aligned}$$

since $H_2(\delta)$ is a constant bounded away from 1. This completes the proof of the theorem. □

Proof of Lemma 6.6. Recall that we assume that $\mathbf{1}$ is one of the rows of A and that A has a full row rank. Let A' be the submatrix of A consisting of all the rows

of A, except $\mathbf{1}$. Clearly, $\mathrm{sp}(A') \subseteq \mathrm{sp}(A)$ and $\mathbf{1} \notin \mathrm{sp}(A')$. It follows that the event $\{\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}(A) = \{\mathbf{0}, \mathbf{1}\}\}$ is the same as the event $\{\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}(A') = \{\mathbf{0}\}\}$. Thus, from now on we will think of Z as an indicator random variable of the latter.

Observe that since n is a multiple of 4, the parity of the bits of $\mathbf{w}$ is always equal to the parity of the bits of $\mathbf{x}$. The parity of the bits of $\mathbf{x}$ is exactly the inner product $\mathbf{1}^t \cdot \mathbf{x}$, which is one of the bits in the vector $A\mathbf{x}$. It follows that there is a 1-1 mapping f such that $f(A(\mathbf{x}), \mathbf{w}) = (A'(\mathbf{x}), \mathbf{w})$. By the second data processing inequality (Theorem 2.1, part 5), we can therefore rewrite $\mathrm{H}(B \mid A\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 1)$ as $\mathrm{H}(B \mid A'\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 1)$.

By the definition of mutual information,

$$\begin{aligned}&\mathrm{H}(B \mid A'\mathbf{X}, \mathbf{M}, \mathbf{W}, Z = 1)\\ &= \mathrm{H}(B \mid \mathbf{M}, \mathbf{W}, Z = 1) - \mathrm{I}(B\ ;\ A'\mathbf{X} \mid \mathbf{M}, \mathbf{W}, Z = 1).\end{aligned}$$

The next proposition shows that the random variables $B, \mathbf{M}$, and $\mathbf{W}$ are mutually independent given the event $\{Z = 1\}$, which together with Theorem 2.1, part 2, implies that $\mathrm{H}(B \mid \mathbf{M}, \mathbf{W}, Z = 1) = \mathrm{H}(B|Z = 1)$. Since B and Z are independent (Z is a function of $\mathbf{M}$ only), $\mathrm{H}(B|Z = 1) = \mathrm{H}(B) = 1$. Thus, in order to prove the lemma it would suffice to show that $\mathrm{I}(B\ ;\ A'\mathbf{X} \mid \mathbf{M}, \mathbf{W}, Z = 1) = 0$.

PROPOSITION 6.8. *The random variables $B, \mathbf{M}$, and $\mathbf{W}$ are mutually independent, given the event $\{Z = 1\}$.*

Proof. We will show that the random variables $B, \mathbf{M}$, and $\mathbf{W}$ are mutually independent unconditionally. This independence would then hold even given the event $\{Z = 1\}$, because this event is a function of $\mathbf{M}$ only.

The random variables B and $\mathbf{M}$ are independent, by definition. Let M be any value of the random variable $\mathbf{M}$, and let b be any value of the random variable B. In order to show the desired independence, we need to prove that for any possible value $\mathbf{w}$ of $\mathbf{W}$, $\Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b) = \Pr(\mathbf{W} = \mathbf{w})$.

Using conditional probability, we can rewrite $\Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b)$ as follows:

$$\begin{aligned}&\Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b)\\ &= \sum_{\mathbf{x} \in \{0,1\}^n} \Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b, \mathbf{X} = \mathbf{x})\\ &\qquad \cdot \Pr(\mathbf{X} = \mathbf{x} \mid \mathbf{M} = M, B = b).\end{aligned}$$

Since $\mathbf{X}, \mathbf{M}$, and B are mutually independent by definition, $\Pr(\mathbf{X} = \mathbf{x} \mid \mathbf{M} = M, B = b) = \Pr(\mathbf{X} = \mathbf{x}) = 1/2^n$. $\Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b, \mathbf{X} = \mathbf{x}) = 1$ only if $\mathbf{w} = M\mathbf{x} \oplus \mathbf{b}$, and it is 0 otherwise. The number of $\mathbf{x}$'s that satisfy this condition is the number of solutions to the linear system $M\mathbf{x} = \mathbf{w} \oplus \mathbf{b}$ over Z_2^n. Since M is an $\frac{n}{2} \times n$ matrix that has a full row rank, this number is $2^{n/2}$. Therefore, $\Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b) = 2^{n/2}/2^n = 1/2^{n/2}$.

Consider now the quantity $\Pr(\mathbf{W} = \mathbf{w})$. Using conditional probability we can rewrite it as

$$\begin{aligned}&\Pr(\mathbf{W} = \mathbf{w})\\ &= \sum_{M,b} \Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b) \cdot \Pr(\mathbf{M} = M, B = b).\end{aligned}$$

We already proved that for all M and b, $\Pr(\mathbf{W} = \mathbf{w} \mid \mathbf{M} = M, B = b) = 1/2^{n/2}$. Therefore, also $\Pr(\mathbf{W} = \mathbf{w}) = 1/2^{n/2}$, completing the proof. ∎

Next we prove $\mathrm{I}(B\ ;\ A'\mathbf{X} \mid \mathbf{M}, \mathbf{W}, Z = 1) = 0$. By the chain rule for mutual information,

$$\begin{aligned}&\mathrm{I}(B, \mathbf{M}, \mathbf{W}\ ;\ A'\mathbf{X} \mid Z = 1)\\ &= \mathrm{I}(\mathbf{M}, \mathbf{W}\ ;\ A'\mathbf{X} \mid Z = 1) + \mathrm{I}(B\ ;\ A'\mathbf{X} \mid \mathbf{M}, \mathbf{W}, Z = 1).\end{aligned}$$

Since mutual information is always a nonnegative quantity, it would thus suffice to show that $\mathrm{I}(B, \mathbf{M}, \mathbf{W}\ ;\ A'\mathbf{X} \mid Z = 1) = 0$.

The function $f(b, M, \mathbf{w}) = (b, M, \mathbf{w} \oplus \mathbf{b})$ is a 1-1 function. Note that $f(B, \mathbf{M}, \mathbf{W}) = (B, \mathbf{M}, \mathbf{W} \oplus \mathbf{B}) = (B, \mathbf{M}, \mathbf{MX})$. Therefore, by the second data processing inequality (Theorem 2.1, part 5), we have

$$\mathrm{I}(B, \mathbf{M}, \mathbf{W}\ ;\ A'\mathbf{X} \mid Z = 1) = \mathrm{I}(B, \mathbf{M}, \mathbf{MX}\ ;\ A'\mathbf{X} \mid Z = 1).$$

Using again the chain rule for mutual information we have

$$\begin{aligned}&\mathrm{I}(B, \mathbf{M}, \mathbf{MX}\ ;\ A'\mathbf{X} \mid Z = 1)\\ &= \mathrm{I}(B, \mathbf{M}\ ;\ A'\mathbf{X} \mid Z = 1) + \mathrm{I}(\mathbf{MX}\ ;\ A'\mathbf{X} \mid B, \mathbf{M}, Z = 1).\end{aligned} \tag{7}$$

We next show that each of the above mutual information quantities is 0. By the definition of the input distribution μ, the random variables $B, \mathbf{M}$, and $\mathbf{X}$ are mutually independent. This holds even given the event $\{Z = 1\}$, because the latter is a function of $\mathbf{M}$ only. It follows that $B, \mathbf{M}$, $A'\mathbf{X}$ are also mutually independent given the event $\{Z = 1\}$, and thus $\mathrm{I}(B, \mathbf{M}\ ;\ A'\mathbf{X} \mid Z = 1) = 0$ (Theorem 2.1, part 7).

As for the second mutual information quantity on the right-hand side of (7), we use again the independence of $B, \mathbf{M}$, and $A'\mathbf{X}$ given $\{Z = 1\}$ as well as Theorem 2.1, part 8, to derive $\mathrm{I}(\mathbf{MX}\ ;\ A'\mathbf{X} \mid B, \mathbf{M}, Z = 1) = \mathrm{I}(\mathbf{MX}\ ;\ A'\mathbf{X} \mid \mathbf{M}, Z = 1)$. The following proposition proves that for any matching M satisfying the condition indicated by the event $\{Z = 1\}$, the random variables $M\mathbf{X}$ and $A'\mathbf{X}$ are independent. It then follows that $\mathrm{I}(\mathbf{MX}\ ;\ A'\mathbf{X} \mid \mathbf{M}, Z = 1) = 0$.

PROPOSITION 6.9. *For any matching $M \in \mathcal{M}_n$ satisfying the condition $\mathrm{sp}(M) \cap \mathrm{sp}(A') = \{\mathbf{0}\}$, the random variables $M\mathbf{X}$ and $A'\mathbf{X}$ are independent.*

Proof. Let $\mathbf{z}$ be any possible value for the random variable $M\mathbf{X}$, and let $\mathbf{y}$ be any possible value for the random variable $A'\mathbf{X}$. In order to prove the independence, we need to show that $\Pr(M\mathbf{X} = \mathbf{z} \mid A'\mathbf{X} = \mathbf{y}) = \Pr(M\mathbf{X} = \mathbf{z})$.

M is an $\frac{n}{2} \times n$ Boolean matrix that has a full row rank. Therefore, the number of solutions to the linear system $M\mathbf{x} = \mathbf{z}$ over Z_2^n is exactly $2^{n/2}$. Recall that $\mathbf{X}$ was chosen uniformly at random from Z_2^n. Therefore, $\Pr(M\mathbf{X} = \mathbf{z}) = 1/2^{n/2}$.

By the definition of conditional probability, $\Pr(M\mathbf{X} = \mathbf{z} \mid A'\mathbf{X} = \mathbf{y}) = \Pr(M\mathbf{X} = \mathbf{z}, A'\mathbf{X} = \mathbf{y}) / \Pr(A'\mathbf{X} = \mathbf{y})$. Since A' is a $(c-1) \times n$ Boolean matrix and has a full row rank, the same argument as above shows that $\Pr(A'\mathbf{X} = \mathbf{y}) = 1/2^{n-c+1}$. Let D be an $(\frac{n}{2} + c - 1) \times n$ matrix, which is created by putting M on top of A'. Since $\mathrm{sp}(M) \cap \mathrm{sp}(A') = \{\mathbf{0}\}$, D has a full row rank. We thus obtain $\Pr(M\mathbf{X} = \mathbf{z}, A'\mathbf{X} = \mathbf{y}) = \Pr(D\mathbf{X} = (\mathbf{z}, \mathbf{y})) = 1/2^{n/2-c+1}$. Hence, $\Pr(M\mathbf{X} = \mathbf{z} \mid A'\mathbf{X} = \mathbf{y}) = 2^{n/2-c+1}/2^{n-c+1} = 1/2^{n/2} = \Pr(M\mathbf{X} = \mathbf{z})$. The proposition follows. □

This completes the proof of Lemma 6.6. □

We now turn to the proof of Lemma 6.7.

Proof of Lemma 6.7. Denote the event $\{\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}(A) \neq \{\mathbf{0}, \mathbf{1}\}\}$ by E. We would like to prove $\Pr(E) \leq O(c^3/(n \log c))$. For $0 \leq k \leq n$, define $\mathrm{sp}_k(A)$ to be the vectors in $\mathrm{sp}(A)$ whose Hamming weight is k. Define E_k to be the event $\{\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}_k(A) \neq \emptyset\}$.

Since $\mathrm{sp}_0(A) = \{\mathbf{0}\}$ and $\mathrm{sp}_n(A) = \{\mathbf{1}\}$, the event E can be rewritten as $\bigvee_{k=1}^{n-1} E_k$. Thus, using the union bound, we can bound the probability of E as follows:

$$\Pr(E) \leq \sum_{k=1}^{n-1} \Pr(\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}_k(A) \neq \emptyset). \tag{8}$$

Let M be any matching in $\mathcal{M}_n$. Any vector $\mathbf{v}$ in $\mathrm{sp}(M)$, when viewed as a set $S_{\mathbf{v}}$ (i.e., $\mathbf{v}$ is the characteristic vector of $S_{\mathbf{v}}$), is a disjoint union of edges from M. We thus immediately conclude that $\mathbf{v}$ has to have an even Hamming weight. This implies that for all odd $1 \leq k \leq n-1$,

$$\Pr(\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}_k(A) \neq \emptyset) = 0. \tag{9}$$

Consider then an even k, and let $\mathbf{v}$ be any vector in $\mathrm{sp}_k(A)$. If $\mathbf{v}$ belongs to $\mathrm{sp}(M)$, then M can be partitioned into two perfect "submatchings": a perfect matching on $S_{\mathbf{v}}$ and perfect matching on $[n] \setminus S_{\mathbf{v}}$. We conclude that the number of matchings M in $\mathcal{M}_n$, for which $\mathbf{v} \in \mathrm{sp}(M)$, is exactly $m_k \cdot m_{n-k}$, where m_ℓ is the number of perfect matchings on ℓ nodes. Note that $m_\ell = \frac{\ell!}{(\ell/2)!2^{\ell/2}}$, and thus,

$$\Pr(\mathbf{v} \in \mathrm{sp}(\mathbf{M})) = \frac{m_k \cdot m_{n-k}}{m_n} = \frac{\binom{\frac{n}{2}}{\frac{k}{2}}}{\binom{n}{k}}.$$

It follows, by the union bound, that for any even k,

$$\Pr(\mathrm{sp}(\mathbf{M}) \cap \mathrm{sp}_k(A) \neq \emptyset) \leq |\mathrm{sp}_k(A)| \cdot \frac{\binom{\frac{n}{2}}{\frac{k}{2}}}{\binom{n}{k}}. \tag{10}$$

Since $\mathbf{1} \in \mathrm{sp}(A)$, $|\mathrm{sp}_k(A)| = |\mathrm{sp}_{n-k}(A)|$ for all $0 \leq k \leq n$. Combining this and (8), (9), and (10), it would thus suffice to prove the following:

$$\sum_{j=1}^{n/4} |\mathrm{sp}_{2j}(A)| \cdot \frac{\binom{\frac{n}{2}}{j}}{\binom{n}{2j}} \leq O\left(\frac{c^3}{n \log c}\right). \tag{11}$$

We start by bounding the ratio in each of the terms:

$$\begin{aligned} \frac{\binom{\frac{n}{2}}{j}}{\binom{n}{2j}} &= \frac{(\frac{n}{2})! \cdot (2j)! \cdot (n-2j)!}{(\frac{n}{2}-j)! \cdot j! \cdot n!} \\ &= \frac{\frac{n}{2} \cdots (\frac{n}{2}-j+1) \cdot (2j) \cdots (j+1)}{n \cdots (n-2j+1)} \\ &\leq \left(\frac{1}{2}\right)^j \cdot \left(\frac{2j}{n-j}\right)^j = \left(\frac{j}{n-j}\right)^j \leq \left(\frac{4j}{3n}\right)^j. \end{aligned} \tag{12}$$

The last inequality follows from the fact $j \leq n/4$. We next bound $|\mathrm{sp}_{2j}(A)|$ for small values of j.

PROPOSITION 6.10. *For every* $1 \leq j \leq \lfloor c/2 \rfloor$, $|\mathrm{sp}_{2j}(A)| \leq \sum_{i=1}^{2j} \binom{c}{i}$.

Proof. Using just the elementary row operations of Gaussian elimination, we can transform A into a matrix A' which has exactly the same span as A and has the $c \times c$ identity matrix as a submatrix. (Recall that A has a full row rank of c.) It follows

that any linear combination of t rows of A' results in a vector of Hamming weight at least t. Therefore, the only linear combinations to give vectors in $\mathrm{sp}_{2j}(A)$ are ones that use at most $2j$ rows of A'. The proposition follows, since the number of the latter is $\sum_{i=1}^{2j} \binom{c}{i}$. ∎

We conclude that for $1 \le j \le \lfloor c/2 \rfloor$, $|\mathrm{sp}_{2j}(A)| \le \sum_{i=1}^{2j} c^i = \frac{c^{2j}-1}{c-1} \cdot c \le 2c^{2j}$ (assuming $c \ge 2$). On the other hand, we have for all $1 \le j \le n/4$, $|\mathrm{sp}_{2j}(A)| \le |\mathrm{sp}(A)| \le 2^c$. Note that the quantity $2c^{2j}$ exceeds 2^c, when $j \ge \frac{c-1}{2\log c}$. We thus define $\ell \stackrel{\text{def}}{=} \lfloor \frac{c-1}{2\log c} \rfloor$ and break the sum on the right-hand side of (11), which we need to bound, into two parts as follows:

$$\begin{aligned}
&\sum_{j=1}^{n/4} |\mathrm{sp}_{2j}(A)| \cdot \frac{\binom{\frac{n}{2}}{j}}{\binom{n}{2j}} \\
&= \sum_{j=1}^{\ell} |\mathrm{sp}_{2j}(A)| \cdot \frac{\binom{\frac{n}{2}}{j}}{\binom{n}{2j}} + \sum_{j=\ell+1}^{n/4} |\mathrm{sp}_{2j}(A)| \cdot \frac{\binom{\frac{n}{2}}{j}}{\binom{n}{2j}} \\
&\le \sum_{j=1}^{\ell} (2c^{2j}) \cdot \left(\frac{4j}{3n}\right)^j + 2^c \cdot \max_{\ell < j \le n/4} \left(\frac{4j}{3n}\right)^j. \qquad (13)
\end{aligned}$$

The last inequality follows from (12), from Proposition 6.10, and from the fact that $\sum_{j=\ell+1}^{n/4} |\mathrm{sp}_{2j}(A)| \le |\mathrm{sp}(A)| \le 2^c$. We bound each of the terms on the right-hand side of (13) separately. We start with the first one:

$$\sum_{j=1}^{\ell} (2c^{2j}) \cdot \left(\frac{4j}{3n}\right)^j = 2 \cdot \sum_{j=1}^{\ell} \left(\frac{4c^2 j}{3n}\right)^j \le 2 \cdot \sum_{j=1}^{\ell} \left(\frac{4c^2 \ell}{3n}\right)^j.$$

Recall that we assumed $c^3/\log c \le 3n/4$. Hence, $4c^2\ell/(3n) \le 2c^3/(3n \log c) \le 1/2$. We can thus bound the geometric series as follows:

$$\begin{aligned}
2 \cdot \sum_{j=1}^{\ell} \left(\frac{4c^2\ell}{3n}\right)^j &\le 2 \cdot \frac{4c^2\ell}{3n} \cdot \frac{1}{1 - \frac{4c^2\ell}{3n}} \le \frac{16c^2\ell}{3n} \\
&\le \frac{8c^3}{3n \log c}. \qquad (14)
\end{aligned}$$

We now turn to bounding the second term on the right-hand side of (13).

PROPOSITION 6.11. *The function $g(j) = (aj)^j$, where $a > 0$, has a local minimum at $j^* = \frac{1}{ae}$ in the interval $(0, \infty)$.*

Proof. We rewrite g as follows: $g(j) = e^{j \ln(aj)}$. The derivative of g is the following:

$$g'(j) = e^{j \ln(aj)} \cdot (\ln(aj) + 1).$$

Thus, g has a local extremum at $j^* = \frac{1}{ae}$. We next verify that it is a local minimum. The second derivative of g is the following:

$$g''(j) = g'(j) \cdot (\ln(aj) + 1) + g(j) \cdot \frac{1}{j} = g(j) \cdot \left((\ln(aj) + 1)^2 + \frac{1}{j}\right).$$

Since g is positive in the interval $(0, \infty)$, $g''(j) > 0$ for all j in this interval. In particular, $g''(j^*) > 0$, implying that j^* is a local minimum. ∎

Proposition 6.11 shows that the function $g(j) = (aj)^j$ has a local minimum at $j^* = \frac{1}{ae}$ in the interval $(0, \infty)$. In our case $a = \frac{4}{3n}$, and thus $j^* = 3n/(4e) \geq n/4$. Therefore the maximum of $(\frac{4j}{3n})^j$ in the interval $[\ell, n/4]$ is obtained at $j = \ell$. We conclude that

$$\begin{aligned} 2^c \cdot \max_{\ell < j \leq n/4} \left(\frac{4j}{3n}\right)^j &\leq 2^c \cdot \left(\frac{4\ell}{3n}\right)^\ell \leq 2^c \cdot \left(\frac{2c}{3n \log c}\right)^{\frac{c}{2 \log c}} \\ &\leq \left(\frac{4c}{3n \log c}\right)^c \leq \left(\frac{2c}{n}\right)^c \leq \frac{c}{n} \\ &\leq \frac{c^3}{n \log c}. \end{aligned} \tag{15}$$

In the next-to-last inequality we used the fact $2 \leq c \leq n/4$. Combining (13), (14), and (15), we have

$$\sum_{j=1}^{n/4} |\operatorname{sp}_{2j}(A)| \cdot \frac{\binom{\frac{n}{2}}{j}}{\binom{n}{2j}} \leq \frac{8c^3}{3n \log c} + \frac{c^3}{n \log c} \leq O\left(\frac{c^3}{n \log c}\right).$$

This completes the proof of Lemma 6.7. □

Remark. As mentioned previously, the randomized lower bound for HM_n also holds for a restricted version of the Hidden Matching Problem (which we denoted by RHM_n), in which Bob's input is a matching M taken from a small set $\mathcal{M}$ of $\Theta(n)$ disjoint matchings only. One can define an analogous restricted version of BHM_n, denoted RBHM_n. However, in this case the complexities of BHM_n and RBHM_n are entirely different. By Aaronson's result [1], and by our $O(\log n)$-bit quantum upper bound for BHM_n (which, of course, also works for RBHM_n), the deterministic one-way communication complexity of RBHM_n is only $O(\log^2 n \cdot \log \log n)$. On the other hand, the deterministic (and even 0-error randomized) one-way communication complexity of BHM_n is $\Omega(n)$ (Theorem 6.2). Hence, there is an exponential gap between the two.

7. Open problems. The main question in quantum communication complexity is to characterize its power in relation to classical communication complexity. For partial Boolean functions it was known that quantum two-way communication complexity could be exponentially lower than the classical complexity [20]. Here we prove a similar result for a relation for one-way communication complexity and SM. The main open question is to find the relation between quantum and classical communication complexity for total functions. Are they polynomially related for all total functions? Is this relationship even tighter in the case of one-way communication complexity? Moreover, can we show an exponential separation between quantum one-way communication complexity and randomized two-way communication complexity?

Acknowledgments. We would like to thank Umesh Vazirani, Ashwin Nayak, and Kunal Talwar for helpful discussions.

REFERENCES

[1] S. Aaronson, *Limitations of quantum advice and one-way communication*, in Proceedings of the 19th IEEE Conference on Computational Complexity (CCC), 2004, pp. 320–332.

[2] F. Ablayev, *Lower bounds for one-way probabilistic communication complexity and their application to space complexity*, Theoret. Comput. Sci., 157 (1996), pp. 139–159.

[3] A. Ambainis, A. Nayak, A. Ta-Shma, and U. V. Vazirani, *Dense quantum coding and quantum finite automata*, J. ACM, 49 (2002), pp. 496–511.

[4] A. Ambainis, L. J. Schulman, A. Ta-Shma, U. Vazirani, and A. Wigderson, *The quantum communication complexity of sampling*, SIAM J. Comput., 32 (2003), pp. 1570–1585.
[5] L. Babai and P. G. Kimmel, *Randomized simultaneous messages: Solution of a problem of Yao in communication complexity*, in Proceedings of the 12th IEEE Conference on Computational Complexity (CCC), 1997, pp. 239–246.
[6] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, *Quantum fingerprinting*, Phys. Rev. Lett., 87 (2001), 167902.
[7] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum versus classical communication and computation*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), 1998, pp. 63–68.
[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
[9] P. Frankl and R. M. Wilson, *Intersection theorems with geometric consequences*, Combinatorica, 1 (1981), pp. 357–368.
[10] D. Gavinsky, J. Kempe, O. Regev, and R. de Wolf, *Bounded-error quantum state identification and exponential separations in communication complexity*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), 2006, pp. 594–603.
[11] I. Kerenidis and R. de Wolf, *Exponential lower bound for 2-query locally decodable codes via quantum argument*, in Proceedings of the 35th ACM Annual Symposium on Theory of Computing (STOC), 2003, pp. 106–115.
[12] H. Klauck, *On quantum and probabilistic communication: Las Vegas and one-way protocols*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), 2000, pp. 644–651.
[13] I. Kremer, *Quantum Communication*, Master's thesis, The Hebrew University of Jerusalem, Jerusalem, 1995.
[14] I. Kremer, N. Nisan, and D. Ron, *On randomized one-round communication complexity*, Comput. Complexity, 8 (1999), pp. 21–49.
[15] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[16] I. Newman, *Private versus common random bits in communication complexity*, Inform. Process. Lett., 39 (1991), pp. 67–71.
[17] I. Newman and M. Szegedy, *Public versus private coin flips in one round communication games*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), 1996, pp. 561–570.
[18] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[19] C. H. Papadimitriou and M. Sipser, *Communication complexity*, J. Comput. System Sci., 28 (1984), pp. 260–269.
[20] R. Raz, *Exponential separation of quantum and classical communication complexity*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), 1999, pp. 358–367.
[21] P. Sen and S. Venkatesh, *Lower bounds in the quantum cell probe model*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 358–369.
[22] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[23] A. C-C. Yao, *Some complexity questions related to distributive computing*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC), 1979, pp. 209–213.
[24] A. C-C. Yao, *Lower bounds by probabilistic arguments*, in Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1983, pp. 420–428.
[25] A. C-C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 352–361.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 385–397

THE EUCLIDEAN ORIENTEERING PROBLEM REVISITED*

KE CHEN† AND SARIEL HAR-PELED†

Abstract. We consider the *rooted orienteering problem*: Given a set P of n points in the plane, a starting point $r \in P$, and a length constraint $\mathcal{B}$, one needs to find a path starting from r that visits as many points of P as possible and of length not exceeding $\mathcal{B}$. We present a $(1-\varepsilon)$-approximation algorithm for this problem that runs in $n^{O(1/\varepsilon)}$ time; the computed path visits at least $(1-\varepsilon)k_{\text{opt}}$ points of P, where k_{opt} is the number of points visited by an optimal solution. This is the first polynomial time approximation scheme for this problem. The algorithm also works in higher dimensions.

Key words. orienteering problem, k-TSP, approximation algorithms, PTAS

AMS subject classifications. 68Q25, 68W25, 68W40

DOI. 10.1137/060667839

1. Introduction.

Consider a traveling salesperson who has a fixed amount of gasoline and wants to maximize the number of customers visited under this constraint. This is an instance of the *orienteering problem* that requires us to design a network that visits a maximum number of points, subject to an upper bound on the total length of the network. This problem is "dual" to the classical *k-TSP problem* [3, 10, 9], which asks for a minimum length path visiting at least k points.

In this paper, we consider the *rooted path orienteering problem* in the plane, specified by P, r, and $\mathcal{B}$, where P is a set of n points in the plane, r is the starting point, and $\mathcal{B} > 0$ is the maximum length allowed. The solution to this problem is a path that starts at r and visits as many points of P as possible, such that the path length does not exceed $\mathcal{B}$. The effect of fixing the starting point is significant as far as approximation algorithms are concerned. Indeed, approximation algorithms for k-TSP extend easily to the *unrooted orienteering problem*, where there is no fixed starting point, while the approximation algorithm for the rooted orienteering problem is more challenging. The difficulty stems from the fact that an optimal path may visit a large number of points that lie in a small cluster at a distance nearly $\mathcal{B}$ from r, thus making it difficult to visit at least a large fraction of these points unless the path is very efficient [2].

Some related problems include the *prize-collecting traveling salesman problem* and the *vehicle routing problem*. They arise from real world applications such as delivering goods to locations or assigning technicians to maintenance service jobs. A substantial amount of work on heuristics for these problems can be found in the operations research literature [12].

Arkin, Mitchell, and Narasimhan [2] were the first to design approximation algorithms for the rooted orienteering problem. They considered the rooted orienteering problem for points in the plane when the underlying network is a path, a cycle, or

*Received by the editors August 19, 2006; accepted for publication (in revised form) August 8, 2007; published electronically April 23, 2008. A preliminary version of this paper appeared in *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry*, 2006, pp. 247–254.
http://www.siam.org/journals/sicomp/38-1/66783.html

†Department of Computer Science, University of Illinois, 201 N. Goodwin Avenue, Urbana, IL 61801 (kechen@uiuc.edu, http://www.uiuc.edu/~kechen, sariel@uiuc.edu, http://www.uiuc.edu/~sariel). The second author's research was partially supported by NSF CAREER award CCR-0132901.

a tree. Their algorithms provide a 2-approximation for the rooted path orienteering problem, and a 2- (resp., 3-) approximation when the networks considered are cycles (resp., trees). Blum et al. [6] proposed the first constant-factor approximation algorithm for the rooted path orienteering problem when the points lie in a general metric space. Bansal et al. [5] improved the approximation factor to 3. Arkin, Mitchell, and Narasimhan [2] asked whether a better approximation is possible in Euclidean spaces.

One difficulty in assailing this problem is the relative lack of algorithmic tools to handle rigid budget constraints. Since the development of $(1+\varepsilon)$-approximation algorithms for TSP [3, 10], a large class of the problems that aim to minimize the tour length, subject to certain constraints on the points visited by the tour, have been resolved. See the surveys by Mitchell [11] and Arora [4] for further information. In contrast, the behavior of the optimization problems that seek to maximize some function on the points visited, subject to constraints on the length of the path used or the timespans the points are being visited [7, 5, 8], are not as well understood.

The main idea in the previous approximation algorithms for the orienteering problem [2, 6, 5] was to transform an approximation algorithm for the *rooted k-TSP problem* into an approximation algorithm for the orienteering problem. In particular, Blum et al. [6] formulated the notion of the *excess* for a path (which is defined to be the difference between the length of a path and the distance between the endpoints of the path) and then combined dynamic programming with the use of k-TSP to obtain an algorithm for orienteering in a metric space. These techniques were also implicitly used in the work of Arkin, Mitchell, and Narasimhan [2].

Our results. To obtain a polynomial time approximation scheme (PTAS) for the orienteering problem, we extend the concept of the excess of a path into the $\mathfrak{u}$-*excess* of a path, which is (loosely) the difference in lengths between π and the best approximation to π by a polygonal line having $\mathfrak{u}$ vertices; see Figure 1. (Therefore, the previous notion of excess is 2-excess in our notation.)

To this end, we revisit the rooted k-TSP problem in the plane and show that Mitchell's algorithm [10] computes an $(\varepsilon, \mathfrak{u})$-approximation for rooted k-TSP; that is, the algorithm outputs a rooted path of length $\leq \|\pi\| + \varepsilon \cdot \mathcal{E}_{\pi,\mathfrak{u}}$, where π is any path that starts from the root and visits k points, $\|\pi\|$ denotes the length of π, and $\mathcal{E}_{\pi,\mathfrak{u}}$ is the $\mathfrak{u}$-excess of π. Note that the quantity $\mathcal{E}_{\pi,\mathfrak{u}}$ might be smaller than $\|\pi\|$ by several orders of magnitude. Therefore, we show that Mitchell's algorithm provides a much tighter approximation for k-TSP than what was previously known. See section 3.

Armed with the new approximation algorithm for k-TSP, it is now possible to reduce the orienteering problem to an instance of k-TSP. The PTAS for orienteering is presented in section 5.

The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 presents the new analysis of Mitchell's algorithm. Section 4 extends the new k-TSP algorithm into higher dimensions by combining Mitchell's technique and Arora's k-TSP algorithm. Section 5 gives the PTAS for the orienteering problem. We conclude in section 6.

2. Problem statement and definitions. Let $\pi = \langle p_1, p_2, \ldots, p_k \rangle$ be a path that visits k points of P, starting at p_1 and ending at p_k. The *length* of π is denoted by $\|\pi\| = \sum_{i=1}^{k-1} \|p_{i+1} - p_i\|$. More generally, for a collection S of segments, $\|S\|$ denotes the total length of segments in S. Let $1 = i_1 < \cdots < i_{\mathfrak{u}} = k$ be a sequence of $\mathfrak{u} \leq k$ integers. The path $\langle p_{i_1}, p_{i_2}, \ldots, p_{i_{\mathfrak{u}}} \rangle$ is a $\mathfrak{u}$-*skeleton* of π. The *optimal* $\mathfrak{u}$-*skeleton* of π is the $\mathfrak{u}$-skeleton of π with maximum total length, denoted by $\mathcal{S}^{\mathfrak{u}}_{\mathrm{opt}}(\pi)$. See Figure 1.

The $\mathfrak{u}$-*excess* of a path π is the difference between the length of π and its optimal

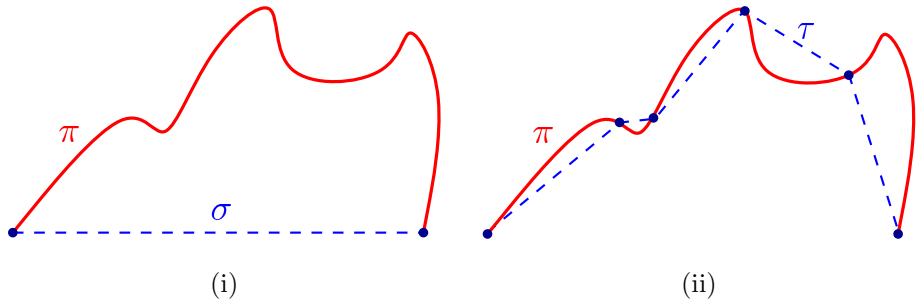


FIG. 1. (i) *The segment σ is the 2-skeleton of the path π. The 2-excess of π, namely, $\mathcal{E}_{\pi,2}$, is the difference between the length of π and the length of σ.* (ii) *The polygonal line τ forms a 6-skeleton of π. The 6-excess of π, namely, $\mathcal{E}_{\pi,6}$, is at most $\|\pi\|-\|\tau\|$.*

$\mathfrak{u}$-skeleton, that is, $\mathcal{E}_{\pi,\mathfrak{u}} = \|\pi\| - \left\|\mathcal{S}^{\mathfrak{u}}_{\text{opt}}(\pi)\right\|$. Note that the $\mathfrak{u}$-excess of π may be considerably smaller than the length of π.

Given a set P of n points and a starting point $r \in P$, the *rooted k-TSP problem* is to find a shortest path that visits k points of P starting at r. An $(\varepsilon, \mathfrak{u})$-*approximation* to the rooted k-TSP is a path ϕ that visits k points of P starting at r, such that the length of ϕ is no more than $\|\mathcal{T}\| + \varepsilon \cdot \mathcal{E}_{\mathcal{T},\mathfrak{u}}$, for any path $\mathcal{T}$ that visits k points of P starting at r.

DEFINITION 2.1 (the rooted orienteering problem). *Given a set P of n points, a budget $\mathcal{B}$, and a starting point $r \in P$, the* rooted orienteering problem *is to find a path ω_{opt} that visits as many points of P as possible, under the constraint that the length of ω_{opt} is at most $\mathcal{B}$. Let k_{opt} denote the number of points visited by ω_{opt}. A* $(1-\varepsilon)$-approximation *to the rooted orienteering problem is a path ω (starting at r) that visits at least $(1-\varepsilon)k_{\text{opt}}$ points of P, such that the length of ω is at most $\mathcal{B}$.*

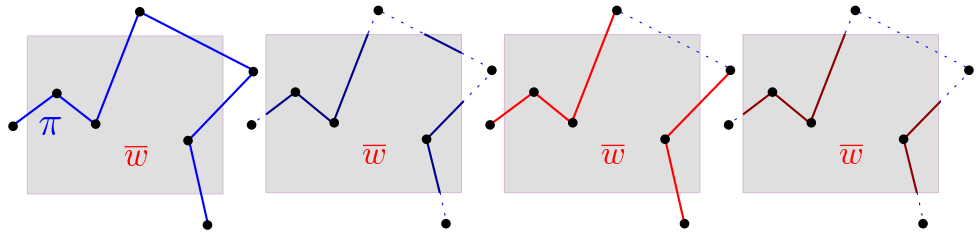
3. An $(\varepsilon, \mathfrak{u})$-approximation algorithm for k-TSP. In this section, we present an $(\varepsilon, \mathfrak{u})$-approximation algorithm for k-TSP. The algorithm is the k-TSP algorithm of Mitchell [10], and our contribution is the new tighter analysis of its performance (see section 3.3). In the following, we first review Mitchell's algorithm and then present our new improved analysis.

3.1. Preliminaries. In the following, m is a fixed constant. We assume, without loss of generality, that the points of P all have distinct x and y coordinates and that P is contained in an axis-parallel square $\mathcal{Q}$. Let π be a given path visiting k points of P.

DEFINITION 3.1. *A closed, axis-parallel rectangle $\overline{w}$ is a* window *if $\overline{w} \subseteq \mathcal{Q}$. The* extent *of a window $\overline{w}$ is the larger of the width and height of $\overline{w}$ and is denoted by $\Delta_{\overline{w}}$. Let $\pi(\overline{w})$ denote the subset of π consisting of the union of segments of π having at least one endpoint inside (or on the boundary of) $\overline{w}$. Given a collection S of segments, we slightly abuse notation and denote the set of segments of S clipped to $\overline{w}$ by $S \cap \overline{w}$. See Figure 2.*

A line ℓ is a cut *for π, with respect to $\overline{w}$, if ℓ is a horizontal or vertical line and ℓ intersects $\overline{w}$; ℓ is an m-*perfect cut *for π, with respect to $\overline{w}$, if ℓ intersects the segments of $\pi(\overline{w}) \cap \overline{w}$ at most m times.*

DEFINITION 3.2. *The combinatorial type of a window $\overline{w}$ with respect to π is the subset of P inside (or on the boundary of) $\overline{w}$ and a listing, for each of the four sides of $\overline{w}$, of the identities of the line segments of $\pi(\overline{w})$ that intersect it. (In particular, if a segment of π intersects $\overline{w}$, but both its endpoints are outside $\overline{w}$, then the segment is*



(i) π (ii) $\pi \cap \overline{w}$ (iii) $\pi(\overline{w})$ (iv) $\pi(\overline{w}) \cap \overline{w}$

FIG. 2. *The different ways of clipping a path π to a window $\overline{w}$.*

not considered in the combinatorial type of $\overline{w}$.) We say that $\overline{w}$ is a minimal window *if there is no window $\overline{w}'$ that is strictly contained in $\overline{w}$ with the same combinatorial type as $\overline{w}$.*

For a minimal window $\overline{w}$, if there is no m-perfect cut for π, with respect to $\overline{w}$, then it is m-dense. *Namely, any horizontal or vertical line that intersects $\overline{w}$ has more than m intersection points with $\pi(\overline{w}) \cap \overline{w}$.*

Given a window $\overline{w}$, Mitchell [10] described how to "shrink" $\overline{w}$ into a minimal window by "pinning" all four sides of $\overline{w}$. It is not hard to see that the number of all possible minimal windows is $O(n^4)$, since (intuitively) it has four degrees of freedom.

CLAIM 3.3. *If a window $\overline{w}$ is m-dense, then $\|\pi(\overline{w}) \cap \overline{w}\| \geq m \cdot \Delta_{\overline{w}}$.*

Proof. Assume, without loss of generality, that the width of $\overline{w}$ is greater than the height of $\overline{w}$, and the interval $[x_1, x_2]$ is the projection of $\overline{w}$ onto the x-axis. Let $f(\alpha)$ denote the number of segments of $\pi(\overline{w})$ within $\overline{w}$ that intersect the vertical line $x = \alpha$. By the density of $\overline{w}$, $f(x) > m$ for $x \in [x_1, x_2]$. The total length of the segments of $\pi(\overline{w}) \cap \overline{w}$ is lower bounded by the integral of $f(x)$ over $[x_1, x_2]$, which in turn is lower bounded by $m(x_2 - x_1) = m \cdot \Delta_{\overline{w}}$. ∎

3.2. Review of the *k*-TSP algorithm. The "new" $(\varepsilon, \mathfrak{u})$-approximation algorithm for k-TSP is the algorithm of Mitchell [10] for k-TSP, and we review it here only for the sake of completeness. We remind the reader that m is a fixed (constant) number.

A problem instance of **WindowTSP** consists of (i) a (minimal) window $\overline{w}$ that contains at least one point of P, with its boundaries determined by (up to) four points of P, (ii) an integer $h \geq 0$, indicating how many points interior to $\overline{w}$ should be visited, (iii) boundary information specifying at most m crossing segments (each determined by a pair of points of P, one interior to or on the boundary of $\overline{w}$, and another outside $\overline{w}$) for each side of the boundary of $\overline{w}$, and (iv) connectivity constraints, indicating which pairs of crossing segments are required to be connected within $\overline{w}$. The solution to the **WindowTSP** problem is a set of (hopefully short) paths inside window $\overline{w}$ such that (i) all of the boundary constraints are satisfied, (ii) all of the connectivity constraints within $\overline{w}$ are satisfied, and (iii) h points of P are visited by the paths inside $\overline{w}$. See Figure 3.

Clearly, the rooted k-TSP problem can be formulated as a **WindowTSP** instance consisting of a bounding box $\mathcal{Q}$ of P, a parameter k, empty boundary information, and connectivity constraints requiring that k points of P inside $\mathcal{Q}$ must be connected by a single path, with r as an endpoint of the path.

The recursive algorithm for **WindowTSP** works as follows. If the window $\overline{w}$ has at most m points of P in its interior, then the problem is solved by enumeration of

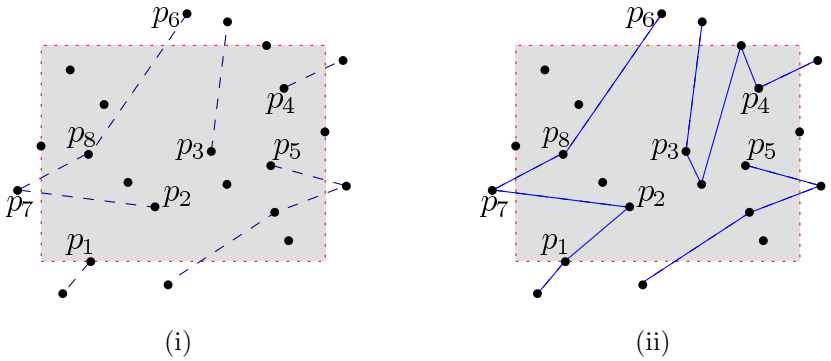


FIG. 3. (i) *An instance of the* **WindowTSP** *problem. The segments specify how the solution crosses the boundary of* $\overline{w}$. *The connectivity constraints are as follows:* p_1 *is required to connect to* p_2 *(possibly via other points within* $\overline{w}$*),* p_3 *is required to connect to* p_4 *(possibly via other points within* $\overline{w}$*), and* p_5 *is the starting point* r *of the path (namely, the degree of* p_5 *is* 1*). The multipaths are required to visit nine points in the window.* (ii) *A possible solution.*

all possible solutions. Otherwise, the algorithm tries all possible cuts for the current window recursively, enumerating over all the possible choices of valid boundary information along this cut and computing the cheapest option. If there is no m-perfect cut, then the algorithm performs a cut and reduces the intersection by introducing bridges; see Remark 1 below. For our analysis, we care only whether a cut used by the algorithm is an m-perfect cut or a more complicated cut.

When a cut divides a window into two smaller windows, we need to "shrink" those two windows into minimal windows. In particular, segments that just pass through a window are ignored during the shrinking. This is a small but important technicality. See Figure 4.

Remark 1. The algorithm of Mitchell [10] also introduces bridges (close to, or) on the boundary of the window $\overline{w}$, where a bridge is a vertical or horizontal segment. To simplify our exposition, we ignored those bridges in describing the algorithm. Of course, for a correct working implementation those bridges are necessary. See [10] for full details. See also Remark 2 below.

3.3. Analysis of the algorithm. The key observation in our analysis is that the approximation algorithm does not introduce any error when a cut is m-perfect. Thus, the error is introduced only when the algorithm works inside an m-dense window, but such windows have high "excess."

DEFINITION 3.4. *For a set* S *of segments, let* $I_x(S)$ *denote the projection of* S *onto the* x*-axis; namely,* $I_x(S)$ *is the set of all points* α *(on the* x*-axis), such that the vertical line* $x = \alpha$ *intersects the segments of* S. *Let* $\mathrm{len}_x(S)$ *denote the total length of* $I_x(S)$. *Note that* $I_x(S)$ *is a set of (disjoint) intervals on the real line, and* $\mathrm{len}_x(S)$ *is the total length of these intervals. We define* $I_y(S)$ *and* $\mathrm{len}_y(S)$ *in a similar fashion.*

DEFINITION 3.5. *For a window* $\overline{w}$ *and a path* π, *the* surplus *of* π *in* $\overline{w}$ *is*

$$\rho(\overline{w}, \pi) = \|\pi \cap \overline{w}\| - \sqrt{(\mathrm{len}_x(\pi \cap \overline{w}))^2 + (\mathrm{len}_y(\pi \cap \overline{w}))^2}.$$

It is easy to verify that the surplus is always nonnegative. See Figure 5.

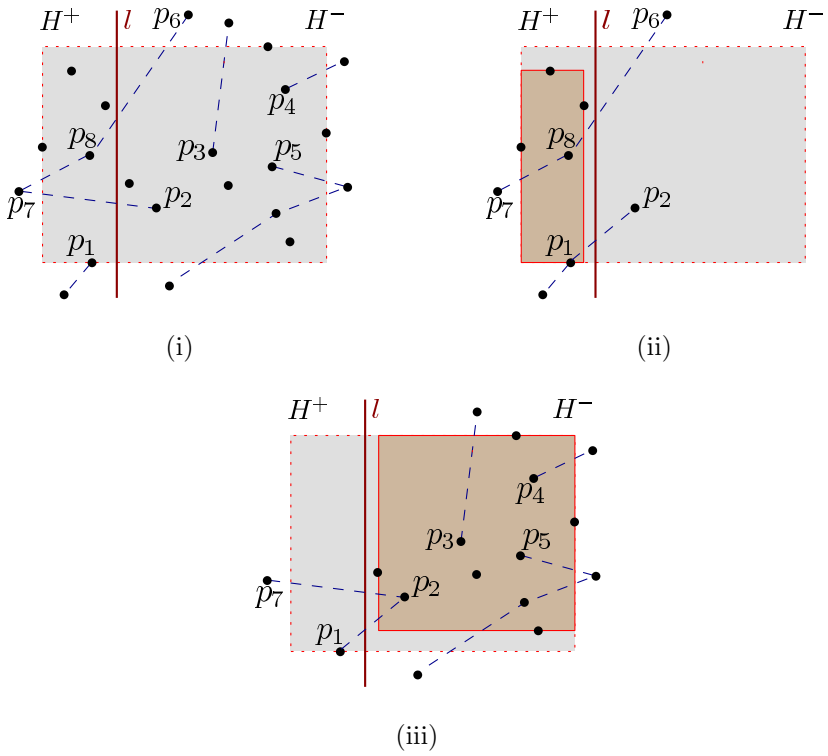


FIG. 4. *Performing a cut on the* **WindowTSP** *instance of Figure* 3. (i) *A cut l divides the window $\overline{w}$ into smaller windows $\overline{w} \cap H^+$ and $\overline{w} \cap H^-$.* (ii) *The minimal window for $\overline{w} \cap H^+$. The segments represent the crossing boundary information for the new window. In particular, the segment p_1p_2 is introduced when "guessing" the boundary information along the cut l before the recursive call. The other segments intersecting the boundary are inherited from the original instance.* (iii) *The minimal window for $\overline{w} \cap H^-$.*

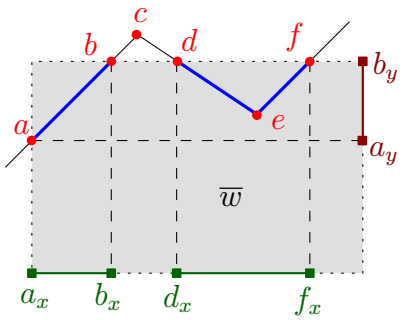


FIG. 5. *Illustrating the intersection of a polygonal line $\pi = \langle a, b, c, d, e, f \rangle$ with a window $\overline{w}$. The set $I_x(\pi \cap \overline{w})$ consists of the segments a_xb_x and d_xf_x; the set $I_y(\pi \cap \overline{w})$ consists of segment a_yb_y. The surplus of π in $\overline{w}$ is $\|a-b\| + \|d-e\| + \|e-f\| - \sqrt{(\|a_x - b_x\| + \|d_x - f_x\|)^2 + \|a_y - b_y\|^2}$.*

LEMMA 3.6. *If $X, Y, X_1, \ldots, X_n, Y_1, \ldots, Y_n$ are nonnegative real numbers such that*

$$\sum_{i=1}^{n} X_i \geq X \quad \text{and} \quad \sum_{i=1}^{n} Y_i \geq Y,$$

then $\sum_{i=1}^{n} \sqrt{X_i^2 + Y_i^2} \geq \sqrt{X^2 + Y^2}$.

Proof. This is an easy application of the triangle inequality. Indeed, let q_i be the point $(\sum_{j=1}^{i} X_j, \sum_{j=1}^{i} Y_j)$ in the plane for $1 \leq i \leq n$, and let $q_0 = (0,0)$. Consider the path $\pi = \langle q_0, q_1, \ldots, q_n \rangle$. Clearly, we have that

$$\|\pi\| = \sum_{i=1}^{n} \sqrt{X_i^2 + Y_i^2} = \sum_{i=1}^{n} \|q_i - q_{i-1}\| \geq \|q_n - q_0\|$$
$$= \sqrt{\left(\sum_{i=1}^{n} X_i\right)^2 + \left(\sum_{i=1}^{n} Y_i\right)^2} \geq \sqrt{X^2 + Y^2},$$

as required. □

LEMMA 3.7. *Let $\overline{\mathsf{D}}$ be a set of interior disjoint windows (inside $\mathcal{Q}$), and let π be a polygonal path inside $\mathcal{Q}$. We have that $\mathcal{E}_{\pi,2} \geq \sum_{\overline{w} \in \overline{\mathsf{D}}} \big(\|\pi \cap \overline{w}\| - \sqrt{2}\Delta_{\overline{w}} \big)$.*

Proof. Let Ψ be a decomposition of $\mathcal{Q}$ into interior disjoint axis-parallel rectangles such that Ψ contains all of the rectangles of $\overline{\mathsf{D}}$. Let $X = \operatorname{len}_x(\pi \cap \mathcal{Q})$ and $Y = \operatorname{len}_y(\pi \cap \mathcal{Q})$. For a window $\overline{w}$, let $X_{\overline{w}} = \operatorname{len}_x(\pi \cap \overline{w})$ and $Y_{\overline{w}} = \operatorname{len}_y(\pi \cap \overline{w})$. Clearly, $X \leq \sum_{\overline{w} \in \Psi} X_{\overline{w}}$ and $Y \leq \sum_{\overline{w} \in \Psi} Y_{\overline{w}}$, since $I_x(\pi) = \bigcup_{\overline{w} \in \Psi} I_x(\pi \cap \overline{w})$ and $I_y(\pi) = \bigcup_{\overline{w} \in \Psi} I_y(\pi \cap \overline{w})$. Let s and t be the two endpoints of π. We have that

$$\mathcal{E}_{\pi,2} = \|\pi\| - \|s - t\| = \|\pi\| - \sqrt{\operatorname{len}_x(st)^2 + \operatorname{len}_y(st)^2} \geq \|\pi\| - \sqrt{X^2 + Y^2},$$

since $\operatorname{len}_x(st) \leq X$ and $\operatorname{len}_y(st) \leq Y$. On the other hand, by Lemma 3.6, we get $\sqrt{X^2 + Y^2} \leq \sum_{\overline{w} \in \Psi} \sqrt{X_{\overline{w}}^2 + Y_{\overline{w}}^2}$, since $\sum_{\overline{w} \in \Psi} X_{\overline{w}} \geq X$ and $\sum_{\overline{w} \in \Psi} Y_{\overline{w}} \geq Y$. Therefore,

$$\mathcal{E}_{\pi,2} \geq \|\pi\| - \sqrt{X^2 + Y^2} \geq \|\pi\| - \sum_{\overline{w} \in \Psi} \sqrt{X_{\overline{w}}^2 + Y_{\overline{w}}^2} = \sum_{\overline{w} \in \Psi} \left(\|\pi \cap \overline{w}\| - \sqrt{X_{\overline{w}}^2 + Y_{\overline{w}}^2} \right)$$
$$= \sum_{\overline{w} \in \Psi} \rho(\overline{w}, \pi) = \sum_{\overline{w} \in \overline{\mathsf{D}}} \rho(\overline{w}, \pi) + \sum_{\overline{w} \in \Psi \setminus \overline{\mathsf{D}}} \rho(\overline{w}, \pi) \geq \sum_{\overline{w} \in \overline{\mathsf{D}}} \rho(\overline{w}, \pi),$$

because the surplus $\rho(\overline{w}, \pi)$ is always nonnegative. Now, since

$$\rho(\overline{w}, \pi) = \|\pi \cap \overline{w}\| - \sqrt{X_{\overline{w}}^2 + Y_{\overline{w}}^2} \geq \|\pi \cap \overline{w}\| - \sqrt{\Delta_{\overline{w}}^2 + \Delta_{\overline{w}}^2} = \|\pi \cap \overline{w}\| - \sqrt{2}\Delta_{\overline{w}},$$

we obtain $\mathcal{E}_{\pi,2} \geq \sum_{\overline{w} \in \overline{\mathsf{D}}} \rho(\overline{w}, \pi) \geq \sum_{\overline{w} \in \overline{\mathsf{D}}} \big(\|\pi \cap \overline{w}\| - \sqrt{2}\Delta_{\overline{w}} \big)$, as claimed. □

THEOREM 3.8. *Let $\pi = \langle p_1, p_2, \ldots, p_k \rangle$ be an arbitrary path that visits k points of P, and let $\mathfrak{u} \geq 2$ be an arbitrary fixed integer. One can compute, in $n^{O(\mathfrak{u})}$ time, a path that starts at p_1 and visits k points of P, with length at most $\|\pi\| + \mathcal{E}_{\pi,\mathfrak{u}}/\mathfrak{u}$.*

Proof. Set $m = \lceil 2\sqrt{2}\mathfrak{u} \rceil$, and use the algorithm presented above. The running time bound follows readily. Thus, we need only argue that the path computed is indeed within the claimed bound on the length.

Thus, consider the (conceptual) execution of the recursive algorithm over the path π, performing the recursive calls according to π. Specifically, let $\overline{w}$ be a *minimal* window that is visited by the recursive algorithm when applied to π. If an m-perfect cut (for π) exists with respect to $\overline{w}$, then we use it to cut the window $\overline{w}$ into two parts and proceed recursively on each side of the cut. If such an m-perfect cut does not exist for $\overline{w}$ (that is, $\overline{w}$ is m-dense), then we (conceptually) stop, and use the results returned by the recursive call on this window. We claim that for these specific choices, the recursive algorithm computes a path σ, such that $\|\sigma\|$ is as required. Since the recursive algorithm returns a path no longer than σ, this would imply the theorem.

Let $\overline{\mathsf{D}}$ be the set of m-dense windows (which by the algorithm execution are minimal windows) visited by the algorithm when applied to π. Let $\mathcal{S} = \mathcal{S}^{\mathfrak{u}}_{\mathrm{opt}}(\pi)$ be an optimal $\mathfrak{u}$-skeleton for π, and let $\pi_1, \ldots, \pi_{\mathfrak{u}-1}$ be the breakup of π into subpaths by the vertices of $\mathcal{S}$. By Lemma 3.7, we have that

$$\begin{aligned}\mathcal{E}_{\pi,\mathfrak{u}} &= \sum_{j=1}^{\mathfrak{u}-1} \mathcal{E}_{\pi_j,2} \geq \sum_{j=1}^{\mathfrak{u}-1} \sum_{\overline{w}\in\overline{\mathsf{D}}} \Big(\|\pi_j \cap \overline{w}\| - \sqrt{2}\Delta_{\overline{w}} \Big) = \sum_{\overline{w}\in\overline{\mathsf{D}}} \sum_{j=1}^{\mathfrak{u}-1} \Big(\|\pi_j \cap \overline{w}\| - \sqrt{2}\Delta_{\overline{w}} \Big) \\ &= \sum_{\overline{w}\in\overline{\mathsf{D}}} \Big(\|\pi \cap \overline{w}\| - \sqrt{2}(\mathfrak{u}-1)\Delta_{\overline{w}} \Big) \geq \sum_{\overline{w}\in\overline{\mathsf{D}}} \frac{\|\pi \cap \overline{w}\|}{2},\end{aligned}$$

since $\|\pi \cap \overline{w}\| \geq m\Delta_{\overline{w}}$, for each $\overline{w} \in \overline{\mathsf{D}}$ (by Claim 3.3), and $m = \lceil 2\sqrt{2}\mathfrak{u} \rceil \geq 2\sqrt{2}\mathfrak{u}$. Now, note that $\pi(\overline{w}) \cap \overline{w}$ is a subset of $\pi \cap \overline{w}$, and henceforth it holds that

$$\mathcal{E}_{\pi,\mathfrak{u}} \geq \sum_{\overline{w}\in\overline{\mathsf{D}}} \frac{\|\pi \cap \overline{w}\|}{2} \geq \sum_{\overline{w}\in\overline{\mathsf{D}}} \frac{\|\pi(\overline{w}) \cap \overline{w}\|}{2}. \tag{1}$$

For an m-dense window $\overline{w} \in \overline{\mathsf{D}}$, the path σ output by the algorithm (when applied to π) inside $\overline{w}$ is of length $\leq (1 + 1/m) \cdot \|\pi(\overline{w}) \cap \overline{w}\|$, as this is the performance guarantee provided by Mitchell's analysis [10]. Namely, the error introduced by the approximation inside $\overline{w}$ is bounded by $\|\pi(\overline{w}) \cap \overline{w}\| / m$. For windows (visited by the algorithm when applied to π) that are not m-dense, the path σ within them is identical to the path π. Thus, for the path σ, it follows from (1) that

$$\|\sigma\| - \|\pi\| \leq \sum_{\overline{w}\in\overline{\mathsf{D}}} \frac{\|\pi(\overline{w}) \cap \overline{w}\|}{m} = \frac{2}{m} \sum_{\overline{w}\in\overline{\mathsf{D}}} \frac{\|\pi(\overline{w}) \cap \overline{w}\|}{2} \leq \frac{2\mathcal{E}_{\pi,\mathfrak{u}}}{m} < \frac{\mathcal{E}_{\pi,\mathfrak{u}}}{\mathfrak{u}},$$

since $m \geq 2\sqrt{2}\mathfrak{u}$. ∎

It is possible to prove Lemma 3.7 and Theorem 3.8 directly, by arguing that the skeleton can be replaced by an alternative skeleton that is longer and is still shorter, by the excess in the dense windows, than an optimal path. (Since excess is a global property that is not directly defined for windows, the resulting argument is somewhat more complicated.) We provide the more technical proof above, since it brings to the forefront the notion of surplus. Note that the surplus is decomposition sensitive; as such, it might be much smaller than the excess. Therefore, the analysis of Theorem 3.8 is probably loose, as the bound on the error depends solely on the surplus in the dense windows.

Remark 2. As mentioned in Remark 1, we ignored the use of bridges in describing Mitchell's algorithm [10]. Our analysis implies that the use of those bridges is restricted only to dense windows, where all we need is the performance guarantees

already provided by Mitchell's analysis. In particular, for those dense windows, we can also use Arora's algorithm. This is the main insight we use in extending our algorithm to higher dimensions.

4. An $(\varepsilon, \mathfrak{u})$-approximation algorithm for k-TSP in $\mathbf{R}^d$. In this section, we present an $(\varepsilon, \mathfrak{u})$-approximation algorithm for k-TSP in higher dimensions, by combining Mitchell's methods together with Arora's k-TSP algorithm [3]. Throughout the section, we are concerned with $\mathbb{R}^d$, where $d > 2$ is a fixed constant.

Let $\mathcal{Q}$ be an axis-parallel d-dimensional hypercube that contains the point set P. In the following, m is a fixed constant, and π is a given path visiting k points of P. The following definitions are analogous to the ones in section 3.

DEFINITION 4.1. *A closed, axis-parallel d-dimensional box $\overline{w}$ is a* window *if $\overline{w} \subseteq \mathcal{Q}$. The* extent *of a window $\overline{w}$ is the largest side length of $\overline{w}$ and is denoted by $\Delta_{\overline{w}}$.*

A $(d-1)$-dimensional hyperplane ℓ is a cut *for π, with respect to $\overline{w}$, if ℓ is axis-parallel and ℓ intersects $\overline{w}$; ℓ is an m-*perfect cut *for π, with respect to $\overline{w}$, if ℓ intersects the segments of $\pi(\overline{w}) \cap \overline{w}$ at most m times.*

DEFINITION 4.2. *The combinatorial type of a window $\overline{w}$ with respect to π is the subset of P inside (or on the boundary of) $\overline{w}$ and a listing, for each facet of $\overline{w}$, of the identities of the line segments of $\pi(\overline{w})$ that intersect it. We say that $\overline{w}$ is a* minimal window *if there is no window $\overline{w}'$ that is strictly contained in $\overline{w}$ with the same combinatorial type as $\overline{w}$.*

*For a minimal window $\overline{w}$, if there is no m-perfect cut for π, with respect to $\overline{w}$, then it is m-*dense*. Namely, any axis-parallel $(d-1)$-dimensional hyperplane that intersects $\overline{w}$ has more than m intersection points with $\pi(\overline{w}) \cap \overline{w}$.*

The following claim is an immediate extension of Claim 3.3.

CLAIM 4.3. *If a window $\overline{w}$ is m-dense, then $\|\overline{w} \cap \pi\| \geq m \cdot \Delta_{\overline{w}}$.*

To bootstrap our algorithm, we need a $(1+\varepsilon)$-approximation algorithm for the **WindowTSP** problem in $\mathbb{R}^d$. To this end, note that the **WindowTSP** problem seeks a set of $O(md)$ paths (with prespecified endpoints) that collectively visits a prespecified number of points inside the given window; it is not hard to adapt Arora's technique to solve the **WindowTSP** problem in $n^{O(md)} \cdot O(m \log n)^{(m\sqrt{d})^{O(d)}}$ time, such that the solution has a total length $\leq (1 + 1/m)L(\overline{w})$, where $L(\overline{w})$ denotes the length of an optimal solution inside the specified window $\overline{w}$. (This problem can be solved even faster, but it has no impact on the overall performance of our algorithm.) The required adaption is straightforward, and we omit the tedious but easy details; see [3, 1]. We denote this subroutine by k**DenseAprxTSP**.

For an instance of the **WindowTSP** problem, the algorithm works as follows. If $\overline{w}$ has at most m points of P in its interior, then the subproblem is solved by brute force. Otherwise, the algorithm chooses the smaller value returned by the following two options.

(a) Use k**DenseAprxTSP** to solve this problem, providing a solution with total length at most $(1 + 1/m)L(\overline{w})$, where $L(\overline{w})$ denotes the length of an optimal solution inside $\overline{w}$.

(b) Solve the problem recursively, optimizing over all choices associated with an m-perfect cut of window $\overline{w}$. (As in the $\mathbb{R}^2$ case, before performing the recursive calls, we need to shrink the windows formed by the cut into minimal windows.)

  (i) There are $O(d \cdot n^2)$ choices for a cut. More specifically, there are d choices of (axial) directions, and we can always let the cut pass through either

a point of P or an intersection point between $\pi(\overline{w})$ and the boundary of $\overline{w}$. Since $\pi(\overline{w})$ is a subset of the set of $\binom{n}{2}$ possible segments (namely, all segments connecting a pair of points of P), it follows that there are $O(n^2)$ possible intersection points between $\pi(\overline{w})$ and the boundary of $\overline{w}$.

(ii) There are $O(k)$ choices of the number of points visited in new subproblems, subject to the requirement that the total number of points visited within the two subproblems is equal to the number specified in the given instance.

(iii) There are $O(n^{2m})$ choices of new boundary information on the cut. Specifically, we select $\leq m$ segments (each determined by a pair of points of P) that cross the cut. We require that the boundary information of the new subproblems be consistent with the boundary information of the given instance.

(iv) There are a constant number of choices (since m and d are fixed constants) of connectivity constraints for the two new subproblems determined by the cut, subject to the requirement that these constraints be consistent with the constraints of the given instance.

Let k**TSPAprxAlg** denote this recursive algorithm. One can easily use memoization to turn it into an efficient dynamic programming algorithm. There are $O\big(k \cdot n^{2d} \cdot (n^{2m})^{2d}\big) = O\big(k\, n^{(4m+2)d}\big)$ possible subproblems, since there are $O(k)$ choices for the number of points that should be visited within $\overline{w}$, $O(n^{2d})$ choices of $\overline{w}$, and $O(n^{2m})$ choices of crossing segments on each of the $2d$ facets of $\overline{w}$. (The number of possible connectivity constraints is a constant since m and d are fixed constants.)

Remark 3. Before analyzing this algorithm, observe that it can be viewed as the combination of Mitchell's method [10] with Arora's k-TSP algorithm [3]. Specifically, the algorithm's top structure follows Mitchell's method. However, conceptually, whenever the algorithm "encounters" a dense window, it uses k**DenseAprxTSP** (which is an easy extension of Arora's k-TSP algorithm).

To see why we had to modify Mitchell's algorithm, observe that the algorithm in section 3 cannot be used in higher dimensions directly, because part of Mitchell's k-TSP algorithm relies on a crucial property of m-guillotine subdivisions in the plane. Namely, it introduces (and accounts for the additional length of) bridges on the path to decrease the interaction of the path with the outside world when considering dense windows. It is not known how to extend this directly to higher dimensions. However, the need for bridges arises only when a window is dense. In a dense window (in higher dimensions) we can circumvent this issue altogether by using Arora's algorithm (namely k**DenseAprxTSP**). Similarly, using Arora's algorithm on its own does not suffice here, since it introduces errors (by deflecting paths through "portals") even in windows which are not dense.

Analysis. To analyze the algorithm, we extend the definition of surplus (see Definition 3.5) to higher dimensions in a natural way. The following lemma is the analogue of Lemma 3.7.

LEMMA 4.4. *Let $\overline{\mathsf{D}}$ be a set of interior disjoint windows (inside $\mathcal{Q}$), and let π be a polygonal path inside $\mathcal{Q}$. We have that $\mathcal{E}_{\pi,2} \geq \sum_{\overline{w} \in \overline{\mathsf{D}}} (\|\pi \cap \overline{w}\| - \sqrt{d}\Delta_{\overline{w}})$.*

The following theorem is similar to Theorem 3.8.

THEOREM 4.5. *Let $\pi = \langle p_1, p_2, \ldots, p_k \rangle$ be an arbitrary path that visits k points of P, and let $\mathfrak{u} \geq 2$ be an arbitrary fixed integer. One can compute, in $n^{O(\mathfrak{u}d\sqrt{d})} \cdot (\mathfrak{u}\sqrt{d}\log n)^{(\mathfrak{u}d)^{O(d)}}$ time, a path that starts at p_1 and visits k points of P, with length at most $\|\pi\| + \mathcal{E}_{\pi,\mathfrak{u}}/\mathfrak{u}$.*

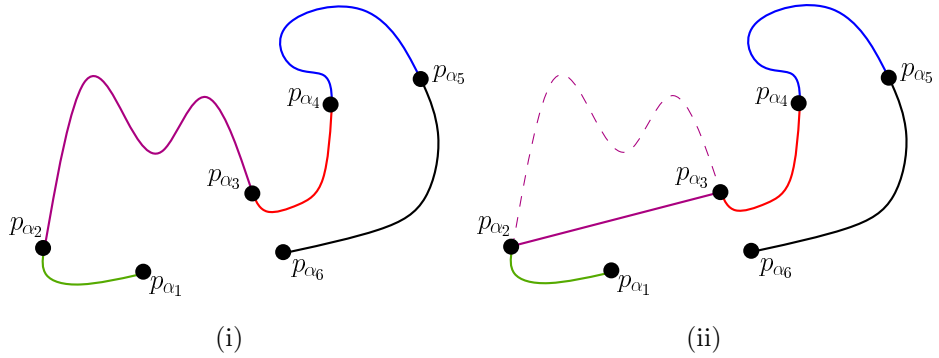


FIG. 6. (i) *The path* π^*_{opt} *is divided into* $\mathfrak{u} = 5$ *subpaths, each of which visits a (roughly) equal number of points.* (ii) *Since* $\mathcal{E}_2 \geq \sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i/\mathfrak{u}$, *we obtain the desired path* π' *by connecting* p_{α_2} *to* p_{α_3}.

Proof. Set $m = \left\lceil 2\sqrt{d} \cdot \mathfrak{u} \right\rceil$. Observe that the algorithm k**TSPAprxAlg** uses the algorithm k**DenseAprxTSP** inside the dense windows, which provides the required approximation guarantee. The argument now follows the proof of Theorem 3.8 (almost) verbatim, and is thus omitted. □

Note that the algorithm in this section also works for the planar case (namely, $d = 2$).

5. A PTAS for orienteering. Next, we apply the algorithm of Theorem 4.5 to the rooted orienteering problem.

LEMMA 5.1. *Given a set* P *of* n *points in* $\mathbb{R}^d$, *a budget* $\mathcal{B}$, *and a root* $r = p_1$, *let* $\pi^*_{\mathrm{opt}} = \langle p_1, p_2, \ldots, p_k \rangle$ *be an optimal rooted orienteering path starting at* r *with budget* $\mathcal{B}$. *One can compute, in* $n^{O(d\sqrt{d}/\varepsilon)} \cdot (\sqrt{d} \log n/\varepsilon)^{(d/\varepsilon)^{O(d)}}$ *time, a path such that it starts at* r *and visits at least* $(1-\varepsilon)k$ *points of* P, *with length at most* $\mathcal{B}$.

Proof. Set $\mathfrak{u} = \lceil 2/\varepsilon \rceil$. Let $\pi^*_{\mathrm{opt}}(i,j) = \langle p_i, p_{i+1}, \ldots, p_j \rangle$ denote the portion of the path π^*_{opt} from p_i to p_j, and let $\mathcal{E}(i,j) = \left\|\pi^*_{\mathrm{opt}}(i,j)\right\| - \|p_i - p_j\|$ denote its 2-excess. Let $\alpha_i = \lceil (i-1)(k-1)/\mathfrak{u} \rceil + 1$. By the definition, we have $\alpha_1 = 1$ and $\alpha_{\mathfrak{u}+1} = k$, and furthermore, each subpath $\pi^*_{\mathrm{opt}}(\alpha_i, \alpha_{i+1})$ visits

$$(2) \quad \alpha_{i+1} - \alpha_i - 1 = \left(\left\lceil \frac{i(k-1)}{\mathfrak{u}} \right\rceil + 1 \right) - \left(\left\lceil \frac{(i-1)(k-1)}{\mathfrak{u}} \right\rceil + 1 \right) - 1 \leq \left\lfloor \frac{k-1}{\mathfrak{u}} \right\rfloor$$

points (excluding the endpoints p_{α_i} and $p_{\alpha_{i+1}}$).

Consider the subpaths $\pi^*_{\mathrm{opt}}(\alpha_1, \alpha_2), \ldots, \pi^*_{\mathrm{opt}}(\alpha_{\mathfrak{u}}, \alpha_{\mathfrak{u}+1})$ of π^*_{opt} and their 2-excesses $\mathcal{E}_1 = \mathcal{E}(\alpha_1, \alpha_2), \ldots, \mathcal{E}_{\mathfrak{u}} = \mathcal{E}(\alpha_{\mathfrak{u}}, \alpha_{\mathfrak{u}+1})$, respectively. Clearly, there exists an index ν, $1 \leq \nu \leq \mathfrak{u}$, such that $\mathcal{E}_\nu \geq (\sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i)/\mathfrak{u}$.

By connecting the vertex p_{α_ν} directly to the vertex $p_{\alpha_{\nu+1}}$ in π^*_{opt}, we obtain a new path $\pi' = \langle p_1, p_2, \ldots, p_{\alpha_\nu}, p_{\alpha_{\nu+1}}, p_{\alpha_{\nu+1}+1}, \ldots, p_k \rangle$. Observe that $\|\pi'\| = \left\|\pi^*_{\mathrm{opt}}\right\| - \mathcal{E}_\nu$, and by (2), π' visits at least $k - (\alpha_{\nu+1} - \alpha_\nu - 1) \geq k - \lfloor (k-1)/\mathfrak{u} \rfloor \geq (1 - 1/\mathfrak{u})k$ points of P. See Figure 6.

Consider the $(\mathfrak{u}+1)$-skeleton $\mathcal{S}' = \langle p_{\alpha_1}, p_{\alpha_2}, \ldots, p_{\alpha_{\mathfrak{u}+1}} \rangle$ of π'. By the definition of $\mathcal{E}_i$, we have that $\|\mathcal{S}'\| = \left\|\pi^*_{\mathrm{opt}}\right\| - \sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i$. Therefore, by the definition of $\mathcal{E}_{\pi',\mathfrak{u}+1}$, we have that

$$\mathcal{E}_{\pi',\mathfrak{u}+1} \leq \|\pi'\| - \|\mathcal{S}'\| = \left(\left\|\pi^*_{\mathrm{opt}}\right\| - \mathcal{E}_\nu \right) - \left(\left\|\pi^*_{\mathrm{opt}}\right\| - \sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i \right) = \sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i - \mathcal{E}_\nu.$$

By applying Theorem 4.5 to the path π', one can compute a path ξ that visits $(1-1/\mathfrak{u})k \geq (1-\varepsilon)k$ points of P, of length

$$\|\xi\| \leq \|\pi'\| + \frac{\mathcal{E}_{\pi',\mathfrak{u}+1}}{\mathfrak{u}+1} \leq \left(\left\|\pi^*_{\mathrm{opt}}\right\| - \mathcal{E}_\nu\right) + \frac{1}{\mathfrak{u}+1}\left(\sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i - \mathcal{E}_\nu\right)$$
$$= \left\|\pi^*_{\mathrm{opt}}\right\| + \frac{1}{\mathfrak{u}+1}\left(\sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i - (\mathfrak{u}+2)\mathcal{E}_\nu\right) \leq \left\|\pi^*_{\mathrm{opt}}\right\| \leq \mathcal{B},$$

since $\sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i - (\mathfrak{u}+2)\mathcal{E}_\nu \leq 0$, implied by $\mathcal{E}_\nu \geq (\sum_{i=1}^{\mathfrak{u}} \mathcal{E}_i)/\mathfrak{u}$. □

Of course, the value of k is not known in advance. Therefore, the algorithm tries all possible values of k from 1 to n and returns the maximum value such that k points of P can be visited within the budget $\mathcal{B}$.

THEOREM 5.2. *Given a set P of n points in $\mathbb{R}^d$, a budget $\mathcal{B}$, and a root r, let k_{opt} be the number of points of P visited by an optimal orienteering path starting at r with budget $\mathcal{B}$. One can compute, in $n^{O(d\sqrt{d}/\varepsilon)} \cdot (\sqrt{d}\log n/\varepsilon)^{(d/\varepsilon)^{O(d)}}$ time, a path that starts at r and visits at least $(1-\varepsilon)k_{\mathrm{opt}}$ points of P, with length at most $\mathcal{B}$.*

6. Conclusions. In this paper, we defined the notion of $(\varepsilon, \mathfrak{u})$-approximation to k-TSP, and showed that Mitchell's k-TSP algorithm [10] actually works as an $(\varepsilon, \mathfrak{u})$-approximation algorithm for the k-TSP problem in the plane. We used it to develop a $(1-\varepsilon)$-approximation algorithm for the orienteering problem. The analysis easily extends to handling the case where both the starting and ending vertices of the orienteering problem are specified. In particular, the algorithm can approximate the best orienteering cycle rooted at a point r.

Our algorithm sheds a light on the power of Mitchell's approach [10], which has the advantage that it introduces errors only when the underlying path is "dense." This is in contrast to Arora's technique [3], which inherently introduces error in the approximation generated.

In the new analysis of the k-TSP algorithm the notion of surplus emerges naturally. We expect it to be much smaller than the excess in a lot of cases, and it might be of independent interest and useful in analyzing other algorithms.

There are numerous problems for further research, including the following:

- Can the running time be significantly improved?
- Can one extend the algorithms presented here to the problem of visiting points with time window constraints [7, 5, 8], where one has to visit a point inside a prespecified time window? This problem seems to be more challenging. Currently, even a constant-factor approximation algorithm is not known for the simple case of visiting points on the line.

Acknowledgment. The authors would like to thank the anonymous referees for their useful comments.

REFERENCES

[1] S. ARORA AND G. KARAKOSTAS, *Approximation schemes for minimum latency problems*, SIAM J. Comput., 32 (2003), pp. 1317–1337.

[2] E. M. ARKIN, J. S. B. MITCHELL, AND G. NARASIMHAN, *Resource-constrained geometric network optimization*, in Proceedings of the 14th Annual ACM Symposium on Computational Geometry, 1998, pp. 307–316.

[3] S. ARORA, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, J. ACM, 45 (1998), pp. 753–782.

[4] S. Arora, *Approximation schemes for NP-hard geometric optimization problems: A survey*, Math. Program, 97 (2003), pp. 43–69.

[5] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, *Approximation algorithms for deadline-TSP and vehicle routing with time-windows*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 166–174.

[6] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, *Approximation algorithms for orienteering and discounted-reward TSP*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 46–55.

[7] R. Bar-Yehuda, G. Even, and S. Shahar, *On approximating a geometric prize-collecting traveling salesman problem with time windows*, J. Algorithms, 55 (2005), pp. 76–92.

[8] C. Chekuri and M. Pál, *A recursive greedy algorithm for walks in directed graphs*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, 2005, pp. 245–253.

[9] N. Garg, *Saving an epsilon: A 2-approximation for the k-MST problem in graphs*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 396–402.

[10] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. Comput., 28 (1999), pp. 1298–1309.

[11] J. S. B. Mitchell, *Geometric shortest paths and network optimization*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 633–701.

[12] P. Toth and D. Vigo, eds., *The Vehicle Routing Problem*, SIAM Monogr. Discrete Math. Appl. 9, SIAM, Philadelphia, 2002.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 398–410

LOWER BOUND FOR THE ONLINE BIN PACKING PROBLEM WITH RESTRICTED REPACKING*

JÁNOS BALOGH†, JÓZSEF BÉKÉSI†, GÁBOR GALAMBOS†, AND GERHARD REINELT‡

Abstract. In 1996 Ivkovič and Lloyd [*A fundamental restriction on fully dynamic maintenance of bin packing*, Inform. Process. Lett., 59 (1996), pp. 229–232] gave the lower bound $\frac{4}{3}$ on the asymptotic worst-case ratio for so-called fully dynamic bin packing algorithms, where the number of repackable items in each step is restricted by a constant. In this paper we improve this result to about 1.3871. We present our proof for a semionline case of the classical bin packing, but it works for fully dynamic bin packing as well. We prove the lower bound by analyzing and solving a specific optimization problem. The bound can be expressed exactly using the Lambert W function.

Key words. bin packing, semionline algorithm, worst-case behavior, lower bound

AMS subject classifications. 68Q25, 68W25, 68W40

DOI. 10.1137/050647049

1. Introduction.

The classical one-dimensional bin packing problem is among the most frequently studied combinatorial optimization problems. In its traditional definition a list $L = \{x_1, x_2, \ldots, x_n\}$ of elements (also called items) with sizes in the interval $(0, 1]$ and an infinite list of unit capacity bins are given. Each element x_i from the list L has to be assigned to a unique bin such that the sum of the sizes of the elements in a bin does not exceed the bin capacity. The size of an element will also be denoted by x_i. The *bin packing problem* consists of packing the items into the bins in such a way that as few bins as possible are used.

It is well known that finding an optimal packing is *NP-hard* [8]. Consequently, a large number of papers have been published which look for polynomial time algorithms that find feasible solutions with an acceptable approximation quality.

For measuring the efficiency of algorithms there are two general methods: the investigation of the worst-case behavior or—assuming some probability distribution of the elements—a probabilistic analysis. In this paper we will concentrate on the asymptotic worst-case ratio of an algorithm. For a given list L, denote by $A(L)$ and $OPT(L)$ the number of bins used by algorithm A and the number of bins used in an optimal packing, respectively. Then the *asymptotic worst-case ratio (AWR)* of algorithm A is

$$R_A := \limsup_{k\to\infty} \left\{ \max_L \left\{ \frac{A(L)}{k} \,\middle|\, OPT(L) = k \right\} \right\}.$$

If an algorithm has an AWR of R_A, we also say that it is R_A*-competitive*.

*Received by the editors December 8, 2005; accepted for publication (in revised form) August 20, 2007; published electronically April 23, 2008. This research was supported by the Hungarian National Research Fund (projects T 048377 and T 046822) and by the MÖB-DAAD Hungarian-German Researcher Exchange Program (project 21). A preliminary version of the paper was presented at the 29th Hungarian Conference of Mathematics, Physics and Computer Science Teachers and appeared in electronic form in the proceedings of the conference published by the organizers.

http://www.siam.org/journals/sicomp/38-1/64704.html

†Department of Computer Science, Gyula Juhász Faculty of Education, H-6701 Szeged, Hungary (balogh@jgypk.u-szeged.hu, bekesi@jgypk.u-szeged.hu, galambos@jgypk.u-szeged.hu).

‡Institute for Computer Science, University of Heidelberg, D-69120 Heidelberg, Germany (gerhard.reinelt@informatik.uni-heidelberg.de).

In recent decades many different types of algorithms have been proposed and also several variants of the classical problem have been introduced and studied [2]. Since the terminology used for classifying algorithms or problems is not unique, we start the paper with a proposal for a new classification based on a different view of the problem. To this end, we distinguish two contributors: the *scheduler*, who provides information on the problem data, and the *loader*, who packs the items. The actual problem then depends on how the scheduler gives the data and which strategies the loader uses for packing.

We suppose that the scheduler reveals information about the items at subsequent discrete time steps. The information consists of messages that certain elements (with respective sizes) arrive and/or that certain elements can be deleted. The total number of such time steps is at most n because information about several items can be given in the same step. If the element x_i appears in step j to be added to the list, then we say that its arrival time a_i is j. In some step k, the scheduler may also inform about the fact that the item x_i can be deleted. We say that the deletion time d_i of x_i is k. From this time the item may be deleted from the bin into which it was packed earlier. Of course, $a_i < d_i$, for $1 \leq i \leq n$. By setting $d_i = \infty$ we indicate that no deletion time for item x_i has been given.

At every time step the loader decides on its actions for packing the items. If some elements are packed and some others are deleted, we assume that first the deletion operations are performed.

In short, the scheduler decides on how the problem data is given, and the loader finds a feasible packing of all items depending on arrival and deletion times. Special strategies now lead to different variants of bin packing.

If $a_i = 1$ and $d_i = \infty$ for every i, $1 \leq i \leq n$, then the loader immediately receives complete knowledge about the list. In this case the scheduler produces an *offline problem* and the loader applies an *offline algorithm*.

If $a_i = i$ and $d_i = \infty$ and the loader packs the items one by one in each step without knowing anything about subsequent elements (neither the sizes nor the number of the elements), and the packed items may not be repacked anymore during the algorithm, then we speak about an *online problem* and an *online algorithm*.

In the case of *semionline (SOL) algorithms* the scheduler specifies the problem online as above, but the loader is allowed to apply at least one of the following operations: a repacking of some items, a lookahead into the next several elements, or some kind of preordering. SOL algorithms allowing only a restricted number of elements to be repacked in each step are called *c-repacking SOL algorithms*.

If the scheduler also allows the deletion of elements, then we speak about a *dynamic bin packing problem (DBP)*, and if the loader exploits this additional information for deleting items then his algorithm is a *dynamic bin-packing algorithm*. In this case $A(L)$ is considered as the maximum number of used bins during the packing. An offline DBP basically would be a special type of scheduling problem. Therefore, in the context of bin packing, only *online and SOL dynamic problems* are investigated.

If, in the SOL case, only repacking is allowed for the loader, and the scheduler may specify some elements to be deleted, then Ivkovič and Lloyd in [12] use the terms *fully dynamic bin packing problem (FDBP)* and *fully dynamic bin packing algorithm*. The c-repacking FDBP can be defined similarly to the definition of c-repacking SOL problems.

For offline bin packing algorithms the best results were achieved in [5] and [13], where the authors proved that for any $\varepsilon > 0$, there are algorithms which can solve

the bin packing problem in linear time, while their AWR is $1+\varepsilon$. Recently, the best online algorithm was given by Seiden in [15] and its AWR is 1.58889. The best-known lower bound 1.5401 is due to van Vliet [16].

SOL algorithms were investigated in [9], [10], and [6]. The first and the second references use different types of lookaheads, and the last one allows repacking. Gambosi, Postiglione, and Talamo [7] also analyzed certain SOL algorithms. In their algorithms repacking was allowed in a special way: those elements which were "large enough" were repacked one by one, while the "small" elements were moved together in a bundle between the bins. In this sense these algorithms can repack even $O(n)$ elements in one step. They analyzed two algorithms. The faster one has linear time and is $\frac{3}{2}$-competitive, and the other one runs in $O(n \log n)$ time and is $\frac{4}{3}$-competitive. Recently, except for trivial results, no good lower bound has been found for the efficiency of SOL algorithms.

Various dynamic approximation algorithms were investigated in [3].

Ivkovič and Lloyd constructed an approximation algorithm to solve the FDBP (see [12]). Similar to the technique used in [7], they also repacked the small elements using a "bundle technique." The competitive ratio of their algorithm is $\frac{5}{4}$ and requires $\Theta(\log n)$ time per step; i.e., its running time is $\Theta(n \log n)$. Investigating the lower bounds, they proved that there is no FDBP algorithm with deletions and where only a constant number of elements may be repacked with a better competitive ratio than $\frac{4}{3}$ [11] (i.e., for the c-repacking FDBP for any c). Using a similar construction in the proof, we can apply this bound to SOL algorithms as well. This fact was also mentioned by Csirik and Woeginger in [4].

In this paper we improve the lower bound to 1.3871 for the c-repacking SOL algorithms, but it follows from the construction that this lower bound is also valid for c-repacking fully dynamic bin packing algorithms. This last fact is coming from the following observation. While we try to construct lower bounds either for the c-repacking SOL problem or the c-repacking FDBP, we realize that the two constructions differ slightly from each other: to allow some deletions will not spoil the c-repacking SOL lower-bound construction for the c-repacking FDBP case.

During our analysis we will use different instruments: LP techniques will be combined with results from linear algebra, and finally we will solve a nonlinear optimization problem.

2. Construction of the linear program. For $k \geq 1$, $n \geq 1$, let $x_1, x_2, \ldots, x_k$ $\left(\frac{1}{2} \leq x_1 < x_2 < \cdots < x_k < 1\right)$ be fixed real numbers and $c \geq 1$ be an arbitrary integer. Define $y_i = 1 - x_i \, (i = 1, \ldots, k)$ and $y_{k+1} = 0$. Furthermore, let $\varepsilon < \min_{j=1,\ldots,k} \{y_j - y_{j+1}\}$ be an arbitrary positive number and $\varepsilon_j := \frac{\varepsilon}{\lceil \frac{n}{2y_j} \rceil}$.

Denote the first list by L_0. Define L_0 as a list of N items of size a, where $a < \frac{\varepsilon_k}{\lceil \frac{n}{y_k} \rceil c}$ and $N := \lfloor \frac{\frac{n}{2} - \varepsilon}{a} \rfloor$. We can see that L_0 contains very small elements with equal, suitably chosen size. The cumulative size of the small items converges to $\frac{n}{2}$ from below if $n \to \infty$. Denote by L_i $(1 \leq i \leq k, k \geq 2)$ the lists containing big elements. Define L_i as a list of $\lceil \frac{n}{2y_i} \rceil$ items of size $x_i + \varepsilon_i$. This means that the sizes of the elements are equal in list L_i. Suppose we need to pack the elements of the lists $L_0, L_1, \ldots, L_k$ with an SOL algorithm which can repack only a constant number of elements. We call these kinds of algorithms c-repacking algorithms. We analyze the behavior of an arbitrary c-repacking algorithm for the lists L_0 and $L_0 L_i$ $(1 \leq i \leq k)$. In our analysis $size(B)$ denotes the total size of items in bin B. We present the lower bound as a solution of a linear programming problem.

LEMMA 1.

$$\limsup_{n\to\infty}\frac{A(L_0)}{OPT(L_0)} \geq 1+\sum_{j=1}^{k}2z_j\left(\frac{1}{y_j}-1\right).$$

Proof. Consider an arbitrary c-repacking algorithm A that has to pack list L_0 first. It is clear that the above-defined a is a very small number, and the total size of the repackable items in $\lceil\frac{n}{2y_j}\rceil$ $(j=1,\ldots,k)$ steps is less than ε_k, which is the smallest among $\varepsilon_1,\ldots,\varepsilon_k$. We want to give a lower bound on $A(L_0)$. Denote by z_in the cumulative size of the items that have been packed in y_i-type bins $(i=1,\ldots,k)$. Bin B is called a y_i-type bin if $size(B)\in(y_{i+1},y_i]$. It is easy to see that the total number of y_i-type bins is at least $\frac{z_in}{y_i}$. The cumulative size of the items that have not been packed in any y_i-type bin is $Na-n\sum_{j=1}^{k}z_j$. So we get that

$$A(L_0)\geq\sum_{j=1}^{k}\frac{z_jn}{y_j}+Na-n\sum_{j=1}^{k}z_j=Na+n\sum_{j=1}^{k}z_j\left(\frac{1}{y_j}-1\right).$$

Because of the definition of N we have $Na\geq\frac{n}{2}-\varepsilon-a$. This means that

$$A(L_0)\geq n\left(\frac{1}{2}+\sum_{j=1}^{k}z_j\left(\frac{1}{y_j}-1\right)\right)-\varepsilon-a. \tag{1}$$

In the following we investigate the behavior of the optimal algorithm on L_0. In the optimal packing each bin B is loaded such that $size(B)>1-a$. So we get

$$OPT(L_0)\leq\frac{Na}{1-a}+1\leq\frac{\frac{n}{2}-\varepsilon}{1-a}+1\leq\frac{\frac{n}{2}}{1-a}+1. \tag{2}$$

From (1) and (2) it follows that

$$\limsup_{n\to\infty}\frac{A(L_0)}{OPT(L_0)}\geq 1+\sum_{j=1}^{k}2z_j\left(\frac{1}{y_j}-1\right). \tag{3}$$ □

LEMMA 2. *For $i=1,\ldots,k$,*

$$\limsup_{n\to\infty}\frac{A(L_0L_i)}{OPT(L_0L_i)}\geq 1+y_i+2y_i\left(\sum_{j=1}^{i-1}z_j\left(\frac{1}{y_j}-1\right)-\sum_{j=i}^{k}z_j\right).$$

Proof. Consider now the packing of the concatenated list L_0L_i $(1\leq i\leq k)$ for fixed i. Let B be a bin which contains an element from L_i. Because the size of the big item is $x_i+\varepsilon_i$, the total size of the small elements in the bin is at most $y_i-\varepsilon_i$. This can also happen in such a way that, after L_0 has been packed, the bin contains more items, but during the processing of L_i, some of them have been repacked to other bins. Since the list L_i has exactly $\lceil\frac{n}{2y_i}\rceil$ elements, at most $c\lceil\frac{n}{2y_i}\rceil$ elements could have been repacked. The size of each small item is a, so from the definition of a and ε_i we get that A can repack small items with at most ε_i cumulative size. So we can say that after packing L_0 $size(B)\leq y_i$ must hold for B. From this it follows that the cumulative size of the small items packed together with an item from L_i is $\sum_{j=i}^{k}z_jn$.

Other small items from L_0 can be packed in y_j-type bins, where $j = 1, \dots, i-1$. The total size of the remaining elements of L_0 is $Na - n\sum_{j=1}^{k} z_j$. So we can estimate the number of bins used by A in the following way:

$$A(L_0L_i) \geq \frac{n}{2y_i} + n\sum_{j=1}^{i-1}\frac{z_j}{y_j} + Na - n\sum_{j=1}^{k} z_j \quad (i = 1, \dots, k). \tag{4}$$

For the optimal packing of L_0L_i $(1 \leq i \leq k)$ we get that

$$OPT(L_0L_i) \leq \left\lceil \frac{n}{2y_i} \right\rceil + S_i \leq \frac{n}{2y_i} + S_i + 1, \tag{5}$$

where S_i is the total number of bins which contain only items from L_0 in an optimal packing. A bin containing an L_i-element will be denoted by B_i. We obtain for the cumulative size $small(B_i)$ of the small elements in B_i that

$$small(B_i) \geq \left\lceil \frac{n}{2y_i} \right\rceil (y_i - \varepsilon_i - a) \geq \frac{n}{2} - \left\lceil \frac{n}{2y_i} \right\rceil \varepsilon_i - \left\lceil \frac{n}{2y_i} \right\rceil a$$

$$\geq Na - \left\lceil \frac{n}{2y_i} \right\rceil a \geq Na - \varepsilon,$$

because $a < \frac{\varepsilon}{\left\lceil \frac{n}{2y_i} \right\rceil}$. So

$$S_i \leq 1. \tag{6}$$

From (5) and (6)

$$OPT(L_0L_i) \leq \frac{n}{2y_i} + 2. \tag{7}$$

Combining (4) and (7) we get that

$$\limsup_{n\to\infty} \frac{A(L_0L_i)}{OPT(L_0L_i)} \geq 1 + y_i + 2y_i \left(\sum_{j=1}^{i-1} z_j \left(\frac{1}{y_j} - 1 \right) - \sum_{j=i}^{k} z_j \right) \tag{8}$$

$$(i = 1, \dots, k). \qquad \square$$

THEOREM 3. *The solution of the following linear programming problem is a lower bound for any SOL c-repacking algorithm:*

$$\min b$$

$$b \geq 1 + y_i + 2y_i \left(\sum_{j=1}^{i-1} z_j \left(\frac{1}{y_j} - 1 \right) - \sum_{j=i}^{k} z_j \right) \quad (i = 1, \dots, k),$$

$$b \geq 1 + \sum_{j=1}^{k} 2z_j \left(\frac{1}{y_j} - 1 \right), \tag{9}$$

$$z_i \geq 0 \ (i = 1, \dots, k),$$

$$\sum_{j=1}^{k} z_j < \frac{1}{2}.$$

Proof. The definition of z_j $(j = 1, \ldots, k)$ obviously implies that $\sum_{j=1}^{k} z_j < \frac{1}{2}$. The statement of the theorem then follows from Lemmas 1 and 2. □

3. Solution of the linear program. To simplify our model we introduce a new variable z, which is the sum of all z_j's, i.e., $z = \sum_{j=1}^{k} z_j$. Using this substitution and some reordering, we can rewrite (9) in the following form:

$$\begin{aligned} &\min\ b \\ -2y_i z + 2y_i \sum_{j=1}^{i-1} \frac{z_j}{y_j} - b &\le -(1+y_i) \quad (i = 1, \ldots, k), \\ -2z + 2\sum_{j=1}^{k} \frac{z_j}{y_j} - b &\le -1, \\ -z + \sum_{j=1}^{k} z_j &= 0, \\ z_i &\ge 0 \ \ (i = 1, \ldots, k), \\ z &< \frac{1}{2}. \end{aligned} \tag{10}$$

Instead of (10) we first solve a linear system of equations related to our problem. The number of equations is $k + 2$. The variables of the system are $z, z_1, \ldots, z_k, b$, while $y_1, \ldots, y_k$ are some fixed parameters, satisfying the conditions of Theorem 3.

$$\begin{aligned} -2y_i z + 2y_i \sum_{j=1}^{i-1} \frac{z_j}{y_j} - b &= -(1+y_i) \quad (i = 1, \ldots, k), \\ -2z + 2\sum_{j=1}^{k} \frac{z_j}{y_j} - b &= -1, \\ -z + \sum_{j=1}^{k} z_j &= 0. \end{aligned} \tag{11}$$

LEMMA 4. *The system* (11) *has a unique solution.*

Proof. Let M be the matrix of (11). Then

$$M = \begin{bmatrix} -2y_1 & 0 & \ldots & 0 & 0 & -1 \\ -2y_2 & 2\frac{y_2}{y_1} & \ldots & 0 & 0 & -1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -2y_k & 2\frac{y_k}{y_1} & \ldots & 2\frac{y_k}{y_{k-1}} & 0 & -1 \\ -2 & \frac{2}{y_1} & \ldots & \frac{2}{y_{k-1}} & \frac{2}{y_k} & -1 \\ -1 & 1 & \ldots & 1 & 1 & 0 \end{bmatrix}. \tag{12}$$

Denote by $\det(M)$ the determinant of M. We prove that $\det(M) < 0$, and so from Cramer's rule the statement of the lemma follows. Let us expand $\det(M)$ using the last column of M. So we obtain

$$\det(M) = \sum_{i=1}^{k+1} -1\,(-1)^{k+2+i} \det(M_i) = \sum_{i=1}^{k+1} (-1)^{k+1+i} \det(M_i), \tag{13}$$

where M_i is a $(k+1) \times (k+1)$ matrix, which is obtained from M by deleting its last column and ith row. Expand $\det(M_i)$ by its last row. So

$$\det(M_i) = (-1)(-1)^{k+2} \det(M_{i1}) + \sum_{j=2}^{k+1} (-1)^{k+1+j} \det(M_{ij}), \tag{14}$$

where M_{ij} is a $k \times k$ matrix, which is obtained from M_i by deleting its last row and jth column. Now we investigate how M_{ij} looks:

$$M_{ij} = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_{i-1} & 2\frac{y_{i-1}}{y_1} & \dots & 2\frac{y_{i-1}}{y_{j-2}} & 2\frac{y_{i-1}}{y_j} & \dots & 0 & 0 \\ -2y_{i+1} & 2\frac{y_{i+1}}{y_1} & \dots & 2\frac{y_{i+1}}{y_{j-2}} & 2\frac{y_{i+1}}{y_j} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ -2y_k & 2\frac{y_k}{y_1} & \dots & 2\frac{y_k}{y_{j-2}} & 2\frac{y_k}{y_j} & \dots & 2\frac{y_k}{y_{k-1}} & 0 \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{j-2}} & \frac{2}{y_j} & \dots & \frac{2}{y_{k-1}} & \frac{2}{y_k} \end{bmatrix}.$$

Case A. Suppose that $j \neq i$ and $j \neq i+1$ and consider only the ith and $(i+1)$th columns of M_{ij}. The two columns are the following:

$$\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 2\frac{y_{i+1}}{y_{i-1}} & 2\frac{y_{i+1}}{y_i} \\ \vdots & \vdots \\ 2\frac{y_k}{y_{i-1}} & 2\frac{y_k}{y_i} \\ \frac{2}{y_{i-1}} & \frac{2}{y_i} \end{bmatrix}.$$

It is easy to see that these columns are not independent, so we obtain that $\det(M_{ij}) = 0$ if $j \neq i$ and $j \neq i+1$.

Case B. If $j = i$, then M_{ii} is a lower triangular matrix of the form

$$M_{ii} = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_{i-1} & 2\frac{y_{i-1}}{y_1} & \dots & 2\frac{y_{i-1}}{y_{i-2}} & 0 & \dots & 0 & 0 \\ -2y_{i+1} & 2\frac{y_{i+1}}{y_1} & \dots & 2\frac{y_{i+1}}{y_{i-2}} & 2\frac{y_{i+1}}{y_i} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ -2y_k & 2\frac{y_k}{y_1} & \dots & 2\frac{y_k}{y_{i-2}} & 2\frac{y_k}{y_i} & \dots & 2\frac{y_k}{y_{k-1}} & 0 \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{i-2}} & \frac{2}{y_i} & \dots & \frac{2}{y_{k-1}} & \frac{2}{y_k} \end{bmatrix}.$$

From this we get that $\det(M_{11}) = 2^k \frac{1}{y_1}$, $\det(M_{(k+1)(k+1)}) = -2^k y_k$, and that for $2 \le i \le k$, $\det(M_{ii}) = -2^k \frac{y_{i-1}}{y_i}$.

Case C. Similarly, if $j = i + 1$, then $M_{i(i+1)}$ is also a lower triangular matrix:

$$M_{i(i+1)} = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -2y_{i-1} & 2\frac{y_{i-1}}{y_1} & \dots & 2\frac{y_{i-1}}{y_{i-2}} & 0 & \dots & 0 & 0 \\ -2y_{i+1} & 2\frac{y_{i+1}}{y_1} & \dots & 2\frac{y_{i+1}}{y_{i-2}} & 2\frac{y_{i+1}}{y_{i-1}} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ -2y_k & 2\frac{y_k}{y_1} & \dots & 2\frac{y_k}{y_{i-2}} & 2\frac{y_k}{y_{i-1}} & \dots & 2\frac{y_k}{y_{k-1}} & 0 \\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{i-2}} & \frac{2}{y_{i-1}} & \dots & \frac{2}{y_{k-1}} & \frac{2}{y_k} \end{bmatrix}.$$

It is easy to see that $\det(M_{i(i+1)}) = -2^k$ if $i \neq 1$ and $\det(M_{12}) = 2^k$.

Using these results, we get from (14) that

$$\det(M_1) = -2^k \left((-1)^{k+2} \frac{1}{y_1} + (-1)^{k+3} \right),$$

$$\det(M_i) = -2^k \left((-1)^{k+1+i} \frac{y_{i-1}}{y_i} + (-1)^{k+1+i+1} \right), \quad i = 2, \dots, k,$$

$$\det(M_{k+1}) = -2^k y_k.$$

So from (13) we obtain

$$\begin{aligned}\det(M) &= -2^k\left((-1)^{2(k+1)}\,\frac{1}{y_1}+(-1)^{2(k+1)+1}\right)\\ &\quad + -2^k\left(\sum_{i=2}^{k}\left((-1)^{2(k+1+i)}\,\frac{y_{i-1}}{y_i}+(-1)^{2(k+1+i)+1}\right)+(-1)^{2(k+1)}\,y_k\right)\\ &= -2^k\left(\frac{1}{y_1}+\sum_{i=2}^{k}\frac{y_{i-1}}{y_i}+y_k-k\right).\end{aligned}$$

Since $y_{i-1} > y_i$ $(i = 2,\dots,k)$ and $\frac{1}{2} \geq y_1$ we get $\frac{1}{y_1}+\sum_{i=2}^{k}\frac{y_{i-1}}{y_i} > k$, and so $\det(M) < 0$. □

LEMMA 5. *The solution of system* (11) *satisfies the conditions of the linear program* (10) *and*

$$b = 1 + \frac{1-y_k}{\frac{1}{y_1}+\sum\limits_{i=2}^{k}\frac{y_{i-1}}{y_i}+y_k-k}.$$

Proof. By substituting the right-hand side of (11) into M, we obtain the matrix M_b,

$$M_b = \begin{bmatrix} -2y_1 & 0 & \dots & 0 & 0 & -1-y_1\\ -2y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & -1-y_2\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots\\ -2y_k & 2\frac{y_k}{y_1} & \dots & 2\frac{y_k}{y_{k-1}} & 0 & -1-y_k\\ -2 & \frac{2}{y_1} & \dots & \frac{2}{y_{k-1}} & \frac{2}{y_k} & -1\\ -1 & 1 & \dots & 1 & 1 & 0 \end{bmatrix},$$

and

$$\det(M_b) = \det(M) + \sum_{i=1}^{k} y_i\,(-1)^{k+1+i}\det(M_i) = \det(M) - 2^k\,(1-y_k).$$

Using again Cramer's rule,

$$b = \frac{\det(M_b)}{\det(M)} = 1 + \frac{1-y_k}{\frac{1}{y_1}+\sum\limits_{i=2}^{k}\frac{y_{i-1}}{y_i}+y_k-k}.$$

By a similar analysis we can prove that the other conditions of (10) also hold. As an example we present only the inequality

$$z = \frac{\det(M_z)}{\det(M)} = \frac{\frac{1}{2}\det(M) - 2^{k-1}\frac{y_k-1}{y_1}}{\det(M)} = \frac{1}{2} + \frac{y_k-1}{2y_1\left(\frac{1}{y_1}+\sum\limits_{i=2}^{k}\frac{y_{i-1}}{y_i}+y_k-k\right)} < \frac{1}{2},$$

where

$$M_z = \begin{bmatrix} -1-y_1 & 0 & \dots & 0 & 0 & -1-y_1 \\ -1-y_2 & 2\frac{y_2}{y_1} & \dots & 0 & 0 & -1-y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -1-y_k & 2\frac{y_k}{y_1} & \dots & 2\frac{y_k}{y_{k-1}} & 0 & -1-y_k \\ -1 & \frac{2}{y_1} & \dots & \frac{2}{y_{k-1}} & \frac{2}{y_k} & -1 \\ 0 & 1 & \dots & 1 & 1 & 0 \end{bmatrix}.$$

The validity of the other inequalities can be shown easily. □

LEMMA 6. *The solution of system* (11) *is an optimal solution of the linear program* (10).

Proof. Consider the solution vector v of (11). Because of Lemma 5 it is a feasible solution of (10). Let v be a basis solution. The objective function contains only b, which has a positive coefficient, satisfying the optimum criterion. So we get the statement of the lemma. □

4. Getting the lower bound. In this section we deal with the optimal choice of the variables $y_1, y_2, \dots, y_k$. To choose them in an optimal way, we have to solve the following nonlinear optimization problem:

$$\max\left\{1 + \frac{1-y_k}{\frac{1}{y_1} + \sum_{i=2}^{k} \frac{y_{i-1}}{y_i} + y_k - k}\right\} \tag{15}$$

$$\text{subject to } \frac{1}{2} \geq y_1 > \dots > y_k > 0.$$

It does not seem to be easy to solve (15) analytically for arbitrary fixed k, but it can be done numerically by global optimization tools. Details can be found in [1]. In this paper we analyze (15) for the case when $k \to \infty$.

LEMMA 7. *The optimal solution of* (15) *converges to the maximum of the function*

$$f(x) := 1 + \frac{1-x}{x + 1 + \ln\left(\frac{1}{x}\right) - \ln(2)}$$

in the interval $\left(0, \frac{1}{2}\right]$ *if* $k \to \infty$.

Proof. We distinguish two cases.

Case 1. In this case we assume that $y_2 \leq \frac{y_1}{2}$. Then $2 \leq \frac{y_1}{y_2}$. Using $y_1 \leq \frac{1}{2}, y_2 \leq \frac{1}{4}, y_k \leq y_2$, $4 \geq 2 + y_k$, and applying the inequality between the arithmetic and geometric means for the numbers $\frac{y_2}{y_3}, \frac{y_3}{y_4}, \dots, \frac{y_{k-1}}{y_k}, y_k$ in the denominator of (15), we obtain the upper approximation (see [14])

$$1 + \frac{1-y_k}{\frac{1}{y_1} + \sum_{i=2}^{k} \frac{y_{i-1}}{y_i} + y_k - k} \leq 1 + \frac{1-y_k}{4 + \sum_{i=3}^{k} \frac{y_{i-1}}{y_i} + y_k - k}$$

$$\leq 1 + \frac{1-y_k}{2 + y_k - k + (k-1)\sqrt[k-1]{y_k}}.$$

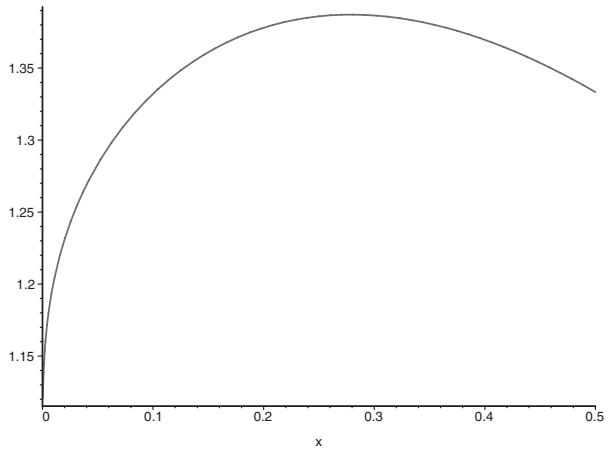


FIG. 1. *Graph of the function $f(x)$ in the interval $\left(0, \frac{1}{2}\right]$.*

Using that $\lim_{k\to\infty} (k-1)\ \sqrt[k-1]{x} - k = \ln(x) - 1$ and $\ln(x) < -\frac{5}{4}$ if $x \le \frac{1}{4}$, we obtain that

$$\lim_{k\to\infty} 1 + \frac{1-y_k}{\frac{1}{y_1} + \sum_{i=2}^{k} \frac{y_{i-1}}{y_i} + y_k - k} \le 1 + \frac{1-y_k}{y_k + 1 + \ln(y_k)} < 1.$$

Case 2. Suppose that $y_2 > \frac{y_1}{2}$. From this we get $2y_2(1-2y_1) \ge y_1(1-2y_1)$, and so

$$\frac{1}{y_1} + \frac{y_1}{y_2} \ge 2 + \frac{1}{2y_2}. \tag{16}$$

Using (16) and $y_1 \le \frac{1}{2}$ and then applying again the inequality between the arithmetic and geometric means for the numbers $\frac{1}{2y_2}, \frac{y_2}{y_3}, \ldots, \frac{y_{k-1}}{y_k}$ in the denominator of (15), we obtain the upper approximation

$$\begin{aligned} 1 + \frac{1-y_k}{\frac{1}{y_1} + \sum_{i=2}^{k} \frac{y_{i-1}}{y_i} + y_k - k} &\le 1 + \frac{1-y_k}{2 + \frac{1}{2y_2} + \sum_{i=3}^{k} \frac{y_{i-1}}{y_i} + y_k - k} \\ &\le 1 + \frac{1-y_k}{2 + y_k - k + (k-1)\ \sqrt[k-1]{\frac{1}{2y_k}}}. \end{aligned}$$

Since $\lim_{k\to\infty} (k-1)\ \sqrt[k-1]{\frac{1}{2x}} - k = \ln\left(\frac{1}{x}\right) - \ln(2) - 1$, we get that

$$\lim_{k\to\infty} 1 + \frac{1-y_k}{\frac{1}{y_1} + \sum_{i=2}^{k} \frac{y_{i-1}}{y_i} + y_k - k} \le 1 + \frac{1-y_k}{y_k + 1 + \ln\left(\frac{1}{y_k}\right) - \ln(2)}.$$

Since $\frac{1-x}{x+1+\ln(\frac{1}{x})-\ln(2)} \ge 0$ if $x \in (0, \frac{1}{2}]$, we get that the maximum of the limit comes from Case 2, which proves the statement of the lemma. The graph of function $f(x)$ of Case 2 can be seen in Figure 1.

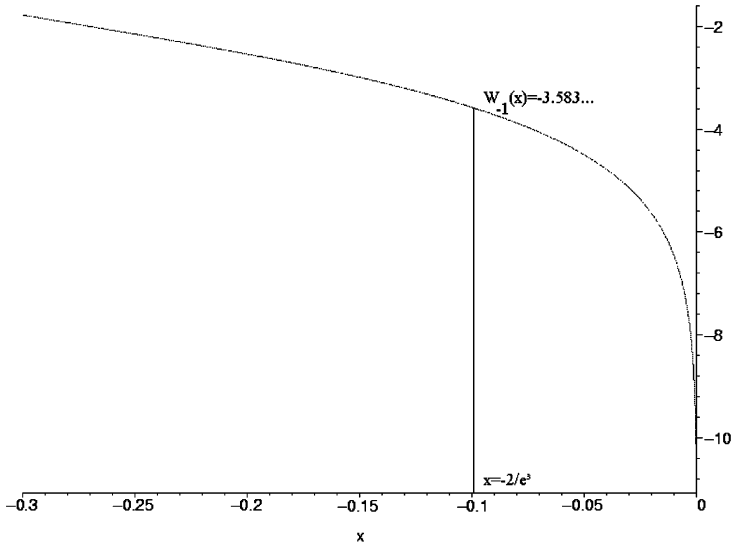


FIG. 2. *Graph of the function* $W_{-1}(x)$ *in the interval* $[-0.3, 0)$.

Here we used the fact that, if we compute the maximum of $f(x)$ in the given interval, then we can construct a series of values $y_1, y_2, \ldots, y_k$ which gives an optimal solution for (15). The construction sets y_1 to $\frac{1}{2}$ and defines the geometrical series $y_i = \frac{1}{2}(2y_k)^{\frac{i-1}{k-1}}$, $i = 2, \ldots, k$, which ends with y_k. □

LEMMA 8. *The maximum of the function* $f(x)$ *is* $1 - \frac{1}{W_{-1}\left(\frac{-2}{e^3}\right)+1} \approx 1.3871$ *in the interval* $\left(0, \frac{1}{2}\right]$, *where* $W_{-1}(x)$ *is the real branch of the Lambert* W *function for which* $W(x) \leq -1$ *holds.*

Proof. The proof is straightforward by taking the derivative of $f(x)$ and investigating the function analytically. Figure 2 displays the graph of the function $W_{-1}(x)$ in the interval $[-0.3, 0)$. □

5. Conclusions. In this paper we discussed improved lower bounds for the c-repacking version of the classical online bin packing problem. Note, however, that these bounds are valid for c-repacking fully dynamic bin packing algorithms as well. Namely, when deletions are allowed, our construction can be applied in the following way. We first give the list L_0, then insert L_1, and after deleting this, we insert L_2, etc. By examining the list L_0 and the L_0L_1, $L_0L_2, \ldots, L_0L_k$ configurations, we obtain the same lower bound with similar arguments as above.

REFERENCES

[1] J. BALOGH, J. BÉKÉSI, G. GALAMBOS, AND M. MARKÓT, *Improved lower bounds for semi-on-line bin packing problems*, submitted for publication (2007); available online at http://www.jgytf.u-szeged.hu/∼balogh/babegama.ps.

[2] E. G. COFFMAN, G. GALAMBOS, S. MARTELLO, AND D. VIGO, *Bin packing approximation algorithms: Combinatorial analysis*, in Handbook of Combinatorial Optimization, D.-Z. Du and P. M. Pardalos, eds., Kluwer, Dordrecht, The Netherlands, 1999, pp. 151–208.

[3] E. G. COFFMAN, JR., M. R. GAREY, AND D. S. JOHNSON, *Dynamic bin packing*, SIAM J. Comput., 12 (1983), pp. 227–258.

[4] J. Csirik and G. J. Woeginger, *On-line packing and covering problems*, in On-Line Algorithms, Lecture Notes in Comput. Sci. 1442, A. Fiat and G. Woeginger, eds., Springer-Verlag, Berlin, 1998, pp. 147–177.
[5] W. Fernandez de la Vega and G. S. Lueker, *Bin packing can be solved within* $1+\varepsilon$ *in linear time*, Combinatorica, 1 (1981), pp. 349–355.
[6] G. Galambos and G. J. Woeginger, *Repacking helps in bounded space on-line bin packing*, Computing, 49 (1993), pp. 329–338.
[7] G. Gambosi, A. Postiglione, and M. Talamo, *Algorithms for the relaxed online bin-packing model*, SIAM J. Comput., 30 (2000), pp. 1532–1551.
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability (A Guide to the Theory of NP-Completeness)*, W.H. Freeman and Company, San Francisco, 1979.
[9] E. F. Grove, *On-line bin packing with lookahead*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1995, pp. 430–436.
[10] G. Gutin, T. Jensen, and A. Yeo, *Batched bin packing*, Discrete Optimization, 2 (2005), pp. 71–82.
[11] Z. Ivković and E. L. Lloyd, *A fundamental restriction on fully dynamic maintenance of bin packing*, Inform. Process. Lett., 59 (1996), pp. 229–232.
[12] Z. Ivković and E. L. Lloyd, *Fully dynamic algorithms for bin packing: Being (mostly) myopic helps*, SIAM J. Comput., 28 (1998), pp. 574–611.
[13] N. Karmarkar and R. Karp, *An efficient approximation scheme for the one-dimensional bin packing problem*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1982, pp. 312–320.
[14] M. Markót, *private communication*, 2005.
[15] S. S. Seiden, *On the on-line bin packing problem*, J. ACM, 49 (2002), pp. 640–671.
[16] A. van Vliet, *An improved lower bound for on-line bin packing algorithms*, Inform. Process. Lett., 43 (1992), pp. 277–284.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 411–428

AN APTAS FOR GENERALIZED COST VARIABLE-SIZED BIN PACKING*

LEAH EPSTEIN† AND ASAF LEVIN‡

Abstract. Bin packing is a well-known problem which has a large number of applications. Classical bin packing is a simple model in which all bins are identical. In the bin packing problem with variable-sized bins, we are given a supply of a variety of sizes. This latter model assumes, however, that the cost of a bin is always defined to be its exact size. In this paper we study the more general problem where an available bin size is associated with a fixed cost, which may be smaller or larger than its size. The costs of different bin sizes are unrelated. This generalized problem has various applications in storage and scheduling. In order to generalize previous work, we design new rounding and allocation methods. Our main result is an asymptotic polynomial time approximation scheme for the generalized problem.

Key words. worst case analysis, approximation algorithm, bin packing

AMS subject classifications. 68Q25, 68W25, 68W40

DOI. 10.1137/060670328

1. Introduction.

Bin packing is a natural and well-studied problem which has applications in computer storage, bandwidth allocation, stock cutting, transportation, and many other important fields. The study of bin packing started more than 30 years ago [5, 7]. Since then, a large amount of research has been dedicated to this problem and its variants (see, e.g., [1, 3, 10]).

An interesting variant of this problem is variable-sized bin packing, in which a supply of containers of some fixed (finite) number of given sizes is available, instead of just a supply of a single bin type. The cost of using a bin is simply its size. The first to investigate the variable-sized bin packing problem were Friesen and Langston [4]. Several papers studied this problem in offline and online environments [12, 2, 13, 14].

We consider the following variant of one-dimensional bin packing, which is a natural generalization of both classical bin packing and variable-sized bin packing. We are given an infinite supply of bins of r types whose sizes are denoted by $b_r < \cdots < b_1 = 1$. We denote $B = \{b_1, \ldots, b_r\}$. Items of sizes in $(0, 1]$ are to be partitioned into subsets. The set of items is denoted S, and the items have indices in the set $\{1, 2, \ldots, n\}$. The size of item j is denoted by s_j. Each subset J in the partition has to be assigned (packed) to some bin type i, such that the set of items fits into an instance of this bin type, i.e., $\sum_{j \in J} s_j \leq b_i$. A bin type i is associated with a cost c_i. We assume $c_1 = 1$. Thus, the cost of a solution is the sum of costs of the bins used, taking multiple subsets which are using the same bin type into account. That is, if the subsets are $J_1, \ldots, J_k$, and subset ℓ is packed into a bin of type i_ℓ (for $1 \leq \ell \leq k$), we get the cost $\sum_{\ell=1}^{k} c_{i_\ell}$. The goal is to find a feasible solution whose total cost is minimized. Without loss of generality we let $b_{r+1} = c_{r+1} = 0$. We call this problem generalized cost variable-sized bin packing (GCVS).

This problem clearly generalizes the classical bin packing problem (where $B = \{b_1\}$) and variable-sized bin packing (where $c_i = b_i$ for $1 \leq i \leq r$). Classical bin

*Received by the editors September 20, 2006; accepted for publication (in revised form) August 29, 2007; published electronically April 23, 2008.
http://www.siam.org/journals/sicomp/38-1/67032.html
†Department of Mathematics, University of Haifa, 31905 Haifa, Israel (lea@math.haifa.ac.il).
‡Department of Statistics, The Hebrew University, Jerusalem, Israel (levinas@mscc.huji.ac.il).

packing assumes a very simple model with uniform bin sizes. Although a large number of real-world problems can indeed be defined as such a problem, an even larger number of problems involve bins of various sizes. This started the study of variable-sized bin packing. To simplify reality, such models usually assume that the cost per unit of storage area is constant. Once again, this is not always the case in real life, as can be easily seen from prices of memory sticks and portable hard disks. Moreover, typically a smaller container has a larger cost per unit of storage, as is the case with various current storage devices. On the other hand, we sometimes encounter a phenomenon where we find that due to technological barriers, the cost of a memory storage device with a capacity twice as large is more than twice the cost of the more modest device.

These are just examples that in reality the cost of storage containers just cannot be assumed to be linear in their sizes. Previous studies of generalized costs functions assumed that the price per unit decreases as the bin size grows. Kang and Park [9], who studied a generalized problem with cost functions satisfying $\frac{c_i}{b_i} \leq \frac{c_j}{b_j}$ for $i > j$, suggested an algorithm of asymptotic approximation ratio $\frac{3}{2}$. Another possibly reasonable assumption is a concave cost function (see [11]). From the scenarios stated above, showing that pricing policies can be arbitrary, we deduce that neither option describes the typical real situation. This leads to the study of general cost functions.

It is known that no approximation algorithm for the classical bin packing problem can have a cost within a constant factor r of the minimum number of required bins for $r < \frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$. This leads to the usage of the standard quality measure for the performance of bin packing algorithms which is the *asymptotic approximation ratio* or *asymptotic performance guarantee*. For an algorithm $\mathcal{A}$, we denote its cost on an input $\mathcal{X}$ by $\mathcal{A}(\mathcal{X})$. An optimal algorithm is denoted by OPT, and its cost of input $\mathcal{X}$ is denoted by $\text{OPT}(\mathcal{X})$.

The asymptotic approximation ratio for an algorithm $\mathcal{A}$ is defined to be

$$\mathcal{R}(\mathcal{A}) = \limsup_{n\to\infty} \sup_{\mathcal{X}} \left\{ \frac{\mathcal{A}(\mathcal{X})}{\text{OPT}(\mathcal{X})} \middle| \text{OPT}(\mathcal{X}) = n \right\}.$$

The natural question, which was whether this measure allows one to find an approximation scheme for classical bin packing, was answered affirmatively by Fernandez de la Vega and Lueker [3]. They designed an algorithm whose output never exceeds $(1+\varepsilon)\text{OPT}(I) + f(\varepsilon)$ bins for an input I and a given $\varepsilon > 0$. The running time was linear in n but depended exponentially on ε. Such a class of algorithms is considered to be an asymptotic polynomial time approximation scheme (APTAS).

Karmarkar and Karp [10] developed an asymptotic fully polynomial time approximation scheme (AFPTAS) for the same problem. This means that using a similar (but much more complex) algorithm, it is possible to achieve a running time which depends on $\frac{1}{\varepsilon}$ polynomially, without any loss in the approximation ratio. Karmarkar and Karp [10] also designed an algorithm which uses at most $\text{OPT}(I) + \log^2[\text{OPT}(I)]$ bins for an input I.

Murgolo [12] designed an APTAS and an AFPTAS for the bin packing problem with variable-sized bins. These results are relatively similar to those of [3, 10] and rely heavily on the fact that the cost of a bin equals its size.

Outline. In this paper, we design an APTAS for GCVS. In section 2 we state and prove some reductions which allow us to seek a slightly simpler structure of solution. These reductions need to be handled carefully to enable the usage of some simplifying assumptions later. In particular, we define a structure for optimal packings, which

turns out to give a solution which is not very different from an overall optimal solution but allows one to simplify the search for optimal solutions. Such a solution allows one to pack an item into a bin that is much larger than the smallest bin that can contain this item or whose cost is not more than a constant multiplicative factor (in terms of ε) away from the cost of such a minimal bin. Before describing the scheme in full detail, we present an outline in section 3. In section 4 we show how to apply grouping and rounding procedures on the input. We partition the input into sets as a function of the smallest bin they can fit into. To be able to apply rounding techniques on each set, we show that the largest items of every group can be packed according to one of two very different packing rules. We note that the number of item sizes resulting from the rounding procedure is not a constant. Therefore, many of the known methods for solving such rounded problems cannot be applied to solve our rounded problem. One example of a standard approach which fails is to formulate an integer programming formulation for the rounded problem. However, in our case its dimension is not constant, and therefore we cannot simply use Lenstra's algorithm [8] to solve the rounded instance and find an optimal packing for it. Despite these difficulties, we show in section 5 that we can get a near-optimal solution to the original problem using a shortest path computation. We use the properties proved in section 2 to reduce to a polynomial size the size of the graph in which we look for the shortest path. Our shortest path computation allocates items to bins, where bin sizes are considered along the path in an increasing order of costs, starting with the cheapest bins. We prove the correctness of our scheme in section 6. We conclude this paper with some remarks in section 7.

This algorithm uses methods of rounding and grouping that are based on ideas from [3] and [12]. However, as the adaptation of these ideas into a scheme with general costs requires a treatment of the bin types in a sorted order, we apply a layered graph based scheme for this. Such a scheme (for a scheduling problem) was given by Hochbaum and Shmoys [6]. To be able to design a solution for the most general problem with no assumptions on the cost function, we additionally apply some novel methods. Note that the running time of our APTAS depends on the number of bin types, r, polynomially (i.e., r is seen as a part of the input). Throughout this paper we denote by ε a fixed positive constant such that $\varepsilon < \frac{1}{100}$ and $\frac{1}{\varepsilon}$ is an integer.

2. Some reductions. In this section we show a series of modifications on B and restrictions on the optimal solution. We do not apply any modifications on the input items at this time. In the following sections we will compute a solution that uses only the modified set of bins and approximates an optimal solution among the possible solutions under the specified restrictions. The first reduction keeps at least one optimal solution unaffected, whereas the other reductions change the optimal solutions and moreover result in an increase in the total cost of an optimal solution. However, we will show that this increase is bounded by a (multiplicative) factor of $1 + \varepsilon$.

LEMMA 1. *Without loss of generality we assume that the values c_i are monotonically decreasing (i.e., for $i < j$, $c_i > c_j$).*

Proof. To achieve this we show that, given a set of bin types and a solution, we can omit some bins from B and change any solution (to a given input) into a solution that does not use bins removed from B and whose cost is not larger than the cost of the original solution. To achieve this, we apply the following process on B. While there exist i, j such that $i < j$ but $c_i \leq c_j$, remove bin type j from B. Note that since $b_i > b_j$, we can move the contents of every bin of size b_j into a bin of size b_i

without increasing the cost of a given solution. This is done until no such pair i, j exists and thus results in a set B' where the sequence of values c_i is monotonically decreasing. □

Since the process described in Lemma 1 can be applied to B without changing the cost of optimal solutions, we use the notation B for the set of bin type to which the process was already applied. We assume in the remainder of the paper that the values c_i are monotonically decreasing.

The following reductions increase the cost of an optimal solution by a factor of at most $1+\varepsilon$. In our analysis, we compare the cost of our approximation algorithm to the cost of an optimal solution for the instance resulting from the reductions and prove an approximation ratio of $1+O(\varepsilon)$. We get that even though the "real" approximation ratio (i.e., the approximation ratio with respect to an optimal solution of the original problem) may be slightly larger, it is at most $1+\varepsilon$ times the approximation ratio that we prove, and thus our analysis results in an approximation factor of $1+O(\varepsilon)$. Therefore, since we are interested in designing an APTAS for the problem, the reductions are harmless from our point of view. The next lemma shows that we can assume that the sequence of bin costs decreases geometrically or faster.

LEMMA 2. *Without loss of generality we may assume that for all i, $\frac{c_i}{c_{i+1}} \geq 1+\varepsilon$.*

Proof. If the claim does not already hold for the input bin types, we apply the following process on the bin types of the input. Traverse the list of bin types from the largest bin (i.e., $j=1$) to the smallest bin ($j=r$). During the traversal keep only a subset of the types, and remove the other types from B. We keep the first bin type ($j=1$) and recursively assume that the last bin type that is kept has index j. Then, given the value j for $i=j+1, j+2, \ldots$, as long as $\frac{c_j}{c_i} < 1+\varepsilon$ and $i > j$ we remove the ith type from the list of bin types. We always keep the bin type with smallest index i such that $\frac{c_j}{c_i} \geq 1+\varepsilon$ and i becomes the new value of j. If there is no such value of i, we remove all bin types $j+1, j+2, \ldots, r$ from B.

Consider a feasible solution that packs the set of items S using a bin of type i that is removed during this process. Then, the set of resulting bins contains a bin type j such that $\frac{c_j}{1+\varepsilon} < c_i < c_j$. Then, we modify the solution by using a bin of type j to pack all the items in S. Applying this procedure on all bins of types that were removed from B results in a feasible solution to the new instance, whose total cost is at most $1+\varepsilon$ times the cost of the original solution. Therefore, the cost of an optimal solution for the new instance is at most $1+\varepsilon$ times the cost of an optimal solution for the original instance. Thus, if we design APTAS for instances satisfying the assumption of the lemma, then the resulting solution will also be an APTAS for the original instances. We conclude that it suffices to consider instances satisfying the property. □

We say that a feasible solution is *nice* if it satisfies the following condition for every pair of a bin and an item. Assume that the solution uses a bin of type i to pack a set of items S. For a given $j \in S$, let k_j be the maximum index such that $b_{k_j} \geq s_j$ (i.e., b_{k_j} is the smallest bin size where j can be packed). Then, either $b_{k_j} \leq \varepsilon^6 b_i$ or $c_{k_j} \geq \varepsilon^8 c_i$ (or both). This means either that an item packed in a bin can fit into a much smaller bin or that the smallest (and thus cheapest) bin that can accommodate this item has a cost which differs from the cost of the current bin by a constant factor. Note that an item that does not fit into any bin of index strictly larger than i immediately satisfies the second condition.

The next lemma shows that we can restrict ourselves to looking for an approximated nice solution.

LEMMA 3. *Given an instance of the GCVS problem, denote by* OPT_n *the minimum cost of a nice solution and by* OPT *the cost of an optimal solution (which is not necessarily nice). Then,* $\text{OPT}_n \leq (1+3\varepsilon)\text{OPT}$.

Proof. Fix an optimal solution O whose cost is OPT. It suffices to show how to transform it into a nice solution whose cost is at most $(1+3\varepsilon)$OPT. We do the transformation for each packed bin in O separately. Assume that O uses a bin of type i to pack the item set which is denoted by C. We use a set of bins to pack C in order to convert the packing of C into a nice packing. To do so, we first identify the sequence $i_1, i_2, \ldots$. This sequence is independent of C and can be computed as a function of i as follows. Let i_ℓ be the smallest value of an index q such that $c_q \leq \varepsilon^{\ell+6} \cdot c_i$. Then, instead of using just one bin (of type i) to pack the items in C, we use one bin of type i and in addition, for each $\ell = 1, 2, \ldots$, we use $\frac{2}{\varepsilon^6}$ bins of type i_ℓ. This set of bins is used to pack C as follows. We use the single bin of type i to pack two sets of items. The first set consists of all items of C with size larger than b_{i_1}. The second set consists of all items j such that $b_{k_j} \leq \varepsilon^6 b_i$ (where k_j is as defined above, the index of a smallest type of bin into which item j can fit). Note that for such items it holds that $s_j \leq \varepsilon^6 b_i$. Denote the set of items that we pack using this unique bin of type i by C_0. The rest of the items from C is partitioned into classes, where the ℓth class, denoted by C_ℓ, contains all items whose size is between $b_{i_{\ell+1}}$ and b_{i_ℓ}. That is, $C_\ell = \{a \in C \setminus C_0 : b_{i_{\ell+1}} < s_a \leq b_{i_\ell}\}$.

Then, the items of set C_0 clearly fit into the bin of type i, since C was originally packed into this bin and $C_0 \subseteq C$. The items of C_ℓ for $\ell \geq 1$ are packed using the first-fit algorithm into at most $\frac{2}{\varepsilon^6}$ bins of type i_ℓ. To see this last claim, note that if $C_\ell \neq \emptyset$, then $b_{i_\ell} \geq \varepsilon^6 b_i$. Therefore, the total size of the items in C_ℓ is at most $\frac{b_{i_\ell}}{\varepsilon^6}$ (as they are packed into a single bin of type i). Since first-fit opens a new bin only if the total size of the items in the previous bin and the new item is at least the capacity of the bin, it has at most one bin with a total size of less than half the size of the bin, and therefore we conclude that first-fit, when applied to C_ℓ and bins of type i_ℓ, will use at most $\frac{2}{\varepsilon^6}$ bins.

Therefore, instead of using one bin of type i whose cost is c_i, we use a set of bins whose total cost is at most $c_i + \sum_{\ell=1}^{\infty} \frac{2}{\varepsilon^6} \cdot c_{i_\ell} \leq c_i + \sum_{\ell=1}^{\infty} \frac{2}{\varepsilon^6} \cdot \varepsilon^{\ell+6} \cdot c_i = c_i \cdot (1 + 2\sum_{\ell=1}^{\infty} \varepsilon^\ell) = c_i(1 + \frac{2\varepsilon}{1-\varepsilon}) \leq c_i(1+3\varepsilon)$, where the last inequality holds since $\varepsilon \leq \frac{1}{3}$.

It is clear that the resulting packing of C_0 into the bin of type i satisfies the conditions of a nice packing since all items violating the condition were removed from this bin. We next prove that the packing of every newly created bin satisfies the conditions of a nice packing as well. Consider an item $j \in C_\ell$ (which is packed into a bin of type i_ℓ). We show that this item satisfies the second condition of nice packings. By definition, $c_{i_\ell} \leq \varepsilon^{\ell+6} c_i$. Note that $i_{\ell+1}$ must exist since $c_{r+1} = 0$. Consider the bin type k_j. By definition of C_ℓ, $s_j > b_{i_{\ell+1}}$, and thus $k_j < i_{\ell+1}$. However, $i_{\ell+1}$ is the smallest index q for which $c_q \leq \varepsilon^{\ell+7} \cdot c_i$, and thus $c_{k_j} > \varepsilon^{\ell+7} \cdot c_i \geq \varepsilon c_{i_\ell} \geq \varepsilon^8 c_{i_\ell}$.

Application of the above transformation on all the bins of O results in a feasible solution that is also nice (as shown above, by the definition of the sets C_ℓ for $\ell \geq 0$) whose cost is at most $(1+3\varepsilon)$OPT. ∎

In what follows we assume that the instance satisfies the assumptions of Lemma 2. We approximate the minimum cost nice solution whose cost is denoted by OPT_n, and we construct a feasible solution whose cost is at most $(1+O(\varepsilon)) \cdot \text{OPT}_n + f(\frac{1}{\varepsilon})$, where f is some function (which will turn out to be polynomial). Note that the solution that we obtain is not necessarily nice, since the original problem does not require this. (It is possible, however, to convert it into a nice solution in polynomial time by applying a construction as above.)

3. Outline of the scheme. In this section we provide the outline of the scheme. The complete details will be given in the following sections.

The first step of the scheme is to preprocess the list of bin sizes so that this list will satisfy the properties of Lemmas 1 and 2.

We next partition the set of items into types. For a bin of type i, we say that an item of size s_j is *large for a bin of type* i if $\varepsilon^6 b_i \le s_j \le b_i$. It is *small for a bin of type* i if $s_j < \varepsilon^6 b_i$, and otherwise it is *huge for a bin of type* i. An item j is *large* if there is a type i such that it is large for a bin of type i. We denote by $\mathcal{L}$ the set of large items. We partition $\mathcal{L}$ into sets: for all i, $\mathcal{L}_i$ consists of all the large items for a bin of type i that are huge for a bin of type $i+1$, and $\mathcal{L}_r$ consists of all the large items for a bin of type r.

The next step of the scheme is to apply linear grouping for each $\mathcal{L}_i$ separately. We denote by S_1^i the set of the largest items resulting from the linear grouping of $\mathcal{L}_i$.

Our scheme looks for solutions that satisfy an additional property. That is, for each i we consider only two possibilities for packing S_1^i: either we have $|S_1^i|$ dedicated bins of size b_i, each of which contains exactly one item from the set S_1^i and no other item is packed into such a special bin, or the items of S_1^i are packed as small items in much larger bins. (Note that we do not allow mixtures of the two options for a given value of i.) We will show that there exists such a solution that does not cost much more than an optimal nice solution.

To find our solution we construct a layered graph. The graph is split into levels, where each level i corresponds to decisions regarding the packing of bins of type i. Each level consists of $3n+1$ layers, where each layer is associated with packing at most one bin of the corresponding type. At the entry for each level we decide whether S_1^i is packed in dedicated bins or the items of S_1^i are packed as small items in much larger bins (the graph contains edges of both possibilities). Each vertex encodes the number of (large) items of each rounded size that still needs to be packed. A vertex encodes also the rounded total size of the small items (i.e., small for the bin type of its level) that still need to be packed. Each vertex needs to recall the subset of the indices i such that S_1^i is packed as small items only if for the current level the items of S_1^i are still large items (and not small items).

We then look for a shortest path in this (very large but still polynomial size) layered graph. This shortest path corresponds to a well-defined packing of the items that are packed as large items. Afterwards, the remaining items need to be distributed to the empty slots in the resulting packing. To this end, additional bins (of each size) are used, if the process of packing small items as indicated by our path did not result in packing a large enough total size of small items. We show that these additional bins have a small cost and do not hurt the returned solution too much. Thus we show that the resulting solution is a good approximation of an optimal solution.

4. Linear grouping. Recall that for a bin of type i, we say that an item of size s_j is *large for a bin of type* i if $\varepsilon^6 b_i \le s_j \le b_i$. It is *small for a bin of type* i if $s_j < \varepsilon^6 b_i$, and otherwise it is *huge for a bin of type* i. An item j is *large* if there is a type i such that it is large for a bin of type i. We denote by $\mathcal{L}$ the set of large items.

We next partition $\mathcal{L}$ into subsets according to the size of the items. $\mathcal{L}_r$ is the set of large items for a bin of type r. If we defined $\mathcal{L}_{j+1}, \ldots, \mathcal{L}_r$, then $\mathcal{L}_j$ is defined as the intersection of the set of large items for a bin of type j and the set $\mathcal{L} \setminus (\mathcal{L}_{j+1} \cup \cdots \cup \mathcal{L}_r)$.

For each i such that $|\mathcal{L}_i| \le \frac{1}{\varepsilon^{16}}$ we pack each item of $\mathcal{L}_i$ in a bin of type i by itself. (Such a bin is called a dedicated bin.) Such a class $\mathcal{L}_i$ with at most $\frac{1}{\varepsilon^{16}}$ elements is called *thin*.

LEMMA 4. *The total cost of packing each item of a thin class into a dedicated bin is at most* $\frac{1+\varepsilon}{\varepsilon^{17}}$.

Proof. Since each bin type is used to pack at most $\frac{1}{\varepsilon^{16}}$ items, the total cost of these bins is at most the total cost of using $\frac{1}{\varepsilon^{16}}$ copies of every bin in the input sequence; i.e., it is at most $\frac{1}{\varepsilon^{16}} \cdot \sum_{i=1}^{r} c_i \leq \frac{1}{\varepsilon^{16}} \cdot c_1 \cdot \sum_{i=0}^{r-1} (\frac{1}{1+\varepsilon})^i \leq \frac{1}{\varepsilon^{16}} \cdot c_1 \cdot \sum_{i=0}^{\infty} (\frac{1}{1+\varepsilon})^i = \frac{1+\varepsilon}{\varepsilon^{17}}$, where the first inequality holds by Lemma 2. ∎

By Lemma 4, we can assume without loss of generality that for each nonempty class of large items, the class has at least $\frac{1}{\varepsilon^{16}}$ elements.

Next, we perform a linear grouping of each class $\mathcal{L}_i$ separately. More precisely, let $|\mathcal{L}_i| = n_i$ (recall that we assume that $n_i \geq \frac{1}{\varepsilon^{16}}$). We sort the elements $a_1^i, a_2^i, \ldots a_{n_i}^i$ of $\mathcal{L}_i$ according to their size. That is, we denote the size of the element a_j^i by s_j^i, and we assume without loss of generality that $s_1^i \geq s_2^i \geq \cdots \geq s_{n_i}^i$. We partition $\mathcal{L}_i$ into $\frac{1}{\varepsilon^{16}}$ subclasses denoted by $\bar{S}_1^i, \bar{S}_2^i, \ldots, \bar{S}_{1/\varepsilon^{16}}^i$. The partition is defined by the following two conditions: $|\bar{S}_p^i| = \lfloor n_i \varepsilon^{16} \rfloor$ or $|\bar{S}_p^i| = \lceil n_i \varepsilon^{16} \rceil$ for all $p \geq 1$, and if $p < q$, then $|\bar{S}_p^i| \geq |\bar{S}_q^i|$ (thus we always have $|\bar{S}_1^i| = \lceil n_i \varepsilon^{16} \rceil$). Moreover, we require that if $a_j^i \in \bar{S}_p^i$ and $a_k^i \in \bar{S}_q^i$ such that $p < q$, then $s_j^i \geq s_k^i$. Thus $\bar{S}_1^i$ is a set which contains the largest $\lceil n_i \varepsilon^{16} \rceil$ elements of $\mathcal{L}_i$ (breaking ties arbitrarily). In general, we partition $\mathcal{L}_i$ to approximately equal-size sets (sets of lower indices may have one additional item compared to sets of higher indices) so that $\bar{S}_j^i$ contains the largest elements from $\mathcal{L}_i \setminus (\bar{S}_1^i \cup \cdots \cup \bar{S}_{j-1}^i)$. We note that $n_i \varepsilon^{16} \leq |\bar{S}_1^i| \leq \lfloor n_i \varepsilon^{16} \rfloor + 1 \leq 3|\mathcal{L}_i \setminus \bar{S}_1^i| \cdot \varepsilon^{16}$, where the last inequality can be proved using simple algebra and the properties $|\mathcal{L}_i \setminus \bar{S}_1^i| = n_i - |\bar{S}_1^i| \geq n_i - (n_i \varepsilon^{16} + 1)$, $\varepsilon < \frac{1}{3}$, and $n_i \geq \frac{1}{\varepsilon^{16}}$.

For all i and all $j \geq 2$, we round up the size of all the elements of $\bar{S}_j^i$ to the size of the largest element of $\bar{S}_j^i$, the set of rounded items is denoted by S_j^i, and we denote by σ_j^i the rounded up size of an item in S_j^i. We also define $S_1^i = \bar{S}_1^i$. The set of rounded (large) items, resulting from $\mathcal{L}_i$ is denoted by $\mathcal{L}_i'$, and the set of all rounded (large) items is denoted by $\mathcal{L}'$. In the APTAS of [3] for the classical bin packing problem, the set of the largest elements in the linear grouping is packed using a separate bin for each such item. This packing is applied also for the later APTAS of Murgolo [12] for the variable-sized bin packing problem. In both cases, such a packing increases the cost of a packing only by an arbitrarily small factor. In the generalized problem, the important property that a small number of items of $\mathcal{L}_i$ can be packed into separate bins of size b_i does not hold, since an optimal packing may pack many items of $\mathcal{L}_i$ into larger bins. Instead, for each i, we allow the algorithm to choose between two possibilities: in the first possibility the algorithm packs S_1^i using a separate bin of size b_i for each item in S_1^i. This is the typical packing of the set of largest items of the linear grouping, and it is useful (by the analysis of the algorithm) in cases where most of the items of $\mathcal{L}_i$ are packed in OPT_n into bins of size smaller than $\frac{b_i}{\varepsilon^6}$. For other cases, we allow the algorithm to pack the set S_1^i, seeing them as a part of the set of small items of larger bins. In this case the algorithm will pack the elements of S_1^i using bins of size at least $\frac{b_i}{\varepsilon^6}$. That is, in our scheme the total size of elements of S_1^i will be added to the total size of small elements that the algorithm needs to pack using bins whose size is at least $\frac{b_i}{\varepsilon^6}$. To state the next lemma we consider a fixed optimal nice solution (the optimum among the nice solutions) denoted by OPT_n. We use $\mathcal{A}_i$ to denote the subset of elements of $\mathcal{L}_i \setminus \bar{S}_1^i$ which OPT_n packs into bins of size smaller than $\frac{b_i}{\varepsilon^6}$. If $|\mathcal{A}_i| \geq \frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|$, we say that $\mathcal{L}_i$ is *good*.

LEMMA 5. *Assume that $\mathcal{L}_i$ is good. Consider a packing where each item of S_1^i is packed into a separate bin of size b_i. In this case, the total cost of bins containing*

the items of S_1^i is at most $6 \cdot \varepsilon^4 \cdot \psi_i$, where ψ_i is the total cost of the subset of bins in OPT_n, that consists of all bins that contain at least one item of $\mathcal{A}_i$.

Proof. We first argue that each bin that is used to pack an element of $\mathcal{A}_i$ has size of at least b_i. Since $\mathcal{A}_i \subset \mathcal{L}_i$, such an item is not large for b_{i+1}, and since $b_{i+1} < b_i$ this means that it is huge for it and does not fit into smaller-size bins. Therefore, the cost for such a bin is at least c_i. Next note that each bin that is used to pack an element of $\mathcal{A}_i$ packs at most $\frac{1}{\varepsilon^{12}}$ such elements. This follows since each such element has size of at least $\varepsilon^6 b_i$, and the size of such bin is smaller than $\frac{b_i}{\varepsilon^6}$ (due to the definition of $\mathcal{A}_i$). Therefore, there exist at least $\frac{|\mathcal{L}_i \setminus \bar{S}_1^i|}{2}\varepsilon^{12}$ bins of OPT_n, where each one of them costs at least c_i, and each one of them contains at least one element of $\mathcal{A}_i$. So $\psi_i \geq \frac{|\mathcal{L}_i \setminus \bar{S}_1^i|}{2}\varepsilon^{12} c_i$. As already stated in the proof of Lemma 4, $|S_1^i| = |\bar{S}_1^i| \leq 3|\mathcal{L}_i \setminus \bar{S}_1^i|\varepsilon^{16}$, and therefore $|S_1^i|$ is at most $6 \cdot \varepsilon^4$ times the number of bins that are used by OPT_n to pack the elements of $\mathcal{A}_i$. The claim follows since each bin used to pack an element of $\mathcal{L}_i$ costs at least c_i which is exactly the cost of the bins used to pack each element of S_1^i. ∎

LEMMA 6. *The total cost of assigning each element of S_1^i a separate bin, for all values of i such that $\mathcal{L}_i$ is good, is at most $3\varepsilon^2 \cdot \text{OPT}_n$.*

Proof. By Lemma 5, the total cost incurred by S_1^i for a value of i such that $\mathcal{L}_i$ is good is at most $6\varepsilon^4$ times the total cost of the bins that OPT_n uses to pack elements of $\mathcal{A}_i$. Since OPT_n is nice, and the first property of nice packings does not hold for the bin sizes that we consider here, we conclude that each of these bins has a cost of at least c_i and at most $\frac{c_i}{\varepsilon^8}$. By Lemma 2, there are at most $\left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon^8} \right\rceil$ bin types with cost in the interval $\left[c_i, \frac{c_i}{\varepsilon^8}\right]$. Therefore, a specific bin can be used by OPT_n to pack elements from at most $\left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon^8} \right\rceil$ different sets $\mathcal{A}_i$. Thus, the total cost of assigning each element of S_1^i a separate bin for all i such that $\mathcal{L}_i$ is good is at most $6\varepsilon^4 \cdot \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon^8} \right\rceil \cdot \text{OPT}_n \leq 3\varepsilon^2 \text{OPT}_n$, where the last inequality can be easily verified and holds since $\varepsilon < \frac{1}{100}$. ∎

By Lemma 6, if we allow our scheme to choose one of the two possibilities described above for each i, then the cost of the largest sets in the linear groupings can be disregarded if the class is good. We argue in the next lemma that if the class $\mathcal{L}_i$ is not good, then by deciding to pack S_1^i in bins of size at least $\frac{b_i}{\varepsilon^6}$, we increase the space demand for such bins by a multiplicative factor of at most $(1 + 6\varepsilon^{10})$. This increase does not cause any harm, since in any case we will have more dominant rounding errors (that will still lead to an $(1 + O(\varepsilon))$-approximate solution with respect to the asymptotic approximation ratio).

LEMMA 7. *If $\mathcal{L}_i$ is not good, then the total size of the elements of S_1^i is at most $6\varepsilon^{10}$ times the total size of elements of $\mathcal{L}_i \setminus \bar{S}_1^i$ that are packed in OPT_n into bins which are of size at least $\frac{b_i}{\varepsilon^6}$.*

Proof. Using $\mathcal{A}_i < \frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|$, we get $|\mathcal{L}_i \setminus (\bar{S}_1^i \cup \mathcal{A}_i)| \geq \frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|$ and thus $\frac{|S_1^i|}{|\mathcal{L}_i \setminus (\bar{S}_1^i \cup \mathcal{A}_i)|} \leq \frac{3|\mathcal{L}_i \setminus \bar{S}_1^i|\varepsilon^{16}}{\frac{1}{2}|\mathcal{L}_i \setminus \bar{S}_1^i|} \leq 6\varepsilon^{16}$. Since $\varepsilon^6 b_i \leq s_j^i \leq b_i$ for all j, the claim follows. ∎

The next lemma holds by a similar argument to the arguments of [3]. Namely, using the fact that if the items of S_1^i are removed for all i, the rounded input can be mapped back into the original input, so that every item is mapped to an item that is no smaller than it.

LEMMA 8. *Given an instance of the GCVS problem, the optimal solution cost of the instance after applying the linear grouping step as described above (with rounded-up sizes), and removing all items in S_1^i for all i, is at most the cost of the optimal solution to the original instance.*

5. The main scheme. In this section we show how to use the linear grouping, as it is described in the previous section, to compute an approximated nice solution. Our method is based on constructing a layered graph and then computing a shortest path in this graph. This shortest path is then used to construct a feasible solution of the GCVS problem.

For $i = 1, 2, \ldots, r$ we denote by d_i the total size of elements whose individual sizes are in the interval $(b_{i+1}, \varepsilon^6 b_i)$. Such elements are huge with respect to bin types $i+1, i+2, \ldots, r$ and small with respect to bin type i. They are not large with respect to any type of bin. If $b_{i+1} \geq \varepsilon^6 b_i$, then $d_i = 0$.

A level in a layered graph is defined as a consecutive set of layers. Our layered directed graph $G = (V, E)$ is composed of $r+1$ levels, where the ith level corresponds to decisions regarding the packing of elements into bins of type i. (Recall that $b_{r+1} = 0$ and hence there are no elements packed into bins of type $r+1$. We add this level to unify the presentation of our scheme.) Each level consists of $3n+1$ layers, where n is the total number of items in the input. Each edge connects a vertex of one layer with a vertex of the consecutive layer, where the layers are ordered so that first there are the $3n+1$ layers of level $r+1$, then the layers of level r, and so on up to the layers of level 1. We add to G one additional vertex denoted by t, which is defined to be the very last layer. We partition the levels into phases as follows. A level i is a *phase p level* if $b_i \in \left(\varepsilon^p, \varepsilon^{p-1}\right]$. For a bin type i, we denote by $B_i = \{k : b_k \in \left[\varepsilon^6 b_i, b_i\right] \text{ and } c_k \in \left[\varepsilon^8 c_i, c_i\right]\}$.

For a phase p level i, a *pattern of level i* corresponds to a packing of a bin of size b_i with elements resulting from the linear grouping, which are of size in the interval $[\varepsilon^6 b_i, b_i]$. These are elements in $\bigcup_{\substack{k \in B_i \\ j \geq 2}} S_j^k \cap \{a \in S : \sigma_a \in [\varepsilon^6 b_i, b_i]\}$. Each such bin can contain space for smaller items as well, where the space defined by the pattern is an integer multiple of $\varepsilon^6 b_i$. This integer is called *the number of slots for small items of the pattern* and is denoted by n_{slot}. Formally, a pattern of level i is $(n_{slot}, (n_j^k)_{\substack{k \in B_i \\ j \geq 2}})$, where n_j^k is the number of items from S_j^k that are packed into this bin. In this notation we assume that $n_j^k = 0$ if $\sigma_j^k < \varepsilon^6 b_i$.

LEMMA 9. *The number of possible patterns of level i is $O((n\varepsilon^{16}+2)^{\frac{8\log_{1+\varepsilon}\frac{1}{\varepsilon}+1}{\varepsilon^{16}}} \cdot (\frac{1}{\varepsilon^6}+1))$; i.e., it is bounded by a polynomial in the input size.*

Proof. The number n_{slot} is an integer in the interval $[0, \frac{1}{\varepsilon^6}]$. The number n_j^k is an integer in the interval $[0, \lceil n\varepsilon^{16} \rceil]$. By Lemma 2, $|B_i|$ is at most $\log_{1+\varepsilon}\frac{1}{\varepsilon^8}+1$. Since each such class (whose index belong to B_i) has at most $\frac{1}{\varepsilon^{16}}$ different values of element size, we conclude that the number of possible patterns of level i is at most $O((n\varepsilon^{16}+2)^{\frac{8\log_{1+\varepsilon}\frac{1}{\varepsilon}+1}{\varepsilon^{16}}} \cdot (\frac{1}{\varepsilon^6}+1))$. ∎

We next describe the vertex set of our layered graph G. A vertex u of the ith level, which is a phase p level, is associated with a label consisting of the following information, which has four parts. The first part contains information on the number $n_j^k(u)$ of items from S_j^k that still needs to be packed, for relevant values of k and j, if the current vertex is reached. That is, for every $k \in B_i$ and for all $j \geq 2$, the label contains the number $n_j^k(u)$ of remaining such elements. The second part of the label contains information on the total size D of small elements for a bin of type i that needs to be packed. This information in the label is $n_{slot}(u) = \lfloor \frac{D}{b_i \varepsilon^6} \rfloor$. Note that D is an approximated sum of such items, which never exceeds the real amount to be packed. The third part of the label contains approximated sums of items that we decided to pack as small items but that are still considered to be large items for bins

of type i, and thus this information should be carried over to future phases. To be more precise, if level i is a phase p level, then the label contains six integer numbers $n^p_{slot}(u)$, $n^{p+1}_{slot}(u), \ldots, n^{p+5}_{slot}(u)$, where $n^q_{slot}(u) \cdot b_i\varepsilon^6$ (for $q = p, p+1, \ldots, p+5$) is an approximated sum of the sizes of all items whose sizes are in $\left(\varepsilon^q, \varepsilon^{q-1}\right]$, and the smallest bin that is large enough to contain such items is of type k for some $k > i$ such that $c_k < \varepsilon^8 c_i$ (and therefore $k \notin B_i$). Note that such items are packed as small items in a nice solution; however, they are still large items for a bin of type i (this is so since for an item x we have $\varepsilon^q < s_x \leq b_k < b_i \leq \varepsilon^{p-1} \leq \varepsilon^{q-6}$ and therefore $s_x \geq \varepsilon^6 b_i$). The fourth part of the label is described next. For all $k \in B_i$, the label contains the information regarding whether we decide to pack each member of S^k_1 using its own bin of size b_k, or all elements of S^k_1 are packed as small items in bins of size at least $\frac{b_k}{\varepsilon^6}$. Therefore, the label encodes a subset $I(u)$ of the index set B_i and if k belongs to this subset, we decide to pack each element of S^k_1 using a dedicated bin of size b_k.

LEMMA 10. *The number of vertices in the graph is polynomial in the input size.*

Proof. Consider a phase p level i and a given layer of this level. In this layer there is a vertex for each possible value of the label. Since the number of levels is $r+1$ and the number of layers in each level is $3n+1$ (plus one last layer which consists of a single vertex), in order to prove the claim it suffices to show that the number of possible labels for level i is polynomial. The number $n^k_j(u)$ is an integer in the range $[0, n_i]$, and therefore there are at most $n+1$ such possibilities. We have to identify this number for the pairs of indices k and j such that $b_k \in [\varepsilon^6 b_i, b_i]$, $c_k \in [\varepsilon^8 c_i, c_i]$ and $j = 2, 3, \ldots, \frac{1}{\varepsilon^{16}}$, and by Lemma 2 there are at most $1 + \log_{1+\varepsilon} \frac{1}{\varepsilon^8}$ values that k can have. Therefore, the number of possibilities for the first part of the label is at most $(n+1)^{(1+\log_{1+\varepsilon} \frac{1}{\varepsilon^8}) \cdot \frac{1}{\varepsilon^{16}}}$ and since ε is a fixed constant, this number is polynomial in the input size. Next, consider the number of possibilities for $n_{slot}(u)$. Note that the number of small elements for a bin of type i is at most n, and each one of them has size less than $\varepsilon^6 b_i$. Therefore, $D < nb_i\varepsilon^6$, and we conclude that $n_{slot}(u) = \lfloor \frac{D}{b_i\varepsilon^6} \rfloor$ is an integer in the interval $[0, n-1]$. Hence, the number of possibilities for $n_{slot}(u)$ is at most n. We next bound the value of $n^q_{slot}(u)$ for $q = p, p+1, \ldots, p+5$. Note that each item considered so far has a size smaller than b_i, and there are at most n such items. Therefore, the total size of these items (that still needs to be packed) is less than nb_i. Hence, $n^q_{slot}(u) \in [0, \frac{nb_i}{b_i\varepsilon^6})$. Since this is an integer, we get that the number of possibilities for this value is polynomial (for a fixed value of ε). It remains to bound the number of possibilities for $I(u)$. By Lemma 2, $|B_i| = \log_{1+\varepsilon} \frac{1}{\varepsilon^8} + O(1)$, and the number of possibilities for $I(u)$ is the number of subsets of B_i that is $O(2^{\log_{1+\varepsilon} \frac{1}{\varepsilon^8}})$ that is a constant (for a fixed value of ε). ∎

We define five types of edges. The first two types are edges connecting two vertices from a common level (and consecutive layers). The last three types of edges connect vertices from consecutive levels. The first type has the purpose of assigning a set of items into a bin. The second type allows us to bypass a bin and not use it. The next two types translate configurations of different levels. The last type allows us to terminate all paths in the target vertex t.

1. The first type of an edge connects two vertices from two consecutive layers of a common level i, where i is a phase p level. It corresponds to a packing of a single bin of size b_i according to a pattern of level i denoted by $(n_{slot}, (n^k_j)_{\substack{k \in B_i \\ j \geq 2}})$. The two vertices connected by such an edge (u, v) have labels that differ only in the first and second part of the label (so $I(u) = I(v)$ and $n^q_{slot}(u) = n^q_{slot}(v)$ for $q = p, p+1, \ldots, p+5$). The first part of the label changes according to $n^k_j(v) = \left(n^k_j(u) - n^k_j\right)^+$ for all $k \in B_i$ and for all

$j \geq 2$ (where for a real number a we let $a^+ = \max\{a, 0\}$). That is, the first part of the the label of v results from the label of u by decreasing the number of items that need to be packed from the set S_j^k by exactly the number of elements from S_j^k that are packed by the pattern corresponding to this edge. Similarly, $n_{slot}(v) = (n_{slot}(u) - n_{slot})^+$. Such an edge has a cost of c_i.

2. The second type of an edge is an edge connecting two vertices in consecutive layers of a common level with equal labels. Such an edge has a zero cost. Such an edge corresponds to the decision not to pack an additional bin of the corresponding size.

3. The third type of an edge connects a vertex u that belongs to the last layer of phase p level $i+1$ to a vertex v that belongs to the first layer of level i, where level i is also a phase p level. In order to obtain the label of v from the label of u we need to apply the following changes. For each value of $k \in B_{i+1} \setminus B_i$, we remove the entries $n_j^k(u)$, for all $j \geq 2$, from the first part of the label of u. These are items that lost their opportunity to be packed as large items and must be packed as small items to maintain the properties of a nice packing. Out of these elements, we compute the total size of elements that need to be packed as small elements according to u and they are already small elements for bin of type i. We need to add to this total size the value of d_i, which are items that are immediately small without being large for any previous bin. However, since the two levels are of the same phase, b_i and b_{i+1} are within a factor of $\frac{1}{\varepsilon}$, which implies that such items do not exist and so $d_i = 0$. Denote the resulting value, which is the sum of all items that became small, by T. Then, the second part of the label of v is exactly $n_{slot}(v) = \lfloor \frac{T}{\varepsilon^6 b_i} + \frac{n_{slot}(u) \cdot b_{i+1}}{b_i} \rfloor$. The first part of the label of v is augmented with new entries for the number of elements of S_j^i that need to be packed for $j \geq 2$; we showed how to compute this number earlier, and it is either $n_j^i = \lfloor n_i \varepsilon^{16} \rfloor$ or $n_j^i = \lceil n_i \varepsilon^{16} \rceil$ for all $j \geq 2$. Note that $B_i \setminus B_{i+1} = \{i\}$; thus these are the only new entries. For $q = p, p+1, \ldots, p+5$, we define $n_{slot}^q(v)$ as follows: for each $k \in B_{i+1}$ such that $c_k < \varepsilon^8 c_i$, we compute $\text{Size}_k^q(u) = \sum_{j \geq 2 : \sigma_j^k \in (\varepsilon^q, \varepsilon^{q-1}]} n_j^k(u) \cdot \sigma_j^k$. We sum up all the values $\text{Size}_k^q(u)$ for all k such that $k \in B_{i+1}$ and $c_k < \varepsilon^8 c_i$, and we denote this sum by $E_q(u)$. That is, $E_q(u) = \sum_{k \in B_{i+1} : c_k < \varepsilon^8 c_i} Size_k^q(u)$. Then we let $n_{slot}^q(v) = \lfloor \frac{n_{slot}^q(u) \cdot b_{i+1}}{b_i} + \frac{E_q(u)}{b_i \varepsilon^6} \rfloor$. We finish the definition of the label of v by setting either $I(v) = I(u) \cap B_{i+1} \cap B_i$ or $I(v) = (I(u) \cap B_{i+1} \cap B_i) \cup \{i\}$ (both edges are constructed, having the exact same cost). The cost of the edge (u, v) is $\sum_{k \in I(u) \setminus I(v)} |S_1^k| \cdot c_k$, and this cost reflects the cost of packing each element of S_1^k in a separate bin of type k if $k \in I(u)$. We charge this packing of the S_1^k to an edge of type 3 or 4, which is a transition between levels, at the time that the packing of S_1^k stops being indicated in the label.

4. The fourth type of edge connects a vertex u that belongs to the last layer of phase p level $i+1$ to a vertex v that belongs to the first layer of level i, where level i is a phase p' level for $p' \leq p-1$. In order to obtain the label of v from the label of u we need to apply the following changes. For each value of $k \in B_{i+1} \setminus B_i$, we remove from the first part of the label of u the entries $n_j^k(u)$ for all $j \geq 2$. Out of these items, we compute the total size of such elements that need to be packed according to u and that are already small for a bin of type i, and we add to this total size the value of d_i. We denote the resulting value by T. We need to take into account the term $Y = \sum_{q=\max\{p, p'+6\}}^{p+5} n_{slot}^q(u)$. The last term corresponds to items that we previously decided to pack as small items but that were still large for bins of type $i+1$, and now such an item x satisfies $s_x < \varepsilon^5 b_i$, as $s_x \leq \varepsilon^{q-1}$ and

$b_i > \varepsilon^{p'} \geq \varepsilon^{q-6}$. Note that we slightly relax the condition of packing items as small and sometimes allow items that are smaller than a bin by a factor of at least ε^5 (instead of ε^6) to be packed as small items in this bin. Then, the second part of the label of v is exactly $n_{slot}(v) = \lfloor \frac{T}{\varepsilon^6 b_i} + \frac{(n_{slot}(u)+Y)\cdot b_{i+1}}{b_i} \rfloor$. The first part of the label of v is augmented with new entries for the number of elements of S_j^i that needs to be packed for $j \geq 2$, and this number is either $n_j^i = \lfloor n_i \varepsilon^{16} \rfloor$ or $n_j^i = \lceil n_i \varepsilon^{16} \rceil$ for all $j \geq 2$. For $q = p', p'+1, \ldots, p'+5$, we define $n_{slot}^q(v)$ as follows. For a given value of q, denote by Y_q the set of pairs (k,j) such that $(k,j) \in Y_q$ if and only if $k \in B_{i+1} \setminus B_i$ and $\sigma_j^k \in (\varepsilon^q, \varepsilon^{q-1}]$. We compute $E_q(u) = \sum_{(k,j)\in Y_q} n_j^k(u) \cdot \sigma_j^k$. For $q \notin \{p, p+1, \ldots, p+5\}$, we denote $n_{slot}^q(u) = 0$. Then, for $q = p', p'+1, \ldots, p'+5$, we let $n_{slot}^q(v) = \lfloor \frac{n_{slot}^q(u)\cdot b_{i+1}}{b_i} + \frac{E_q(u)}{\varepsilon^6 b_i} \rfloor$. Note that every entry $n_{slot}^q(u)$ was translated either into a part of the small jobs or into a part of an entry $n_{slot}^q(v)$ but not to both.

We finish the definition of the label of v by setting either $I(v) = I(u) \cap B_{i+1} \cap B_i$ or $I(v) = (I(u) \cap B_{i+1} \cap B_i) \cup \{i\}$ (again, both edges exist with the same cost). The cost of the edge (u,v) is $\sum_{k \in I(u)\setminus I(v)} |S_1^k| \cdot c_k$ and this cost reflects the cost of packing each element of S_1^k in a separate bin of type k if $k \in I(u)$. As in the previous type of edge, we charge this packing of the S_1^k items to the edges the level transition, where the packing of S_1^k stops being indicated in the label.

5. The last type of edge connects vertices u of the last layer of level 1 to t. Such a vertex u is adjacent to t only if all parts of the label of u except for the last one are either empty sets or are equal to zero. The cost of such an edge if it exists is $\sum_{k \in I(u)} |S_1^k| \cdot c_k$.

Note that $b_{r+1} = 0$ and thus no items can fit into this bin (which was created so that the levels of the graph can be created uniformly). Let a be the vertex of the first layer of level $r+1$ whose label is defined as follows. The first part, which indicates the number of items in S_k^j for $2 \leq j \leq \frac{1}{\varepsilon^{16}}$ and $k \in B_{r+1}$, is empty as $B_{r+1} = \emptyset$. The second part is zero and so are the six amounts in the third part. Finally, $I(a) = \emptyset$. Our scheme finds the minimum cost path $\mathcal{P}$ from a to t. Note that items to be packed are added to the graph as soon as the first level of any bin that can accommodate them is reached.

We next construct a feasible solution based on the path found by the algorithm. If the path uses an edge connecting (u,v), where both of them belong to a common level i and the label of u differs from the label of v, then this edge corresponds to a pattern $(n_{slot}, (n_j^k)_{\substack{k\in B_i \\ j\geq 2}})$. We open a bin of type i and allocate it exactly n_j^k items of S_j^k for all $k \in B_i$ and $j \geq 2$. (The actual allocation is of the $\bar{S}_j^k$ items, but we may assume that each of them occupies the same space which is the size of an S_j^k item.) We also reserve a space of size $n_{slot}\varepsilon^6 b_i$ for small items (which will be allocated later to these spaces). We apply this for all edges of the first type that $\mathcal{P}$ uses. Edges of the second type in $\mathcal{P}$ do not affect the solution (they have cost zero with an empty configuration and are associated with bins that are bypassed). Next assume that $\mathcal{P}$ uses an edge (u,v) of the third or fourth type, where u belongs to the $(i+1)$th level (and v to level i). The effect of such an edge except for translation between levels is to assign large items from a set S_1^k such that $k \in B_{i+1}$ and $k \notin B_i$, which are supposed to be packed in their own bins, one item per bin. In this case for all $k \in I(u) \setminus I(v)$ we open $|S_1^k|$ bins of size b_k and allocate them the items of S_1^k. It remains to consider the edge of the last type that $\mathcal{P}$ uses. Assume that this edge is (u,t); then for all $k \in I(u)$ we open $|S_1^k|$ bins of size b_k and allocate them the items of S_1^k. Note that the total cost of the bins that we open is exactly the cost of $\mathcal{P}$.

We next consider the nonallocated items (which are supposed to be packed as small items as implied by the chosen path). We sort the nonallocated items in a nonincreasing size order. We start to allocate them into the space kept for small items as follows.

Let μ_i be the number of slots for small items in bins of level i in the solution as implied by the sum of values n_{slot} in the patterns. Let R_i be the number of edges of the first type in the path (inside level i) that contain at least one slot for small items. We would like to allow a total size of at least $\varepsilon^6 b_i(\mu_i + 37)$ that will be allocated to items to be packed, and for that we open $R_i\varepsilon^5(1+\varepsilon)+2$ new bins of type i for small items.

We start to allocate the largest remaining item to the space in the smallest-type bin (the largest bin) and keep allocating items according to the sorted list into the space in the bins. While allocating items into bins of level i, if there is not enough room for the current item in the current space we move to the next space. We do this as long as there are free slots in the bins that correspond to edges of the first type. We use the new bins of size b_i and assign small items into them until either we run out of small items to be packed completely or we have used all new bins that were opened for level i as stated above. A new bin that is opened in this step is called a *small item bin of type i*. We later show that our bound (i.e., the number of new bins per level) is large enough, and we bound the additional cost caused by the new bins.

We conclude this section by noting that since the layered graph has polynomial size, and its construction takes polynomial time, finding the shortest path in G takes a polynomial time and constructing the solution based on this path is also polynomial. Therefore, our scheme (for a fixed value of ε) is a polynomial time algorithm. Hence we establish the following corollary.

COROLLARY 11. *Given a fixed value of ε, the time complexity of the scheme is polynomial.*

6. Analysis. In this section we prove that our algorithm is an APTAS for GCVS. We first bound the cost of $\mathcal{P}$. To do so, we present a path from a to t in G whose cost is $(1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$.

LEMMA 12. *There exists a path $\tilde{\mathcal{P}}$ from a to t whose cost is at most* $(1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$.

Proof. Consider an optimal solution OPT_n. From this solution, remove all items which belong to thin classes, as such items do not exist in the layered graph.

Before we can define a path in the graph, we make some adaptations to the solution, and, specifically, we convert it into a packing of the rounded items, where some of the items that are packed as small items are converted into tiny items in the packing.

This packing is later translated into a path in the graph, where items that are packed as small need to be considered carefully. We are going to use the space allocated in the solution to the items of $\bar{S}_j^k$ to accommodate the items of S_{j+1}^k, which are (by definition) no larger of items of $\bar{S}_j^k$. This leaves the items of $\bar{S}_1^k$ unpacked. Thus we create a new instance of each item of $\bar{S}_1^k = S_1^k$, the additional set with the new instances of these items is denoted by $\mathcal{S}^k$. The room taken by each item of the original set $\bar{S}_1^k$ will be used later for items of S_2^k. No adaptations are performed on the placement of a new set $\mathcal{S}^k$. The replacement of $\bar{S}_j^k$ items into the S_{j+1}^k items is done after a packing for the new set $\mathcal{S}^k$ is created.

We next define a packing of the items in $\mathcal{S}^i$ for $1 \le i \le r$. For every such i such

that $\mathcal{L}_i$ is good, i.e., if OPT_n packs at least half of the elements of $\mathcal{L}_i$ using bins of size smaller than $\frac{b_i}{\varepsilon^6}$, we put each item of $\mathcal{S}^i$ into a bin of size b_i. We earlier showed in Lemma 6 that this transformation increases the cost of the solution by at most an additive factor of $3\varepsilon^2\text{OPT}_n$.

We would like to pack the items of $\mathcal{S}^i$ for values of i such that $\mathcal{L}_i$ is not good. This packing is composed of several steps. These items are converted into "sand," which are infinitely small items. We insert a set of empty bins into the packing and bound the resulting increase in the cost. Next, we show that these bins are sufficient to accommodate all sand resulting from items of the sets $\mathcal{S}^i$ for values of i, such that $\mathcal{L}_i$ is not good. We do not convert the items into their original sizes since these items are small for the bins into which they are packed, and the graph takes into account only an approximated sum of sizes of such items rather than the specific sizes. Given the list of bins used by OPT_n we do the following. Let N_i be the number of bins of type i used by OPT_n. We open another $\lceil 2N_i\varepsilon^6 \rceil \leq 2N_i\varepsilon^6 + 1$ bins of type i. This increases the cost of the solution by an additive factor of at most $2\varepsilon^6\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$. (The last term is found by summing up on the costs of all types, similar to the summation in the proof of Lemma 4.)

By Lemma 7, the space required for the sand resulting from a set $\mathcal{S}^i$ (where $\mathcal{L}_i$ is not good) is at most $6\varepsilon^{10}$ times the total size of items in $\mathcal{L}_i \setminus \bar{S}_1^i$ that are packed into bins of size at least $\frac{b_i}{\varepsilon^6}$. The value $D_{i,k}$ is now defined for values of i, k such that $\mathcal{L}_i$ is not good, and $b_k \geq b_i$. We let $D_{i,k}$ be the sum of sizes of the items of $\mathcal{L}_i \setminus \bar{S}_1^i$ that are packed into bins of type k as small items for these bins. For other values of i, k we let $D_{i,k} = 0$. We associate a total size of (at most) $6\varepsilon^{10}D_{i,k}$ of sand created from items of $\mathcal{S}^i$ with these items and, specifically, for every item in $\mathcal{L}_i \setminus \bar{S}_1^i$ packed in a bin of type k such that $D_{i,k} > 0$, which has size x, we associate a total size of sand of $6\varepsilon^{10}x$, which is a part of the sand resulting from $\mathcal{S}^i$. This is done unless the amount of sand that is not associated with any items is smaller, and in this case we associate the remainder and stop. Due to Lemma 7, every portion of the sand is associated with some item. For every bin size b_i, the total size of sand associated with items packed in these bins is therefore at most $6\varepsilon^{10}N_ib_i$. Thus this sand can be packed into the new bins.

We now replace the $\bar{S}_j^k$ items by S_{j+1}^k items. We have a solution for the rounded items, where all items are packed, but some items that are packed as small (not all of them) are seen as sand. We say that an item j is *tiny* for bin i if the smallest bin that can be used for packing j has size b_k such that $b_k < \varepsilon^6 b_i$. We next convert all items that are packed as tiny items for their bins into sand. We would like to convert the total amount of sand in a bin of type k to be an integer multiple of $\varepsilon^6 b_k$. Given a bin with a total amount of sand which is Δ, we keep an amount of $\Delta' = \lfloor \frac{\Delta}{\varepsilon^6 b_k} \rfloor \varepsilon^6 b_k$ and move the remainder of total size $\Delta - \Delta'$ to the new bins of this type. The total size of sand that these new bins need to accommodate increases to at most $6\varepsilon^{10}N_kb_k + \varepsilon^6 b_k N_k \leq 2N_k\varepsilon^6 b_k$. Thus the new bins have enough space to accommodate these items. Empty bins are removed from the solution.

The cost of the current solution is at most $(1 + 3\varepsilon^2 + 2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$. We show a path in the graph with at most the cost of the current solution.

In order to proceed, we identify a subset of the vertex set U as follows. Consider a vertex u of level i. We define a set of indices $J(u) \subseteq B_i$ as follows. For every $k \in B_i$, $k \in J(u)$ if and only if the items of $\mathcal{S}^k$ are packed in separate bins, i.e., if $\mathcal{L}_k$ is good. The set U is defined to contain the vertex u if and only if $I(u) = J(u)$. We also define $t \in U$.

Then, $a \in U$ by definition, we start the path in a and proceed to define the path edge by edge until t is reached. We show a path in the induced subgraph of G over U whose cost is at most the cost of the current solution.

Since the number of layers in a level of the graph is $3n + 1$ (corresponds to the packing of at most $3n$ bins), we show that for all values of k, the number of bins of type k in the current solution never exceeds $3n$. Clearly, $N_k \leq n$, since OPT_n packs at least one item in every bin. Therefore, we get $N_k + \lceil 2N_k\varepsilon^6 \rceil \leq N_k(1 + 2\varepsilon^6) + 1 \leq 3n$.

We are ready to construct the path. The first vertex in the path is a. At each step, we need to show which edges are traversed inside levels and which edges are used between levels.

When we reach a first vertex of a level, we traverse an edge for every bin, except for bins of the solution that contain single items of $\mathcal{S}_i$. The cost of these bins is added to the cost of the path in the edges of the third, fourth, and fifth type. For a given bin f, denote by Δ the total size of the items that are tiny items for a bin of type i, which are packed into this bin. Recall that Δ is an integer multiple of $\varepsilon^6 b_i$, unless this is the last bin of sand. If there is at least one item in the bin that is not sand, we further calculate the number ν_j^k of items from S_j^k for all $k \in B_i$ and for all $j \geq 1$. Then we use in our path an edge of pattern $(n_{slot} = \frac{\Delta}{\varepsilon^6 b_i}, (n_j^k = \nu_j^k))$. For every bin which contains only sand, we traverse an edge of pattern $(n_{slot} = \frac{1}{\varepsilon^6}, (n_j^k = 0))$. The edges that are traversed are those that decrease n_{slot} by the largest possible value (i.e., by $\frac{1}{\varepsilon^6}$, unless n_{slot} is smaller than this value at some point, and then it becomes zero in the next vertex).

The remaining edges connecting vertices inside level i will be the zero cost edges.

Next, we need to define the edges we traverse that connect levels. However, since we use only the subset U of vertices, there is a unique edge connecting the last vertex we reached in a level to a vertex of the next level. For an outgoing edge (u, v) the label of v is defined by the label of u and the vertex set U. Therefore, there is a unique edge left for every u.

Therefore, this identifies a path in the network. The cost of the path up to the last layer of level 1 is identical to the cost of the adapted solution.

Finally, we need to show that the path reaches vertex t. Since there is one-to-one correspondence between the location of items, which are defined to be packed not as tiny items, in the packing and in the path, we need to consider items that are packed as tiny. That is, we need to show that in the last vertex of the last level which we defined for the path, the value of n_{slot} is zero.

Since the sum of items that are to be packed as tiny in the graph is only rounded down and never rounded up, the space in the solution that is used for sand is large enough to pack these items. In the packing, the total size of sand in a bin of type k is always an integer multiple of $\varepsilon^6 b_k$, except for possibly the last bin, which contains only sand. However, we allow the edge of the graph that corresponds to this bin to use the complete bin for sand, which may only result in allocation of additional space for items that are packed as tiny.

Denote by $\mathcal{G}$ the set of indices i such that $\mathcal{L}_i$ is good. Then, we note that nontiny items in OPT_n that are not in $\cup_{i \notin \mathcal{G}} \mathcal{S}^i$ are transformed to sand if and only if they are tiny, and the items in $\cup_{i \notin \mathcal{G}} \mathcal{S}^i$ are transformed to sand (items in $\cup_{i \in \mathcal{G}} \mathcal{S}^i$ are not transformed to sand). □

We next need to consider the path $\mathcal{P}$ that the algorithm finds. For this path we show how a packing is created and bound its cost.

Clearly, the cost of $\mathcal{P}$ is at most the cost of $\tilde{\mathcal{P}}$.

Since the total cost of the bins corresponding to the patterns that belong to $\mathcal{P}$ is at most the total cost of $\tilde{\mathcal{P}}$, we conclude that the total cost of $\mathcal{P}$ is at most $(1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}$. We denote the total cost of $\mathcal{P}$ by $c(\mathcal{P})$. Since the total cost of the solution is partitioned into the total cost of the path $\mathcal{P}$ and the total cost of the small item bins of type i for all i, in order to prove that our scheme is an APTAS, it suffices to show that the total cost of the small item bins is at most $\varepsilon^4 c(\mathcal{P}) + \frac{2}{\varepsilon}$.

LEMMA 13. *The total cost of the small item bins is at most* $\varepsilon^4 c(\mathcal{P}) + \frac{2(1+\varepsilon)}{\varepsilon}$.

Proof. We first prove that items packed as tiny items of bins of type i are of size at most $\varepsilon^5 b_i$. To prove this it is enough to show by induction that for every level i, the items that were supposed to be packed as tiny in levels $1,\ldots,i$ as implied by the selected path can indeed be packed in such bins. These are items that are large for bins $i+1,\ldots,r$ but are not packed as large items there according to the path, and items that belong to some set S_1^ℓ that are supposed to be packed as tiny in bins of types $1,\ldots,i$. The claim must hold for $i=0$ before any tiny items are packed, since no such items exist. Assume that the claim holds for a level $k-1$. This means that the items that need to be packed in level k or in smaller bins are no larger than $\varepsilon^5 b_k$. The loss of a factor of ε (i.e., the reason that we consider $\varepsilon^5 b_k$ and not $\varepsilon^6 b_k$) is due to the fact that levels are partitioned into phases and we may allow a large item for a bin of type k to be packed in bin of type k as tiny, if its size is at most $\varepsilon^5 b_k$ and it is supposed to be packed as tiny in a bin. This happens only if it belongs to some set S_1^ℓ where $c_\ell < \varepsilon^8 c_k$.

The edges of the path assign spaces for the sum of all items (in terms of sand) except an amount that was lost when rounding down was applied. Such rounding for the current level was applied at most 37 times; one such time is in the calculation of the total number of slots needed for the tiny items. The other times were applied on the information on items that had to be packed as tiny, but were not small enough just yet, which were kept in six components. Each such component is updated at most six times (it is changed only between phases and not between every pair of levels) before it is added to the total number of tiny items.

Thus by increasing the space, reserved for items which are packed as tiny in this level, to $\varepsilon^6 b_k(\mu_k+37)$, the allocated space is large enough for the items if they are packed as sand.

We conclude that here may be an additional amount of sand at most $37\varepsilon^6 b_k$. Thus, if we make sure that at least this total sum of items can always be packed in bins of level $k+1$ (unless all tiny items are packed, and no additional tiny items remain to be packed into smaller bins), the inductive claim is proved. We need to take into account that the real items are not sand but are of size at most $\varepsilon^5 b_k$.

If no new bins for small items are opened in level k we are done, since this means that all remaining unpacked items are packed. Assume therefore that in level k, at least one bin for small items was opened. When the items are packed greedily into the existing bins, the space that is reserved in each bin is filled except for possibly a remainder of size at most $\varepsilon^5 b_i$ in a bin of size b_i. The reason is that some item of size at most $\varepsilon^5 b_i$ did not fit in it. This is true for the original bins as well as for the new bins.

The total size of tiny items that can be packed into the original bins of this level is therefore at least $\varepsilon^6 b_k \mu_k - \varepsilon^5 b_k R_k$, where R_k is the number of edges in the path which belong to level k and correspond to nonempty patterns. After the original bins are used, new ones are opened, where each new one is opened only after a previous bin is full up to a level of at least $b_k - \varepsilon^5 b_k$. Thus, if we keep opening

bins until all required items are packed, the number of additional bins is at most $\lceil \frac{\varepsilon^5 b_k R_k + 37\varepsilon^6 b_k}{b_k - \varepsilon^5 b_k} \rceil \leq \varepsilon^5(1+\varepsilon)(R_k + 37\varepsilon) + 1 \leq R_k\varepsilon^5(1+\varepsilon) + 2$. The inequalities hold since $\varepsilon < \frac{1}{100}$. Since the algorithm opens this number of bins, we can conclude that it succeeds to pack all required items. The claim is therefore proved for level k.

We next calculate the additional cost, on top of the cost of the path. Since $\sum_{i=1}^{r} c_i \leq \sum_{i=1}^{r} \frac{1}{(1+\varepsilon)^i} \leq \frac{1+\varepsilon}{\varepsilon}$, we conclude that if we ignore the last two *small item bins* of each type i and consider only the remaining small item bins, it suffices to show that the total cost of these bins is at most $\varepsilon^4 c(\mathcal{P})$. However, this holds since the original cost is at least $\sum_{i=1}^{r} R_i c_i$ (recall that R_i is the number of edges of the first type inside level i that contain some tiny items), and the additional cost is $\sum_{i=1}^{r} \varepsilon^5(1+\varepsilon)(R_i c_i) \leq \sum_{i=1}^{r} \varepsilon^4(R_i c_i)$ (since $\varepsilon(1+\varepsilon) < 1$). ∎

We next show our main result of this paper, i.e., that our scheme is an APTAS for GCVS.

THEOREM 14. *The main scheme is an APTAS for GCVS.*

Proof. Recall that ε is a fixed positive constant. Then, by Corollary 11, finding a feasible solution based on our scheme can be done in polynomial time. It remains to bound the performance guarantee of our algorithm.

By Lemmas 4, 12, and 13, we conclude that the total cost of the solution is at most $(1+\varepsilon^4)((1+3\varepsilon^2+2\varepsilon^6)\text{OPT}_n + \frac{1+\varepsilon}{\varepsilon}) + \frac{(1+\varepsilon)}{\varepsilon^{17}} + \frac{2(1+\varepsilon)}{\varepsilon} \leq (1+4\varepsilon^2)\text{OPT}_n + \frac{8}{\varepsilon^{17}}$ (since $\varepsilon < \frac{1}{100}$). By Lemma 3, $\text{OPT}_n \leq (1+3\varepsilon)\text{OPT}$, and, therefore, the cost of the solution returned by the scheme is at most $(1+4\varepsilon^2)(1+3\varepsilon)\text{OPT} + \frac{8}{\varepsilon^{17}}$. Taking the value of OPT before applying Lemma 2 the performance guarantee increases to $(1+4\varepsilon^2)(1+3\varepsilon)(1+\varepsilon)\text{OPT} + \frac{8}{\varepsilon^{17}}$. The claim follows from $(1+\varepsilon)(1+4\varepsilon^2)(1+3\varepsilon) \leq 1+6\varepsilon = 1+O(\varepsilon)$ (since $\varepsilon < \frac{1}{100}$) and from $\frac{8}{\varepsilon^{17}} = O(1)$. ∎

7. Concluding remarks. We showed how to obtain an APTAS for the variant of variable-sized bin packing where the cost of a bin of size b_i is a general cost and not necessarily b_i. In order to establish our scheme we needed to reduce the problem using Lemmas 2 and 3. We note that even though the reduction in Lemma 2 uses standard tricks, the property of the nice solution is a novel method and is the crucial tool in the construction of our scheme. It is clear that the notion of a nice solution is the main combinatorial structure that allowed us to reduce the time complexity of the scheme into a polynomial scheme. We argue that similar approaches can provide generalizations of approximation results for unweighted problems into approximations for weighted variants (using different notions of nice solutions that would be tailored per problem).

REFERENCES

[1] E. G. COFFMAN, M. R. GAREY, AND D. S. JOHNSON, *Approximation algorithms for bin packing: A survey*, in Approximation Algorithms, D. Hochbaum, ed., PWS Publishing, New York, 1997.

[2] J. CSIRIK, *An online algorithm for variable-sized bin packing*, Acta Inform., 26 (1989), pp. 697–709.

[3] W. FERNANDEZ DE LA VEGA AND G. S. LUEKER, *Bin packing can be solved within* $1+\varepsilon$ *in linear time*, Combinatorica, 1 (1981), pp. 349–355.

[4] D. K. FRIESEN AND M. A. LANGSTON, *Variable sized bin packing*, SIAM J. Comput., 15 (1986), pp. 222–230.

[5] M. R. GAREY, R. L. GRAHAM, AND J .D. ULLMAN, *Worst-case analysis of memory allocation algorithms*, in Proceedings of the 4th Symposium on Theory of Computing (STOC'72), 1972, pp. 143–150.

[6] D. S. Hochbaum and D. B. Shmoys, *A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach*, SIAM J. Comput., 17 (1988), pp. 539–551.
[7] D. S. Johnson, *Near-Optimal Bin Packing Algorithms*, Ph.D. thesis, MIT, Cambridge, MA, 1973.
[8] H. W. Lenstra, Jr., *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.
[9] J. Kang and S. Park, *Algorithms for the variable sized bin packing problem*, European J. Oper. Res., 147 (2003), pp. 365–372.
[10] N. Karmarkar and R. M. Karp, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82), 1982, pp. 312–320.
[11] C.-L. Li and Z.-L. Chen, *Bin-packing problem with concave costs of bin utilization*, Naval Res. Logist., 53 (2006), pp. 298–308.
[12] F. D. Murgolo, *An efficient approximation scheme for variable-sized bin packing*, SIAM J. Comput., 16 (1987), pp. 149–161.
[13] S. S. Seiden, *An optimal online algorithm for bounded space variable-sized bin packing*, SIAM J. Discrete Math., 14 (2001), pp. 458–470.
[14] S. S. Seiden, R. van Stee, and L. Epstein, *New bounds for variable-sized online bin packing*, SIAM J. Comput., 32 (2003), pp. 455–469.

SIAM J. COMPUT.
Vol. 38, No. 1, pp. 429–447

BINARY SPACE PARTITIONS FOR AXIS-ALIGNED FAT RECTANGLES*

CSABA D. TÓTH†

Abstract. It is shown that for any n disjoint axis-aligned fat rectangles in three-space there is a *binary space partition* (BSP) of $O(n \log^8 n)$ size and $O(\log^5 n)$ height and it can be constructed in $O(n \operatorname{polylog} n)$ time. This improves earlier bounds of Agarwal et al. [*SIAM J. Comput.*, 29 (2000), pp. 1422–1448]. On the other hand, for every $n \in \mathbb{N}$, there are n disjoint axis-aligned fat rectangles in $\mathbb{R}^3$ such that their smallest axis-aligned BSP has $\Omega(n \log n)$ size.

Key words. binary space partition, convex partition, fat objects, axis-aligned rectangles

AMS subject classifications. 05A18, 52C45, 68U05

DOI. 10.1137/06065934X

1. Introduction.

For a set of input objects in the Euclidean space, a *binary space partition* (BSP) is a recursive cutting scheme that dissects the space along hyperplanes into convex cells and, in the meantime, dissects the input objects as well. A BSP corresponds to a data structure, called a *BSP tree*, that stores all cutting hyperplanes and the final fragments of the input objects. Initially, the BSP tree data structure was designed for efficient hidden-surface removal from a moving viewpoint by Fuchs, Kedem, and Naylor [16] based on ideas of Schumacker et al. [22] in the computer graphics community. Ever since, BSPs have found innumerable applications in robotics [7], shadow generation [8, 9], solid modeling [23], graph drawing [6], network design [18], collision detection [2, 3], range counting [13], point location [5, 10], and in many other fields.

The efficiency of most applications depends crucially on the space complexity of the underlying BSP tree. The *size* of a BSP is the total number of fragments into which the input objects are partitioned. The BSP size gives an upper bound on the number of leaf nodes in the BSP tree if the BSP does not make redundant cuts; i.e., the cutting hyperplane always partitions the convex hull of the objects. Ideally, none of the input objects are cut during the partitioning, and the BSP size equals the number of input objects. In many cases, however, it is inevitable that input objects are also fragmented, and the size of the BSP becomes superlinear. There are configurations where the smallest BSP for n disjoint convex objects (for instance, line segments in three-space [14]) has $\Theta(n^2)$ size, which is often prohibitive for applications. The *height* of a BSP is the height of the corresponding BSP tree. *Balanced* BSP trees are important for some applications, such as ray tracing and solid modeling, and for the efficient construction of the BSPs [4].

After numerous heuristic algorithms constructing small size BSPs [16, 19, 23], Paterson and Yao [20] were the first to study their combinatorial properties. They showed that for any n disjoint line segments in the plane there is a BSP of $O(n \log n)$ size and $O(\log n)$ height. The currently known best lower bound is $\Omega(n \log n / \log \log n)$ [24].

*Received by the editors May 9, 2006; accepted for publication (in revised form) September 6, 2007; published electronically May 21, 2008. A preliminary version of this paper has appeared in the *Proceedings of the 11th European Symposium on Algorithms*, Budapest, Hungary, 2003, Lecture Notes in Comput. Sci. 2832, Springer-Verlag, Berlin, 2003, pp. 494–505.
http://www.siam.org/journals/sicomp/38-1/65934.html
†Department of Mathematics, MIT, Cambridge, MA 02138 (toth@math.mit.edu).

Paterson and Yao [20] showed that n disjoint triangles in three-space always have a BSP of $O(n^2)$ size and $O(n)$ height. There is a set of n disjoint triangles (each having one side along a hyperbolic paraboloid) for which every BSP has $\Omega(n^2)$ size [14]. The best currently known upper bound for the BSP size of disjoint $(d-1)$-dimensional simplices in $\mathbb{R}^d$, $d \geq 3$, is $O(n^d)$.

Theoretical research focused on object classes where near linear size BSP is possible. There is an $O(n)$ size BSP for n disjoint segments in the plane if they have a constant number of distinct orientations [25], if the ratio of the longest and the shortest segment is bounded by a constant [12], or if they are convexly independent (every line segment has an end point along the convex hull) [12].

Paterson and Yao [21] constructed a BSP of $O(n^{3/2})$ size and $O(\log n)$ height for n disjoint *axis-aligned* objects in $\mathbb{R}^3$. They also showed that this bound is tight in the worst case apart from a constant factor. Dumitrescu, Mitchell, and Sharir [15] extended this result and proved an upper bound of $O(n^{d/(d-k)})$ on the BSP size of n disjoint axis-aligned k-dimensional boxes in $\mathbb{R}^d$, $1 \leq k \leq d-1$. This bound is tight for $k < d/2$. For $d/2 \leq k < d-1$, only one tight bound is known currently: there is an $O(n^{5/3})$ size an $O(\log n)$ height BSP for n disjoint axis-aligned (2-dimensional) rectangles in four-space [15]. Results of Dumitrescu, Mitchell, and Sharir, combined with ideas of Paterson and Yao, imply an $O(n^{d/2})$ upper bound on the minimal size of a BSP for disjoint axis-aligned boxes in d-space [17].

Disjoint fat objects also have small size BSPs in some cases. We say that a k-dimensional object r is *fat* if the ratio of side length of the smallest enclosing cube and the largest inscribed cube in the affine k-dimensional space spanned by r is at most a constant.[1] If this constant is $\alpha \geq 1$, then r is said to be *α-fat*. Specifically, every line segment is 1-fat, and the aspect ratio of every α-fat rectangle is at most α. De Berg [10] constructed an $O(n)$ size BSP for n *full-dimensional* fat objects of constant combinatorial complexity (e.g., fat boxes) in $\mathbb{R}^d$ for any dimension $d \in \mathbb{N}$. This technique can be generalized, and it leads to linear size BSPs for a wider class of input sets, such as *uncluttered scenes* [10] and *guardable scenes* [11]. None of these results gives a bound for $(d-1)$-dimensional or lower-dimensional fat polygonal objects in $\mathbb{R}^d$, $d \geq 3$.

Agarwal et al. [1] studied BSPs for n disjoint *axis-aligned* fat rectangles in three-space. They have constructed a BSP of $n2^{O(\sqrt{\log n})}$ size and $O(\log n)$ height in $n2^{O(\sqrt{\log n})}$ time. Our main result improves upon this bound.

THEOREM 1. *For every set of n disjoint axis-aligned fat rectangles in $\mathbb{R}^3$, there is a BSP of $O(n \log^8 n)$ size and $O(\log^5 n)$ height. Such a BSP can be computed in $O(n \log^{12} n)$ time and $O(n \log^8 n)$ space.*

If the input rectangles are α-fat, $1 \leq \alpha \leq n$, then our proof gives a BSP of $O(n\alpha^4 \log^8 n)$ size and $O(\log^5 n)$ height. Our proof is constructive; we present an explicit algorithm to compute a BSP for fat rectangles. In our algorithm, every partition hyperplane is axis-aligned (i.e., parallel to $(d-1)$ coordinate axes). Such a BSP is called an *axis-aligned BSP*. We complement our upper bound with a lower bound construction and present a set of n disjoint axis-aligned fat rectangles in $\mathbb{R}^3$ that requires a $\Omega(n \log n)$ size axis-aligned BSP.

THEOREM 2. *For every $n \in \mathbb{N}$, there are n pairwise disjoint axis-aligned squares in $\mathbb{R}^3$ such that any axis-aligned BSP for them has $\Omega(n \log n)$ size.*

[1] Fat objects are usually defined in terms of enclosing and inscribed *disks*. Our definition, in terms of *cubes*, is more convenient for our analysis of axis-aligned objects, and it is equivalent to the usual definition (with a possibly different constant).

That is, the minimal size of an axis-aligned BSP for n disjoint axis-aligned fat rectangles in $\mathbb{R}^3$ is always $O(n \log^8 n)$ and it is sometimes $\Omega(n \log n)$. No attempts were made to optimize the degree of the polylogarithmic factors; it is left open for future research.

The and remainder of the paper is organized as follows. In section 2, we define several BSP schemes. We distinguish between different types of axis-aligned rectangles with respect to an axis-aligned box in section 3. We present a hierarchy of recursive partition schemes for special classes of axis-aligned fat rectangles in section 4. Based on these building blocks, we prove Theorem 1 in section 5. We present our lower bound construction and prove Theorem 2 in section 6. Finally, we conclude with some conjectures in section 7.

2. Preliminaries: BSP schemes and BSPs. We start with defining a slightly more general partitioning procedure than the classical BSP. A *binary space partition scheme* (*BSP scheme*) is a recursive partitioning algorithm for a pair (R, C) of an input set R of objects lying in an open bounding volume C. The volume C is called the *cell* associated with the input. (If C is not given with the input set R, then C may be considered as the entire space, a convex hull, or a bounding box of the input.) Unless the input satisfies an *end condition*, the cell C is split along a hyperplane h into two open subcells C_1 and C_2, and then a BSP scheme is applied recursively for (R_1, C_2) and (R_2, C_2), where $R_i = \{r \cap C_i : r \in R\}$, $i = 1, 2$, is the set of object fragments clipped to the subcell C_i. (Notice that the fragments lying *in* the hyperplane h are not fragmented any further.)

A classical BSP is, in our terminology, a BSP scheme with a specific end condition. A cell C is partitioned until the input consists of at most one full-dimensional and no lower-dimensional input object. Specifically, for rectangles in three-space, the end condition requires that the input is *empty*.

Every BSP scheme naturally corresponds to a binary tree. Every node v of the tree corresponds to a subproblem (R_v, C_v): the root corresponds to the initial input $R = R_0$ and its bounding volume C_0. The two children of a nonleaf node v correspond to the two subproblems in the two open subcells of the spitting hyperplane h_v. The BSP scheme can be encoded (or stored) in a data structure, called a *BSP tree*, based on this binary tree: every nonleaf node stores a splitting hyperplane and all fragments of k-dimensional input objects, $k < d$, lying on the splitting plane. Every leaf node stores at most one fragment of a full-dimensional input object, which is the input of the corresponding subproblem (for an input of rectangles in three-space, the leaf nodes do not store any fragment).

The size of a BSP scheme is the total number of fragments of input objects stored in the corresponding BSP tree. If the BSP scheme does not make *redundant cuts*, i.e., if every splitting plane partitions the convex hull of the input objects, then the size of the BSP scheme is an upper bound on the number of nonleaf nodes of the BSP tree. Our explicit algorithms can be implemented with no redundant cuts (cf., Lemma 4), and so our bounds on the size of various BSP schemes lead to the same asymptotic bounds on the size of a corresponding BSP trees.

Every BSP scheme defines a *subdivision* of the input volume C. The subdivision of a BSP scheme is the collection of nonoverlapping cells corresponding to the leaf nodes of the BSP tree. The cells in the subdivision jointly tile the volume C.

Colorful BSPs. We have defined the classical BSP as a BSP scheme with a specific end condition. The (classical) BSP terminates when every subproblem consists of at most one full-dimensional object. In our algorithms, we use another

BSP scheme: a *colorful binary space partition* is defined as a BSP scheme where some input object is colored, each object with at most one color, and the end condition is satisfied if and only if in each subproblem no two objects have different colors. A (classical) BSP for the complete input is, of course, a colorful BSP. A colorful BSP, however, may have a smaller size than a BSP (i.e., it may dissect the input into much fewer fragments).

Overlays of BSPs. We define a powerful tool to concatenate several BSP schemes while keeping the total size small. Consider two BSP schemes P_1 and P_2 (their inputs (R_1, C_1) and (R_2, C_2) may be different, and their end conditions may also be different). For $i = 1, 2$, let $T(P_i)$ denote the BSP tree of P_i, and let $\mathcal{L}_i$ denote the subdivision of P_i. Assume C_2 can be tiled with cells of $\mathcal{L}_1$ (i.e., the closure of C_2 is the union of the closures of some cells of $\mathcal{L}_1$).

The *overlay* BSP scheme $P_1 \circ P_2$ is defined for the input $(R_1 \cup R_2, C_1)$ as follows. First $P_1 \circ P_2$ performs P_1; then in each cell $L \in \mathcal{L}_1$, independently, it performs P_2 restricted to L. Technically, in each cell $L \in \mathcal{L}_1$, we run P_2 on the input set $(\{r \cap L : r \in R_2\}, L)$ and partition L with the splitting planes of P_2 recursively. See Figure 1 for an example in the plane.

For a set A of objects, we also define the *restricted overlay* BSP scheme $P_1 \circ_A P_2$, which first performs P_1; then in each cell $L \in \mathcal{L}_1$ that contains a portion of some object in A, it performs P_2 restricted to L (while cells $L \in \mathcal{L}_1$ that are disjoint from A are not partitioned any further).

We obtain the BSP tree $T(P_1 \circ P_2)$ by appending to every leaf node $v \in T(P_1)$ the BSP tree obtained when we perform the binary partition P_2 restricted to the tile $C_v \in \mathcal{L}_1$ until the end condition of P_2 is satisfied in each subproblem. Notice that the subdivision corresponding to the overlay BSP scheme $P_1 \circ P_2$ satisfies the end conditions of both P_1 and P_2. More importantly, if P_1 is a (classical) BSP *for a set R_1 of $(d-1)$-dimensional objects*, then P_2 does not partition any object in R_1 because they all lie on the boundaries of the tiling $\mathcal{L}_1$. Intuitively, P_1 *eliminates* the objects in R_1 from further fragmentation. Similarly, if P_1 is a colorful BSP for a set A of red objects and a set B of blue objects, and P_2 is a (classical) BSP for A, then the splitting planes of P_2 cannot partition any blue objects in $P_1 \circ_A P_2$. We will use these properties in our algorithms.

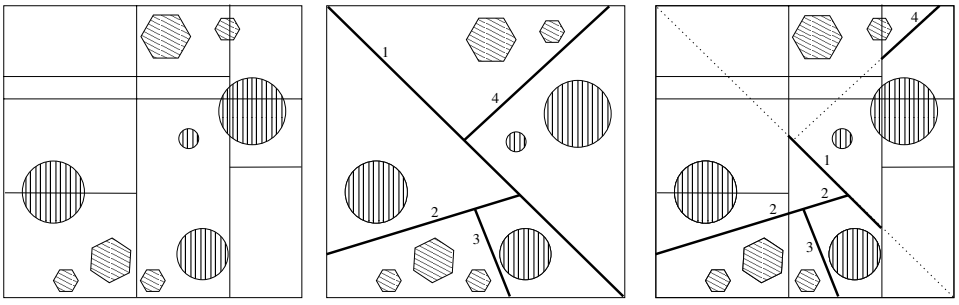


FIG. 1. *An example with an axis-aligned BSP scheme P_1 (whose input objects are not shown in the figure) and a colorful BSP P_2 for a set of disks and hexagons (shown in the figure). (Left) The axis-aligned subdivision $\mathcal{L}_1$ and the input objects of R_2 in their common bounding box $C_1 = C_2$. (Middle) The subdivision $\mathcal{L}_2$ produced by P_2. (Right) The subdivision produced by the overlay BSP scheme $P_1 \circ P_2$. Redundant cuts made by the overlay BSP scheme are drawn with dotted lines.*

The overlay $\bigcirc_{i=1}^{k} P_i = P_1 \circ P_2 \circ \cdots \circ P_k = (\ldots((P_1 \circ P_2) \circ P_3) \circ \cdots \circ P_k)$ of k BSP schemes is obtained by successively overlaying the partition schemes $P_1, P_2, \ldots, P_k$. Let us point out an immediate consequence of our definition.

PROPOSITION 3. *Consider k BSP schemes $P_1, P_2, \ldots, P_k$. The height of the overlay $\bigcirc_{i=1}^{k} P_i$ is no more than the sum of the heights of $P_1, P_2, \ldots, P_k$.*

Redundant cuts of overlays. Even if neither of the BSP schemes P_1 and P_2 makes redundant cuts, the overlay $P_1 \circ P_2$ may perform redundant cuts (e.g., along the dotted lines in Figure 1, right).

Recall that the number of nodes in the BSP tree is proportional to the number of fragments of the input objects only if the BSP makes no redundant cuts. Since we want to use the fragmentation of the objects to measure the size of our data structure, we need to filter out all redundant cuts. This can be easily done by simply skipping all redundant cuts.

LEMMA 4. *Given a BSP scheme P for an input (R, C) that partitions R into a set F of fragments and has height H (but possibly makes redundant cuts), we can construct a BSP scheme P' for (R, C) that has the same end condition as P, partitions R into the same set F of fragments, has height at most H, and makes no redundant cuts.*

Proof. We construct P' by performing the cuts made by P but skipping redundant cuts. We use the following auxiliary data structure: for every intermediate cell C of P, we create an open convex volume $\hat{C} \subseteq C$ that contains all portions of the input objects clipped to C (i.e., $\{r \cap C : r \in R\} \subset \hat{C}$). We construct the cells $\hat{C}$ and the BSP scheme P' as the partitioning algorithm P progresses. If P performs a redundant cut that splits a cell C into subcells C_1 and C_2 such that $\{r \cap C : r \in R\} \subset C_1$, then P' does nothing, but we put $\hat{C}_1 := \hat{C} \cap C_1$ and $\hat{C}_2 = \emptyset$ (in fact, we can remove C_2 from further consideration). If P performs a nonredundant cut that splits a cell C into subcells C_1 and C_2 along a hyperplane h, then we look up the (unique) cell C' at the current level of partition of P' that contains $\hat{C}$ and make P' partition C' along h into subcells C'_1 and C'_2, and we put $\hat{C}_1 := \hat{C} \cap C_1$ and $\hat{C}_2 := \hat{C} \cap C_2$.

Note that by skipping redundant cuts, the cell C' of P' may be much larger than the corresponding cell C of P. Hence, the cut along h can extend beyond the cell C (in Figure 1, e.g., the cut along 3 would extend above 2). This, however, has no effect on the number of fragments of the input objects in R or whether the end condition is reached: P and P' differ only in the way they partition $C' \setminus \hat{C}$ and $C \setminus \hat{C}$ which are empty of input objects. □

In our algorithms, we sometimes overlay several BSP schemes whose input objects are the same (namely, the set of axis-aligned fat rectangles) but whose input cells are different. In this case, a *redundant cut* has the same meaning for each BSP scheme: it is a splitting hyperplane for a cell C that is disjoint from the convex hull of $\{r \cap C : r \in R\}$. In the following sections, we present explicit algorithms that may perform redundant cuts (due to overlaying several BSP schemes). These BSP schemes together with the filtering done by Lemma 4 lead to BSP trees of size proportional to the number of fragments into which the input objects are partitioned.

3. Basic properties of axis-aligned objects. An *axis-aligned box* in $\mathbb{R}^3$ is the cross product of three nonempty intervals, which we call the *extents* of the box. The lengths of the x-, y-, and z-extent of a box B are denoted by $x(B)$, $y(B)$, and $z(B)$, respectively. A box is *full-dimensional* if all three extents have nonzero length. An *axis-aligned rectangle* is a box where exactly two extents have positive lengths and one extent has zero length (i.e., it is a point). Finally, an axis-aligned rectangle is

α-*fat* (or *fat* if α is clear from the context) if its aspect ratio (the ratio of its longer and shorter positive extents) is at most a constant $\alpha \geq 1$.

If all input objects of a BSP scheme are axis-aligned, then we assume that the initial cell C_0 is the axis-aligned bounding box of the input. If C_0 is axis-aligned and every splitting plane is orthogonal to a coordinate axis, then the cell of every subproblem is an open axis-aligned box.

Shelves and bridges. The input of a subproblem (R_v, C_v) is the set of rectangles clipped to the cell C_v. Even if a rectangle r of the initial input is fat, the fragment $r \cap C_v$ clipped to C_v is not necessarily fat. In the following definition, we describe the possible positions of a fat rectangle relative to an axis-aligned cell C.

DEFINITION 5. *Consider an axis-aligned full-dimensional open box C and an axis-aligned closed rectangle r such that $r \cap C$ is nonempty. We say that r is* pass-through *for C, if an extent of r contains the corresponding extent of C. We distinguish three classes of pass-through rectangles for C (see Figure 2):*

- *r is* free *for C if both positive extents of r contain the corresponding extents of C;*
- *r is a* shelf *for C if one extent of r contains the corresponding extent of C and the other positive extent of r contains an end point of the corresponding extent of C;*
- *r is a* bridge *for C if one extent of r contains the corresponding extent of C and the other positive extent of r lies in the interior of the corresponding extent of C.*

Each type of pass-through rectangle has a specific role in our algorithms. Whenever an input rectangle r is free for an intermediate cell C of the partitioning, we split C along $r \cap C$. This partition step is called a *free cut*. It is always a nonredundant cut and it does not increase the number of fragments of the input objects.

The fragment $r \cap C$ of a fat *shelf* r can have an arbitrary aspect ratio, but we can construct an $O(n \log n)$ size BSP for any n shelves in a cell C (cf., Lemma 10). The fragment $r \cap C$ of a *fat bridge* r is not necessarily fat, either, but it preserves some properties of the fatness of r: if r is α-fat, then the extent of $r \cap C$ which is the same as the corresponding extent of C is at most α times as long as the other positive extent (which lies in the interior of that of C).

Agarwal et al. [1] proved that for n disjoint axis-aligned fat rectangles, all of which are pass-through for their bounding box C_0, there is a BSP of $O(n)$ size and $O(n)$ height. (They also give an $O(n \log n)$ size and $O(\log n)$ height BSP for such an input.) These BSPs, however, do not provide a good partitioning for a set of fat axis-aligned rectangles, in general, because they may partition a rectangle lying in the interior of C into $O(n)$ fragments, resulting in a BSP of size $O(n^2)$. We can improve

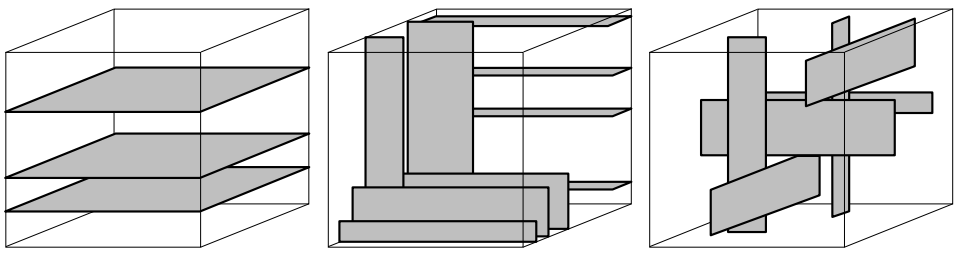


FIG. 2. *Free rectangles, shelves, and bridges clipped to an axis-aligned box.*

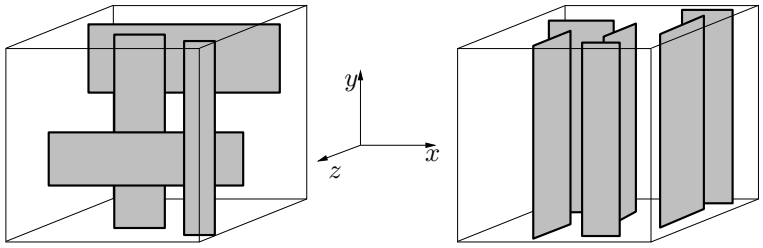


FIG. 3. *Pass-through xy-rectangles (left) and y-class pass-through rectangles (right).*

upon the upper bound $n2^{O(\sqrt{n})}$ of Agarwal et al. because we distinguish shelves and bridges. This allows us to better exploit the particular geometric properties of these objects.

Orientation of edges and rectangles. A line segment (or an edge of a rectangle) is called an x-, y-, or z-segment (x-, y-, or z-edge) if it is parallel to the x-, y-, or z-coordinate axis, respectively. The *orientation* of an axis-aligned rectangle is the pair of orientations of its edges; thus we can talk about xy-, yz-, and xz-rectangles (Figure 3). The *base* of a shelf r for a cell C is a side s of C such that the segment $r \cap s$ is the common extent of C and $r \cap C$. Every shelf for C has a unique base. All shelves based at the same side s of a cell C must have the same orientation: the orientation of every shelf based at s is different from that of s, and if two shelves r_1 and r_2 based at s had the remaining two distinct orientations, then the segments $s \cap r_1$ and $s \cap r_2$ would intersect.

For pass-through (but not free) rectangles with respect to a cell C, we distinguish three *classes* depending on their extent dimension covering that of C: a pass-through rectangle r is in the x-class for C (resp., y-class and z-class) if $x(r \cap C) = x(C)$ (resp., $y(r \cap C) = y(C)$ and $z(r \cap C) = z(C)$), i.e., the x-extent of r contains the corresponding extent of C. (Agarwal et al. [1] also distinguished these classes and called them *front*, *right*, and *top*.)

Computation of BSP size. We do not count directly the number of fragments of input rectangles in the analyses of our algorithms. Instead, we deduce a bound on the BSP size from the number of fragments of segments clipped to the input rectangles.

DEFINITION 6. *Consider a set R of disjoint axis-aligned rectangles in $\mathbb{R}^3$. For every rectangle $r \in R$ and every axis-parallel line ℓ, the line segment $\ell \cap r$ is called a* mast *of R.*

PROPOSITION 7. *Assume that we are given a set R of axis-aligned rectangles in $\mathbb{R}^3$ and an axis-aligned BSP scheme P that applies every possible free cut. If P partitions any* mast *of R into at most k pieces, then it partitions every rectangle of R into $O(k^2)$ fragments.*

Proof. The BSP scheme P restricted to an axis-aligned rectangle $r \in R$ is a 2-dimensional BSP-scheme. During the partition algorithm, only the fragments of r adjacent to the boundary ∂r may be further partitioned: if a fragment of r is not adjacent to any edge of r (see Figure 4(a)), then it is free for the cell in which it is contained, and we assume that P makes a (free) cut along every free fragment of the input.

Consider the steps of the BSP scheme that partition a fragment r' of r. Each of these steps increases the number of fragments of r by one. We distinguish two types of splitting planes: (i) The splitting plane $h \cap r'$ subdivides the common boundary

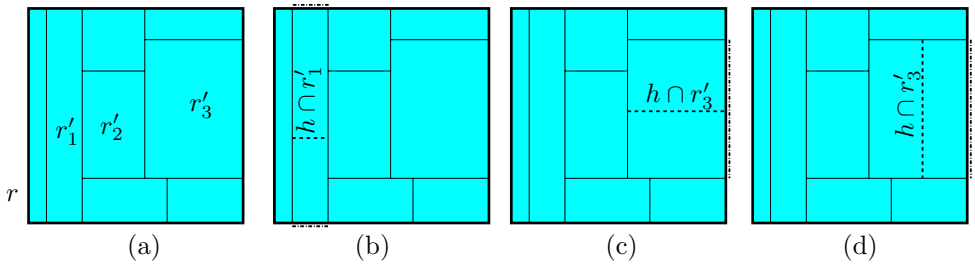


FIG. 4. *The subdivision of a rectangle r after a few steps of a partition algorithm. Fragment r'_2 of r is free* (a). *The splitting plane $h \cap r'$ subdivides the set $\partial r \cap r'$* (b) *and* (c). *The splitting plane $h \cap r'$ does not subdivide $\partial r \cap r'$* (d).

$\partial r \cap r'$ (i.e., $\partial r \cap r'$ has points on both sides of h; see Figures 4(b),(c)). Since every edge of r is subdivided into at most k fragments, at most $4k-4$ steps can partition ∂r this way. (ii) The splitting plane $h \cap r'$ partitions the fragment $r' \subset r$ but does not subdivide the set $\partial r \cap r'$ (see Figure 4(d)). In this case, $h \cap r'$ is parallel to a portion of $\partial R \cap r'$, which is part of an edge e of r. Let us project $h \cap r'$ orthogonally to such an edge e. The projections of all splitting segments of type (ii) give a multiple coverage of e. Consider a final fragment e' of an edge e of r. Note that the projection of every splitting segment either contains e' or is disjoint from e'. If e' is ℓ-fold covered, then any mast of r orthogonal to e' and having an end point on e' is partitioned into at least $\ell + 1$ pieces. Each final fragment of the four edges along ∂r is covered at most $k-1$ times, and so the number of splitting segments of type (ii) is no more than $(4k-4)(k-1)$. □

The upper bound $O(k^2)$ of Proposition 7 is optimal apart from a constant factor. An xy-rectangle might be split along x-lines into k slabs, and then each slab can be split along y-lines into k pieces, which gives a total of k^2 fragments.

4. Building blocks. We present our BSP scheme for fat axis-aligned rectangles in $\mathbb{R}^3$ in the next section. It is composed of several layers of recursive BSP schemes. In this section, we build up these layers gradually, starting with simple BSP schemes and culminating with a BSP for pass-through rectangles for a cell C. We discuss each level in a separate subsection below. In subsection 4.1, we describe how the free cuts are performed. In subsection 4.2, we present a BSP for shelves such that every mast of an input rectangle is partitioned into $O(\log n)$ pieces. In subsections 4.3 and 4.4, we present a colorful BSP separating pass-through rectangles of different classes. Finally, in subsection 4.5, we describe a BSP for z-class pass-through rectangles. All our BSP schemes and their analyses are similar; they apply recursion and partition a cell along some kind of median plane in each recursive step.

4.1. Free cuts. We construct our BSP schemes recursively, and we often have to partition every cell along all free cuts. A free cut does not increase the total number of fragments of input objects, but it increases the height of the corresponding BSP tree. Since several input rectangles may be free for the same cell, it *does* matter how we perform the free cuts.

Consider a set of n disjoint axis-aligned rectangles, and let P be an axis-aligned BSP scheme. For each cell C of the subdivision defined by P, we partition C recursively along all free rectangles of C. Since the input rectangles are disjoint, all free rectangles for C are parallel. In one recursive step, we split C along the *median* free rectangle. Therefore in $O(\log n)$ recursive calls, we eliminate all free cuts.

Let $P \triangleleft \texttt{FreeCut}(R)$ denote the BSP scheme obtained by performing the recursive free cuts described above in each cell of the subdivision of P. The BSP tree $T(P \triangleleft \texttt{FreeCut}(R))$ can be obtained from $T(P)$ by appending a binary tree of height $O(\log f(C))$ to each leaf node of $T(P)$ that corresponds to a cell with $f(C)$ free rectangles.

PROPOSITION 8. *The operation $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$ does not increase the size of P, but it increases its height by $O(\log n)$.*

4.2. BSP for shelves. Given a set R of disjoint axis-aligned rectangles and an axis-aligned cell C, we present the BSP scheme $\texttt{Shelf}(R, C)$, which is a classical BSP for the set $S = \{r \cap C : r \in R \text{ is a shelf for } C\}$. That is, $\texttt{Shelf}(R, C)$ partitions the cell C recursively until every (open) subcell is disjoint from elements of S. A rectangle $r \in R$ may be a shelf for a resulting subcell $C' \subset C$, but $\texttt{Shelf}(R, C)$ eliminates all fragments of $r \cap C$ if r is a shelf *for the input cell* C.

We construct $\texttt{Shelf}(R, C)$ as an overlay of six BSPs for shelves with a common base side. Let s be a side of cell C. We may assume without loss of generality (w.l.o.g.) that s is the lower xz-side of C and the orientation of every shelf with base s is yz (see Figure 5, left). Notice that the x-coordinates of shelves with base s are distinct.

ALGORITHM 1 ($\texttt{OneShelf}_s(R, C)$).

Input: R is a set of disjoint axis-aligned fat rectangles, C is an axis-aligned cell, and s is a side of C. $S_s(C) = \{r \cap C : r \in R \text{ is a shelf for } C \text{ with base side } s\} \neq \emptyset$.

1. Permute the coordinate axes (temporarily, for the sake of this algorithm only) such that s is the lower xz-side of C and every shelf based at s is a yz-rectangle.
2. Let P be a BSP scheme that partitions C along an xz-plane h through an edge of a shelf of $S_s(C)$ with a maximum y coordinate, and then P partitions the subcell between h and s by a yz-plane along the median shelf into two cells C_1 and C_2. (See an example in Figure 5, middle.)
3. $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$.
4. For $i = 1, 2$ do: If a rectangle of $S_s(C)$ intersects C_i, then $P \leftarrow P \circ \texttt{OneShelf}_{s \cap C_i}(R, C_i)$.
5. Return P.

Rounds. We analyze Algorithm 1 (and other recursive BSP schemes) in increments of work done in one level of the recursion, which we call a *round*. In the first round, the algorithm is called for the initial input cell C; the second round calls the algorithm for the cells C_1 and C_2 independently. In round i, in general, Algorithm 1 is called for up to 2^i disjoint cells, in parallel. The work done in round i includes a

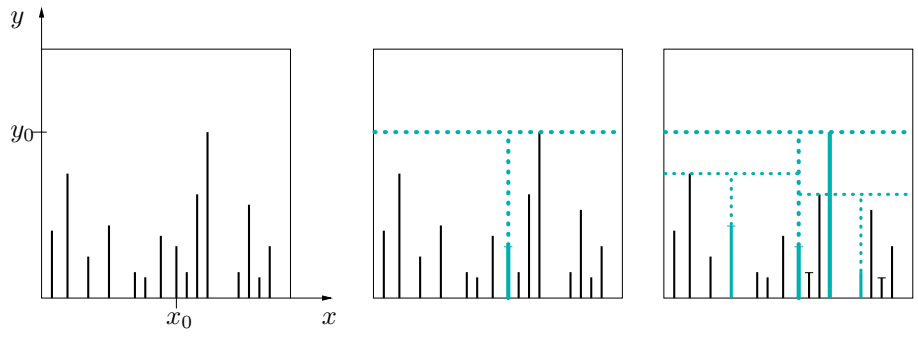


FIG. 5. *The partition done in two consecutive rounds of Algorithm* 2.

median cut and the free cuts performed in each of these cells. (Note that free cuts may partition C_1 and C_2 into several smaller subcells, yet the recursive calls are applied for C_1 and C_2.)

LEMMA 9. *If R is a set of n disjoint axis-aligned rectangles and $S_s = \{r \cap C : r \in R$ is a shelf for C with base side $s\}$ is nonempty, then $\texttt{OneShelf}_s(R, C)$ is a BSP for S_s. It partitions every mast of R into $O(\log(|S_s| + 1))$ fragments, and its height is $O(\log(|S_s| + 1) \cdot \log n) = O(\log^2 n)$.*

Proof. We analyze Algorithm 1 in rounds (increments of work done in one level of the recursion). In each round, the number of elements of $S_s(C_i)$ in a cell C_i (for which the $\texttt{OneShelf}_{s\cap C_i}(R, C_i)$ is applied) decreases by a factor of at least two. After $O(\log(|S_s(C)| + 1))$ rounds, no cell contains any fragment of $S_s(C)$ in its interior. It follows that $\texttt{OneShelf}_s(R, C)$ is a BSP for $S_s(C)$ and it terminates in $O(\log(|S_s(C)| + 1))$ rounds. By Proposition 8, its height increases by $O(\log n)$ in every round, and so its total height is $O(\log n \cdot \log(|S_s(C)| + 1)) = O(\log^2 n)$.

Consider a mast along a rectangle $r \in R$. We assume w.l.o.g. that s is the lower xz-side of C and every shelf based at s is a yz-rectangle. Since we split the cells by xz- and yz-planes in step 2, no z-segment is ever cut. Every y-mast is dissected into $O(\log(|S_s| + 1))$ fragments because it can be cut at most once in every round. Now let e be an x-mast. Consider a fragment $e \cap C_i$ in a cell C_i for which $\texttt{OneShelf}_{s\cap C_i}(R, C_i)$ is called in round j. Observe that $e \cap C_i$ can only be cut if C_i contains an end point of e in its interior: otherwise e is pass-through for C_i, and since e is disjoint from all the shelves of $S_s(C_i)$, it must lie above the highest shelf of $S_s(C_i)$. This implies that e is cut at most twice in every round, and so it is partitioned into $O(\log(|S_s(C)| + 1)) = O(\log n)$ fragments during the entire algorithm. □

By overlaying the BSP scheme $\texttt{OneShelf}_s(R, C)$ for the six sides of a cell C, we obtain a BSP for all portions of shelves for C that lie in cell C.

ALGORITHM 2 ($\texttt{Shelf}(R, C)$).

Input: R is a set of disjoint axis-aligned fat rectangles, and C is an axis-aligned cell.

1. Let J be the set of (at most six) sides of C such that $S_s(C) \neq \emptyset$ for $s \in J$.
2. Return $\bigcirc_{s\in J} \texttt{OneShelf}_s(R, C)$.

By combining Lemma 9 with Proposition 3, we obtain the following result.

LEMMA 10. *If R is a set of n disjoint axis-aligned rectangles, then $\texttt{Shelf}(R, C)$ is a BSP for the set $S = \{r \cap C : r \in R$ is a shelf for $C\}$. It partitions every mast of R into $O(\log n)$ fragments, and its height is $O(\log^2 n)$.*

4.3. A colorful BSP to separate xy-oriented y-class rectangles from the z-class. Next, we present a colorful BSP that separates the y- and z-class pass-through rectangles in a cell C under certain conditions. Consider a set R of n disjoint α-fat axis-aligned rectangles in $\mathbb{R}^3$. Let C be an axis-aligned cell such that the lengths of its three extents are ordered as $x(C) \geq y(C) \geq z(C)$.

We distinguish a *red set* $A = \{r \cap C : r \in R$ is a z-class bridge for $C\}$ and a *blue set* $B = \{r \cap C : r \in R$ is a y-class bridge for C of orientation $xy\}$ (see Figure 6). Project every rectangle $r \in A \cup B$ to the x-axis. Since the projections of red and blue pass-through rectangles do not overlap, we can subdivide the x-extent of C into intervals $x_0x_1, x_1x_2, \ldots, x_kx_{k+1}$ for some $k \in \mathbb{N}$, $0 \leq k \leq n$, such that each interval contains projections of rectangles from one color class only and any two consecutive intervals contain projections of different color classes. We define $X(R, C) = (x_1, x_2, \ldots, x_k)$ to be a sequence of such interval end points.

If we partition C by yz-planes through $x_1, x_2, \ldots, x_k$, then we separate the red and blue rectangles within C. These k yz-planes, however, might partition other

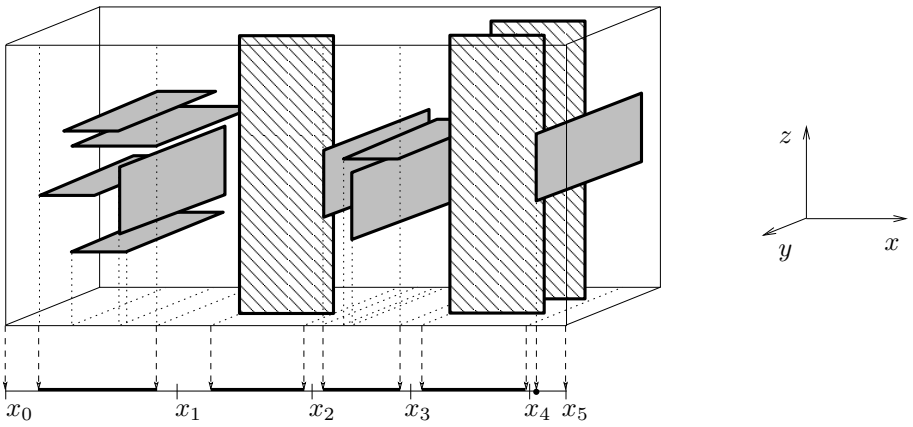


FIG. 6. *Bridges from two classes and a subdivision of the extent $x(C)$ into five intervals.*

rectangles of R into too many fragments. We can show, however, that all α-fat rectangles that intersect more than $2\alpha^2$ such yz-planes must intersect the boundary of the cell C.

PROPOSITION 11. *Consider a cell C such that $x(C) \geq y(C) \geq z(C)$. An α-fat rectangle whose y- and z-extents lie in the interior of the corresponding extents of C intersects at most $2\alpha^2$ yz-planes through points of $X(R, C)$.*

Proof. We have assumed that every *blue* bridge is an α-fat xy-rectangle. The length of the interval $x_i x_{i+1}$ corresponding to projections of blue bridges is at least $y(C)/\alpha$. The x-extent of an α-fat rectangle r whose y- and z-extents lie in the interior of the corresponding extents of C is less than $\alpha \cdot y(C)$ independently of the orientation of r because $y(C) \geq z(C)$. Therefore, r intersects fewer than α^2 intervals corresponding to blue bridges. Since the intervals correspond to red and blue bridges alternately, we conclude that r intersects less than $2\alpha^2$ yz-planes through points of $X(R, C)$. □

Based on the above observations, we may now present a colorful BSP for A and B.

ALGORITHM 3 ($\texttt{Sep2}(R, C)$).

Input: R is a set of disjoint axis-aligned rectangles, and C is an axis-aligned cell such that $x(C) \geq y(C) \geq z(C)$ and $X(R, C)$ is nonempty.

Let P be the trivial BSP scheme for (R, C) (its BSP tree $T(P)$ consists of the root only).

1. If R contains shelves for C, then $P \leftarrow P \circ \texttt{Shelf}(R, C)$.
2. $P \leftarrow P \circ M_{yz}(R, C)$, where $M_{yz}(R, C)$ is a BSP scheme that partitions C into two cells C_1 and C_2 along the yz-plane through the median element of the sequence $X(R, C)$.
3. $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$.
4. For $i = 1, 2$ do: If the sequence $X(R, C_i)$ is nonempty, then $P \leftarrow P \circ \texttt{Sep2}(R, C_i)$.
5. **Return** P.

LEMMA 12. *If R is a set of n disjoint α-fat axis-aligned rectangles and $x(C) \geq y(C) \geq z(C)$, then $\texttt{Sep2}(R, C)$ is a colorful BSP for the red set $A = \{r \cap C : r \in R$ is a z-class bridge for $C\}$ and the blue set $B = \{r \cap C : r \in R$ is a y-class bridge for C of orientation $xy\}$. It partitions every x-mast of R into $O(\alpha^2 \log^2 n)$ fragments, every y- or z-mast of R into $O(\log^2 n)$ fragments, and its height is $O(\log^3 n)$.*

Proof. We analyze Algorithm 3 in rounds (increments of work done in one level

of the recursion). Since the y- and z-extents of C_1 and C_2 are the same as that of C, and $M_{yz}(R, C)$ does not partition any y- or z-bridges for C, we have $X(R, C_1) \cup X(R, C_2) \subset X(R, C)$. Since $M_{yz}(R, C)$ cuts along a median element of $X(R, C)$, we have $|X(R, C_1)| \leq |X(R, C)|/2$ and $|X(R, C_2)| \leq |X(R, C)|/2$. After $\log(|X(R, C)| + 1) \leq O(\log n)$ rounds, Algorithm 3 partitions C along yz-planes through all points of $X(R, C)$. It follows that $\texttt{Sep2}(R, C)$ is a colorful BSP for A and B and it terminates in $O(\log n)$ rounds. Every round may call $\texttt{Shelf}(R, C)$, which produces a BSP-scheme of height $O(\log^2 n)$, so the total height of P amounts to $O(\log^3 n)$.

A y- or z-mast e is never cut by a splitting yz-plane of $M_{yz}(R, C)$, and so it is in the interior of at most one cell C for which $\texttt{Sep2}(R, C)$ is called. In each round, $\texttt{Shelf}(R, C)$ is called for a cell C containing e. By Lemma 10, e is partitioned into $O(\log n)$ fragments in each round. In $O(\log n)$ rounds, the overlay of $O(\log n)$ algorithms $\texttt{Shelf}(R, C_i)$ dissects e into $O(\log^2 n)$ fragments.

Now consider an x-mast f along an input rectangle $r \in R$. First, suppose that the y- and z-extents of r lie in the interior of the corresponding extents of C. By Proposition 11, yz-planes through points of $X(R, C)$ can dissect r into at most $2\alpha^2$ pieces. Each piece is dissected into $O(\log^2 n)$ fragments by the overlay of $O(\log n)$ calls to $\texttt{Shelf}(R, C_i)$, one in each round, which gives a total of $O(\alpha^2 \log^2 n)$ fragments. Next, suppose that the y- or z-extent of r contains an end point of the corresponding extent of C. A cut $M_{yz}(R, C)$ can partition a fragment $f \cap C$ only if C contains one end point of f. If f is pass-through for C, then r is a *shelf* for C and $r \cap C$ has been eliminated by a call to $\texttt{Shelf}(R, C)$ in step 1 of the same round. In each round, $M_{yz}(R, C)$ partitions at most two fragments of f (those incident to the end points of f). In the next round, the resulting four or fewer fragments of f may be partitioned by calls to $\texttt{Shelf}(R, C)$ in step 1. So the mast f is cut $O(\log n)$ times in each round. In a total of $O(\log n)$ rounds, f is partitioned into $O(\log^2 n)$ fragments. □

4.4. A colorful BSP to separate y- and z-class rectangles. In the previous subsection, we separated y- and z-class pass-through rectangles under certain conditions. In this section, we create an environment where those conditions are satisfied, and we separate *all* y- and z-class pass-through rectangles.

We define the BSP scheme $P(\alpha, C)$ that splits C into $(\lfloor \alpha \rfloor + 1)^2$ congruent cells by $\lfloor \alpha \rfloor$ equally spaced xy-planes and by $\lfloor \alpha \rfloor$ equally spaced xz-planes (i.e., $P(\alpha, C)$ splits the y- and z-extents of C while keeping the x-extent). This can be accomplished by a BSP scheme of height $2 \log \lfloor \alpha \rfloor = O(\log \alpha)$. We denote by $\Pi(\alpha, C)$ the resulting set of $(\lfloor \alpha \rfloor + 1)^2$ cells.

PROPOSITION 13. *Consider a cell C such that $x(C) \geq y(C) \geq z(C)$. Every α-fat pass-through rectangle* for *C satisfies one of the following four conditions for every $C_i \in \Pi(\alpha, C)$:*

(i) *it is free for C_i;*
(ii) *it is a shelf for C_i;*
(iii) *it is a z-class bridge for C_i;*
(iv) *it is a y-class bridge for C_i with orientation xy.*

Proof. Consider a subcell $C_i \in \Pi(\alpha, C)$. Since the x-edge of C_i is more than α times as long as its y- and z-edges, there is no bridge for C_i in the x-class. Similarly, the y-edge of the initial cell C is more than α times as long as the z-edge of C_i; therefore, a y-class bridge for C with orientation yz cannot be a bridge for C_i. □

We partition every cell $C_i \in \Pi(\alpha, C)$ so that the x-, y-, and z-class pass-through rectangles are separated. Free rectangles can be eliminated by free cuts. By Lemma 10, the BSP scheme $\texttt{Shelf}(R, C_i)$ can eliminate shelves. By Lemma 12, the *colorful BSP*

$\texttt{Sep2}(R, C_i)$ can separate pass-through rectangles of types (iii) and (iv). We combine these algorithms to separate x-, y-, and z-class bridges, independently of the length of the three extents of the cell C. Since we use the next algorithm as a subroutine of another BSP scheme, it rotates its input cell C in a position where $x(C) \geq y(C) \geq z(C)$.

ALGORITHM 4 ($\texttt{Sep3}(R, C)$).

Input: R is a set of disjoint axis-aligned fat rectangles, and C is an axis-aligned cell.

1. Permute the coordinate axes (temporarily for this algorithm) such that $x(C) \geq y(C) \geq z(C)$.
2. $P \leftarrow P(\alpha, C)$.
3. $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$.
4. For every $C_i \in \Pi(\alpha, C)$ do: If there are shelves for C_i, then $P \leftarrow P \circ \texttt{Shelf}(R, C_i)$.
5. For $C_i \in \Pi(\alpha, C)$ do: If the sequence $X(R, C_i)$ is nonempty, then $P \leftarrow P \circ \texttt{Sep2}(R, C_i)$.
6. **Return** P.

LEMMA 14. *Assume that R is a set of n disjoint α-fat axis-aligned rectangles, $1 \leq \alpha \leq n$, and C is an axis-aligned cell with $x(C) \geq y(C) \geq z(C)$. Then the BSP scheme $\texttt{Sep3}(R, C)$ is a colorful BSP for the red set $A = \{r \cap C : r \in R$ is a y-class bridge for $C\}$ and the blue set $B = \{r \cap C : r \in R$ is a z-class bridge for $C\}$; it is also a (classical) BSP for the set $\{r \cap C : r \in R$ is an x-class bridge for $C\}$. It partitions every mast of R into $O(\alpha^2 \log^2 n)$ fragments, and its height is $O(\log^3 n)$.*

Proof. By Proposition 13, a pass-through α-fat rectangle for C can have four different positions in a subcell $C_i \in \Pi(\alpha, C)$. Free rectangles are eliminated in step 3, and the shelves are eliminated in step 4. Finally, $\texttt{Sep2}(R, C_i)$ separates z-class bridges of C from y-class bridges of C with orientation xy in each subcell $C_i \in \Pi(\alpha, C)$.

The height of the BSP scheme $P(\alpha, C)$ is $O(\log \alpha)$. By Proposition 8, step 3 increases the height by $O(\log n)$. Finally, by Lemma 12 and Proposition 3, the height increases by $O(\log^3 n)$ in step 5.

Step 2 of Algorithm 4 cuts every y- and z-mast into $O(\alpha)$ fragments but does not partition any x-mast. By Lemma 12, $\texttt{Sep2}(R, C_i)$ partitions every x-mast into $O(\alpha^2 \log^2 n)$ fragments and any y- or z-mast into $O(\log^2 n)$ fragments in each $C_i \in \Pi(R, C)$. Altogether, Algorithm 4 partitions every mast (of any orientation) into $O(\alpha^2 \log^2 n) \subset O(\log^2 n)$ fragments. ∎

4.5. BSP for one class of pass-through rectangles. We are now ready to present a (classical) BSP for z-class pass-through rectangles for a cell C. Our algorithm is based on a simple intuition: we partition C recursively along xz-medians of the vertex set of the z-class pass-through rectangles. (The *xz-median* of a set of m points is an xz-plane h with a minimal y-coordinate such that the open half-space behind h contains fewer than $m/2$ points.) In order to avoid the possibility that pass-through rectangles of the y-class are fragmented into too many pieces, we separate z-class pass-through rectangles from any other classes by applying $\texttt{Sep3}(R, C_i)$ in each subcell in every round of the recursion.

ALGORITHM 5 ($\texttt{Long}(R, C)$).

Input: R is a set of disjoint axis-aligned fat rectangles, and C is an axis-aligned cell.

Let $L(R, C) = \{r \cap C : r \in R$ is pass-through for C in class $z\} \neq \emptyset$ and $V(R, C) = \{v \in \mathbb{R}^3 : v$ is a vertex of a $r \in L(R, C)$, v lies on the boundary of C but not on any xz-side of $C\}$. Let P be the trivial BSP scheme for the input (R, C).

1. If the sequence $X(R, C)$ is nonempty, then $P \leftarrow \texttt{Sep3}(R, C)$.

2. $P \leftarrow P \circ_{L(R,C)} N_{xz}(R, C)$, where $N_{xz}(R, C)$ is a BSP scheme that partitions C into two cells C_1 and C_2 along the median xz-plane of the point set $V(R, C)$.
3. $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$.
4. For $i = 1, 2$, do: If $V(R, C_i) \neq \emptyset$, then $P \leftarrow P \circ_{L(R,C)} \texttt{Long}(R, C_i)$.
5. **Return** P.

LEMMA 15. *If R is a set of n disjoint α-fat axis-aligned rectangles, $1 \leq \alpha \leq n$, then the BSP scheme* $\texttt{Long}(R, C)$ *is a BSP for the set $L(R, C) = \{r \cap C : r \in R$ is pass-through for C in class $z\}$. It partitions every mast of R into $O(\alpha^2 \log^3 n)$ fragments, and its height is $O(\log^4 n)$.*

Proof. We analyze Algorithm 5 in rounds (increments of work done in one level of the recursion). Since the z-extents of C_1 and C_2 are the same as that of C, we have $L(R, C_1) \cup L(R, C_2) \subset L(R, C)$. In every round, $N_{xy}(R, C)$ decreases the number of vertices in $V(R, C)$. If $V(R, C) = \emptyset$, then every element of $L(R, C)$ is free for C, which is eliminated in step 3. It follows that Algorithm 5 is a BSP for $L(R, C)$ and it terminates in $O(\log(|V(R, C)| + 1)) = O(\log n)$ rounds. Every round may call $\texttt{Sep3}(R, C)$, which has height $O(\log^3 n)$, so the total height of P is $O(\log^4 n)$.

It remains to show that Algorithm 5 dissects every mast e into $O(\alpha^2 \log^3 n)$ fragments. First, consider an x- or z-mast e. Note that $N_{xz}(R, C_i)$ in step 2 cannot cut e, and so in every round e lies in the interior of at most one cell C_i for which $\texttt{Long}(R, C_i)$ is applied. By Lemma 14, $\texttt{Sep3}(R, C_i)$ can dissect e into $O(\alpha^2 \log^2 n)$ fragments, and so e is partitioned into $O(\alpha^2 \log^3 n)$ fragments in total.

Now consider a y-mast f. A fragment $f \cap C_i$ can be cut if C_i contains an end point of f in its interior. Otherwise f lies along a rectangle r, where $r \cap C_i$ is y-class pass-through for the cell C_i. Therefore $\texttt{Sep3}(R, C_i)$ in step 1 separated $r \cap C_i$ from elements of $L(R, C)$ (i.e., partitioned C_i into subcells whose interiors do not contain fragments of both $r \cap C_i$ and $L(R, C)$). By the definition of restricted overlay BSPs, that means that $N_{xz}(R, C_i)$ does not partition $f \cap C_i$ (in this and all recursive calls). Hence, the cuts $N_{xz}(R, C_i)$ can successively dissect f into at most $O(\log n)$ pieces. In each round, at most four pieces are partitioned by $\texttt{Sep3}(R, C_i)$ in step 2. So the mast f is cut $O(\alpha^2 \log^2 n)$ times in each round. In a total of $O(\log n)$ rounds, f is partitioned into $O(\alpha^2 \log^3 n)$ fragments. ∎

5. Binary space partition algorithm for fat rectangles. In this section we present our main algorithm, a BSP for n disjoint axis-aligned fat rectangles in three-space, and its analysis. The algorithm is based on a simple heuristic: we split the space recursively along the median xy-plane of the clipped rectangles. After each cut, fragments of input rectangles may become pass-through with respect to the resulting cells. All new pass-through rectangles belong to the z-class. In order to prevent subsequent xy-median cuts from further partitioning a pass-through rectangle, we *eliminate* all z-class pass-through rectangles. By Lemma 15, we can remove the set $L(R, C)$ of z-class pass-through rectangles by overlaying the current BSP scheme with $\texttt{Long}(R, C)$.

ALGORITHM 6 ($\texttt{Main}(R, C)$).

Input: R is a set of disjoint axis-aligned fat rectangles, and C is an axis-aligned cell. $R(C) = \{r \cap C : r \in R, r \cap C \neq \emptyset\}$ is nonempty. Initially, let P denote the trivial BSP scheme for the input (R, C).

1. If $L(R, C) \neq \emptyset$, then $P \leftarrow P \circ \texttt{Long}(R, C)$.
2. $P \leftarrow P \circ Q_{xy}(R, C)$, where $Q_{xy}(R, C)$ is a BSP scheme that partitions C into two cells C_1 and C_2 along the median xy-plane of the point set $\{v \in \mathbb{R}^3 : v$ is

a vertex of a rectangle $r \in R(C)$, and v does not lie on any xy-side of $C\}$.

3. $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$.
4. For $i = 1, 2$, do: If $R(C_i) \neq \emptyset$, then $P \leftarrow P \circ \texttt{Main}(R, C_i)$.
5. `Return` P.

LEMMA 16. *If R is a set of n disjoint α-fat axis-aligned rectangles, $1 \leq \alpha \leq n$, then the BSP scheme* $\texttt{Main}(R, C)$ *is a BSP for $R(C) = \{r \cap C : r \in R, r \cap C \neq \emptyset\}$. It partitions every mast of R into $O(\alpha^2 \log^4 n)$ fragments, and its height is $O(\log^5 n)$.*

Proof. We analyze Algorithm 6 in rounds (increments of work done in one level of the recursion). In step 2, $Q_{xy}(R, C)$ decreases the number of rectangle vertices that do not lie on either xy-side of a cell by a factor of at least two. If a clipped rectangle $r \cap C$ intersects C but all its vertices lie on xy-sides of C, then r is pass-through for C in the z-class, and $\texttt{Long}(R, C)$ in step 1 eliminates it. It follows that Algorithm 6 is a BSP for $R(C)$ and it terminates in $O(\log n)$ rounds. Since $\texttt{Main}(R, C)$ calls $\texttt{Long}(R, C)$ in every round, its height is $O(\log n) \cdot O(\log^4 n) = O(\log^5 n)$.

It remains to show that Algorithm 6 partitions every mast into $O(\alpha^2 \log^4 n)$ fragments. Consider an x- or y-mast e. In step 2, $Q_{xy}(R, C)$ does not cut e. Therefore, in each round e is in the interior of at most one cell C_i for which $\texttt{Main}(R, C_i)$ is applied. By Lemma 15, each call to $\texttt{Long}(R, C_i)$ can partition e into $O(\alpha^2 \log^3 n)$ fragments. In a total of $O(\log n)$ rounds, e is partitioned into $O(\alpha^2 \log^4 n)$ fragments.

Now, consider a z-mast f along an input rectangle r. Both $\texttt{Long}(R, C_i)$ and $Q_{xy}(R, C_i)$ can dissect f, and so its fragments may lie in several cells for which $\texttt{Main}(R, C_i)$ is called in a round. A fragment $f \cap C_i$ may be dissected only if an end point of f is in the interior of C_i; otherwise, r is a z-class pass-through rectangle for C_i and was eliminated by $\texttt{Long}(R, C_i)$ in step 1. Thus in each round, step 2 may cut f twice, and $\texttt{Long}(R, C_i)$ may cut it $O(\alpha^2 \log^3 n)$ times. In a total of $O(\log n)$ rounds, f may be partitioned into $O(\alpha^2 \log^4 n)$ fragments. ∎

We are now ready to prove our main result, which we reiterate here.

THEOREM 1. *For every set of n disjoint axis-aligned fat rectangles in $\mathbb{R}^3$, there is a BSP of $O(n \log^8 n)$ size and $O(\log^5 n)$ height. Such a BSP can be computed in $O(n \log^{12} n)$ time and $O(n \log^8 n)$ space.*

Proof. Consider a set R of n disjoint α-fat axis-aligned rectangles, and let C be their axis-aligned bounding box. By Lemma 16, $\texttt{Main}(R, C)$ is a BSP for R. It partitions every mast into $O(\alpha^2 \log^4 n)$ fragments. By Proposition 7, it partitions every rectangle $r \in R$ into $O(\alpha^4 \log^8 n)$ fragments. In particular, for $\alpha = O(1)$, every rectangle $r \in R$ is partitioned into $O(\log^8 n)$ fragments, and the size of the BSP is $O(n \log^8 n)$.

Computational complexity. Our algorithm relies heavily on overlaying several simple BSP schemes. We present a simple data structure that leads to straightforward implementation of Algorithm 6 in $O(n \log^{12} n)$ time and $O(n \log^8 n)$ space. For an input of n disjoint axis-aligned rectangles, Algorithm 6 constructs a BSP tree P starting from a single root node by repeating the following operation: choose a leaf node v (corresponding to a cell C), store a splitting hyperplane h at v, and append two new leaf nodes to v (corresponding to the subcells of C on the two sides of h). In order to keep the size of the BSP tree proportional to the number of fragments of R, we also construct the variant of the main BSP algorithm that skips all redundant cuts, as described in Lemma 4.

Our data structure maintains several subdivisions of the bounding box C of the input rectangles. We maintain the subdivision of the current BSP scheme P. Besides the main BSP scheme P, we have several algorithms (namely, $\texttt{Main}(R, C)$, $\texttt{Long}(R, C)$, $\texttt{Sep3}(R, C)$, $\texttt{Sep2}(R, C)$, $\texttt{Shelf}(R, C)$, and $\texttt{OneShelf}(R, C)$), which are applied in

each round to a family of disjoint cells. Initially all of these algorithms are applied in the bounding box C of the input R, but later they are applied in cells of different subdivisions. For each of these algorithms, we maintain the set of cells in which these algorithms are applied in the current level of recursion and which contain some portions of input objects. Since R is partitioned into $O(n \log^8 n)$ fragments, this is an upper bound on the number of cells that we need to store at any point in our algorithm for each subdivision.

For every cell C in every subdivision that contains any fragment of an input object, we maintain the set $R(C) = \{r \cap C : r \in R, r \cap C \neq \emptyset\}$ of clipped rectangles and the axis-aligned bounding box $\hat{C}$ of $R(C)$. Empty cells are discarded from any further consideration. For each clipped rectangle $r \cap C$, we store its status with respect to C (orientation; pass-through or not; if pass-through then free, shelf, or bridge; x-, y-, or z-class). We maintain a vector for C indicating if $R(C)$ contains any pass-through rectangles; x-, y-, or z-class rectangles; or shelves. From the sorted list of m clipped rectangles we can extract the sorted list of vertices of clipped pass-through rectangles, bridges, and shelves in $O(m)$ time, and so we can quickly find the median planes for the steps $M_{yz}(R, C)$, $N_{xz}(R, C)$, and $Q_{xy}(R, C)$.

Assume that the input rectangles are partitioned into N fragments in the current BSP scheme P. There are two types of operations we apply. The first type is operation $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$, which partitions every cell along all free cuts. We can scan each cell C for free rectangles and partition C along all of them at once. If $R(C)$ contained m fragments, then we can extract the data structure for all of the resulting subcells in $O(m)$ time. Operation $P \leftarrow P \triangleleft \texttt{FreeCut}(R)$ can be done in $O(N)$ time.

The second type of operation corresponds to the median cuts of one round of an algorithm $\texttt{Main}(R, C)$, $\texttt{Long}(R, C)$, $\texttt{Sep3}(R, C)$, $\texttt{Sep2}(R, C)$, $\texttt{Shelf}(R, C)$, or $\texttt{OneShelf}(R, C)$. We are given a family $\mathcal{C}$ of disjoint cells, each cell $C \in \mathcal{C}$ may contain several cells of the subdivision $\mathcal{L}$ of the current BSP scheme P, and we overlay some kind of median cut $h(C)$ for each $C \in \mathcal{C}$ with P. For each cell $C \in \mathcal{C}$ and its splitting plane $h(C)$, we identify all cells in $\mathcal{L}$ that lie in C and intersect $h(C)$. For each such cell $L \in \mathcal{L}$, we decide in $O(1)$ time whether L should be partitioned along $h(C)$ or not. Since each cell of $\mathcal{L}$ is partitioned into at most two subcells, we can update the data structure in $O(N)$ time.

In total, our algorithm has $O(\log^4 n)$ rounds (the height of the BSP tree is $O(\log^5 n)$, where one $O(\log n)$ factor corresponds to the free cuts). Each round can be performed in $O(N) = O(n \log^8 n)$ time, so the total time complexity amounts to $O(n \log^{12} n)$. □

6. Lower bound. We present a set of $3n$ disjoint axis-aligned squares in $\mathbb{R}^3$ such that any axis-aligned BSP for them has $\Omega(n \log n)$ size. The $3n$ squares are grouped into three families, each of size n, as follows (refer to Figure 7):

$$A(n) = \{a_i = [3i, 3i + 3n] \times [3i - 3n, 3i] \times [3i] : i = 0, 1, \ldots, n - 1\},$$
$$B(n) = \{b_i = [3i + 1] \times [3i + 1, 3i + 1 + 3n] \times [3i + 1 - 3n, 3i + 1] : i = 0, 1, \ldots, n - 1\},$$
$$C(n) = \{c_i = [3i + 2 - 3n, 3i + 2] \times [3i + 2] \times [3i + 2, 3i + 2 + 3n] : i = 0, 1, \ldots, n - 1\}.$$

Each of the families $A(n)$, $B(n)$, and $C(n)$ consists of n parallel and pairwise disjoint axis-aligned squares. Every square has a vertex along the line $x = y = z$. Squares of different families are disjoint because any two families are separated by a plane. $A(n)$ and $B(n)$ are separated by the plane $x = y$. Similarly, $A(n)$ and $C(n)$ lie on different sides of $x = z$, while $B(n)$ and $C(n)$ are separated by $y = z$.

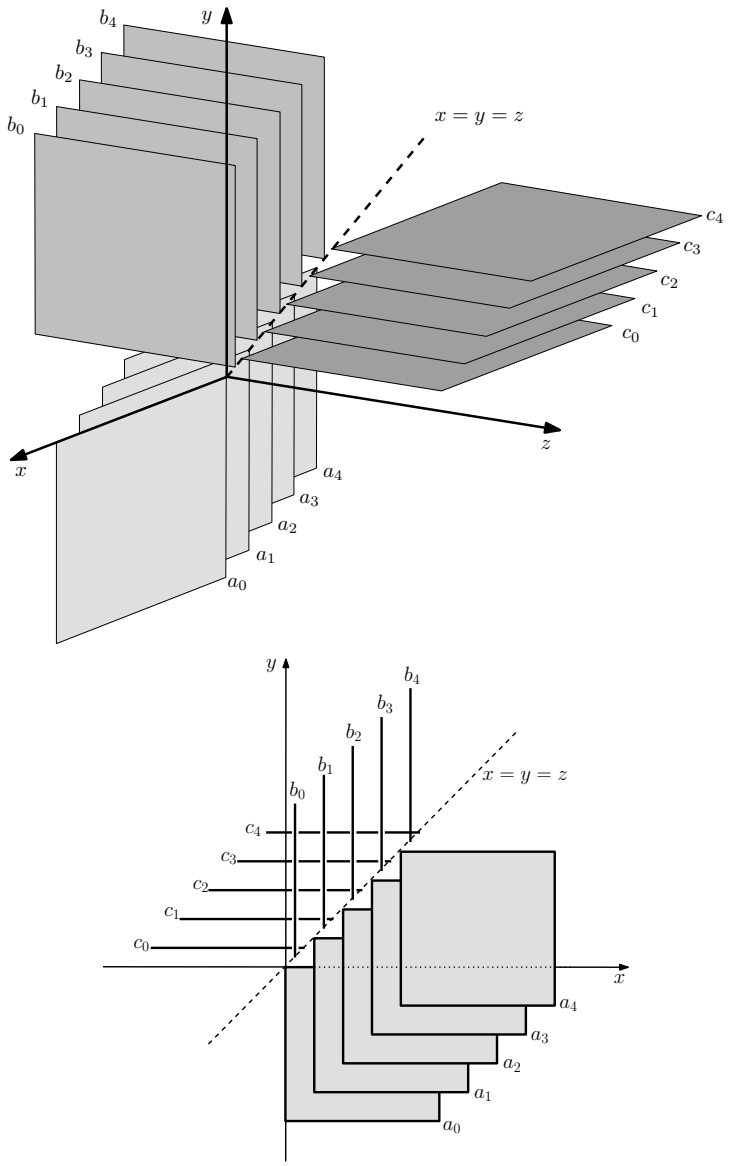


FIG. 7. *The lower bound construction for $n = 5$. View of 15 squares from the negative octant, where the dashed segment lies in the positive octant (top), and a projection to the xy-plane (bottom).*

For every $k = 1, 2, \ldots, n$, let $F(k)$ denote the set of fragments of input squares clipped in the cube $Q(k) = [0, 3k] \times [0, 3k] \times [0, 3k]$. Let $f(k)$ denote the minimum size of an axis-aligned BSP for $F(k)$. We show that the following recursion relations hold:

$$f(0) = 0, \tag{1}$$

$$f(1) \geq 1, \tag{2}$$

$$f(k) \geq k - 1 + \min_{0 \leq i \leq k-1} (f(i) + f(k - 1 - i)) \quad \text{for} \quad 2 \leq k \leq n. \tag{3}$$

The solution to this recursion is $f(n) = \Omega(n \log n)$, and so Theorem 2 follows immediately.

Inequalities (1) and (2) are obvious. For inequality (3), consider a configuration $F(k)$, $1 < k \leq n$, and suppose that an axis-aligned splitting plane h cuts the cube $Q(k)$. Suppose that h cuts the diagonal $[(0,0,0),(3k,3k,3k)]$ of $Q(k)$ at a point on the segment $[(3i,3i,3i),(3i+3,3i+3,3i+3)]$ for some $i \in \{0,1,\ldots,n-1\}$. First, we show that h cuts at least $k-1$ elements of $F(k)$. If h is an xy-plane, then it cuts $c_0, c_1, \ldots, c_{i-1}$ and $b_{i+1}, b_{i+2}, \ldots, b_{k-1}$. If h is an xz-plane, then it cuts $b_0, b_1, \ldots, b_{i-1}$ and $a_{i+1}, a_{i+2}, \ldots, a_{k-1}$. If h is an yz-plane, then it cuts $a_0, a_2, \ldots, a_{i-1}$ and $c_{i+1}, c_{i+2}, \ldots, c_{k-1}$.

Let us denote by Q_1 and Q_2 the cubes spanned by the segments $[(0,0,0),(3i,3i,3i)]$ and $[(3i+3,3i+3,3i+3),(3k,3k,3k)]$, respectively. The set of rectangles of $F(k)$ clipped in Q_1 is $F(i)$, and the set of rectangles of $F(k)$ clipped in Q_2 is a configuration congruent to $F(k-i-1)$. The cubes Q_1 and Q_2 are disjoint from h, and so the subconfigurations $F(i)$ and $F(k-i-1)$ are intact. Hence the size of the axis-aligned BSP $F(k)$ is at least $k-1+f(i)+f(k-i-1)$. This confirms inequality (3) and completes our proof.

7. Conclusion. Let $f(F_n(k,d))$ denote the minimum size of an axis-aligned BSP for a set $F_n(k,d)$ of n disjoint fat axis-aligned k-flats in $\mathbb{R}^d$, and let $f_n(k,d) = \max_{F_n(k,d)}\{f(F_n(k,d))\}$. We have shown that $f_n(2,3) = O(n\log^8 n)$ and $f_n(2,3) = \Omega(n\log n)$. Moreover, there is an $O(n\log^8 n)$ size and $O(\log^5 n)$ height BSP for any $F_n(2,3)$ such that every input rectangle is partitioned into $O(\log^8 n)$ fragments.

In general we would like to determine the asymptotic behavior of $f_n(k,d)$ for every $k,d \in N$, $1 < k < d$, when $n \to \infty$. It is easy to see that we cannot hope for anything better than $f_n(k,d) = \Theta(f_n(1,d-k+1))$. Indeed, consider a worst case construction $F_n(1,d-k+1)$ for n axis-parallel line segments in $\mathbb{R}^{d-k+1}$. Embed $F_n(1,d-k+1)$ into a $(d-k+1)$-dimensional affine subspace H of $\mathbb{R}^d$, and draw a k-dimensional square above every line segment orthogonally to H. We obtain n disjoint k-dimensional squares in $\mathbb{R}^d$. The size of an axis-aligned BSP for this set $F_n(k,d)$ cannot be smaller than $f_n(1,d-k+1)$. We conjecture that this bound can be attained apart from a polylogarithmic factor, i.e., k-dimensional axis-aligned fat rectangles allow (in some sense) for reducing the dimension of the space by $k-1$.

CONJECTURE 1. *For any $d \in \mathbb{N}$, $4 \leq d$, there is a BSP of $O(n\,\mathrm{polylog}\,n)$ size and $O(\mathrm{polylog}\,n)$ height for any set $F_n(d-1,d)$.*

CONJECTURE 2. *For any $k,d \in \mathbb{N}$, where $1 < k$ and $k+1 < d$, there is a BSP of $O(n^{\frac{d-k+1}{d-k}}\,\mathrm{polylog}\,n)$ size and $O(\mathrm{polylog}\,n)$ height for any set $F_n(k,d)$.*

REFERENCES

[1] P. K. AGARWAL, E. F. GROVE, T. M. MURALI, AND J. S. VITTER, *Binary space partitions for fat rectangles*, SIAM J. Comput., 29 (2000), pp. 1422–1448.

[2] S. AR, B. CHAZELLE, AND A. TAL, *Self-customized BSP trees for collision detection*, Comput. Geom., 15 (2000), pp. 91–102.

[3] S. AR, G. MONTAG, AND A. TAL, *Deferred, self-organizing BSP trees*, Comput. Graph. Forum, 21 (2002), pp. 269–278.

[4] S. ARYA, *Binary space partitions for axis-parallel line segments: Size-height tradeoffs*, Inform. Process. Lett., 84 (2002), pp. 201–206.

[5] S. ARYA, T. MALAMATOS, D. M. MOUNT, AND K. C. WONG, *Optimal expected-case planar point location*, SIAM J. Comput., 37 (2007), pp. 584–610.

[6] T. ASANO, M. DE BERG, O. CHEONG, L. J. GUIBAS, J. SNOEYINK, AND H. TAMAKI, *Spanning trees crossing few barriers*, Discrete Comput. Geom., 30 (2003), pp. 591–606.

[7] C. BALLIEUX, *Motion Planning Using Binary Space Partitions*, Technical report Inf/src/93-25, Utrecht University, Utrecht, The Netherlands, 1993.

[8] N. Chin and S. Feiner, *Near real-time shadow generation using BSP trees*, ACM SIGGRAPH Computer Graphics, 23 (1989), pp. 99–106.
[9] N. Chin and S. Feiner, *Fast object-precision shadow generation for area light sources using BSP trees*, Comput. Graph., 25 (1992), pp. 21–30.
[10] M. de Berg, *Linear size binary space partitions for uncluttered scenes*, Algorithmica, 28 (2000), pp. 353–366.
[11] M. de Berg, H. David, M. Katz, M. Overmars, A. F. van der Stappen, and J. Vleugels, *Guarding scenes against invasive hypercubes*, Comput. Geom., 26 (2003), pp. 99–117.
[12] M. de Berg, M. de Groot, and M. Overmars, *New results on binary space partitions in the plane*, Comput. Geom., 8 (1997), pp. 317–333.
[13] M. de Berg and M. Streppel, *Approximate range searching using binary space partitions*, Comput. Geom., 33 (2006), pp. 139–151.
[14] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1997.
[15] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir, *Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles*, Discrete Comput. Geom., 31 (2004), pp. 207–227.
[16] H. Fuchs, Z. M. Kedem, and B. Naylor, *On visible surface generation by a priori tree structures*, Comput. Graph., 14 (1980), pp. 124–133.
[17] J. Hershberger, S. Suri, and Cs. D. Tóth, *Binary space partitions of orthogonal subdivisions*, SIAM J. Comput., 34 (2005), pp. 1380–1397.
[18] C. S. Mata and J. S. B. Mitchell, *Approximation algorithms for geometric tour and network design problems*, in Proceedings of the 11th ACM Symposium on Computational Geometry, ACM Press, New York, 1995, pp. 360–369.
[19] B. Naylor, *Constructing good partitioning trees*, in Proceedings of Graphics Interface, 1993, Canadian Human-Computer Communications Society, Toronto, 1993, pp. 181–191.
[20] M. S. Paterson and F. F. Yao, *Efficient binary space partitions for hidden-surface removal and solid modeling*, Discrete Comput. Geom., 5 (1990), pp. 485–503.
[21] M. S. Paterson and F. F. Yao, *Optimal binary space partitions for orthogonal objects*, J. Algorithms, 13 (1992), pp. 99–113.
[22] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp, *Study for Applying Computer-generated Images to Visual Simulation*, Technical report AFHRL–TR–69–14, U.S. Air Force Human Resources Laboratory, San Antonio, TX, 1969.
[23] W. C. Thibault and B. F. Naylor, *Set operations on polyhedra using binary space partitioning trees*, Comput. Graph., 21 (1987), pp. 153–162.
[24] Cs. D. Tóth, *A note on binary plane partitions*, Discrete Comput. Geom., 30 (2003), pp. 3–16.
[25] Cs. D. Tóth, *Binary space partitions for line segments with a limited number of directions*, SIAM J. Comput., 32 (2003), pp. 307–325.

SPECIAL ISSUE DEDICATED TO THE THIRTY-SEVENTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING (STOC 2005)

This volume comprises the polished and fully refereed versions of a selection of papers presented at the Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC 2005), held in Baltimore, Maryland, May 22–24, 2005. Unrefereed preliminary versions of the papers presented at the symposium appeared in the proceedings of the meeting, published by ACM. The symposium was sponsored by the ACM Special Interest Group on Algorithms and Computation Theory (SIGACT).

The STOC 2005 Program Committee consisted of Gerth Stølting Brodal, Harry Buhrman, Jin-Yi Cai, Cynthia Dwork, Ronald Fagin (chair), Martin Farach-Colton, Anupam Gupta, Sariel Har-Peled, Russell Impagliazzo, Kamal Jain, Adam Tauman Kalai, David Karger, Claire Kenyon, Subhash Khot, Ravi Kumar, Moni Naor, Ryan O'Donnell, Toniann Pitassi, Tim Roughgarden, Alistair Sinclair, and Amnon Ta-Shma.

Out of 290 "Extended Abstracts" submitted to the STOC 2005 Program Committee, 84 were selected for presentation at the symposium. The present volume includes 9 of these papers that were invited to this volume. All papers were refereed in accordance with the *SIAM Journal on Computing*'s stringent standards, and these papers were substantially updated in the process. We take this opportunity to thank all the referees whose anonymous work has significantly contributed to the value of this volume.

Ronald Fagin, the Program Chair of the 2005 STOC Conference, invited three other members of the Program Committee (Anupam Gupta, Ravi Kumar, and Ryan O'Donnell) to assist in editing this special issue of the of the *SIAM Journal on Computing*, and all agreed. We feel that it was an honor to edit this issue.

Ronald Fagin, IBM Almaden Research Center
Anupam Gupta, Carnegie Mellon University
Ravi Kumar, Yahoo! Research
Ryan O'Donnell, Carnegie Mellon University
Guest Editors

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 449–483

AN $O(\log n \log\log n)$ SPACE ALGORITHM FOR UNDIRECTED ST-CONNECTIVITY*

VLADIMIR TRIFONOV†

Abstract. We present a deterministic $O(\log n \log\log n)$ space algorithm for undirected st-connectivity. It is based on a space-efficient simulation of the deterministic EREW algorithm of Chong and Lam [*J. Algorithms*, 18 (1995), pp. 378–402], an approach suggested by Prof. Vijaya Ramachandran, and uses the universal exploration sequences for trees constructed by Koucký in [*Proceedings of the* 16*th Annual IEEE Conference on Computational Complexity*, 2001, pp. 21–27]. Our result improves the $O(\log^{4/3} n)$ bound of Armoni et al. in [*Proceedings of the* 20*th Annual ACM Symposium on Theory of Computing*, 1997, pp. 230–239] and is a big step towards the optimal $O(\log n)$. Independently of our result and using a different set of techniques, the optimal bound was achieved by Reingold in [*Proceedings of the* 37*th Annual ACM Symposium on Theory of Computing*, 2005, pp. 376–385].

Key words. undirected st-connectivity, space-bounded computation

AMS subject classifications. 05C40, 05C85, 68Q25, 68Q15

DOI. 10.1137/050642381

1. Introduction.

The problem we are concerned with is st-connectivity in an undirected graph G with n vertices (USTCONN); i.e., given two vertices s and t of G, we want to answer the question whether there is a path between s and t. This is one of the most basic graph problems with applications ranging from image processing and VLSI design to solving more complex graph problems. Furthermore, st-connectivity plays an important role in complexity theory because directed st-connectivity (STCONN) is **NL**-complete [20] and USTCONN is **SL**-complete [14].

Linear time and space sequential algorithms for solving even the harder STCONN problem have been known for a long time [22]. The problem of developing more efficient space and parallel algorithms was posed. The result of Aleliunas et al. [1] shows that USTCONN can be solved in $O(\log n)$ space with a randomized algorithm with one-sided error, i.e., an algorithm which produces an answer in polynomial time, and, if the two vertices are not connected, then the algorithm is correct; otherwise, it answers incorrectly with probability at most 1/2. The starting point of deterministic space-efficient sequential algorithms is the $O(\log^2 n)$ space algorithm for STCONN of Savitch [20]. For a long time, this was the best result even for USTCONN. The space bound for undirected graphs was first improved by Nisan, Szemerédi, and Wigderson [16] to $O(\log^{3/2} n)$ and then to $O(\log^{4/3} n)$ by Armoni et al. [2]. Both of these results depend on the efficient construction of universal traversal sequences by Nisan [15]. Finally, simultaneous to our result and using different set of techniques, the space complexity of USTCONN was shown by Reingold [18] to be $O(\log n)$.

Developing efficient parallel algorithms for USTCONN has a very rich history. The situation here is complicated further by the existence of multiple models of parallel computation. The models from the PRAM family are generally considered to be of great theoretical value. The results of [10, 19, 4, 11, 12] are concerned with the

*Received by the editors October 10, 2005; accepted for publication (in revised form) January 24, 2007; published electronically May 23, 2008. This work was supported in part by NSF grant CCF-0430695 and by Texas Advanced Research Program grant 003658-0029-1999.
http://www.siam.org/journals/sicomp/38-2/64238.html
†Department of Computer Sciences, University of Texas at Austin, Austin, TX.

CREW PRAM model, [21, 3, 7, 8] with the CRCW PRAM model, and [9, 6, 5, 17] with the EREW PRAM. The state of the art in parallel algorithms for USTCONN are the results of Chong, Han, and Lam [5], which shows that the problem can be solved on the EREW PRAM in $O(\log n)$ time with $O(m+n)$ processors, and Pettie and Ramachandran [17], which demonstrates a randomized EREW PRAM algorithm running in time $O(\log n)$ with optimal number of processors. The algorithms of [5] and [17] actually solve the harder problem of finding the minimum spanning tree of a weighted graph.

The starting point of our algorithm is the $O(\log n \log\log n)$ time deterministic EREW PRAM algorithm with $O(m+n)$ processors of Chong and Lam [6], which we call the CL algorithm. This parallel algorithm can be trivially simulated sequentially in linear space. We use the sequential algorithm to define a mathematical structure called configuration, which captures the state of the algorithm. We define also a sequence of configurations, such that every element of this sequence corresponds to the state of the sequential algorithm at a certain point of its execution. We use the sequence of configurations to trivially define an $O(\log^2 n)$ space algorithm, which instead of storing all of its current state recomputes parts of it when it needs them. This technique is standard for designing space-efficient algorithms. Finally, we modify the $O(\log^2 n)$ space algorithm into an algorithm which uses $O(\log n \log\log n)$ space.

The possibility of simulating parallel algorithms for USTCONN space-efficiently was suggested to the author by Prof. Vijaya Ramachandran in 2000. She conjectured an $O(\log n \log\log n)$ space algorithm derived from the CL algorithm and an alternate simple $O(\log^{3/2} n)$ space algorithm derived from the algorithm of Johnson and Metaxas [11], by using the max-degree hooking scheme of [6]. She observed that the step needing derandomization in [16] is not necessary in a tree-based hook and contract approach, because the trees automatically give rise to disjoint clusters of vertices. The max-degree hooking scheme employed by [6] gives the additional benefit that small trees have small neighborhoods. The main challenge was to implement the levels of recursion, so that they process small trees in $o(\log n)$ space. Solving this problem is the main contribution of this paper.

In the algorithm presented here the space of a level of recursion is between $\Omega(\log\log n)$ and $\Theta(\log n)$, depending on the level. A key tool for our method are the exploration walks on trees defined by Koucký [13]. Exploration walks on trees are similar to the Euler tour technique used by Tarjan and Vishkin [23] in the parallel context. These walks play the role of the edge-list plugging technique and pointer jumping employed by the CL algorithm, because they allow us to traverse trees very efficiently.

Section 2 contains a high-level overview of the CL algorithm. Section 3 defines formally a labeled multigraph and operations on it. Furthermore, it provides a complete description of the sequential version of the CL algorithm. Section 4 gives a detailed description of the space-efficient implementation of the CL algorithm. Section 5 proves the correctness of the sequential CL algorithm. The proofs in section 5 are from [6] and for completeness are adapted here to our framework. Although we make frequent references to [6], the exposition in this paper is self-contained, except for Proposition 3.4, whose proof is not too difficult.

Define $[k] = \{1, \ldots, k\}$ and the double exponential function $\operatorname{dexp}(x) = 2^{2^x}$. All logarithms in the paper are of base 2.

2. The Chong–Lam algorithm. This section gives a high-level overview of the Chong–Lam (CL) algorithm [6] to motivate the definitions in the next section. The CL algorithm uses a hook and contract approach. There are several phases of

hooking and contraction. Before every phase, every vertex of the current graph is in exactly one of three states—active, inactive, and done; all active and inactive vertices have nonzero degree, the done vertices have zero degree, and there are no multiedges between active vertices; the inactive vertices are organized in a set of hooking trees. A hooking tree is a subgraph of the current graph, which is a tree and whose internal edges, which we call hooking edges, are obtained as described below. Initially all vertices with nonzero degree are active, and the rest are done.

In a hooking phase the active vertices in parallel choose to hook to one of their current neighbors, establishing their hooking edges, and thus either become part of existing hooking trees or form new ones. The fact that the connected components formed by the hooking of vertices are trees is ensured by the hooking scheme of the CL algorithm. This hooking scheme uses an ordering $<_d$ of the vertices such that $u <_d v$ iff the degree of u is less than the degree of v or they are the same, but u is less than v, thinking of the vertices as elements of $\mathbb{N}$. To choose their hooking edges, the active vertices of the graph perform in parallel two consecutive, synchronized steps. First, if a vertex v has a neighbor larger according to $<_d$ than itself, then v hooks to the largest such neighbor. Second, if after the first step all neighbors of v are hooked to it, then v hooks to itself; i.e., it does not choose a hooking edge. Otherwise, if after the first step a neighbor u of v is hooked to a vertex different from v, then v hooks to u.

In a contraction phase some of the current hooking trees are contracted to a representative vertex. The representative vertex is the only vertex in the tree which is hooked to itself. Which trees are contracted is determined by a parameter, which depends on the phase and sets an upper bound on the total degree, i.e., the sum of the degrees of the vertices, of the trees which are contracted. For every contracted tree, its representative becomes a new active vertex and the rest of its vertices become done. Also all multiedges between new active vertices are cleaned up. Finally, the vertices of every uncontracted tree become inactive.

The processing required by a hooking phase is performed in parallel time $O(\log d)$, where d is the degree of the active vertex, using pointer jumping. The important part of a contraction phase is checking the degree of a hooking tree. In parallel this could be done in $O(\log c)$ time, where c is the contraction parameter, by using pointer jumping and a constant time edge-list plugging technique.

Finally, the CL algorithm is given by the following recursive procedure. Here `MaxHook` and `Contract`(c) denote correspondingly a hooking and a contracting phase with parameter c.

```
procedure Connect(k)
    MaxHook;
    if k > 0 then
        Contract(dexp(k));
        Connect(k − 1);
        Connect(k − 1);
    Contract(dexp(k+1));
```

The correctness of the CL algorithm ensures that a call to `Connect`$(\lceil \log \log n \rceil)$ contracts every connected component of the graph to a single vertex and all the other vertices are organized in a set of rooted parent trees such that the root of the tree of a vertex u is the vertex to which the connected component of u contracted.

We simulate the CL algorithm trivially with a sequential algorithm using linear space. We fix an ordering on the edges incident to a vertex, and instead of performing the hooking in parallel for all active vertices, we do it sequentially for each of them.

This is possible, because by changing the hooking scheme of CL slightly we can ensure that the hooking of an active vertex does not depend on the hooking of the other active vertices. The new hooking strategy gives preference to neighbors which are inactive or have an inactive neighbor, but it still ensures that small trees composed of active vertices have small degree. The details of the sequential algorithm are given in the following section, and space-efficient versions of the same algorithm are given in section 4.

3. Definitions.

3.1. Multigraphs and exploration walks. An undirected multigraph is a graph with possibly multiple edges between two vertices and such that every edge has a label on each side, where the labels of the edges incident to a vertex v have distinct labels on the side of v. We also have a single self-loop with label 0 at every vertex. Formally, we have the following.

DEFINITION 3.1. *An* undirected multigraph G *is a triple* $\langle V, \delta, \mu\rangle$*, where* V *is a set,* $\delta : V \to \mathbb{N}$*, and* $\mu : E \to E$ *is a bijection such that* $\mu(\mu(e)) = e$ *and* $\mu(v, 0) = (v, 0)$*, where* $E = \{(v, i) : v \in V \text{ and } 0 \le i \le \delta(v)\}$*.*

V *is the* set of vertices *of* G*,* E *is the* set of edges *of* G*,* $\delta(v)$ *is the* degree *of* v*, and* $\mu(e)$ *is the* reverse edge *of* e*. For an edge* e*, call the set* $\{e, \mu(e)\}$ *an* undirected edge*.*

Let $\eta : E \to V$ *and* $\beta : E \to \mathbb{N}$ *be the first and the second component of* μ*, respectively. Then* $\eta(v, i)$ *is the* ith neighbor *of* v*,* i *is the* label *of the edge* (v, i)*, and* $\beta(v, i)$ *is its* back-label*.*

Define the size *of* G*,* $\text{size}(G)$*, to be* $|V|$*.*

In the following a graph means undirected multigraph.

DEFINITION 3.2. *A graph* $G' = \langle V', \delta', \mu'\rangle$ *is a subgraph of a graph* $G = \langle V, \delta, \mu\rangle$ *if* $V' \subseteq V$ *and for every* $u, v \in V'$*,*

$$|\{i : \eta'(u, i) = v\}| \le |\{i : \eta(u, i) = v\}|.$$

Define (simple) path, connected vertices, forest, and tree in the usual way.

DEFINITION 3.3. *Let* G *be a graph. Let* $\Delta : E \times \mathbb{Z} \to E$ *be such that* $\Delta((v, i), j)$ *changes by* j *the label of the edge* (v, i)*. More precisely, for* $i \ne 0$*,*

$$\Delta((v, i), j) = (v, 1 + (i - 1 + j \bmod \delta(v))).$$

Define $\Gamma_{G,k}, \Gamma'_{G,k} : E \to E$ *inductively on* $k \ge 0$*. First,* $\Gamma_{G,0}(e) = \Gamma'_{G,0}(e) = e$*. Now let*

$$\Gamma_{G,k+1}(e) = \Delta(\mu(\Gamma_{G,k}(e)), 1),$$
$$\Gamma'_{G,k+1}(e) = \mu(\Delta(\Gamma'_{G,k}(e), -1)).$$

The sequence $\Gamma_{G,\le l}(e) = \langle \Gamma_{G,0}(e), \Gamma_{G,1}(e), \dots, \Gamma_{G,l}(e)\rangle$ *is called the* exploration walk *of length* $l + 1$ *starting from the edge* e*. We will also refer to the corresponding infinite sequence as* the exploration walk*. Let* $e_k = (v_k, i_k) = \Gamma_{G,k}(e)$*. Then* v_k *and* e_k *are correspondingly the* k*th vertex and the* k*th edge visited by the exploration walk starting from* e*. We have the equivalent notions for the reverse exploration walk, where* Γ *is replaced with* Γ'*.*

Exploration walks were introduced by Koucký [13]. We define them only for the exploration sequence which always changes by one the label of the current edge because this is the case of interest to us. The fact that exploration walks are reversible,

i.e., $\Gamma'_{G,l}(\Gamma_{G,l}(e)) = e$, was noticed by Koucký and is what makes them important to us. We will use the following property of exploration walks on trees.

PROPOSITION 3.4 (see [13]). *Let G be a tree with at most one undirected edge between any two vertices, let $e = (v, i) \in E$, $i \neq 0$, be an edge of G, and let $l = 2(\text{size}(G) - 1)$. Then the exploration walk $\Gamma_{G,\leq l-1}(e)$ of length l starting from e visits every edge of G which is not a self-loop exactly once. Furthermore, this is the shortest exploration walk which visits v exactly $\delta(v) + 1$ times.*

3.2. Operations on graphs.

3.2.1. Configuration. A configuration is the state of the sequential algorithm outlined at the end of section 2. Formally, we have the following.

DEFINITION 3.5. *A* configuration *is a tuple $\mathcal{C} = \langle G, A, I, D, H, R\rangle$, where G is a graph with $V = [n]$, for some $n \in \mathbb{N}$. A, I, and D form a partition of V. $v \in D$ iff $\delta(v) = 0$. $H : V \to \mathbb{N}$ and $R : V \to V$ are such that $H(v) \leq \delta(v)$, and if $R(v) \neq v$, then $v \in D$.*

The elements of A, I, and D are called correspondingly the active, *the* inactive, *and the* done *vertices of G. For $u \in V$, $(u, H(u))$ is the* hooking edge *of u, and $R(u)$ is the* immediate representative *of u.*

We require that H and R do not have nontrivial cycles in the following sense. Let $v_1, \ldots, v_k \in V$, $k \geq 2$. Then

(i) *if $v_{i+1} = \eta(v_i, H(v_i))$, $i \in [k-1]$, and $v_1 = \eta(v_k, H(v_k))$, then $H(v_1) = 0$;*
(ii) *if $v_{i+1} = R(v_i)$, $i \in [k-1]$, and $v_1 = R(v_k)$, then $R(v_1) = v_1$.*

We also require that there is at most one undirected edge between any two active vertices, i.e., if $u, v \in A$, then $|\{i : \eta(v, i) = u\}| \leq 1$, and that there is no hooking edge from an inactive to an active vertex, i.e., if $u \in I$, then $\eta(u, H(u)) \in I$.

Define the representative of v according to R to be

$$\text{rep}_R(v) = \begin{cases} v, & R(v) = v, \\ \text{rep}_R(R(v)) & \text{otherwise.} \end{cases}$$

Definition 3.5(ii) ensures the correctness of this definition.

DEFINITION 3.6. *Let $\mathcal{C} = \langle G, A, I, D, H, R\rangle$ be a configuration. H defines a subforest $F = \langle V, \delta_F, \mu_F\rangle$ of G, called the* hooking forest *of $\mathcal{C}$, with at most one undirected edge between any two vertices in the following way. Fix $v \in V$. Let ε be 1, if $H(v) \neq 0$, and 0 otherwise. Let $0 < i_1 < \cdots < i_k$ be such that*

$$\{i_1, \ldots, i_k\} = \{i : \exists u \in V \text{ such that } \mu(u, H(u)) = (v, i)\}.$$

First, define $\delta_F(v) = k + \varepsilon$. Now define $\eta_F(v, j) = \eta(v, i_j)$, for $1 \leq j \leq k$, and, if $\varepsilon = 1$, $\eta_F(v, k + \varepsilon) = \eta(v, H(v))$. Finally, define $\beta_F(v, j) = i$, where $\eta_F(v, j) = u$ and $\eta_F(u, i) = v$.

Let T be a maximal connected subtree of the forest F. We call T a hooking tree *in $\mathcal{C}$. The* root *of T,* root(T), *is the only vertex v in T such that $H(v) = 0$. The* degree *of T,* deg(T), *is $\sum_{v \in V_T} \delta(v)$. For a vertex $v \in V$, we denote with T_v the subtree of F which contains v.*

The correctness of this definition and the fact that F is a forest with at most one undirected edge between any two vertices follow from Definition 3.5(i).

3.2.2. Hooking. We will define Hook$(\mathcal{C})$ so that if $\mathcal{C}$ describes the state of the sequential algorithm, then Hook$(\mathcal{C})$ is its state after a hooking phase.

Being elements of $\mathbb{N}$, the elements of V are ordered. Define the linear ordering $<_d$ on V so that

$$u <_d v \quad \text{iff} \quad \delta(u) < \delta(v) \quad \text{or} \quad \delta(u) = \delta(v) \quad \text{and} \quad u < v.$$

The hooking operation $\mathrm{Hook}(\mathcal{C})$ produces the configuration $\langle G, A, I, D, H', R\rangle$ defined in the following way. If v is inactive, then $H'(v) = H(v)$. If v is active, let $v_1, \dots, v_{\delta(v)}$ be the neighbors of v, i.e., $v_i = \eta(v, i)$. For the rest of the definition when we have to choose an index i, we always pick the smallest one with the corresponding property. If v has an inactive neighbor v_i, let $H'(v) = i$. If all neighbors of v are active, let v_i be the largest according to $<_d$ amongst the neighbors of v. If $v <_d v_i$, let $H'(v) = i$. If all neighbors of v are active and smaller than v according to $<_d$, then if v has a neighbor v_i which has an inactive neighbor, let $H'(v) = i$. If all neighbors of v and their neighbors are active, then if v has a neighbor v_i which has a neighbor larger than v according to $<_d$, let $H'(v) = i$. Finally, if all neighbors of v and their neighbors are active and smaller than v according to $<_d$, define $H'(v) = 0$.

The hooking strategy described above differs from the hooking strategy of the CL algorithm because it gives preference to neighbors which are inactive or have an inactive neighbor. For example, in our hooking strategy a vertex hooks to an inactive neighbor, if it has one, regardless of its degree. In the hooking strategy of the CL algorithm, a vertex hooks to its largest according to $<_d$ neighbor, regardless of its state. The new hooking strategy does not change the correctness of the CL algorithm because first an active vertex still hooks to itself iff all of its neighbors are active and hooked to it, and second along a sequence of hooking edges of active vertices the vertices increase according to $<_d$ the same way as in the CL algorithm. The reason we chose the new strategy is to ensure that we do not look up the degree of an inactive vertex. The essential properties of the hooking operation are given by the following lemmas. The proofs are the same as in [6].

LEMMA 3.7. *Let $k \geq 3$ and $v_i \in A$, $i \in [k]$, be distinct and such that $v_{i+1} = \eta(v_i, H'(v_i))$, $i \in [k-1]$. Then $v_1 <_d \max_d\{v_{k-1}, v_k\}$.*

Proof. The proof is by induction on k. For $k = 3$, the statement holds, because v_1 is hooked to v_2, either because $v_2 >_d v_1$ or because $v_2 <_d v_1$, but v_2 had an inactive neighbor or an active neighbor larger than v_1. Since v_2 is hooked to v_3, the latter must be the case. So $v_1 <_d \max_d\{v_2, v_3\}$. For the inductive step, we have that $v_{k-1} <_d \max_d\{v_k, v_{k+1}\}$ and $v_1 <_d \max_d\{v_{k-1}, v_k\}$, and so $v_1 <_d \max_d\{v_k, v_{k+1}\}$. □

COROLLARY 3.8. $\mathrm{Hook}(\mathcal{C})$ *is a configuration.*

Proof. Since Hook changes only the hooking edges of active vertices, the only thing we have to check is that it does not create nontrivial cycles of active vertices. Assume that there is $k \geq 2$, and $v_i \in A$, $i \in [k]$, distinct and such that $v_{i+1} = \eta(v_i, H'(v_i))$ and $v_1 = \eta(v_k, H'(v_k))$. The case $k = 2$ is impossible because we must have either $v_1 <_d v_2$ or $v_1 >_d v_2$. In the first case v_2 will hook to v_1 only if v_1 is hooked to a vertex different from v_2. The second case is also impossible, and hence $k \geq 3$. By Lemma 3.7, $v_1 <_d \max_d\{v_{k-1}, v_k\}$ and $v_2 <_d \max_d\{v_k, v_1\}$. If $v_{k-1} <_d v_k$, then $v_1, v_2 <_d v_k$, a contradiction with Lemma 3.7 for v_k, v_1, and v_2. If $v_k <_d v_{k-1}$, then $v_1, v_k <_d v_{k-1}$, a contradiction with Lemma 3.7 for v_{k-1}, v_k, and v_1. Thus we must have that $H'(v_1) = 0$. □

LEMMA 3.9. *Let T be a hooking tree in* $\mathrm{Hook}(\mathcal{C})$ *composed of active vertices. Then* $\mathrm{size}(T) \geq 2$ *and* $\deg(T) < \mathrm{size}^2(T)$.

Proof. Let $r = \mathrm{root}(T)$. r hooks to itself, i.e., $H'(r) = 0$, iff all of its neighbors are active and hooked to it. So T must contain at least two vertices—r and its neighbors.

Let us see now that for $v \in V_T$,

$$\delta(v) \le \delta(r). \tag{3.1}$$

If v is a neighbor of r in T, then since r is hooked to itself, we must have that $v <_d r$ and hence that $\delta(v) \le \delta(r)$. Otherwise, let u be the neighbor of r on the path in T from v to r. Then, by Lemma 3.7, $v <_d \max_d\{u, r\}$, and so again $\delta(v) \le \delta(r)$.

We have that

$$\deg(T) = \sum_{v \in V_T} \delta(v) \le \text{size}(T)\delta(r) < \text{size}^2(T),$$

where in the first inequality we use (3.1), and the second follows because, as explained earlier, T must contain all neighbors of r. $\delta(r)$ is exactly the number of distinct neighbors of r, since there are no multiple edges between active vertices. □

3.2.3. Contraction. We will define Contract$(\mathcal{C}, d)$ so that if $\mathcal{C}$ describes the state of the sequential algorithm, then Contract$(\mathcal{C}, d)$ is its state after a contraction phase with parameter d. A hooking tree T in $\mathcal{C}$ is *d-contractable* if $\deg(T) \le d$.

The result of Contract$(\mathcal{C}, d)$ is the configuration $\mathcal{C}' = \langle G', A', I', D', H', R' \rangle$ defined in the following way. First, define

$$\begin{aligned} A'' &= \{v : v \notin D \text{ and } \deg(T_v) \le d \text{ and } \text{root}(T_v) = v\}, \\ I' &= \{v : v \notin D \text{ and } \deg(T_v) > d\}, \\ D'' &= \{v : v \in D \text{ or } \deg(T_v) \le d \text{ and } \text{root}(T_v) \ne v\}. \end{aligned}$$

Now define

$$H'(v) = \begin{cases} H(v), & v \in I', \\ 0 & \text{otherwise} \end{cases}$$

and

$$R'(v) = \begin{cases} R(v), & v \in D, \\ \text{root}(T_v), & v \in D'' - D, \\ v & \text{otherwise.} \end{cases}$$

Let T be a hooking tree in $\mathcal{C}$ and $s = \text{size}(T)$. Let $\langle v_1, \ldots, v_s \rangle$ be the sequence of the vertices of T in the order in which they are visited by the exploration walk on T of length $2(s-1)$ starting from the edge $(\text{root}(T), 1)$ of T, where we include a vertex only the first time it is visited by the exploration walk. Let $\langle e_1, \ldots, e_k \rangle$ be the sequence of the edges of G incident to the vertices of T defined in the following way—list in order of their labels all edges incident to v_1, then all edges incident to v_2, and so on. Obviously, $k = \deg(T)$.

Let us now define G'. For $u \in A''$, define $l_u \ge 0$ and the sequence of edges $\langle e_{u,1}, \ldots, e_{u,l_u} \rangle$. First, consider the sequence $\langle e_1, \ldots, e_k \rangle$ of the edges of G incident to the vertices of T_u from the previous paragraph. Remove from this sequence all edges which are internal to T_u, i.e., such that $\eta(e_i) \in V_{T_u}$. From every subsequence of edges whose other end belongs to the same d-contractable hooking tree, leave only the first edge; i.e., for every d-contractable hooking tree $T \ne T_u$ of $\mathcal{C}$, if the sequence $\langle e_{i_1}, \ldots, e_{i_h} \rangle$, $i_1 < \cdots < i_h$, contains all the edges from the sequence $\langle e_1, \ldots, e_k \rangle$ such that $\eta(e_{i_j}) \in V_T$, then remove e_{i_j}, $j > 1$. Let l_u be the number of remaining edges

in the sequence and $\langle e_{u,1}, \dots, e_{u,l_u} \rangle$ be the resulting sequence. Naturally, we call the edges $e_{u,j}$ the *remaining edges of* T_u.

Define

$$A' = A'' - \{v \in A'' : l_v = 0\},$$
$$D' = D'' \cup \{v \in A'' : l_v = 0\}$$

and

$$\delta'(v) = \begin{cases} l_v, & v \in A', \\ \delta(v), & v \in I', \\ 0, & v \in D'. \end{cases}$$

We are left to define $\mu'(v,i)$. First, assume $v \in A'$. Let $(u,j) = \mu(e_{v,i})$. If T_u is not d-contractable, then define $\mu'(v,i) = (u,j)$. If T_u is d-contractable, then define $\mu'(v,i) = (w,k)$, where $w = \mathrm{root}(T_u)$ and k is the only index such that $\eta(e_{w,k}) \in V_{T_v}$. Now assume $v \in I'$. Let $(u,j) = \mu(v,i)$. If T_u is not d-contractable, then define $\mu'(v,i) = (u,j)$. If T_u is d-contractable, let $\mu'(v,i) = (w,k)$, where $w = \mathrm{root}(T_u)$ and k is the only index such that $e_{w,k} = (u,j)$.

LEMMA 3.10. *Let* $\mathcal{C}' = \mathrm{Contract}(\mathcal{C}, d)$. *Then* $\mathcal{C}'$ *is a configuration such that* $\delta'(v) \le d$ *for* $v \in A'$, *and* $\deg(T_v) > d$ *for* $v \in I'$. *Furthermore, in the hooking forest of* $\mathcal{C}'$ *every* $v \in A' \cup D'$ *is in a hooking tree which contains only* v.

Proof. The proof is immediate from the definition of $\mathrm{Contract}(\mathcal{C}, d)$. □

3.3. Sequential version of the CL algorithm. For every $k \in \mathbb{N}$, we will define a sequence of configurations $\mathcal{C}_l = \langle G_l, A_l, I_l, D_l, H_l, R_l \rangle$, $0 \le l \le r(k)$, where $r(k) = 5 \cdot 2^k - 3$.

First, define recursively a sequence of pairs, the first element of which is always either *Hook* or *Contract*, and the second is a natural number, in the following way:

$$S_k = \begin{cases} (\langle Hook, 0 \rangle, \langle Contract, 1 \rangle), & k = 0, \\ (\langle Hook, k \rangle, \langle Contract, k \rangle, S_{k-1}, S_{k-1}, \langle Contract, k+1 \rangle), & k > 0. \end{cases}$$

The definition of S_k is obtained by "linearizing" the recursive definition of `Connect`(k).

By induction, S_k has $r(k)$ elements $P_1, \dots, P_{r(k)}$. Let $P_l = \langle Op_l, Arg_l \rangle$, $1 \le l \le r(k)$. Let $\mathcal{C}_0$ be some configuration. Assume that we have already defined $\mathcal{C}_0, \dots, \mathcal{C}_{l-1}$ for $1 \le l \le r(k)$. Define

$$\mathcal{C}_l = \begin{cases} \mathrm{Hook}(\mathcal{C}_{l-1}), & Op_l = Hook, \\ \mathrm{Contract}(\mathcal{C}_{l-1}, \mathrm{dexp}(Arg_l + 1)) & \text{otherwise.} \end{cases}$$

The sequential version of the CL algorithm for testing whether two vertices u and v are connected in a graph G consists of computing the sequence of configurations $\mathcal{C}_0, \dots, \mathcal{C}_{r(k)}$, where $\mathcal{C}_0$ is initialized according to G and $k = \lceil \log \log n \rceil$, and checking whether u and v have the same representative in $C_{r(k)}$: the two vertices are connected iff they have the same representative. The correctness of this algorithm follows from the statements below (see section 5 for their proofs). We make the following definition first.

DEFINITION 3.11. *A configuration* $\mathcal{C}_l$, $0 \le l < r(k)$, *is* nice *if the following hold:*

(i) *if* $Op_{l+1} = Hook$, *then for every* $v \in I_l$, $\mathrm{size}(T_v) > \mathrm{dexp}(Arg_{l+1} + 1)$ *and* $\deg(T_v) > \mathrm{dexp}(Arg_{l+1} + 2)$; *and*

(ii) *for every* $v \in A_l$, $\delta_l(v) \le \mathrm{dexp}(Arg_{l+1} + 2)$.

By definition, if $Op_{l+1} = Hook$, then $\mathcal{C}_l$ is nice iff the state described by it fulfills the preconditions given in [6] for executing $\texttt{Connect}(Arg_{l+1})$. These preconditions ensure the correctness of the algorithm and that it can be executed efficiently in parallel. Considering the correspondence between the sequence $\mathcal{C}_0, \ldots, \mathcal{C}_{r(k)}$ and the $\texttt{Connect}(k)$ procedure of the CL algorithm, the following theorem is a consequence of the results in [6].

THEOREM 3.12. *If $\mathcal{C}_0$ is a nice configuration, then $\mathcal{C}_l$ is nice, for every $1 \leq l < r(k)$, $|A_{r(k)}| \leq \max\{|A_0|/\mathrm{dexp}(k), 1\}$, and $\mathrm{size}(T_v) > \mathrm{dexp}(k+1)$ for $v \in I_{r(k)}$.*

Finally, again similarly to [6], we have the following corollary, which says that if we initialize $\mathcal{C}_0$ according to some undirected graph G, then in $\mathcal{C}_r$ all components of G are contracted.

COROLLARY 3.13. *Let G be a graph with at most one undirected edge between any two vertices and $V = [n]$. Let $k = \lceil \log \log n \rceil$ and $\mathcal{C}_0 = \langle G, A, I, D, H, R \rangle$, where $A = \{v : \delta(v) \neq 0\}$, $I = \emptyset$, $D = V - A$, $H(v) = 0$, and $R(v) = v$, for $v \in V$. Then u and v are connected in G iff $\mathrm{rep}_{R_{r(k)}}(u) = \mathrm{rep}_{R_{r(k)}}(v)$.*

4. Space-efficient algorithm. Since giving directly the Turing machine which solves the problem and reasoning about its space complexity is rather cumbersome, we define the algorithms outlined in sections 4.1 and 4.2 using pseudocode and then explain how to translate the pseudocode to a Turing machine. The algorithms, as given by the pseudocode, are almost a literal rephrasing of the definitions given in sections 3.2 and 3.3 into a precise language in which analyzing space complexity is possible. The deviations from a literal rephrasing exist only to decrease the space requirements of the algorithms. Besides that, the correctness of the algorithms follows essentially from Corollary 3.13. In fact, the algorithm in section 4.1 can be thought of as a literal $O(\log^2 n)$ space implementation of the sequential algorithm described in section 3.3.

In section 4.1 we outline an $O(\log^2 n)$ space implementation of the sequential algorithm derived from the definitions in sections 3.2 and 3.3, which instead of storing all of the current configuration recomputes parts of it when it needs them. In section 4.2 we describe the changes we make to the algorithm from section 4.1 to reduce its space complexity to $O(\log n \log \log n)$. Section 4.4 discusses in detail the pseudocode for the algorithm from section 4.2.

4.1. An $O(\log^2 n)$ space algorithm. Let G be a graph with $V = [n]$ and with at most one undirected edge between any two vertices. Let

$$r = 5 \cdot 2^{\lceil \log \log n \rceil} - 3.$$

Consider the sequence of configurations $\mathcal{C}_0, \ldots, \mathcal{C}_r$ from Corollary 3.13 for k equal to $\lceil \log \log n \rceil$. The starting point for a space-efficient algorithm comes directly from the definition of this sequence. More precisely, we define functions $\texttt{Active}(l, v)$, $\texttt{Inactive}(l, v)$, $\texttt{Done}(l, v)$, $\texttt{Degree}(l, v)$, $\texttt{Neighbor}(l, v, i)$, $\texttt{BackLabel}(l, v, i)$, $\texttt{Rep}(l, v)$, and $\texttt{Hook}(l, v)$, where $0 \leq l \leq r$, v is a vertex, and i is the label of an edge incident to v, which return the corresponding component of $\mathcal{C}_l$. Namely, $\texttt{Active}(l, v)$, $\texttt{Inactive}(l, v)$, and $\texttt{Done}(l, v)$ check, correspondingly, whether v is active, inactive, or done in $\mathcal{C}_l$, i.e., whether it belongs to A_l, I_l, or D_l; $\texttt{Degree}(l, v)$ returns $\delta_l(v)$, the degree of v in G_l; $\texttt{Neighbor}(l, v, i)$ returns $\eta_l(v, i)$, the ith neighbor of v in G_l; $\texttt{BackLabel}(l, v, i)$ returns $\beta_l(v, i)$, the back-label of the ith edge of v; $\texttt{Hook}(l, v)$ returns $H_l(v)$, the label of the hooking edge of v in $\mathcal{C}_l$; $\texttt{Rep}(l, v)$ returns $R_l(v)$, the immediate representative of v in $\mathcal{C}_l$.

Call the parameter l of the above functions the *level of recursion*. Thus the levels of recursion of our algorithm correspond to the elements of the sequence $\mathcal{C}_0, \ldots, \mathcal{C}_r$. For $l = 0$, the bottom of the recursion, all of these functions just use the input graph G (see Corollary 3.13). According to the definitions in section 3.3, $\mathcal{C}_{l+1}$ is derived from $\mathcal{C}_l$, and thus the outputs of the functions mentioned in the previous paragraph for level $l + 1$ is determined recursively from their output for level l. Using these functions, we apply Corollary 3.13 to solve undirected st-connectivity.

If $Op_l = \mathit{Contract}$, then $\mathcal{C}_l$ is obtained from $\mathcal{C}_{l-1}$ by contracting some of its hooking trees as defined in section 3.2.3. In this case for a hooking tree T in $\mathcal{C}_{l-1}$, we must determine $\deg(T)$ and be able to enumerate the vertices of T as they are visited by the exploration walk on T starting from $(v, 1)$ for $v \in V_T$. For these purposes, we define $\texttt{TreeSize}(l, v)$ and $\texttt{TreeWalk}(l, v, i)$. Let T be the hooking tree in $\mathcal{C}_{l-1}$ containing v. If s is the size of T, $\texttt{TreeSize}(l, v)$ returns $2(s-1)$. Notice that $2(s-1)$ is the length of the exploration walk on T given in Proposition 3.4. $\texttt{TreeWalk}(l, v, i)$ returns $\Gamma_{T,i}(v, 1)$.

Going over the details of the definitions of each of the functions mentioned above, it is not hard to see that each level of recursion can be implemented in $O(\log n)$ space, and since we have $r = O(\log n)$ levels of recursion, this results in an $O(\log^2 n)$ algorithm for USTCONN. We omit those details, because they are rather immediate and are superceded by the discussion in the next section and the precise definitions in section 4.4.

4.2. The $O(\log n \log\log n)$ space algorithm. The $O(\log n)$ space per level of recursion in the algorithm outlined in the previous section comes mainly from having to store vertices in the local variables of the functions, since each vertex takes $\Theta(\log n)$ space. To see more precisely what is going on, consider the following. The definition of $H_l(v)$ contains a comparison of the ith neighbor of v in $\mathcal{C}_{l-1}$ with its jth neighbor; i.e., the definition of $\texttt{Hook}(l, v)$ contains a comparison $\texttt{Neighbor}(l-1, v, i) = \texttt{Neighbor}(l-1, v, j)$. Let us say that v is passed to `Hook` through a global variable. Obviously, this global variable must be stored locally before the execution of such comparison, because otherwise its value might be overwritten during the two calls to `Neighbor`. To take care of this bottleneck, we define the functions so that they never store a vertex in their local variables.

The first step towards such definitions is to remove the vertex v from the argument list of the functions. Instead of this argument, we maintain one current vertex in a global variable, and all functions return information about this vertex. A function which otherwise must return a vertex is defined so that after its execution the current vertex is its result (in this case we say that the function moves the current vertex). It is a responsibility of the calling function to keep enough information locally to restore the original current vertex if it needs to. Denote the current vertex with cv.

To implement this, first we change the definitions of some of our functions. Instead of $\texttt{Neighbor}(l, v, i)$, we have $\texttt{Neighbor}(l, i)$, which moves the current vertex to $\eta_l(cv, i)$, its ith neighbor in $\mathcal{C}_l$. Let T be the hooking tree of cv in $\mathcal{C}_{l-1}$. Instead of $\texttt{TreeWalk}(l, v, i)$ we have $\texttt{TreeForward}(l, i)$, which returns j and moves the current vertex to u, where $(u, j) = \Gamma_{T,i}(cv, 1)$. Similarly, we have $\texttt{TreeBack}(l, i, j)$, which moves the current vertex to the vertex of $\Gamma'_{T,i}(cv, j)$.

The most important part of our idea to avoid storing vertices locally is to be able to move the current vertex temporarily, perform something at the new current vertex, and then return to the original current vertex. For this, define $\texttt{Move}(l, i)$ to return $\beta_{l-1}(cv, i)$, i.e., $\texttt{BackLabel}(l-1, i)$, and move the current vertex to $\eta_{l-1}(cv, i)$,

i.e., a call to `Neighbor`$(l-1,i)$. Call `Move`(l,i) and `TreeForward`(l,i) *forward moves*. For a forward move M, let `Reverse`(M,j) be its reverse, i.e., it is correspondingly `Move`(l,j) or `TreeBack`(l,i,j), where j is the result of M. We use the reversibility of exploration walks here, so that `TreeBack` reverts `TreeForward`. Finally, we have a "dummy" forward move `Current` which does not change the current vertex and whose purpose is to address the current vertex. The reverse of `Current` is `Current` again.

We use forward moves to change the current vertex and their reverses to restore it. Call a sequence of forward moves *path description relative to the current vertex*. If `P` is a path description relative to the current vertex and `B` is some instruction(s), then define **after** `P` **do** `B` to change the current vertex according to `P`, perform `B`, and then use the reverses of the moves in `P` to restore the current vertex.

A simple example of the use of **after** is the comparison operator `=`, which compares two vertices given their path descriptions relative to the current vertex and returns true iff they are the same. Using **after**, we can move to the first vertex and store it in a local variable, and then go to the second vertex and compare the two. This takes $\Theta(\log n)$ space. Instead of this, going back and forth between the two vertices, using the reversibility of the moves along the edges and the exploration walks on the trees, we perform the comparison bit by bit. Aside from the information stored for the ways back, this takes only the $\Theta(\log\log n)$ space necessary to store the index of a bit. This way the bottleneck of $\Omega(\log n)$ space is reduced to $\Omega(\log\log n)$.

For example, the `=` operator is used in the definition of `TreeSize`(l), which determines the size of the hooking tree T of cv in $\mathcal{C}_{l-1}$, in the following way. Starting at cv and using Proposition 3.4, we can make steps from the exploration walk on T until we go back to cv sufficiently many times. For this, we can store cv, so that we can compare it with each new vertex of the walk. This takes $\Theta(\log n)$ space, independent of the size of T. Instead, we incrementally find i with properties as in Proposition 3.4, where to check if the ith vertex of the walk is equal to cv we keep the current vertex at cv and use the `=` operator, as defined above, to compare it to the vertex with path description `TreeForward`(l,i). More precisely, we use the comparison `Current=TreeForward`(l,i) to check whether the ith vertex of the walk is equal to cv, where the semantics of the comparison in this case is explained in the previous paragraph. Thus, if $\text{size}(T) \le s$, then we can find its size in space $O(\log s + \log\log n)$ (the $\log\log n$ appears because of counters used during comparison of vertices). Alternatively, if $\text{size}(T) > s$, we can still use only $O(\log s + \log\log n)$ space to learn this without actually computing $\text{size}(T)$, because it is enough to stop the exploration walk on T as soon as we learn that $\text{size}(T) > s$.

The second part of our idea to reduce the space of the algorithm is to have an upper bound $v(l)$ on the values which variables can take at level l; i.e., during the execution of all functions at level l, the values of their local variables are at most $v(l)$. We set

$$v(l) = 2 \cdot \text{dexp}(Arg_l + 2).$$

Call a number x *valid* for level l if it is at most $v(l)$. A vertex v is *valid* for level l if its degree $\delta_{l-1}(v)$ is valid for level l.

Using the concept of a current vertex, we can eliminate the need to store a vertex in a local variable, and thus our local variables contain essentially only degrees of vertices, labels and back-labels of edges, and lengths of exploration walks on hooking trees. For example, the information stored for reversing a forward move is a back-label (for `Move`) or a tree-edge label (for `TreeForward`). We still have to make sure

that every time we store a value in a local variable it is valid. For this the following observation is helpful.

OBSERVATION 4.1.

(i) *The labels of the edges incident to vertex v valid for level l are valid for level l. This is not necessarily true for their back-labels.*

(ii) *All vertices which are active in $\mathcal{C}_{l-1}$ are valid for level l.*

(iii) *If $Op_l = Contract$, then if a hooking tree T in $\mathcal{C}_{l-1}$ has degree at most $\operatorname{dexp}(Arg_l + 1)$, then all of its vertices are valid for level l.*

The first item of the observation is trivial, the second follows from Theorem 3.12, because all $\mathcal{C}_l$ are nice, and the third follows because $\operatorname{dexp}(Arg_l + 1) < v(l)$.

Our goal has become to prove the following lemma. The functions mentioned in the lemma are the ones outlined previously in this section. Their precise definitions are given in section 4.4. Let T be a hooking tree in $\mathcal{C}_{l-1}$. T is *contractable for level l* if $\deg(T) \le \operatorname{dexp}(Arg_l + 1)$. Let (v, i) be an edge of T and $u = \eta_T(v, i)$. A move along (v, i) is *possible for level l* if u is valid for level l.

LEMMA 4.2. *Let T be the hooking tree in $\mathcal{C}_{l-1}$ of the current vertex cv.*

(i) `Active`(l), `Inactive`(l), `Done`(l), `Hook`(l), *and* `Degree`(l) *correctly return the value of the corresponding component of $\mathcal{C}_l$ for cv.*

(ii) *If T is uncontractable for level l or $cv \in D_{l-1}$, then* `Root`(l) *returns* 0*; otherwise, it returns the index of the first occurrence of* $\operatorname{root}(T)$ *in the exploration walk on T starting from $(cv, 1)$.*
`TreeSize`(l) *returns* $2(\texttt{size}(T) - 1)$*, if T is contractable for level l, and* `null` *otherwise.*
Assume that cv is valid for level l. If all moves of $\Gamma_{T,\le i}(cv, 1)$ are possible for level l and it ends in (v, j), then `TreeForward`(l, i) *moves the current vertex to v and returns j. If all moves of $\Gamma'_{T,\le i}(cv, j)$ are possible for level l and it ends in v, then* `TreeBack`(l, i, j) *moves the current vertex to v.*

(iii) *Let $(v, j) = \mu_l(cv, i)$.*
`Neighbor`(l, i) *moves the current vertex to v.*
`BackLabel`(l, i) *returns j, if j is valid for the level at which* `BackLabel`(l, i) *was called (see the discussion on returning values in section* 4.4.2*), and* `null` *otherwise.*

All local variables are valid.

The proof of the lemma is done by induction on the level of recursion. For this, we need the correctness of the functions which a given function calls. Sometimes we have to use correctness for the same level of recursion, but this does not result in a circular reasoning because for any two functions `F` and `G`, there are no chains of function calls within the same level of recursion both from `F` to `G` and from `G` to `F`.

The correctness essentially follows from the correctness of the CL algorithm (Corollary 3.13). It can be easily seen that the introduction of one global current vertex and always returning information about this vertex maintains the faithfulness of our implementation to the CL algorithm—the current vertex is an implicit argument to all functions describing a configuration, and calling it "current" just facilitates our intuition about how the algorithm proceeds.

The only real deviation from the definitions given in section 3.2 is that we have an upper bound on the numerical values which can be stored at a level, and so we might be unable to process the result of a function if it is invalid for the current level of recursion. Actually, as can be seen from Lemma 4.2, some functions are specified to return `null` if their result is invalid for the level requesting it. By Observation 4.1, the only information we can derive from an invalid or `null` result is that either the

current vertex is invalid (if `Degree`(l) is invalid), a neighbor of the current vertex is invalid (if `BackLabel`(l, i) is `null`), or the current vertex is part of an uncontractable tree (if `TreeSize` is `null`). This information is enough to define the functions as in Lemma 4.2. First, we never move to a vertex from which we cannot return, i.e., along an edge with an invalid back-label, so that we never have to store an invalid back-label locally. Second, during contraction, vertices which are either invalid or part of an uncontractable tree will become inactive and thus, by definition, will inherit their properties, e.g., degree and hooking edge, from the previous level. In a hooking operation (section 3.2.2), we do not need to look up the degree of an invalid (and even inactive) vertex. In a contraction operation (section 3.2.3), we can stop the exploration walk on a tree as soon as the walk runs into an invalid vertex, because then the tree is clearly uncontractable.

To ensure that all local variables are valid for the current level of recursion, we use Observation 4.1 in the following way. First, notice that since the value returned from a function resides in a global variable we can check whether it is valid by simply inspecting this global variable. Furthermore, some functions (e.g., `TreeSize` and `BackLabel`) return `null` if their result is invalid for the current level of recursion. In any case, we can easily learn when the return value of a function is invalid without having to store it locally. According to Observation 4.1(i), if the result of `BackLabel` is `null`, then the corresponding neighbor is invalid and we never move the current vertex along such edges. Also, Observation 4.1(ii) ensures that we can always process locally active vertices. Finally, according to Observation 4.1(iii), if the result of `TreeSize` is `null`, then the hooking tree of the current vertex is uncontractable.

4.3. Solving undirected st-connectivity. Using Lemma 4.2, we can prove the main theorem of this paper.

THEOREM 4.3. *USTCONN on a graph with n vertices can be solved in space $O(\log n \log\log n)$.*

Let $m = \lceil \log\log n \rceil$. Recall that $r = 5 \cdot 2^m - 3 = O(\log n)$. By Corollary 3.13, s and t are connected iff $\mathrm{rep}_{R_r}(s) = \mathrm{rep}_{R_r}(t)$. Thus to solve USTCONN it is enough to define a function `MoveToRep`(l) which moves the current vertex cv to its representative $\mathrm{rep}_{R_l}(cv)$ in $\mathcal{C}_l$. Then, to check whether s and t are connected, we call `MoveToRep`(r) twice, once from s and once from t, and compare the two resulting final current vertices (see the defintion of `Connected` in section 4.4.10). `MoveToRep` is defined by first calling itself recursively. This moves the current vertex cv to its representative $v = \mathrm{rep}_{R_{l-1}}(cv)$ in $\mathcal{C}_{l-1}$. Then, if $Op_l = Contract$ and the hooking tree T in $\mathcal{C}_l$ of the new current vertex v is contractable for level l, `MoveToRep` moves v to the root of T. We use here that the root of T will be the immediate representative of v after contracting T. See section 4.4.10 for a precise definition of `MoveToRep`.

The space complexity of the algorithm is dominated by the space taken by the execution stack. From Lemma 4.2, it follows that each local variable at level l is at most $v(l)$. Since there are a constant number of local variables per function and the length of every chain of function calls within the same level of recursion is bounded by a constant, the space taken by level l is $O(\log v(l) + m)$ (the additional $O(m)$ space appears because of the counters used during comparison of vertices). Since $\log v(l) = 4 \cdot 2^{Arg_l} + 1$, $1 \le l \le r$, and $r \cdot m = O(\log n \log\log n)$, we have to prove that

$$\sum_{l=1}^{r} 2^{Arg_l} = O(\log n \log\log n).$$

Consider the recurrence given by

$$S(k) = \begin{cases} 3, & k = 0, \\ 2S(k-1) + 2 \cdot 2^k + 2^{k+1}, & k > 0. \end{cases}$$

From the recursive definition of S_m given in section 3.3, it follows that the left-hand side of the equation which we want to prove is exactly $S(m)$. Finally, it is not hard to prove by induction on k that $S(k) = 2^k(4k + 3)$. Hence $S(m) = O(\log n \log\log n)$.

4.4. Pseudocode. The language of the pseudocode is self-explanatory. The difference between a function and a procedure, marked with **function** and **procedure** correspondingly, is that the first returns a value and the second does not. In the following we will refer to both as functions. Operators, marked with **operator**, are functions for which a more traditional in-fix position of their usage is emphasized. The semantics of statements, marked with **statement**, is closer to what is traditionally known as macroses than to procedures and functions; namely, they do not allocate a new stack frame. Blocks are marked by indentation, **break** exits the closest surrounding loop, **continue** continues with the next cycle of the execution of the closest surrounding loop, and **return** exits the current function, returning a value, if necessary. We use a fixed width font to denote the names of functions and variables from the pseudocode, e.g., `F` and `i`, and a roman italics font for mathematical functions and variables, e.g., f and i.

4.4.1. Execution of the pseudocode. The variable usage and execution of the pseudocode is standard. During its execution, a function can use only its own local variables and the global variables. For the execution of the functions, we use a stack which contains the local variables of the functions executed at the moment. The part of the stack devoted to the execution of a function is called the *stack frame of the function*. Statements use the stack frame of their surrounding function. The top of this stack contains the stack frame of the function being executed at the moment. When the current function calls another function, first a new stack frame is allocated on the top of the stack and then the new function is executed. Part of the stack frame of a function contains information about the address (the place in the program) from which the function was called. Once the current function is finished, its stack frame is removed from the top of the stack, and the execution resumes from the address from which the current function was called.

In addition to functions which are executed using the stack, we have *global functions*, which do not use the stack for their execution. Such functions use only global variables for their execution; i.e., their "local" variables are in fact global variables which are visible only from the particular global function. Since in what follows we are concerned only with the contents of the stack, we will concentrate on functions which use the stack.

The execution of the pseudocode proceeds in the following way. Every function is executed at some level of recursion. Every function has an argument l, which contains the current level of recursion. During its execution, a function can either call other functions on the same level of recursion, i.e., calls like $\mathtt{F}(l, \ldots)$, or make calls to the previous level of recursion, i.e., calls like $\mathtt{F}(l-1, \ldots)$. There is a constant c such that the length of a chain of function calls within the same level of recursion is at most c. The stack frames of functions executed in the same level of recursion are consecutive on the stack. We call such a sequence of stack frames the *stack frame for the level*. From the fact that any stack frame for a level contains at most c stack frames for functions and that each function uses a constant number of variables, it follows that there is a constant d such that the stack frame of every level contains a total of at

most d variables.

The local and global variables contain only numerical and boolean values, and a special value `null`, different from any other value. For every level of recursion l, we have an upper bound $v(l)$, computed by a global function, on the numerical values which are valid for this level; i.e., all numerical values at level l are at most $v(l)$. Boolean values and `null` are always valid. The counters used for comparison of vertices are at most $\lceil \log n \rceil$.

4.4.2. Passing arguments and returning values. We have two methods of passing arguments to a function and returning a value. The first is through global variables, and the second is through global arrays which contain an entry for every level of recursion.

The method which uses global variables is straightforward. We have global variables which are set to the arguments of the function before it is called. We denote the global variables for the arguments of a function `F` with `arg1F`, `arg2F`, and so on. During its execution, a function can look up its arguments from these global variables. It might decide to store them as local variables, but this might not always be possible, because the arguments might be invalid for the current level of recursion.

To return a value a function sets a global variable. After a return from a call of a function, the calling function decides whether it wants to store the returned value locally. We have a special assignment operator, `::=`, which assigns the return value r of a function returning through a global variable to a local variable in the following way: if r is valid for the current level of recursion, the result of the assignment is r; otherwise, it is `null`.

The method which uses arrays is more subtle. Let `F` be a function which uses this method of passing arguments and returning values. This method can be used only if all calls to `F` at the current level of recursion l are recursive; i.e., all calls to `F` look like `F`$(l-1,\ldots)$. We have two global arrays—one, `argF`, for passing arguments to `F` and one, `retF`, for returning a value from it. Those arrays contain exactly one entry for every possible level of recursion, and each entry could be marked. Also each entry holds values which are valid for the corresponding level of recursion, so that the space taken by each such array is the same as the space taken by the execution stack.

Let `H` call `F`. If `H` uses the value returned by `F`, then before the call to `F` the entry of `retF` for the current level of recursion is marked; otherwise, it is left unmarked. When `F` produces a result at level l, it finds the smallest index $i > l$ such that the ith entry of `retF` is marked and tries to store its result there. If the value produced is too large for the corresponding entry, `F` writes `null`. After the call to `F` returns, `H` unmarks the entry of `retF` for the current level of recursion. The only time when `H` does not use the value returned by `F` is when `H` is actually `F` and the call to `F` is a recursive call whose result is passed back without modification, e.g., a call like **return** `F`$(l-1,\ldots)$ in the definition of `F`.

Similarly, if `H` provides arguments to `F`, then before the call to `F`, `H` marks the entry of `argF` for the current level of recursion and provides values for it. When `F` wants to access its arguments at level l, it finds the smallest index $i > l$ such that the ith entry of `argF` is marked and uses the values stored in the entry as its arguments. After the call to `F` returns, `H` unmarks the entry of `argF` for the current level of recursion. Again we have that the only time when `H` does not provide arguments to `F` is when `H` is `F` and the call to `F` is a recursive call whose arguments are the same as for the current call to `F`, e.g., a call like `F`$(l-1,y)$ in the definition of `F`(l,x) at a place where it is always true that $x = y$.

Since this method is used only if all calls to `F` are recursive, there is no danger of overwriting `F`'s arguments or returned value. For passing arguments, this method helps when we need to store an argument only at the level which generates it, where it is valid, but still allow for lower levels of recursion to access it, where it might be invalid. For returning values, this method helps when we want to return a value exactly at the level which requested it originally. This method allows for nested calls from different levels of recursion to the same function, which the ordinary method of passing variables through global variables does not always allow.

The restriction we have on the space of a level is the reason why we chose those methods of passing arguments and returning value. The method using arrays is more unusual, and it is used only in two functions, `BackLabel` and `BackLabelAux`. The reason why we introduced it is explained in the notes for those functions.

In the code, we use $\texttt{F}(l, x_1, \ldots, x_k)$ to call a function `F` at level l with arguments $x_1, \ldots, x_k$. If a function `F` takes arguments, but is called without ones, it uses the values currently located in the global variables (or the arrays) for `F`.

4.4.3. Translation of the pseudocode to a Turing machine. Let us address now the issue of translating the pseudocode to a Turing machine with a binary alphabet. Most of the details, like doing arithmetic and performing conditionals, are rather straightforward, and so we skip them and concentrate only on variable usage. Numerical values are represented in binary. To represent the `null` value, we use one additional bit to designate whether or not the value is `null`. We have a separate tape for each global variable (there are only a constant number of them). The space taken by each global variable is $O(\log n)$.

There is a tape assigned for the stack, and the head of this tape is positioned at the stack frame of the current function, i.e., at the top of the stack. The stack frame of a function contains the state to which the Turing machine must return after the execution of the function (this takes a constant number of bits depending only on the Turing machine) and the values of the local variables of the function.

The space taken by each local variable depends on the level of recursion at which the stack frame occurs. Since at a level l the value of every valid local variable is at most $v(l)$, the space $s(l)$ taken by such a variable at level l is $O(\log v(l))$. This space is known to the current function, because it can be computed by a global function from the current level of recursion. So to use the ith local variable the current function must move the head of the stack tape to the place where the variable is located. This place can be computed by the current function from i and $s(l)$. As discussed earlier, there is a constant d such that the stack frame of the lth level of recursion contains at most d variables. Thus the space taken by the stack frame of level l is $O(\log v(l) + \log\log n)$. The $O(\log\log n)$ appears because of comparison of vertices, as explained in section 4.2.

After the execution of the current function, the state of the Turing machine is restored to the value stored on the stack, and the head of the stack tape is moved to the stack frame of the caller.

4.4.4. Preliminaries. To simplify the exposition of the algorithm, we remove the level of recursion from the argument lists of the functions. Instead we have one global variable, `level`, which contains the current level of recursion. `level` is set to $5 \cdot 2^{\lceil \log\log n \rceil} - 3$ initially. Let `F` be a function which calls a function `G` on the previous level of recursion. This task is performed by `Prev`; namely, `Prev`(`G`(...)) passes arguments to `G`, decreases the current level of recursion, calls `G`, and upon return from `G` increases the current level of recursion. This is the only way that the

current level of recursion is changed—all functions can look up the value of `level`, but none of them changes it. We denote the current level of recursion with cl.

The global variable `currvertex` contains the current vertex cv.

In the following, T denotes the hooking tree of the current vertex in $\mathcal{C}_{cl-1}$. Unless specified otherwise, the exploration walk refers to the exploration walk on T starting from $(cv, 1)$. A hooking tree in $\mathcal{C}_{cl-1}$ is called contractable if its degree is at most $\mathrm{dexp}(Arg_{cl} + 1)$. A value is valid if it is at most $v(cl)$.

We use the following observation. It follows from Observation 4.1 and the correctness of the functions mentioned in it. `MoveValid`(i) checks whether the back-label $\beta_{cl-1}(cv, i)$ of the ith edge of cv in $\mathcal{C}_{cl-1}$ is valid, i.e., whether $\beta_{cl-1}(cv, i) \leq v(cl)$.

OBSERVATION 4.4.

(i) *If* `TreeSize` $\neq$ `null`*, then all moves of* `TreeForward`(i) *are possible for level* cl*.*

(ii) *If* `MoveValid`(i) *is true, then the result of* `Move`(i) *is not* `null` *and valid for level* cl*; otherwise,* $\beta_{cl-1}(cv, i)$ *is invalid for level* cl*.*

By this observation, `TreeSize` and `MoveValid` serve as "safeguard" checks for forward moves. Thus, before making a forward move to change the current vertex, if we want to be able to return, e.g., in an **after** statement, we always first make sure that the forward move returns a valid result. In this case we say that the forward move is valid.

All functions, except `BackLabel` and `BackLabelAux`, take arguments and return values through global variables.

Every function is preceded by paragraphs which give its specification. Also notes are made on the definition and correctness of the function, and on the validity of its local variables. In the notes we use interchangeably the name of a local variable, given in fixed font, and its value at a particular point of the execution of the function. To facilitate the reading of the pseudocode, we have annotated the meaning of the local variables of the nonglobal functions.

4.4.5. Important functions. Global function `ArgOp`. **Input** l. **Output** $2 \cdot Arg_l + \varepsilon_l$, where ε_l is 0, if $Op_l = Hook$, and 1 otherwise. **Assumes** $1 \leq l \leq 5 \cdot 2^{\lceil \log\log n \rceil} - 3$.

```
global function ArgOp
    l := argArgOp;
    k := ⌈log log n⌉;
    while true
        if k = 0 then return 2·(l − 1) + (l − 1);
        else
            if l ≤ 2 then return 2·k + (l − 1);
            else
                if l = 5·2^k − 3 then return 2·(k + 1) + 1;
                else
                    if l ≥ 5·2^(k−1) then l := l − 5·2^(k−1) + 1;
                    else l := l − 2;
                    k := k − 1;
```

Global functions `Valid`, `ContractDegree`, and `Contraction`. **Input** None. **Output** Correspondingly $2 \cdot \mathrm{dexp}(Arg_{cl} + 2)$, $\mathrm{dexp}(Arg_{cl} + 1)$, and whether $Op_{cl} = Contract$.

```
global function Valid
    return 2·dexp(2 + ArgOp(level) div 2);
```

```
global function ContractDegree
    return dexp(1 + ArgOp(level) div 2);

global function Contraction
    return ArgOp(level) mod 2 = 1;
```

Statement "**after** $M_1, M_2, \ldots, M_k$ **do** B." **Input** $M_1, \ldots, M_k$—path description relative to the current vertex, B some instruction(s). **Output** Moves the current vertex according to the forward moves in $M_1, \ldots, M_k$, executes B, and finally restores the original current vertex. **Assumes** All forward moves are valid. **Local variables** $l_1, \ldots, l_k$ (results of the forward moves).

```
statement after M1, M2, ..., Mk do B
    l1 := M1; l2 := M2; ...; lk := Mk;
    B;
    Reverse(Mk, lk); ...; Reverse(M2, l2); Reverse(M1, l1);
```

Operators <, =, and $<_d$. **Input** P_1 and P_2—path descriptions relative to the current vertex. **Output** Let v_1 and v_2 be the end vertex of P_1 and P_2, correspondingly. The three operators check correspondingly, whether $v_1 < v_2$, $v_1 = v_2$, and $v_1 <_d v_2$. **Assumes** All forward moves are valid. $<_d$ assumes also that v_1 and v_2 are valid. **Local variables** i (counter for indices of bits), b_1 (ith bit of v_1), and b_2 (ith bit of v_2) for < and =. d_1 (degree of v_1) and d_2 (degree of v_2) for $<_d$. **Notes** Bit(s, t) returns the sth most significant bit of t. b_1 and b_2 are single bits.

```
operator P1 < P2
    for i := 1 to ⌈log n⌉ do
        after P1 do b1 := Bit(i, currvertex);
        after P2 do b2 := Bit(i, currvertex);
        if b1 ≠ b2 then return b1 < b2;
    return false;
operator P1 = P2
    for i := 1 to ⌈log n⌉ do
        after P1 do b1 := Bit(i, currvertex);
        after P2 do b2 := Bit(i, currvertex);
        if b1 ≠ b2 then return false;
    return true;
operator P1 <d P2
    after P1 do d1 ::= Prev(Degree);
    after P2 do d2 ::= Prev(Degree);
    return (d1 < d2) or (d1 = d2 and P1 < P2);
```

4.4.6. State functions. Functions Done, Active, and Inactive. **Input** None. **Output** Correspondingly whether $cv \in D_{cl}$, $cv \in A_{cl}$, and $cv \in I_{cl}$. **Assumes** None. **Local variables** None.

```
function Done
    return (level = 0 and Degree = 0) or
        (level > 0 and not Contraction and Prev(Done)) or
        (level > 0 and Contraction and
            (Prev(Done) or Root ≠ 0 or Degree = 0));
function Active
    return (level = 0 and Degree ≠ 0) or
        (level > 0 and not Contraction and Prev(Active)) or
        (level > 0 and Contraction and
```

```
            not Done and TreeSize ≠ null);
function Inactive
    return (level > 0 and not Contraction and Prev(Inactive)) or
        (level > 0 and Contraction and
            not Done and TreeSize = null);
```

4.4.7. Hooking. Function Hook. **Input** None. **Output** $H_{cl}(cv)$. **Assumes** None. **Local variables** $\mathtt{d}_1$ (degree of cv), i (counter for neighbors of cv), $\mathtt{d}_2$ (degree of the ith neighbor of cv), j (counter for neighbors of the ith neighbor of cv), and m (current candidate neighbor for hooking). **Notes** Hook is defined as given in section 3.2.2. In line 9 we use Observation 4.1(i) to deduce that $\eta_{cl-1}(cv, \mathtt{i}) \in I_{cl-1}$. At line 5 $cv \in A_{cl-1}$, and hence $\mathtt{d}_1$ is valid and non-null. So i and m are also valid. At line 12 we have that $\eta_{cl-1}(cv, \mathtt{i}), \eta_{cl-1}(cv, \mathtt{m}) \in A_{cl-1}$. At line 14 all neighbors of cv are in A_{cl-1}. Hence $\mathtt{d}_2$ and j are valid.

```
   function Hook
 1     if level = 0 then return 0;
 2     if not Contraction and Prev(Inactive) then return Prev(Hook);
 3     if Contraction and Inactive then return Prev(Hook);
 4     if Prev(Done) or Contraction then return 0;

 5     d1 ::= Prev(Degree);
 6     m := 0;
 7     for i := 1 to d1 do
 8         // if the ith neighbor is inactive, then hook to it
 9         if not MoveValid(i) then return i;
10         after Move(i) do fl := Prev(Inactive);
11         if fl then return i;

           // otherwise, check if it is bigger than the current biggest
           // active neighbor
12         if Move(m) <d Move(i) then m := i;
13     if m > 0 then return m;

14     for i := 1 to d1 do
15         after Move(i) do d2 ::= Prev(Degree);
16         for j := 1 to d2 do
               // if the jth neighbor of the ith neighbor
               // is inactive, then hook to the ith neighbor
17             after Move(i) do fl := MoveValid(j);
18             if not fl then return i;
19             after Move(i), Move(j) do fl := Prev(Inactive);
20             if fl then return i;

               // otherwise, hook to the ith neighbor
               // if its jth neighbor is bigger than currvertex
21             if Current <d (Move(i), Move(j)) then m := i;
22     return m;
```

Function IsHooked. **Input** i. **Output** Let $(v, j) = \mu_{cl-1}(cv, i)$ and $h = H_{cl-1}(v)$. IsHooked is true iff $h = j$. **Assumes** $cl \geq 1$, $0 \leq i \leq \delta_{cl-1}(cv)$, and cv is valid or $\delta_{cl-1}(cv) \leq \delta_{cl-1}(v)$. **Local variables** i, j, and h (contain their equiv-

alents from the definition of the output). **Notes** `i` is valid because of line 2, `j` is valid because of the assignment in line 5, and `h` is valid because of the assignment in line 9.

```
   function IsHooked
 1     if argIsHooked = 0 then return Prev(Hook) = 0;
 2     if argIsHooked > Valid then return Prev(IsHooked);
 3     i := argIsHooked;
 4     if Prev(Degree) > Valid then return Prev(IsHooked(i));

 5     j := Prev(BackLabel(i));
 6     if j = null then
 7         if Prev(Active) then return false;
 8         return Prev(IsHooked(i));
 9     after Move(i) do h ::= Prev(Hook);
10     return h = j;
```

Correctness We will prove the correctness of `IsHooked` by induction on cl. Notice that for $cl = 1$, the checks in lines 2, 4, and 6 all fail because at level 1 all vertices are valid, and we compare `h` and `j` in line 10.

First, consider the case when $cv \in A_{cl-1}$. In this case cv is valid by Observation 4.1(ii). If `j` is invalid, then $v \in I_{cl-1}$, and it is not hooked to cv (otherwise, $cv \in I_{cl-1}$ by Definition 3.5). We catch this in line 7. If `j` is valid, then in line 10 we check whether it is equal to `h`.

Now let $cv \in I_{cl-1}$ and $v \in A_{cl-1}$. Since v is valid, cv is valid also, because this follows from v being valid, and cv being valid or $\delta_{cl-1}(cv) \le \delta_{cl-1}(v)$. So `i`, `j`, and `h` are valid, and we can compare `h` and `j` in line 10.

Assume now that $cv, v \in I_{cl-1}$. In this case the only way `IsHooked` returns an answer without calling recursively is in line 10; then `j` is valid, and we have compared it to `h`. Notice now that if `IsHooked` calls itself recursively, then $\delta_{cl-1}(cv) \le \delta_{cl-1}(v)$. This is true for the calls in lines 2 and 4, because then cv is invalid. For the call in line 8, this is true, because cv is valid and v is invalid. Since $cv, v \in I_{cl-1}$, we have that $cl \ge 2$, $\delta_{cl-2}(cv) = \delta_{cl-1}(cv)$, $\delta_{cl-2}(v) = \delta_{cl-1}(v)$, $(v, \mathtt{j}) = \mu_{cl-2}(cv, \mathtt{i})$, and $\mathtt{h} = H_{cl-2}(v)$. Thus the correctness in this case is ensured by the inductive hypothesis.

4.4.8. Exploration walk. The functions in this section come from the definition of a hooking forest of a configuration given in section 3.2.1. Throughout, it is assumed that $cl \ge 1$ and $Op_{cl} = Contract$.

Function `TreeDegree`. **Input** None. **Output** If cv is valid, `TreeDegree` returns $\delta_T(cv)$; otherwise, it returns `null`. **Assumes** None. **Local variables** `i` (counter for steps of the exploration walk), `d` (degree of cv), and `td` (output). **Notes** In line 5 we use that cv is valid to apply the correctness of `IsHooked`. `d` is valid because of the assignment in line 1, and `i` and `td` are valid because at line 3 cv is valid.

```
   function TreeDegree
       // if currvertex is invalid, return null
 1     d ::= Prev(Degree);
 2     if d = null then return null;

 3     td := 0;
       // count the number of neighbors which are hooked to currvertex
 4     for i := 1 to d do
 5         if IsHooked(i) then td := td + 1;
       // add 1 if currvertex did not hook to itself
```

```
6       if Prev(Hook) ≠ 0 then td := td + 1;
7       return td
```

Function `TreeMove`. **Input** i. **Output** Let $(v, r) = \mu_T(cv, i)$. If a move along (cv, i) is possible, i.e., v is valid, `TreeMove` returns r and moves the current vertex to v; otherwise, it does not change the current vertex and returns `null`. **Assumes** $0 \le i \le \delta_T(cv)$, cv is valid. **Local variables** `i` (input), `j` (graph-edge label of the `i`th T-edge of cv), `k` (counter for the neighbors of v), `l` (counter for vertices which are hooked to cv), `d` (degree of cv), $\mathtt{d}_1$ (degree of v), and `r` (back-label of the `i`th T-edge of cv).

Notes Lines 2–7 convert from the label `i` of a T-edge $e = (v, \mathtt{i})$ to a label `j` of an edge in the graph. In line 5 we use that cv is valid to apply `IsHooked`. Lines 8–10 handle the case when v is invalid. Lines 11–26 compute the tree back-label `r` of e and move the current vertex to v. Lines 11–13 handle the case when v is hooked to the current vertex. Lines 14–26 handle the case when e is the hooking edge of the current vertex. In lines 19–21 we use that if the `k`th neighbor of v is invalid, then it is not cv, because cv is valid.

`i`, `j`, `l`, and `d` are valid, because cv is valid (the assignments in lines 3 and 7 are non-`null`). $\mathtt{d}_1$ is valid because of the assignment in line 9. `k` and `r` are valid because at line 14 v is valid.

```
   function TreeMove
        i := argTreeMove;
1       if i = 0 then return 0;

        // convert from T-edge label to graph-edge label
2       l := i;
3       d ::= Prev(Degree);
4       for j := 1 to d do
5           if IsHooked(j) then l := l - 1;
6           if l = 0 then break;
7       if j > d then j ::= Prev(Hook);

        // if the new vertex is invalid, return null
8       if not MoveValid(j) then return null;
9       after Move(j) do d1 ::= Prev(Degree);
10      if d1 = null then return null;

11      if i < TreeDegree or (i = TreeDegree and Prev(Hook) = 0) then
            // e goes to a neighbor which is hooked to currvertex
12          Move(j);
13          return TreeDegree;

        // e is the hooking edge of currvertex

        // compute the tree back-label
14      r := 1;
15      for k := 1 to d1 do
16          after Move(j) do fl := IsHooked(k);
17          if not fl then continue;
            // enumerate all the edges with which neighbors
            // of the new vertex hooked to it
```

```
18          after Move(j) do fl := MoveValid(k);
19          if not fl then
20              // the kth neighbor of the new vertex is invalid
21              r := r + 1;
22              continue;

            // check if this is the edge with which currvertex
            // is hooked to the new vertex
23          if (Move(j), Move(k)) = Current then break;
24          r := r + 1;

        // move to the new vertex
25      Move(j);
        // return the tree back-label
26      return r;
```

Function `TreeForwardStep`. **Input** i. **Output** Let $(v, j) = \Gamma_{T,1}(cv, i)$. If a move along (cv, i) is possible, i.e., v is valid, then `TreeForwardStep` returns j and moves the current vertex to v; otherwise, it returns `null` and does not change the current vertex. **Assumes** $0 \le i \le \delta_T(cv)$, cv is valid. **Local variables** `j` (back-label of the ith T-edge of cv).

```
function TreeForwardStep
    j := TreeMove(argTreeForwardStep);
    if j = null then return null;
    j := j + 1;
    if j > TreeDegree then j := 1;
    return j;
```

Function `TreeForward`. **Input** i. **Output** If $(v, j) = \Gamma_{T,i}(cv, 1)$, then the function `TreeForward` returns j and moves the current vertex to v. **Assumes** cv and i are valid, all moves of $\Gamma_{T,\le i}(v, 1)$ are possible. **Local variables** `i` (input), `j` (output), and `k` (counter for steps of the exploration walk).

```
function TreeForward
    i := argTreeForward;
    j := 1;
    for k := 1 to i do
        j := TreeForwardStep(j);
    return j;
```

Function `TreeBack`. **Input** i, j. **Output** If v is the vertex of $\Gamma'_{T,i}(cv, j)$, then `TreeBack` moves the current vertex to v. **Assumes** $0 \le j \le \delta_T(cv)$, cv and i are valid, all moves of $\Gamma'_{T,\le i}(v, j)$ are possible. **Local variables** `i` (input), `j` (input), and `k` (counter for steps of the exploration walk). **Notes** `TreeBack` is defined in a way similar to `TreeForward` using a function `TreeBackStep`.

```
function TreeBackStep
    j := argTreeBackStep - 1;
    if j = 0 then j := TreeDegree;
    return TreeMove(j);

procedure TreeBack
    i := arg1TreeBack; j := arg2TreeBack;
    for k := 1 to i do
        j := TreeBackStep(j);
```

Function `TreeSize`. **Input** None. **Output** $2(\texttt{size}(T)-1)$, if T is contractable, and `null` otherwise. **Assumes** None. **Local variables** `i` (length of the exploration walk), $\texttt{i}_1$ (counter for steps of the exploration walk), $\texttt{k}_1$ (T-edge label of the $(\texttt{i}-1)$st edge of the exploration walk), $\texttt{k}_2$ (T-edge label of the `i`th edge of the exploration walk), `d` (T-degree of cv), $\texttt{d}_1$ (T-degree of the `i`th vertex of the exploration walk), and `td` (degree of T).

Notes $2(\texttt{size}(T)-1)$ is the length of the exploration walk given in Proposition 3.4. The method to compute it is provided by the same proposition; i.e., `TreeSize` incrementally finds (lines 6–22) the length of a walk which visits the current vertex exactly the number of times equal to its tree-degree plus 1 (the check is done in lines 13 and 14). Before increasing the length of the walk, we first make sure that the next move is possible (lines 7–12). If it is not, `TreeSize` returns `null`. This is correct, because if T has an invalid vertex, it is not contractable (`Valid` > `ContractDegree`). Otherwise, it checks, if the walk went back to the starting vertex and returns, if the starting vertex was visited sufficiently many times. Also when `TreeSize` visits a vertex for the first time (lines 15–17), it adds its degree to the current total degree of T and returns `null` if the total degree becomes larger than `ContractDegree` (lines 18–21).

The condition of the loop in line 6 makes sure that the current length `i` of the exploration walk is valid. If it is not, line 23 returns `null` because T is uncontractable. This is correct because, on one hand, $\text{size}(T) \le \deg(T)$ (at line 6 T has at least one edge) and, on the other, by Proposition 3.4, exploration walk of length $2(\text{size}(T)-1)$ visits all vertices of a tree of size $\text{size}(T)$ and returns to the starting vertex sufficiently many times. Since `Valid` $\ge$ 2 `ContractDegree`, if the length of the exploration walk becomes bigger than `Valid`, then $\text{size}(T) >$ `ContractDegree`, so that $\deg(T) >$ `ContractDegree` and T is uncontractable.

`i` and $\texttt{i}_1$ are valid because of the condition of the loop in line 6. $\texttt{k}_1$ is valid because of the assumption that all vertices visited by the exploration walk of length $\texttt{i}-1$ in line 7 are valid. $\texttt{k}_2$ is valid because of the condition on the output of `TreeForwardStep` in line 8. `d` is valid because of the condition on the output of `TreeDegree` in line 1. `td` is valid because at line 20 both `td` and $\texttt{d}_1$ are at most `ContractDegree`, and since `Valid` $\ge$ 2 `ContractDegree`, the addition in line 20 produces a valid result.

```
   function TreeSize
 1     d := TreeDegree;
       // if currvertex is invalid, then the tree is uncontractable
 2     if d = null then return null;
 3     if d = 0 then return 0;

 4     i := 1;
 5     td := 0;
 6     while i ≤ Valid do
           // check if we can make one more step
           // from the exploration walk
 7         k1 := TreeForward(i-1);
 8         k2 := TreeForwardStep(k1);
 9         if k2 = null then
               // if we cannot, then the tree is uncontractable
10             TreeBack(i-1, k1);
11             return null;
12         TreeBack(i, k2);
```

```
          // check if we have visited the starting vertex
          // sufficiently many times
13        if TreeForward(i) = Current then d := d - 1;
          // if yes, then return the current length of
          // the exploration walk
14        if d = 0 then return i;

          // check if the end of the current exploration walk
          // is visited for the first time
15        for i1 := 0 to i - 1 do
16            if TreeForward(i) = TreeForward(i1) then break;
17        if i1 = i then
              // if it is, add its degree to the total degree
18            after TreeForward(i) do d1 ::= Prev(Degree)
              // if the total degree becomes too large,
              // then the tree is uncontractable
19            if d1 > ContractDegree then return null;
20            td := td + d1;
21            if td > ContractDegree then return null;

          // increase the length of the exploration walk by 1
22        i := i + 1;
23    return null;
```

Function `Root`. **Input** None. **Output** If T is uncontractable or $cv \in D_{cl-1}$, then `Root` returns 0; otherwise, it returns the index of the first occurrence of root(T) in the exploration walk. **Assumes** None. **Local variables** `d` ($2(\text{size}(T) - 1)$) and `i` (counter for steps of the exploration walk).

Notes According to the definition of root(T) given in section 3.2.1, `Root` enumerates the vertices of T using the exploration walk starting from $(cv, 1)$ (lines 3–5) and finds the first vertex which is hooked to itself (line 4). `d` is valid because of the assignment in line 1, and `i` is valid because at line 3 T is contractable.

```
  function Root
      // check if T is contractable
1     d := TreeSize;
2     if d = null or Prev(Done) then return 0;

      // if it is, find the first vertex in it which is hooked
      // to itself
3     for i := 0 to d-1 do
4         after TreeForward(i) do fl := (Prev(Hook) = 0);
5         if fl then return i;
```

4.4.9. Contraction. The definitions of the functions in this section come from the definition of the contraction operation given in section 3.2.3.

Function `IsEdge`. **Input** i and j. **Output** If v is the vertex of $\Gamma_{T,i}(cv, 1)$, then `IsEdge` returns true iff (v, j) is a remaining edge of T (see the definition in section 3.2.3). **Assumes** i is valid, $0 \le j \le \delta_{cl-1}(v)$, $cv = \text{root}(T)$, and T is contractable. **Local variables** `i` (input), `j` (input), `k` (counter for steps of the exploration walk), $\texttt{j}_1$ (counter for neighbors of the `k`th vertex of the exploration walk), $\texttt{k}_1$ (counter for steps of the exploration walk on the tree of the `j`th neighbor of v), `d` ($2(\text{size}(T) - 1)$),

$\mathtt{d}_1$ (degree of the kth vertex of the exploration walk), and $\mathtt{d}_2$ ($2(s-1)$, where s is the size of the tree of the jth neighbor of v).

Notes The definition of `IsEdge` follows exactly the definition of the remaining edges of T given in section 3.2.3. Let $e = (v, j)$, $w = \eta_{cl-1}(e)$, and T' be the hooking tree of w in $\mathcal{C}_{cl-1}$. Lines 2–5 check if T' is contractable. Line 7 checks if e is internal. Let u be the kth vertex in the exploration walk of T starting from cv. Because of lines 9 and 10, at line 12 $(u, \mathtt{j}_1)$ is an edge before e in the sequence of the remaining edges of T given in section 3.2.3. Let $u' = \eta_{cl-1}(u, \mathtt{j}_1)$. Lines 14–16 check whether u' is in T'. In line 13 we use that if the hooking tree of u' in $\mathcal{C}_{cl-1}$ is uncontractable, then u' is not from T', because at this point T' is contractable.

i, j, and d are valid because T is contractable. $\mathtt{d}_2$ and $\mathtt{k}_1$ are valid, because at line 6 T' is contractable. Lines 9 and 10 ensure the validity of $\mathtt{d}_1$ and $\mathtt{j}_1$.

```
   function IsEdge
        i := arg1IsEdge; j := arg2IsEdge;
 1      d := TreeSize;
        // if T' is uncontractable, then e remains
 2      after TreeForward(i) do fl := MoveValid(j);
 3      if not fl then return true;
 4      after TreeForward(i), Move(j) do d2 := TreeSize;
 5      if d2 = null then return true;

        // T' is contractable
 6      for k := 0 to d-1 do
            // e does not remain if it is an internal edge
 7          if TreeForward(k) = (TreeForward(i), Move(j)) then
                return false;
 8          if k > i then continue;

 9          if k = i then d1 := j - 1;
            else
10              after TreeForward(k) do d1 ::= Prev(Degree);

            // e does not remain if it is not the first edge
            // from T to T'
11          for j1 := 1 to d1 do
12              after TreeForward(k) do fl := MoveValid(j1);
13              if not fl then continue;

14              for k1 := 0 to d2-1 do
15                  if (Treeforward(i), Move(j), TreeForward(k1)) =
                            (TreeForward(k), Move(j1)) then
16                      return false;
17      return true;
```

Statement "**after** P **for every edge** (i, j) **do** B." **Input** P a path description relative to the current vertex, i and j names of local variables using this statement, B instruction(s) which might depend on the variables i and j. **Output** Let v be the vertex with path description P, and T' is its hooking tree in $\mathcal{C}_{cl-1}$. This statement executes B for all possible values of (i, j) such that $(u, \mathtt{j})$ is a remaining edge of T', where u is the vertex of $\Gamma_{T',\mathtt{i}}(v, 1)$. **Assumes** $cl \geq 1$, all forward moves in P are valid, T' is contractable and $v = \mathrm{root}(T')$. **Local variables** $\mathtt{i}_1$ (counter for steps of the

exploration walk on T' starting from $(v,1)$), $\mathtt{d}_1$ ($2(\text{size}(T')-1)$), and $\mathtt{d}_2$ (degree of the $\mathtt{i}$th vertex of the exploration walk on T' starting from $(v,1)$).

Notes Lines 3–5 check if this is the first time the exploration walk on T' starting from $(v,1)$ visits the $\mathtt{i}$th vertex v. If so, lines 7–9 enumerate the remaining edges of T' incident to v. All local variables, and `i` and `j`, are valid because T' is contractable.

```
  statement after P for every edge (i, j) do B
1     after P do d1 := TreeSize;
2     for i := 0 to d1 - 1 do
          // visit only once every vertex of T'
3         for i1 := 0 to i - 1 do
4             if (P, TreeForward(i)) = (P, TreeForward(i1)) then
                  break;
5         if i1 < i then continue;

6         after P, TreeForward(i) do d2 ::= Prev(Degree);
7         for j := 1 to d2 do
8             after P do fl := IsEdge(i, j);
9             if not fl then continue;

              // if (i, j) is a remaining edge, then execute B
10            B;
```

Function `Degree`. **Input** None. **Output** $\delta_{cl}(cv)$. **Assumes** None. **Local variables** `i` (counter for steps of the exploration walk), `j` (counter for neighbors of the $\mathtt{i}$th vertex of the exploration walk), and `d` (degree of cv). **Notes** To obtain the degree of the current vertex, we just enumerate all remaining edges of T. If T is not contractable, then, by definition, the degree comes from a previous level (line 2). Line 2 handles the case when $cv \in I_{cl}$ and line 3 the case when $cv \in D_{cl}$. All local variables are valid because at line 3 T is contractable.

`GraphDegree` returns the degree of the current vertex in the input graph G.

```
  function Degree
1     if level = 0 then return GraphDegree;
2     if not Contraction or TreeSize = null then return Prev(Degree);
3     if Prev(Done) or Root ≠ 0 then return 0;

4     d := 0;
5     after Current for every edge (i,j) do d := d + 1;
6     return d;
```

Procedure `Neighbor`. **Input** i. **Output** Moves the current vertex to $\eta_{cl}(cv,i)$. **Assumes** $0 \le i \le \delta_{cl}(cv)$. **Local variables** `i` (input), `l` (counter for steps of the exploration walk), `j` (counter for neighbors of the $\mathtt{l}$th vertex of the exploration walk), and `d` ($2(\text{size}(T)-1)$).

Notes The definition of `Neighbor` follows the definitions in section 3.2.3. First, we make sure that T is contractable (lines 4 and 10). If not, then we call recursively. Otherwise, i is the index of a remaining edge e of T, and we locate e and move along it (lines 14–20). Once we move along e, we move the current vertex to the representative of the new current vertex, i.e., the root of the new current hooking tree T', if it is contractable (lines 6, 12, and 19).

`i` is valid because at line 8 `argNeighbor` is valid, and the other local variables are valid because at line 14 T is contractable.

`GraphNeighbor`(i) moves the current vertex to its ith neighbor in the input

graph G.

```
   procedure Neighbor
 1     if level = 0 then GraphNeighbor(argNeighbor);
       if not Contraction then Prev(Neighbor);
 2     // handle the self-loop case
 3     if argNeighbor = 0 then return;

 4     if argNeighbor > Valid then
           // if T is uncontractable, call recursively
 5         Prev(Neighbor);
           // if T' is contractable, move to its root
 6         if TreeSize ≠ null then TreeForward(Root);
 7         return;
 8     i := argNeighbor;

 9     d := TreeSize;
10     if d = null then
           // T is uncontractable
11         Prev(Neighbor(i));
12         if TreeSize ≠ null then TreeForward(Root);
13         return;

       // T is contractable
14     after Current for every edge (l, j) do
15         i := i - 1;
           // check if (l, j) is e
16         if i > 0 then continue;

           // move to e and then along e
17         TreeForward(l);
18         Prev(Neighbor(j));
           // move to the root of T'
19         if TreeSize ≠ null then TreeForward(Root);
20         return;
```

Function `BackLabel`. **Input** i. **Output** $\beta_{cl}(cv, i)$. Uses the array method described in section 4.4.2 of taking arguments and returning values. **Assumes** $0 \leq i \leq \delta_{cl}(cv)$. **Local variables** `i` (input), `l` (counter for steps of the exploration walk), $\mathtt{k}_1$ (same as `l`), `j` (counter for the neighbors of the `l`th vertex of the exploration walk), `d` ($2(\text{size}(T) - 1)$), `nd` (output), `k` (counter for steps of the exploration walk on T' starting from $(u, 1)$), $\mathtt{j}_1$ (counter for neighbors of the `k`th vertex of the exploration walk on T' starting from $(u, 1)$), and `r` (index of the first occurrence of the root of T' in the exploration walk on T' starting from $(u, 1)$), where u is the `j`th neighbor of the `l`th vertex of the exploration walk and T' is the hooking tree of u.

Notes The first case of `BackLabel` is when T is contractable. In this case we find the remaining edge e of T with index i (lines 12–14). Let $v = \eta_{cl-1}(e)$ and T' be the hooking tree of v in $\mathcal{C}_{cl-1}$. If T' is uncontractable, then we call recursively, because in this case the back-label comes from the previous level of recursion (line 22). Otherwise, we have to find the index `nd` of the first remaining edge e' of T' which goes from T' to T (lines 24–32). This is the new back-label. To find the index of e', first we find the root of T' (line 20) and then enumerate all remaining edges of T' (lines

25–32). For each remaining edge of T', we check if it goes to T (line 30–32). In lines 27–29, we use that if a remaining edge of T' goes to an uncontractable hooking tree, then it does not go to T, because at this point T is contractable. The case when T is uncontractable is handled by `BackLabelAux` (lines 5 and 10).

`i` is valid because of line 4. `d` is valid because of the assignment in line 8. `l`, `j`, and $\texttt{k}_1$ are valid because at line 12 T is contractable. `r` is valid because of line 20. $\texttt{j}_1$, `k`, and `nd` are valid because at line 24 T' is contractable.

The recursive call in line 22 does not assign the returned value to a local variable; i.e., this call returns a value at some higher level of recursion, depending on the array for returning values of `BackLabel`. This call is the reason why `BackLabel` returns through an array instead of a global variable. The conventional thing to do is to store the result of this call locally, and once the **after** statement has restored the original current vertex, return the stored value. This will not work for us, because the value returned from the recursive call might be invalid. Instead, using that the only reason why we store the returned value is to pass it back, when `BackLabel` produces a result we let it store the result at the level at which it is requested. This works because `BackLabel` is always called on the previous level of recursion.

`GraphBackLabel`(i) returns the back-label of the ith edge incident to cv in the input graph G.

```
   function BackLabel
 1     if level = 0 then return GraphBackLabel(argBackLabel);
 2     if not Contraction then return Prev(BackLabel);
 3     if argBackLabel = 0 then return 0;

       // if currvertex is invalid, call BackLabelAux
 4     if argBackLabel > Valid then
 5         BackLabelAux;
 6         return;
 7     i := argBackLabel;

       // if T is uncontractable, call BackLabelAux
 8     d := TreeSize;
 9     if d = null then
10         BackLabelAux;
11         return;

       // T is contractable
12     after Current for every edge (l, j) do
13         i := i - 1;
           // find e
14         if i > 0 then continue;

15         after TreeForward(l) do
16             fl := MoveValid(j);
17             if fl then
18                 after Move(j) do
19                     fl := (TreeSize ≠ null);
20                     if fl then r := Root;

21         if not fl then
```

```
            // if T' is uncontractable, call recursively
22          after TreeForward(l) do Prev(BackLabel(j));
23          return;

    // T' is contractable
24      nd := 0;
        // find the first edge of T' which goes to T and
        // return its index
25      after TreeForward(l), Move(j), TreeForward(r)
                for every edge (k, j_1) do
26          nd := nd + 1;
27          after TreeForward(l), Move(j),
                        TreeForward(r), TreeForward(k) do
28              fl := MoveValid(j_1);
29          if not fl then continue;

30          for k_1 := 0 to d-1 do
31              if (TreeForward(l), Move(j),
                      TreeForward(r), TreeForward(k), Move(j_1)) =
                    TreeForward(k_1) then
32                  return nd;
```

Function `BackLabelAux`. **Input** i. **Output** $\beta_{cl}(cv)$. To take argument and return value, `BackLabelAux` uses the arrays of `BackLabel`. **Assumes** $0 \le i \le \delta_{cl}(cv)$, T is uncontractable. **Local variables** `l` (counter for steps of the exploration walk on T' starting from $(v, 1)$), `j` (counter for neighbors of the `l`th vertex of the exploration walk on T' starting from $(v, 1)$), `k` (same as `l`), `bl` ($\beta_{cl-1}(cv, i)$), `nbl` (output), `r` (index of the first occurrence of the root of T' in the exploration walk on T' starting from $(v, 1)$), and `d` ($2(\text{size}(T') - 1)$), where T' and v are as in the note for `BackLabel`.

Notes The definition of `BackLabelAux` follows the definitions given in section 3.2.3 when T is uncontractable. Let v and T' be as in the note for `BackLabel`. If T' is uncontractable, the back-label is inherited from the previous level of recursion, and so we call `BackLabel` recursively (lines 3 and 10). Otherwise, at line 12 T' is contractable, the current vertex is v (because of line 5), and `bl` is the back-label of e (because of line 1). So we have to find the index of $(v, \texttt{bl})$ in T' ($(v, \texttt{bl})$ is a remaining edge of T' because T is uncontractable). Line 12 finds the root of T', and lines 13 and 14 find the index `k` of the first occurrence of v in the exploration walk of T' starting from its root. Lines 16–20 enumerate the remaining edges of T' until we find $(v, \texttt{bl})$.

`bl` is valid by the assumption for the return convention of `BackLabel` for line 1. `d` is valid because of the assignment in line 6. `r`, `l`, `j`, `k`, and `nbl` are valid because at line 12 T' is contractable.

Just like for `BackLabel`, the calls to `BackLabel` in lines 3 and 10 return values at some higher level of recursion. The calls to `BackLabel` in lines 1, 3, and 10 do not have arguments—by convention this means that the argument to `BackLabel` comes from a higher level of recursion.

The case when T is uncontractable is the reason why the argument to `BackLabel` is passed through an array instead of a global variable. More precisely, the problem is when the current vertex is invalid; then the argument i to `BackLabel`, which is the label of an edge incident to cv, might be invalid and storing it locally will be impossible. In this case we still want to be able to use the value of i after calling

functions which can potentially change the value of a global argument to `BackLabel`. The decision is to let the value of the argument stay at the level which produced it, because it certainly is valid for this level. For this to work, it is important that the value of the argument stored in the array is not changed while processing the call to `BackLabel`. Fortunately, this does not happen, because `BackLabel` is always called on the previous level of recursion.

```
   function BackLabelAux
 1     bl := Prev(BackLabel);
 2     if bl = null then
           // if T' is uncontractable, call recursively
 3         Prev(BackLabel);
 4         return;

       // move along e
 5     Prev(Neighbor(argBackLabel));

 6     d := TreeSize;
 7     if d = null then
 8         // if T' is uncontractable, go back and call recursively
 9         Prev(Neighbor(bl));
10         Prev(BackLabel);
11         return;

       // T' is contractable
12     r := Root;
       // find the index of the first occurrence of v in
       // the exploration walk on T' starting from (r,1)
13     for k := 0 to d - 1 do
14         if TreeForward(r), TreeForward(k) = Current then break;

       // compute the new back-label
15     nbl := 0;
16     after TreeForward(r) for every edge (l, j) do
           // increase the new back-label by one for every edge
           // that happens before e
17         nbl := nbl + 1;

18         if l = k and j = bl then
               // if we are at (v,bl), move back and return
               // the new back-label
19             Prev(Neighbor(bl));
20             return nbl;
```

Function `Move`. **Input** i. **Output** Let $(v, j) = \mu_{cl-1}(cv, i)$. `Move` returns j and moves the current vertex to v. **Assumes** $cl \geq 1$, $0 \leq i \leq \delta_{cl-1}(cv)$, i and j valid. **Local variables** `i` (input) and `j` (output), which are valid by the assumption about the argument of `Move`.

```
function Move
    i := argMove;
    j := Prev(BackLabel(i));
    Prev(Neighbor(i));
```

```
    return j;
```

Function `MoveValid`. **Input** i. **Output** True iff $\beta_{cl-1}(cv, i)$ is valid. **Assumes** $cl \geq 1$, $0 \leq i \leq \delta_{cl-1}(cv)$, i valid. **Local variables** None.

```
function MoveValid
    return Prev(BackLabel(argMoveValid)) ≠ null;
```

4.4.10. Solving undirected st-connectivity. Procedure `MoveToRep`. **Input** None. **Output** Moves the current vertex to $\mathrm{rep}_{R_{cl}}(cv)$. **Assumes** None. **Local variables** None.

```
procedure MoveToRep
    if level > 0 then
        Prev(MoveToRep);
        if Contraction and TreeSize ≠ null then TreeForward(Root);
```

Global function `Connected`. **Input** s and t. **Output** True iff s and t are connected in G.

```
global function Connected
    level := 5 · 2^⌈log log n⌉ − 3;
    currvertex := arg1Connected; MoveToRep;
    v := currvertex;
    currvertex := arg2Connected; MoveToRep;
    return v = currvertex;
```

5. Proofs of Theorem 3.12 and Corollary 3.13. The proofs in this section are a direct translation into our notation of equivalent statements from [6] and are given here only for completeness. We use the notation of section 3.3.

LEMMA 5.1. *Assume that $\mathcal{C}_l$ is nice and $P_{l+1} = \langle Hook, k\rangle$.*

(i) *If $v \in I_l$, then $v \in I_{l+r(k)}$, $\delta_{l+r(k)}(v) = \delta_l(v)$, and $H_{l+r(k)}(v) = H_l(v)$.*

(ii) *If $v \in I_{l+r(k)} - I_l$, then $\delta_{l+r(k)}(v) \leq \mathrm{dexp}(k+2)$.*

(iii) *$\mathcal{C}_{l+1}$ is nice. If $k > 0$, then $\mathcal{C}_{l+2}$ is nice.*

Proof. (i) The state and degree of a vertex change only during a contraction operation. The same holds for the hooking edges of inactive vertices. Let T be the hooking tree of v in $\mathcal{C}_l$. Since $\mathcal{C}_l$ is nice, $\deg(T) > \mathrm{dexp}(k+2)$. For $l+1 \leq i \leq l+r(k)$, we have that $Arg_i \leq k+1$, so that trees of degree more than $\mathrm{dexp}(k+2)$ are never contracted. Therefore the status, degree, and hooking edge of v are never changed.

(ii) Since $v \in I_{l+r(k)} - I_l$ and because a done vertex stays done, we have that $v \in A_l$, and so $\delta_l(v) \leq \mathrm{dexp}(k+2)$, because $\mathcal{C}_l$ is nice. Hooking does not change degrees. For $l+1 \leq i \leq l+r(k)$, we have that $Arg_i \leq k+1$, so that any active vertex appearing after a contraction must have degree at most $\mathrm{dexp}(k+2)$. Let i be the largest $l \leq i < l+r(k)$ such that $v \in A_i$. Then $\delta_i(v) \leq \mathrm{dexp}(k+2)$, and its degree does not change afterwards. Hence $\delta_{l+r(k)}(v) \leq \mathrm{dexp}(k+2)$.

(iii) We have that $P_{l+2} = \langle Contract, k\rangle$ or $P_{l+2} = \langle Contract, 1\rangle$, if $k = 0$. Since a hooking operation changes only the hooking edges of active vertices, $\mathcal{C}_{l+1}$ is nice. Furthermore, for a hooking tree T' in $\mathcal{C}_{l+1}$ which contains an inactive vertex, we have that $\mathrm{size}(T') > \mathrm{dexp}(k+1)$

If $k > 0$, then $P_{l+3} = \langle Hook, k-1\rangle$. Let T be a hooking tree in $\mathcal{C}_{l+2}$ composed of inactive vertices. Because $P_{l+2} = \langle Contract, k\rangle$, we have that $\delta_{l+2}(v) \leq \mathrm{dexp}(k+1)$, for $v \in A_{l+2}$, and $\deg(T) > \mathrm{dexp}(k+1)$. There are two possibilities for T—either it was a hooking tree in $\mathcal{C}_{l+1}$ composed of active vertices or it contains a hooking tree T' of $\mathcal{C}_{l+1}$ which contained an inactive vertex. In the first case by Lemma 3.9, $\mathrm{size}(T) > \mathrm{dexp}(k)$. This is also true in the second case because, as mentioned at the

end of the previous paragraph, $\text{size}(T) > \text{dexp}(k+1) > \text{dexp}(k)$. Therefore $\mathcal{C}_{l+2}$ is nice. □

LEMMA 5.2. *Assume that $\mathcal{C}_l$ is nice and $P_{l+1} = \langle Hook, k\rangle$.*

(i) *If T is a hooking tree in $\mathcal{C}_{l+r(k)}$ composed of inactive vertices, then* $\text{size}(T) > \text{dexp}(k+1)$ *and* $\deg(T) > \text{dexp}(k+2)$.

(ii) *If $k > 0$, then $\mathcal{C}_{l+2+r(k-1)}$ and $\mathcal{C}_{l+2+2r(k-1)}$ are nice.*

Proof. Let $l_0 = l+2$, $l_1 = l_0 + r(k-1)$, $l_2 = l_1 + r(k-1)$, and $l_3 = l_2 + 1$ $(= l + r(k))$.

(i) Let T be a hooking tree of $\mathcal{C}_{l_3}$ composed of inactive vertices, i.e., $V_T \subseteq I_{l_3}$. Since $P_{l_3} = \langle Contract, k+1\rangle$, we have that

$$\deg(T) > \text{dexp}(k+2), \tag{5.1}$$

which ensures the degree part of the first item of the lemma. We are left with proving the size part.

Case 1: $V_T \cap I_l \neq \emptyset$. By Lemma 5.1(i), the vertices in $V_T \cap I_l$ form a hooking tree T' of $\mathcal{C}_l$ composed of inactive vertices. Since $\mathcal{C}_l$ is nice, $\text{size}(T') > \text{dexp}(k+1)$, and hence $\text{size}(T) > \text{dexp}(k+1)$.

Case 2: $V_T \cap I_l = \emptyset$. We do induction on k. If $k = 0$, then by Lemma 3.9 and (5.1), $\text{size}(T) > \text{dexp}(1)$.

Assume that $k > 0$.

From $\mathcal{C}_l$ nice and $P_{l_0} = \langle Contract, k\rangle$, it follows that $I_l \subseteq I_{l_0}$. Also by Lemma 5.1(i), $I_{l_0} \subseteq I_{l_1} \subseteq I_{l_2}$. Finally, $I_{l_3} \subseteq I_{l_2}$, because $Op_{l_3} = Contract$. Hence $V_T \subseteq I_{l_2}$.

For $l_0 + 1 \leq i \leq l_2$, we have that $Arg_i \leq k$. Therefore we can use the same argument as in the proof of Lemma 5.1(ii) to see that for every $v \in I_{l_2} - I_{l_0}$,

$$\delta_{l_2}(v) \leq \text{dexp}(k+1). \tag{5.2}$$

Case 1.1: $V_T \cap I_{l_0} = \emptyset$. By (5.2), $\delta_{l_2}(v) \leq \text{dexp}(k+1)$ for every $v \in V_T$. Hence $\deg(T) \leq \text{size}(T)\,\text{dexp}(k+1)$. Thus, from (5.1), it follows that $\text{size}(T) > \text{dexp}(k+1)$.

Case 1.2: $V_T \cap I_{l_0} \neq \emptyset$. From Lemma 5.1(i), it follows that $V_T \cap I_{l_0}$ form a hooking tree T' in $\mathcal{C}_{l_0}$. Let $d = \deg(T')/\text{size}(T')$, the average degree of a vertex of T'.

Case 1.2.1: $d \leq \text{dexp}(k+1)$. We have that $\deg(T') \leq \text{dexp}(k+1)\,\text{size}(T')$. Also for every $v \in V_T - V_{T'} \subseteq I_{l_2} - I_{l_0}$, by (5.2), $\delta_{l_2}(v) \leq \text{dexp}(k+1)$. Hence

$$\deg(T) = \sum_{v \in V_{T'}} \delta_{l_2}(v) + \sum_{v \in V_T - V_{T'}} \delta_{l_2}(v) \leq \text{dexp}(k+1)\,\text{size}(T),$$

and so $\text{size}(T) > \text{dexp}(k+1)$, because of (5.1).

Case 1.2.2: $d > \text{dexp}(k+1)$. Since $V_{T'} \subseteq V_T$, we have that $V_{T'} \cap I_l = \emptyset$. Hence $V_{T'} \subseteq A_l$, and so, by Lemma 3.9, $\text{size}(T') > \deg(T')/\text{size}(T') = d > \text{dexp}(k+1)$. Therefore, by (5.2),

$$\begin{aligned}\deg(T) &\leq \deg(T') + \text{dexp}(k+1)(\text{size}(T) - \text{size}(T')) \\ &< \text{size}^2(T') + \text{size}(T')(\text{size}(T) - \text{size}(T')) = \text{size}(T')\,\text{size}(T) \leq \text{size}^2(T).\end{aligned}$$

Hence in this last case and using (5.1) again, $\text{size}(T) > \text{dexp}(k+1)$ as well.

(ii) By Lemma 5.1(iii), $\mathcal{C}_{l_0}$ is nice. Hence, by the first item of the lemma (notice that $P_{l_0+1} = \langle Hook, k-1\rangle$), for any hooking tree T in $\mathcal{C}_{l_1}$ composed of inactive vertices, we have that $\text{size}(T) > \text{dexp}(k)$ and $\deg(T) > \text{dexp}(k+1)$. Furthermore,

since $P_{l_1} = \langle Contract, k\rangle$, for any $v \in A_{l_1}$, $\delta_{l_1}(v) \le \mathrm{dexp}(k+1)$. Therefore $\mathcal{C}_{l_1}$ is nice. $\mathcal{C}_{l_2}$ is nice for similar reasons. ∎

LEMMA 5.3. *Assume that $\mathcal{C}_l$ is nice and $P_{l+1} = \langle Hook, k\rangle$. Then for every $v \in A_{l+r(k)}$,*

$$|\{u \in A_l : \mathrm{rep}_{l+r(k)}(u) = v\}| \ge \mathrm{dexp}(k).$$

Proof. We do induction on k. Let $k = 0$. Consider $v \in A_{l+2}$. v must be a root of a hooking tree T in $\mathcal{C}_{l+1}$ and $V_T \subseteq A_{l+1}$, because $\mathcal{C}_l$ is nice and $P_{l+2} = \langle Contract, 1\rangle$. Hooking does not change the states of vertices, and so $A_l = A_{l+1}$. By Lemma 3.9, T has at least two vertices, and therefore v represents at least two vertices from A_l.

Assume that $k > 0$. Let $l_0 = l + 2$, $l_1 = l_0 + r(k-1)$, $l_2 = l_1 + r(k-1)$, and $l_3 = l_2 + 1$ $(= l + r(k))$. As can be seen in the proof of Lemma 5.2, we have that $A_{l_2} \subseteq A_{l_1} \subseteq A_{l_0} \subseteq A_l$ and $A_{l_2} \subseteq A_{l_3}$. Let $v \in A_{l_3}$. We have that v is in either A_{l_2} or in I_{l_2}.

Assume first that $v \in I_{l_2}$. Since $v \in A_{l_3}$ and $P_{l_3} = \langle Contract, k+1\rangle$, v is a root of a hooking tree T of $\mathcal{C}_{l_2}$ composed of inactive vertices which contracts to v. $V_T \subseteq A_l$, because if there exists $u \in V_T \cap I_l$, then by Lemma 5.1(i), $u \in I_{l_3}$, but we have that $V_T \cap I_{l_3} = \emptyset$. By Lemma 5.2(ii), $\mathcal{C}_{l_1}$ is nice, and therefore by Lemma 5.2(i), $\mathrm{size}(T) > \mathrm{dexp}(k)$. Thus v represents at least $\mathrm{dexp}(k)$ vertices from A_l.

Assume now that $v \in A_{l_2}$. Let

$$U_v = \{u \in A_{l_1} : \mathrm{rep}_{l_2}(u) = v\}.$$

By the inductive hypothesis, $|U_v| \ge \mathrm{dexp}(k-1)$. Let $u \in U_v$. Again by the inductive hypothesis, $|\{w \in A_{l_0} : \mathrm{rep}_{l_1}(w) = u\}| \ge \mathrm{dexp}(k-1)$. Hence

$$|\{w \in A_{l_0} : \mathrm{rep}_{l_2}(w) = v\}| \ge \mathrm{dexp}(k-1)\,\mathrm{dexp}(k-1) = \mathrm{dexp}(k).$$

Since $A_{l_0} \subseteq A_l$, the statement follows in this case as well. ∎

We are ready to prove Theorem 3.12.

Proof of Theorem 3.12. $|A_{r(k)}| \le \max\{|A_0|/\mathrm{dexp}(k), 1\}$ follows from Lemma 5.3 and the fact that the sets of vertices represented by different vertices from $A_{r(k)}$ are disjoint. $\mathrm{size}(T_v) > \mathrm{dexp}(k+1)$, for $v \in I_{r(k)}$, follows from Lemma 5.2(i).

We have to prove now that $\mathcal{C}_l$ is nice for $1 \le l < r(k)$. We prove by induction on t that if $\mathcal{C}_l$ is nice and $P_{l+1} = \langle Hook, t\rangle$, then $\mathcal{C}_i$ is nice for $l+1 \le i < l + r(t)$, and if $0 < t < k$, then $\mathcal{C}_{l+r(t)}$ is also nice.

By Lemma 5.1(iii), $\mathcal{C}_{l+1}$ is nice. This takes care of $t = 0$. Assume that $t > 0$. By Lemma 5.1(iii) and Lemma 5.2(ii), $\mathcal{C}_{l+2}$, $\mathcal{C}_{l+2+r(t-1)}$, and $\mathcal{C}_{l+2+2r(t-1)}$ are nice. By the inductive hypothesis applied twice, $\mathcal{C}_i$ is nice for $l+3 \le i < l+2+r(t-1)$ and $l+3+r(t-1) \le i < l+2+2r(t-1)$. Since $P_{l+r(t)} = \langle Contract, t+1\rangle$, we have that $\delta_{l+r(t)}(v) \le \mathrm{dexp}(t+2)$ for $v \in A_{l+r(t)}$. By Lemma 5.2(i), for any hooking tree of $\mathcal{C}_{l+r(t)}$ composed of inactive vertices, $\mathrm{size}(T) > \mathrm{dexp}(t+1)$ and $\deg(T) > \mathrm{dexp}(t+2)$. If $t < k$, then $P_{l+r(t)+1}$ is either $\langle Contract, t+2\rangle$ or $\langle Hook, t\rangle$, and in both cases $\mathcal{C}_{l+r(t)}$ is nice. ∎

Finally, let us prove Corollary 3.13.

Proof of Corollary 3.13. Let $k = \lceil \log\log n \rceil$. By Theorem 3.12,

$$|A_{r(k)}| \le \max\{n/\mathrm{dexp}(k), 1\} = 1.$$

On the other hand, again by Theorem 3.12, every hooking tree in $\mathcal{C}_{r(k)}$ composed of inactive vertices has size more than $\mathrm{dexp}(k+2) \ge n^2$. Therefore $I_{r(k)} = \emptyset$.

Suppose that u and v are connected in G, but $r_1 = \mathrm{rep}_{r(k)}(u) \neq \mathrm{rep}_{r(k)}(v) = r_2$. As we saw, $|A_{r(k)}| \leq 1$ and $I_{r(k)} = \emptyset$, and so without loss of generality we can assume that $r_1 \in D_{r(k)}$. Let l be the largest such that $r_1 \notin D_l$. $\delta_{l+1}(r_1) > 0$, because $r_1 \neq r_2$, r_1 and r_2 are connected in G, and r_1 inherits the neighbors of the vertices it represents. This contradicts the fact that r_1 becomes done if it is a root of a hooking tree and its degree is 0 (it can also become done if it is a nonroot of a hooking tree which is contracted, but then it would not be a representative). □

Acknowledgments. The author is grateful to Prof. Anna Gál for help in the preparation of this paper and Prof. Vijaya Ramachandran for pointing out the problem and many helpful discussions.

REFERENCES

[1] R. Aleliunas, R. Karp, R. Lipton, L. Lovász, and C. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, in Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science, 1979, pp. 218–223.

[2] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou, $SL \subseteq L^{\frac{4}{3}}$, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1997, pp. 230–239.

[3] B. Awerbuch and Y. Shiloach, *New connectivity and MSF algorithms for ultracomputer and PRAM*, in Proceedings of the IEEE International Conference on Parallel Processing, 1983, pp. 175–179.

[4] F. Chin, J. Lam, and I.-N. Chen, *Efficient parallel algorithms for some graph problems*, Comm. ACM, 25 (1982), pp. 659–665.

[5] K. Chong, Y. Han, and T. Lam, *On the parallel time complexity of undirected connectivity and minimum spanning trees*, J. ACM, 48 (2001), pp. 297–323.

[6] K. Chong and T. Lam, *Finding connected components in $O(\log n \log\log n)$ time on the EREW PRAM*, J. Algorithms, 18 (1995), pp. 378–402.

[7] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree, and graph problems*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491.

[8] H. Gazit, *An optimal randomized parallel algorithm for finding connected components in a graph*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 492–501.

[9] S. Halperin and U. Zwick, *Optimal randomized EREW PRAM algorithms for finding spanning forests and for other basic graph connectivity problems*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 438–447.

[10] D. Hirschberg, A. Chandra, and D. Sarwate, *Computing connected components on parallel computers*, Comm. ACM, 22 (1979), pp. 461–464.

[11] D. Johnson and P. Metaxas, *Connected components in $O(\log^{3/2}|V|)$ parallel time for the CREW PRAM*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 688–697.

[12] D. Karger, N. Nisan, and M. Parnas, *Fast connected components algorithm for the EREW PRAM*, in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 373–381.

[13] M. Koucký, *Universal traversal sequences with backtracking*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, pp. 21–27.

[14] H. Lewis and C. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[15] N. Nisan, *Pseudorandom generators for space-bounded computation*, Combinatorica, 12 (1992), pp. 449–461.

[16] N. Nisan, E. Szemerédi, and A. Wigderson, *Undirected connectivity in $O(\log^{1.5} n)$ space*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 24–29.

[17] S. Pettie and V. Ramachandran, *A randomized time-work optimal parallel algorithm for finding a minimum spanning forest*, SIAM J. Comput., 31 (2002), pp. 1879–1895.

[18] O. Reingold, *Undirected st-connectivity in log-space*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 376–385.

[19] C. Savage and J. Ja'Ja', *Fast, efficient parallel algorithms for some graph problems*, SIAM J. Comput., 10 (1981), pp. 682–691.

[20] W. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.
[21] Y. Shiloach and U. Vishkin, *An $O(\log n)$ parallel connectivity algorithm*, J. Algorithms, 3 (1982), pp. 57–67.
[22] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
[23] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 484–504

THE MIXING TIME OF THE THORP SHUFFLE*

BEN MORRIS†

Abstract. The Thorp shuffle is defined as follows. Cut a deck of cards into two equal piles. Drop the first card from the left pile or the right pile according to the outcome of a fair coin flip, then drop from the other pile. Continue this way until both piles are empty. We show that the mixing time for the Thorp shuffle with 2^d cards is polynomial in d.

Key words. mixing time, Markow chain, card shuffling

AMS subject classification. 60J10

DOI. 10.1137/050636231

1. Introduction.

1.1. The Thorp shuffle. How many shuffles are necessary to mix up a deck of cards? (see section 1.2 for a precise definition). The mathematics of card shuffling has been studied extensively over the past several decades and many of the problems have been solved. Most famously, Bayer and Diaconis [2] (in one of the few mathematical results to have made the front page of the New York Times) gave very precise bounds for the Gilbert–Shannon–Reeds (riffle) shuffle model. Their bounds were correct even up to the constant factors. For many other natural shuffles matching upper and lower bounds are known. However, despite Bayer and Diaconis's definitive analysis of the Gilbert–Shannon–Reeds shuffle, little is known about any variation of this model, and one problem, in particular, has stood out for its resistance to attack.

In 1973, Thorp [11] introduced the following shuffling procedure. Assume that the number of cards, n, is even. Cut the deck into two equal piles. Drop the first card from the left pile or the right pile according to the outcome of a fair coin flip, then drop from the other pile. Continue this way, with independent coin flips deciding whether to drop LEFT-RIGHT or RIGHT-LEFT each time, until both piles are empty.

The Thorp shuffle, despite its simple description, has been hard to analyze. The problem of determining its mixing time (see [1, 4, 9]) is, according to Diaconis [3], the "longest-standing open card shuffling problem." It has long been conjectured that the mixing time is $O(\log^c n)$ for some constant c. However, despite much effort the only known upper bounds are trivial ones of the form $O(n^c)$ that have circulated in the folklore. The main contribution of this paper is to give the first poly log upper bound for the mixing time.

We shall assume that the number of cards is 2^d for a positive integer d. (Thus, our aim is to prove that the mixing time is polynomial in d.) In this case the Thorp shuffle has a very appealing alternative description. By writing the position of each card, from the bottom card (0) to the top card $(2^d - 1)$, in binary, we can view the cards as occupying the vertices of the d-dimensional unit hypercube $\{0, 1\}^d$. The Thorp shuffle proceeds in two stages. In the first stage, an independent coin is flipped for each edge e in direction 1 (i.e., each edge in the cube that connects two vertices that differ in

*Received by the editors July 18, 2005; accepted for publication (in revised form) June 20, 2006; published electronically May 23, 2008.
http://www.siam.org/journals/sicomp/38-2/63623.html
†Department of Mathematics, University of California, Davis, CA 95616 (morris@math.ucdavis.edu). This work was done while the author was at Indiana University and Microsoft Research.

only the first coordinate). If the coin lands heads, the cards at the endpoints of e are interchanged; otherwise the cards remain in place. In the second stage, a "cyclic left bit shift" is performed for each card, where the card in position $(x_1, \ldots, x_d)$ is moved to $(x_2, \ldots, x_d, x_1)$.

We will actually use a slightly modified definition of the Thorp shuffle. Say that an edge in the hypercube *rings* if its endpoints are switched with probability $\frac{1}{2}$. For $j = 1, \ldots, d$, let K_j be the transition kernel for the process in which every edge e in direction j rings. This leads us to the following definition.

Definition: Thorp shuffle. The *Thorp shuffle* is the Markov chain whose transition kernel at time n is K_{j+1} if $j \equiv n \bmod d$.

Note that if X_n is the shuffle described in [11], then this modified process can be written as $\theta^n X_n$, where θ denotes the operation that executes a cyclic right bit shift. Thus d iterations of our shuffle is equivalent to d iterations of the shuffle described in [11], so it is enough to prove a poly(d) mixing time bound for this new model.

It is natural to consider the change in the deck after d shuffles have been performed. (This represents one complete "cycle.") We will call this a *round*. Using the language of network computing, a round of the Thorp shuffle is like passing the cards through d levels of a butterfly network (see, e.g., Knuth's book [6]), where at each stage neighboring cards are interchanged with probability $\frac{1}{2}$.

The main result of this paper is that indeed the mixing time is polynomial in d. Our proof uses *evolving sets*, a technique for bounding mixing times that was introduced by the author and Peres in [8]. Another paper that uses some of the same ideas is [7], in which a variant of evolving sets is used to analyze the exclusion process. Evolving sets are related to the notion of strong stationary duality due to Diaconis and Fill [5].

1.2. Statement of main result. For a Markov chain on state space V with uniform stationary distribution, define the *(uniform) mixing time* by

$$\tau_{\text{mix}} = \min\Big\{n : \Big|p^n(x,y)|V| - 1\Big| \le \tfrac{1}{4} \ \ \forall\, x, y \in V\Big\},$$

where $p^n(x,y)$ is the n-step transition probability from x to y. (This is a stricter definition of mixing time than the usual one involving total variation distance.)

Our main result is the following theorem.

THEOREM 1.1. *The mixing time for the Thorp shuffle is* $O(d^{44})$.

Our approach will be to start by proving a "local" mixing property (i.e., a property involving a limited number of cards) and then build up to a global one. In section 2, we show that the Thorp shuffle (more precisely, a "reversibilized" version of the Thorp shuffle) has the following local property: Let $k > 2^{d-1}$. There is a constant b such that if every set of cards S with $|S| = k$ is almost uniformly distributed after bd^5 steps, then for any such set S, any card $x \in S$ will be roughly mixed within S after bd^5 steps. In section 3, we use this property to show that any collection of cards, if viewed as indistinguishable, mixes in $O(d^5)$ time. Later, we go on to prove that the Thorp shuffle satisfies a global mixing property that relates to an l^2 norm on the transition kernel; a precise statement of this property can be found in section 8. Before we get to the global property, we give some necessary background on evolving sets. Following [8], our mixing time bounds are achieved using an isoperimetric function called the root profile, which is connected to evolving sets. In section 4 we give a brief introduction to evolving sets and show how l^2 techniques can be used to give a bound on the root profile. Next, in section 5 we state the main technical result of this

paper (proved in section 8), which is the global property; then we use this to obtain our bound on the root profile. Next, armed with a good bound on the root profile we prove Theorem 1.1 in section 6. We conclude with proofs of technical lemmas in sections 7 and 8.

2. First step: A local mixing property. The basic aim of this section is to show that the following holds for some constant b: if $k > 2^{d-1}$ and every set of cards S with $|S| = k$ is almost uniformly distributed after bd^5 steps, then for any such set S, any card $x \in S$ will be roughly mixed within S after bd^5 steps. We will actually work with the following reversibilized version of the Thorp shuffle: the shuffle that behaves like the Thorp shuffle for the first d steps $(K_1, \dots, K_d)$, and then like a "reverse Thorp shuffle" for the next d steps $(K_d, \dots, K_1)$. We will call this the zigzag shuffle. Every $2d$ steps of the zigzag shuffle will be called a round. (So a round of the zigzag shuffle is a round of the Thorp shuffle followed by a round of a time-reversed Thorp shuffle.)

2.1. Chameleon process. We would like to have a technique to analyze, for a set S of cards, how a particular card $x \in S$ gets mixed within S. For this reason we will consider an auxiliary process called the *chameleon process*. In the chameleon process the cards move in the same way as in the zigzag shuffle, but they also have colors, which can be red, white, black, or pink. Fix $b > 2^{d-1}$ and let $S = \{x_1, \dots, x_b\}$ be a set of cards. Then the initial colors of the chameleon process are as follows: Cards $x_1, \dots, x_{b-1}$ are colored white, card x_b is colored red, and the remaining cards are colored black. The cards can change color in two ways. The first way is called *pinkening*, which takes place when an edge connecting a red card to a white card rings; in this case both cards are recolored pink. The second way is called *depinking*, which takes place at the end of every $64cd$ rounds of shuffling (where the constant c is specified in the next paragraph); in this case all of the pink cards are collectively recolored red or white, with probability $\frac{1}{2}$ each. (A process of this type was first used in [7] to analyze the exclusion process.) Note that black cards can never change color.

We now specify the constant c that appears above. If α is large enough so that $4\alpha^{-d} \le 2^{-d-1}4^{-d}$ for all $d \ge 1$ and $\beta = 8224 \cdot 64 \cdot 5$, then let c be an integer large enough so that $[4e^{-c}]^d \lceil \beta \log \alpha\, c \rceil d^5 \le \alpha^{-d}$ for all $d \ge 1$.

Let X_n be the zigzag shuffle. For $j = 1, \dots, 2^d$, we will write $X_n(j)$ for the position of card j at time n. If $\Lambda = \{z_1, \dots, z_k\}$ is a set of cards, define $X_n(\Lambda) = \{X_n(z_1), \dots, X_n(z_k)\}$. Let $W_n = X_n(S)$ be the *unordered* set of locations of nonblack cards (i.e., cards that are white, red, or pink in the chameleon process) at time n. For vertices x in the hypercube, define

$$\rho_n(x) = \mathbf{1}(\text{there is a red card at } x \text{ at time } n) + \tfrac{1}{2}\mathbf{1}(\text{there is a pink card at } x \text{ at time } n).$$

The following lemma indicates the fundamental relationship between the chameleon process and the zigzag shuffle.

LEMMA 2.1. *Consider the chameleon process with b nonblack cards. Then*

$$\mathbf{P}\Big(X_n(x_b) = x \,\Big|\, W_1, W_2, \dots\Big) = \mathbf{E}\Big(\rho_n(x) \,\Big|\, W_1, W_2, \dots\Big).$$

Proof. We will use induction on n. The base case $n = 0$ is trivial because there is initially only one red ball which is located at the position of card x_b. Now assume that the result holds for n. Let e be the edge incident to x that rings at time n and let x' be the neighbor of x across e. Let A_1, A_2, and A_3 be the events corresponding to the following three possible values of $(W_n \cap \{x, x'\}, W_{n+1} \cap \{x, x'\})$ when $x \in W_{n+1}$:

1. $(\{x, x'\}, \{x, x'\})$,
2. $(\{x'\}, \{x\})$,
3. $(\{x\}, \{x\})$.

Let $\mathcal{F}_n = \sigma(\rho_n(x), \rho_n(x'))$. Note that

$$\mathbf{E}\Big(\rho_{n+1}(x) \;\Big|\; \mathcal{F}_n, W_1, W_2, \dots\Big) = \Big(\tfrac{1}{2}\rho_n(x) + \tfrac{1}{2}\rho_n(x')\Big)\mathbf{1}(A_1) + \rho_n(x')\mathbf{1}(A_2) + \rho_n(x)\mathbf{1}(A_3). \tag{1}$$

Define $\mu_n(\cdot) = \mathbf{P}\Big(X_n(x_b) = \cdot \;\Big|\; W_1, W_2, \dots\Big)$. Then

$$\mu_{n+1}(x) = \Big(\tfrac{1}{2}\mu_n(x) + \tfrac{1}{2}\mu_n(x')\Big)\mathbf{1}(A_1) + \mu_n(x')\mathbf{1}(A_2) + \mu_n(x)\mathbf{1}(A_3). \tag{2}$$

But by induction we have

$$\mu_n(x) = \mathbf{E}\Big(\rho_n(x) \;\Big|\; W_1, W_2, \dots\Big), \qquad \mu_n(x') = \mathbf{E}\Big(\rho_n(x') \;\Big|\; W_1, W_2, \dots\Big).$$

To complete the proof, take the conditional expectation given $W_1, W_2, \dots$ of both sides of (1) and combine with (2). □

Remark. Note that

$$\mathbf{E}\Big(\sum_x \rho_n(x) \;\Big|\; W_1, W_2, \dots\Big) = \sum_x \mathbf{P}\Big(X_n(b) = x \;\Big|\; W_1, W_2, \dots\Big) = 1. \tag{3}$$

2.2. A useful lemma. Let A and B be disjoint sets of cards. For $x \in A$, say that x is *antisocial in round j* of the zigzag shuffle if at no point in round j does an edge connecting x to a card in B ring. Let $Z(A, B, j)$ denote the number of cards that are antisocial in round j. We say that A *avoids* B if $Z(A, B, j) > \frac{7}{8}|A|$ for $64cd$ consecutive rounds j before time m. If S is a set of cards, say that S *circulates* if there do not exist disjoint sets A, B of cards with $|A| \le \frac{1}{2}|S|$ and $A \cup B = S$ such that A avoids B.

Fix a set of cards $S = \{x_1, \dots, x_k\}$ and consider the corresponding chameleon process. Let $\mathcal{F} = \sigma(X_n(S) : n \ge 0)$. Let $Z_n = \sum_x \rho_{128cd^2 n}(x)$ be the total amount of "red paint" in the system after $64cdn$ rounds of the chameleon process. (Recall that a round is defined as $2d$ steps.) Define $Z_n^\sharp = \min(Z_n, k - Z_n)$. Note that $\lim_{n\to\infty} Z_n^\sharp = 0$ almost surely.

We will need the following lemma.

LEMMA 2.2. *We have*

$$\mathbf{E}\Big(\sqrt{Z_n^\sharp} \;\Big|\; \mathcal{F}, S \text{ circulates}\Big) \le \exp\Big[-\frac{n}{8224d^2}\Big] \tag{4}$$

for all n.

Proof. Fix n such that $64cd^2 n \le m$, and let A_n be either the set of cards that are red or the set of cards that are white at the start of round $64cdn$, according to whether $Z_n \le k/2$ or $Z_n > k/2$, respectively. Let P denote the number of cards pinkened during the next $64cd$ rounds. Let $B_n = S - A_n$. When S circulates, A_n doesn't avoid B_n. We claim that this ensures that $P \ge \frac{|A_n|}{8d}$. Consider a round j such that $Z(A_n, B_n, j) \le \frac{7}{8}|A_n|$. Note that after an edge connecting a card x in A_n to a card y in B_n rings, at least one of the resulting cards is pink. Let us associate that

pink card with x. (If both endpoints are pink, then choose one of them arbitrarily.) Since at least a fraction $1/8$ of the cards in A_n will have a pink card associated to them in this round, and since any given pink card can be associated to at most $2d$ cards in A_n in this round, the number of pink cards at the end of this round must be at least $\frac{|A_n|}{16d}$. It follows that $P \geq \frac{|A_n|}{16d}$.

Note that Z_{n+1} is either $Z_n + \frac{1}{2}P$ or $Z_n - \frac{1}{2}P$, with probability $\frac{1}{2}$ each. Thus, if we write E for the event that S does not circulate, then

$$\mathbf{E}\Big(\sqrt{Z^\sharp_{n+1}}\,\Big|\,P, Z_n, \mathcal{F}, E^c\Big) = \mathbf{E}\Big(\tfrac{1}{2}\sqrt{(Z_n + \tfrac{1}{2}P)^\sharp} + \tfrac{1}{2}\sqrt{(Z_n - \tfrac{1}{2}P)^\sharp}\,\Big|\,Z_n, \mathcal{F}, E^c\Big) \tag{5}$$

$$\leq \sqrt{Z^\sharp_n}\,\frac{\sqrt{1+\frac{1}{32d}} + \sqrt{1-\frac{1}{32d}}}{2} \tag{6}$$

$$\leq \sqrt{Z^\sharp_n}\,\exp\Big[-\frac{1}{8224d^2}\Big], \tag{7}$$

where the first inequality follows from the concavity of the square root, and the second inequality follows from the fact that $\frac{1}{2}\sqrt{1+u} + \frac{1}{2}\sqrt{1-u} \leq \exp(-u^2/8)$ whenever $u \in [0,1]$ (see [8, Lemma 9]). The lemma now follows from (7) and the fact that $Z_0 = 1$. □

2.3. Proof of a local mixing property. If S is a set of cards and S' is a set of locations, then we will write $S \to_m S'$ for the event that $X_m(S) = S'$. For $j \in \{1, \dots, 2^d\}$ define

$$\lambda(j) = \max_{|S|=j} \max_{|S'|=j} \Big|\tbinom{2^d}{j}\mathbf{P}(S \to_m S') - 1\Big|.$$

The following is the principal result of this section.

LEMMA 2.3. *Let $S = \{x_1, \dots, x_k\}$ be a set of cards and suppose that $\lambda(k) \leq \frac{1}{2}$. Then for any set S' of locations and $y \in S'$, we have*

$$\Big|\mathbf{P}\Big(X_m(x_k) = y\,\Big|\,S \to_m S'\Big) - \tfrac{1}{k}\Big| \leq 4\alpha^{-d}. \tag{8}$$

Proof. Define $Z_\infty = \lim_{n\to\infty} Z_n$. (Note that for any S' we have $\mathbf{E}(Z_\infty \,|\, S \to_m S') = 1$; see the remark immediately following Lemma 2.1.) Let $\rho_m = \sum_x \rho_m(x) = Z_{m/128cd^2}$. Lemma 2.1 implies that for all $y \in S'$ we have

$$\Big|\mathbf{P}\Big(X_m(x_k) = y\,\Big|\,S \to_m S'\Big) - \tfrac{1}{k}\Big| = \Big|\mathbf{E}\Big(\rho_m(y) - \tfrac{1}{k}Z_\infty\,\Big|\,S \to_m S'\Big)\Big| \tag{9}$$

$$\leq \mathbf{E}\Big(|\rho_m(y) - \tfrac{1}{k}Z_\infty|\,\Big|\,S \to_m S'\Big) \tag{10}$$

$$\leq \mathbf{P}(\rho_m \notin \{0,k\} \,|\, S \to_m S'). \tag{11}$$

Let E be the event that S does not circulate. Since $\lambda(k) \leq \frac{1}{2}$, Lemma 7.1 in Appendix A gives $\mathbf{P}(E \,|\, S \to_m S') \leq 3\alpha^{-d}$. Hence

$$\mathbf{P}\Big(\rho_m \notin \{0,k\}\,\Big|\,S \to_m S'\Big) \leq \mathbf{P}\Big(E\,\Big|\,S \to_m S'\Big) + \mathbf{P}\Big(\rho_m \notin \{0,k\}\,\Big|\,S \to_m S', E^c\Big) \tag{12}$$

$$\leq 3\alpha^{-d} + \mathbf{P}\Big(\rho_m \notin \{0,k\}\,\Big|\,S \to_m S', E^c\Big). \tag{13}$$

But

$$\mathbf{P}\Big(\rho_m \notin \{0,k\}\,\Big|\,S \to_m S', E^c\Big) \leq \mathbf{E}\Big(\sqrt{Z^\sharp_{m/64cd^2}}\,\Big|\,S \to_m S', E^c\Big) \tag{14}$$

$$\leq \exp\Big[-\frac{m}{8224 \cdot 64 \cdot cd^4}\Big] \leq \alpha^{-d}, \tag{15}$$

where the second inequality follows from Lemma 2.2. Combining (11), (13), and (15) yields the lemma. □

3. Indistinguishable cards mix in poly time. Let Λ be a set of cards. In this section we show that the uniform mixing time for the Markov chain $\{X_n(\Lambda) : n \geq 0\}$ is $O(d^5)$.

Recall that in the previous section we showed that if $k > 2^{d-1}$ and every set S with $|S| = k$ is roughly uniformly distributed after bd^5 steps, then for any such set S, any card $x \in S$ will be roughly mixed after bd^5 steps. But note that if a set of cards S is mixed and $x \in S$ is mixed within S, then $S - \{x\}$ must also be mixed. Iterating this (starting with S being the set of all cards) gives the main result of this section.

LEMMA 3.1. *There is a positive integer constant b such that if $m = bd^5$, then*

$$\max_{\Lambda,\Lambda'} \left| \tbinom{2^d}{|\Lambda|} \mathbf{P}(\Lambda \to_m \Lambda') - 1 \right| \leq \tfrac{1}{4}. \tag{16}$$

Proof. It is enough to consider sets Λ with $|\Lambda| \geq 2^{d-1}$. (Otherwise, consider Λ^c.) Let α, β, and c be defined as in section 2, let $b = \lceil \beta \log \alpha \, c \rceil$, and let $m = bd^5$. Let $\lambda(\cdot)$ be defined as in the previous section. We will show that for all $k \geq 2^{d-1}$, we have

$$\lambda(k) \leq k^\star 4^{-d}, \tag{17}$$

where $k^\star = 2^d - k$. This yields the lemma because the RHS of (17) is at most $\frac{1}{4}$ for all $d \geq 1$.

We will verify (17) by induction on $k^\star$. The base case $k^\star = 0$ ($k = 2^d$) is trivial. Suppose it is true for k, where $k > 2^{d-1}$, and consider $k - 1$. Fix a set of cards Λ with $|\Lambda| = k - 1$ and let $z \notin \Lambda$. Define $\Lambda_z = \Lambda \cup \{z\}$. Fix a set Λ' of vertices of the hypercube with $|\Lambda'| = k - 1$. For $w \notin \Lambda'$, let $\Lambda'_w = \Lambda' \cup \{w\}$, and define

$$x_w = \mathbf{P}(\Lambda_z \to_m \Lambda'_w), \qquad \Delta x_w = x_w - \tbinom{2^d}{k}^{-1}, \tag{18}$$

$$y_w = \mathbf{P}(z \to_m w \,|\, \Lambda_z \to_m \Lambda'_w), \qquad \Delta y_w = y_w - 1/k. \tag{19}$$

Note that $\left|\{w : w \notin \Lambda'\}\right| = k^\star + 1$, and $\frac{k^\star+1}{k}\tbinom{2^d}{k}^{-1} = \tbinom{2^d}{k-1}^{-1}$. It follows that

$$\left|\mathbf{P}(\Lambda \to_m \Lambda') - \tbinom{2^d}{k-1}^{-1}\right| = \left| \sum_{w \notin \Lambda'} \mathbf{P}\Big(\Lambda_z \to_m \Lambda'_w, \{z\} \to_m \{w\}\Big) - \tfrac{1}{k}\tbinom{2^d}{k}^{-1} \right| \tag{20}$$

$$= \left| \sum_{w \notin \Lambda'} x_w y_w - \tfrac{1}{k}\tbinom{2^d}{k}^{-1} \right| \tag{21}$$

$$= \left| \sum_{w \notin \Lambda'} \Delta x_w \tfrac{1}{k} + \Delta y_w \tbinom{2^d}{k}^{-1} + \Delta x_w \Delta y_w \right|. \tag{22}$$

Note that

$$|\Delta x_w| \leq k^\star 4^{-d} \tbinom{2^d}{k}^{-1} \leq \tbinom{2^d}{k}^{-1}, \tag{23}$$

where the first inequality is induction and the second inequality holds because $k^\star \leq 2^d$. Induction also implies that $\lambda(k) \leq \frac{1}{2}$, hence Lemma 2.3 implies that

$$|\Delta y_w| \leq 4\alpha^{-d} \leq \tfrac{1}{2k} 4^{-d} \tag{24}$$

for all $d \geq 1$, where the second inequality follows from the definition of α. Thus, using (23), (24), and the triangle inequality, (22) becomes

$$\begin{aligned}\left|\mathbf{P}(\Lambda \to_m \Lambda') - \tbinom{2^d}{k-1}^{-1}\right| &\leq \frac{k^\star + 1}{k}\left[k^\star 4^{-d}\tbinom{2^d}{k}^{-1} + 4^{-d}\tbinom{2^d}{k}^{-1}\right] \\ &= \frac{1}{k}(k^\star+1)^2 4^{-d}\tbinom{2^d}{k}^{-1} \\ &= (k^\star+1)4^{-d}\tbinom{2^d}{k-1}^{-1} = (k-1)^\star 4^{-d}\tbinom{2^d}{k-1}^{-1}.\end{aligned}$$

Since this is true for all Λ with $|\Lambda| = k-1$, the proof is complete. □

Let K be the transition kernel for one *round* of the Thorp shuffle (i.e., d steps), and let K^t be the transpose of K, defined by $K^t(x,y) = K(y,x)$. Note that K^t is the time-reversal of K. Let $\hat{K} := KK^t$ be the transition kernel for one round of the zigzag shuffle. Let $\{Z_n : n \geq 0\}$ be a Markov chain with transition kernel $\hat{K}$. Then Lemma 3.1 implies that for any set of cards B, the uniform mixing time for the process $\{Z_n(B) : n \geq 0\}$ is at most bd^4. Thus, using standard facts about geometric convergence and the uniform mixing time, we can conclude that for a universal constant C we have

$$\max_{B'} \tbinom{2^d}{|B|}\mathbf{P}(Z_{kCd^4}(B) = B') \leq 1 + e^{-k} \tag{25}$$

for all $k \geq 1$.

Definition: Truncated Thorp shuffle. Fix $d_\star \leq d$. Define the *$d_\star$-truncated Thorp shuffle* as the Markov chain with transition kernel $K_\star = K_1, \ldots, K_{d_\star}$. This is a "partial round" of the Thorp shuffle, with steps $d_\star + 1$ through d censored. To make things irreducible, we define the state space as the set of states reachable from an (arbitrary) fixed starting state.

Define the $d_\star$-truncated zigzag shuffle as the Markov chain with transition kernel $K_\star K_\star^t$. Note that we can think of this shuffle as a product of $2^{d-d_\star}$ copies of a "$d_\star$-dimensional" zigzag shuffle, where the cards occupy $2^{d-d_\star}$ (disconnected) hypercubes of dimension $d_\star$. Combining this observation with (25) yields the following corollary to Lemma 3.1.

COROLLARY 3.2. *Fix $d_\star \geq 2$, let $\{Z_n : n \geq 0\}$ be the $d_\star$-truncated zigzag shuffle, and let $\mathcal{S}$ denote the state space of $\{Z_n\}$. There is a universal constant c such that if $l = kcd(d_\star - 1)^4$, then*

$$|\mathcal{S}| \max_{B'} \mathbf{P}(Z_l(B) = B') \leq \exp(\exp(-k)). \tag{26}$$

Proof. Let $c = 2^5 C$. Then $l \geq 2kdCd_\star^4$, so (25) implies that

$$\begin{aligned}|\mathcal{S}| \max_{B'} \mathbf{P}(Z_l(B) = B') &\leq (1 + e^{-2kd})^{2^{d-d_\star}} \\ &\leq \exp(2^d \exp(-2dk)) \\ &\leq \exp(\exp(-k))\end{aligned}$$

for all $d \geq 1$. □

4. Evolving sets and ℓ^2 techniques. In previous sections we have found that after $O(d^5)$ steps (of the zigzag shuffle), subsets of the deck are fairly well mixed. We would like to show that the Thorp shuffle mixes the deck in the sense of the global

property that every ordering of cards is roughly equally likely. In order to do this, we need to find the best global property of random permutations (and the effect of a Thorp shuffle) to study. In this section we introduce the notion of evolving sets, and proceed to find that the global property of interest will be an ℓ^2 norm on the transition kernel.

4.1. Evolving sets. We will now give a brief overview of evolving sets (see [8] for a more detailed account). Let $\{p(x,y)\}$ be transition probabilities for an irreducible a periodic Markov chain on a finite state space V. Assume that the chain has a uniform stationary distribution (which means that p is doubly stochastic: $\sum_{x\in V} p(x,y) = 1$ for all $y \in V$). For subsets $S \subset V$, define $p(S,y) := \sum_{x\in S} p(x,y)$.

Definition: Evolving sets. The *evolving set* process is the Markov chain $\{S_n\}$ on *subsets* of V with the following transition rule. If the current state S_n is $S \subset V$, choose U uniformly from $[0,1]$ and let the next state S_{n+1} be

$$\widetilde{S} = \{y : p(S,y) \geq U\}.$$

Write $\mathbf{P}_S(\,\cdot\,) := \mathbf{P}(\,\cdot\,|\,S_0 = S)$ and similarly for $\mathbf{E}_S(\,\cdot\,)$. Evolving sets have the following properties (see [8]):

1. The sequence $\{|S_n|\}_{n\geq 0}$ forms a martingale.
2. For all $n \geq 0$ and $x, y \in V$ we have
$$p^n(x,y) = \mathbf{P}_{\{x\}}(y \in S_n).$$
3. The sequence of complements $\{S_n^c\}_{n\geq 0}$ is also an evolving set process, with the same transition probabilities.

As in [8], we will prove our mixing time bound using an isoperimetric quantity that we denote by ψ, which is defined as follows. For $S \subset V$, define

$$\psi(S) := 1 - \mathbf{E}_S\sqrt{\frac{|\widetilde{S}|}{|S|}}.$$

Define $\psi(x)$ for $x \in [0, 1/2]$ by

$$\psi(x) = \inf\{\psi(S) : |S| \leq x|V|\}, \tag{27}$$

and for $x > 1/2$, let $\psi(x) := \psi_* = \psi(\frac{1}{2})$. Observe that ψ is nonnegative and (weakly) decreasing on $[0,\infty)$. We will call the function ψ the *root profile.*

4.2. From ℓ^2 bounds to a bound on ψ. In this section, we show how to use l^2 techniques to obtain a bound on the root profile.

For functions $f : V \to [0,1]$, define $||f||_1 := \frac{1}{|V|}\sum_{x\in V} f(x)$ and $||f||_2 := (\frac{1}{|V|}\sum_{x\in V} f(x)^2)^{1/2}$. For $S \subset V$, define $\mathbf{1}_S : V \to [0,1]$ by

$$\mathbf{1}_S(x) = \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{otherwise.} \end{cases}$$

LEMMA 4.1. *Let $\widetilde{S}$ be the next step in the evolving set process starting from S, i.e., $\widetilde{S} = \{y : p(S,y) > U\}$, where U is uniform. Let $\alpha = \frac{||p(S,\cdot)||_2^2}{||\mathbf{1}_S||_1}$. Then*

$$\mathbf{E}\left(\sqrt{\frac{|\widetilde{S}|}{|S|}}\right) \leq [\alpha(2-\alpha)]^{\frac{1}{4}}.$$

Proof. Let Λ be an independent copy of $\widetilde{S}$, i.e., $\Lambda = \{y : p(S,y) > U'\}$, for an independent uniform random variable U'. Note that either $\widetilde{S} \subseteq \Lambda$ or $\Lambda \subseteq \widetilde{S}$ (depending on which of the uniform variables U, U' is larger). Let $X = |\widetilde{S} \cap \Lambda|$ and $Y = |\widetilde{S} \cup \Lambda|$. Then

$$\left[\mathbf{E}\left(\sqrt{|\widetilde{S}|}\right)\right]^2 = \mathbf{E}\left(\sqrt{|\widetilde{S}||\Lambda|}\right) \tag{28}$$

$$= \mathbf{E}(\sqrt{XY}) \tag{29}$$

$$\leq \sqrt{\mathbf{E}(X)\mathbf{E}(Y)} \tag{30}$$

$$= \sqrt{\mathbf{E}(X)(2|S| - \mathbf{E}(X))}, \tag{31}$$

where the first inequality is Cauchy Schwarz and the second inequality follows from the fact that $\mathbf{E}(X+Y) = 2\mathbf{E}(\widetilde{S}) = 2|S|$. But

$$\mathbf{E}(X) = \sum_{y \in V} \mathbf{P}(y \in \widetilde{S} \cap \Lambda) \tag{32}$$

$$= \sum_{y \in V} \mathbf{P}(y \in \widetilde{S})^2 \tag{33}$$

$$= \sum_{y \in V} p(S,y)^2 = |V| \cdot ||p(S,\cdot)||_2^2, \tag{34}$$

so dividing the LHS of (28) and the RHS of (31) by $|S| = |V| \cdot ||\mathbf{1}_S||_1$ and then taking a square root yields the lemma. □

Remark. If we define $\Delta := 1 - \alpha$, then

$$\left[\alpha(2-\alpha)\right]^{\frac{1}{4}} = (1-\Delta^2)^{\frac{1}{4}} \leq 1 - \frac{\Delta^2}{4}.$$

5. A bound on the root profile. We will need the following technical result, which is proved in Appendix B.

Corollary 8.5. *Fix $S \subset V$ and let $x = \frac{|S|}{(2^d)!} = ||\mathbf{1}_S||_1$. Let $p(\,\cdot\,,\,\cdot\,)$ be the transition kernel for one round of the Thorp shuffle. Then there is a universal constant $C > 0$ such that*

$$||p(S,\cdot)||_2^2 \leq x^{1+C/d^{14}}.$$

We are now ready to obtain a bound on the root profile of the Thorp shuffle.

LEMMA 5.1. *Let ψ be the root profile of the Markov chain that each step performs a round of the Thorp shuffle ($K_1 K_2 \cdots K_d$). There is a universal constant $\gamma > 0$ such that*

$$\psi(x) \geq \max\left(1 - x^{\gamma/2d^{42}}, \gamma d^{-28}\right). \tag{35}$$

Proof. Let C be the constant appearing in Corollary 8.5. We will show that there is a universal constant $B > 0$ such that

$$\psi(x) \geq \max\left(1 - x^{CB/2d^{42}}, Bd^{-28}\right). \tag{36}$$

Setting $\gamma = \min(BC, B)$ will then yield the lemma. First, we show that $\psi_* \geq Bd^{-28}$. Fix S with $\frac{|S|}{(2^d)!} = x \leq \frac{1}{2}$ and let

$$\widetilde{S} = \{y : p(S,y) > U\},$$

where $\{p(x,y)\}$ are the transition probabilities for one round of the Thorp shuffle. The remark following Lemma 4.1 implies that

$$\mathbf{E}\sqrt{\frac{|\widetilde{S}|}{|S|}} \le 1 - \frac{\Delta^2}{4},$$

where $\Delta = 1 - \frac{||p(S,\cdot)||_2^2}{||\mathbf{1}_S||_1}$, and Corollary 8.5 implies that $||p(S,\cdot)||_2^2 \le x^{C/d^{14}}||\mathbf{1}_S||_1 \le 2^{-C/d^{14}}||\mathbf{1}_S||_1$. Thus

$$\Delta \ge 1 - 2^{-Cd^{-14}} \tag{37}$$

$$= 1 - e^{-C\log 2\, d^{-14}} \tag{38}$$

$$\ge Ad^{-14} \tag{39}$$

for a universal constant $A > 0$, and hence $1 - \frac{\Delta^2}{4} \le 1 - Bd^{-28}$ for a universal constant $B \in (0, \frac{1}{4})$. (The fact that we can take $B < \frac{1}{4}$ will be used later on.) Since this holds for all S with $|S| \le \frac{1}{2}(2^d)!$, we conclude that $\psi_* \ge Bd^{-28}$. To complete the proof of Lemma 5.1, we must show that (36) holds when the max is achieved by the first term. Suppose that $1 - x^{CB/2d^{42}} \ge Bd^{-28}$. Then

$$x \le (1 - Bd^{-28})^{2d^{42}/CB} \le \exp(-2C^{-1}d^{14}). \tag{40}$$

Recall that Lemma 4.1 gives

$$\mathbf{E}\sqrt{\frac{|\widetilde{S}|}{|S|}} \le (\alpha(2-\alpha))^{\frac{1}{4}} \le (2\alpha)^{\frac{1}{4}}, \tag{41}$$

where $\alpha = \frac{||p(S,\cdot)||_2^2}{||\mathbf{1}_S||_1}$. Thus, (40) implies that

$$x^{C/2d^{14}} \le e^{-1} < \tfrac{1}{2},$$

and hence

$$2 \le x^{-C/2d^{14}}. \tag{42}$$

Furthermore, Corollary 8.5 implies that $\alpha \le x^{C/d^{14}}$. Plugging this and (42) into (41) gives

$$\mathbf{E}\sqrt{\frac{|\widetilde{S}|}{|S|}} \le (x^{-C/2d^{14}}x^{C/d^{14}})^{\frac{1}{4}} = x^{C/8d^{14}} \le x^{CB/2d^{42}}, \tag{43}$$

since $B < \frac{1}{4}$ (and $x \le 1$). □

6. Proof of main result. *Proof of Theorem* 1.1. We shall start by bounding the mixing time of the Markov chain that does an entire round of the Thorp shuffle each step. Recall that the root profile $\psi : [0,\infty) \to \mathbf{R}$ is defined by

$$\psi(x) = \begin{cases} \inf\{\psi(S) : |S| \le x|V|\} & \text{if } x \in [0, \frac{1}{2}], \\ \psi_* & \text{if } x > \frac{1}{2}, \end{cases}$$

where $\psi_* = \psi(\frac{1}{2})$. Thus ψ is (weakly) decreasing on $[0,\infty)$.

Let $h(z) := 1 - \psi(1/z^2)$. Since $\psi(x) = \psi_*$ for all real numbers $x \geq \frac{1}{2}$, the function h is well defined even for $z \leq 1$. Note that h is nonincreasing. In [8] it is shown (see section 5 and the part of section 3 entitled "Derivation of Theorem 1 from Lemma 3 and Theorem 4") that there is a sequence of random variables $\{Z_n : n \geq 0\}$ that satisfies $Z_0 = \sqrt{|V|}$ and

$$\mathbf{E}\left(\frac{Z_{n+1}}{Z_n}\Big| Z_n\right) \leq h(Z_n) \tag{44}$$

such that

$$\tau_{\text{mix}} \leq 2 \min\{n : \mathbf{E}(Z_n) \leq \tfrac{1}{2}\}. \tag{45}$$

Lemma 5.1 gave the following bound on the root profile:

$$\psi(x) \geq \max\Big(1 - x^{\gamma/2d^{42}}, \gamma d^{-28}\Big) \tag{46}$$

for a universal constant $\gamma > 0$. Thus $h \leq g$, where g is defined by

$$g(z) = \min\Big(z^{-\gamma/d^{42}}, 1 - \gamma d^{-28}\Big),$$

and hence $\mathbf{E}(Z_{n+1}|Z_n) \leq g(Z_n)Z_n$. Let $f(z) = zg(z) = \min(z^{1-\gamma/d^{42}}, z(1 - \gamma d^{-28}))$. Note that f is increasing and, as the minimum of two concave functions, is concave. We claim that $\mathbf{E}(Z_n) \leq f^n(Z_0)$, where f^n is the n-fold iterate of f. We verify this by induction. The base case $n = 0$ is immediate. Suppose that the claim holds for n. Then

$$\mathbf{E}(Z_{n+1}) = \mathbf{E}(\mathbf{E}(Z_{n+1}|Z_n)) \tag{47}$$
$$\leq \mathbf{E}(f(Z_n)) \tag{48}$$
$$\leq f(\mathbf{E}(Z_n)) \tag{49}$$
$$\leq f(f^n(Z_0)) = f^{n+1}(Z_0), \tag{50}$$

where the third line follows from concavity and the last line is the induction hypothesis. Let

$$f_1 = z^{1-\gamma/d^{42}}, \qquad f_2 = z(1 - \gamma d^{-28}),$$

so that $f = \min(f_1, f_2)$. Then for all m, n we have

$$\mathbf{E}(Z_{m+n}) \leq f^{m+n}(\sqrt{Z_0}) \leq f_2^m(f_1^n(Z_0)).$$

But $f_1^n(z) = z^{(1-\gamma/d^{42})^n} \leq z^{\exp(-\gamma n/d^{42})}$, and $Z_0 = \sqrt{|V|} \leq (2^d)^{2^d} = 2^{d2^d}$. Thus, choosing $n \geq \gamma^{-1}d^{43}$ gives

$$f_1^n(Z_0) \leq 2^{d2^d e^{-d}},$$

which is at most 4 for all $d \geq 1$. Finally, since

$$f_2^m(z) = z\Big(1 - \gamma d^{-28}\Big)^m \leq ze^{-\gamma m/d^{28}},$$

we have $f_2^m(4) \leq 4e^{-\gamma m/d^{28}}$, which is at most $\frac{1}{2}$ whenever $m \geq \gamma^{-1}d^{28}\log 8$. Putting this together, we conclude that $\tau_{\text{mix}} \leq 2\gamma^{-1}(d^{43} + d^{28}\log 8) = O(d^{43})$. Since each round corresponds to d Thorp shuffles we conclude that the mixing time for the original model is $O(d^{44})$. □

7. Appendix A. In this section we prove some technical results needed in section 3. We will adopt the notation of that section; for the convenience of the reader, we now give a brief recap. Let A and B be disjoint sets of cards. For $x \in A$, say that x is *antisocial in round* j of the zigzag shuffle if at no point in round j does an edge connecting x to a card in B ring. Let $Z(A, B, j)$ denote the number of cards that are antisocial in round j. We say that A *avoids* B if $Z(A, B, j) > \frac{7}{8}|A|$ for $64cd$ consecutive rounds j before time m. If S is a set of cards, say that S *circulates* if there do not exist disjoint sets A, B of cards with $|A| \le \frac{1}{2}|S|$ and $A \cup B = S$ such that A avoids B.

The main purpose of this section is to prove Lemma 7.1, which was used in the proof of Lemma 2.3 in section 3. The proof of Lemma 7.1 uses some fairly technical large deviation results. For the convenience of the reader, we shall prove Lemma 7.1 first, and the large deviation results second.

LEMMA 7.1. *Fix a set of cards S with $|S| \ge 2^{d-1}$. Then for any set S' of vertices of the hypercube we have*

$$\mathbf{P}\Big(S \text{ does not circulate} \,\Big|\, S \to_m S'\Big) \le \alpha^{-d} \frac{1 + \lambda(|S|)}{1 - \lambda(|S|)}.$$

Proof. Let E be the event that S does not circulate. We have

$$\begin{aligned}\mathbf{P}\Big(E, S \to_m S'\Big) &\le \sum_{k \le \frac{1}{2}|S|} \sum_{A:|A|=k} \mathbf{P}\Big(A \text{ avoids } B, S \to_m S'\Big) \\ &\le 2^{d-1} \max_k \Big[2^{dk} \max_{A:|A|=k} \mathbf{P}\Big(A \text{ avoids } B, S \to_m S'\Big)\Big],\end{aligned}$$

where in the summations we write B for $S - A$, the 2^{d-1} is an upper bound on the number of $k \le \frac{1}{2}|S|$, and the 2^{dk} is an upper bound on the number of sets A with $|A| = k$. Since $|A| \le \frac{1}{2}|S|$ and $A \cup B = S$, we must have $|B| \ge \frac{1}{4}2^d$. Hence if $|A| = k$, then

$$\mathbf{P}\Big(A \text{ avoids } B, S \to_m S'\Big) \le \sum_{B' \subset S'} \mathbf{P}\Big(A \text{ avoids } B, B \to_m B', A \to_m S' - B'\Big) \tag{51}$$

$$\le \sum_{B' \subset S'} \mathbf{P}(B \to_m B') \mathbf{P}\Big(A \text{ avoids } B \,\Big|\, B \to_m B'\Big). \tag{52}$$

But

$$\begin{aligned}&\mathbf{P}\Big(A \text{ avoids } B \;\Big|\; B \to_m B'\Big) \\ &\le \sum_{i=0}^{m} \prod_{j=i}^{i+64cd-1} \mathbf{P}\Big(Z(A, B, j) > \tfrac{7k}{8} \,\Big|\, B \to_m B'\Big) \le m\Big(e^{-k/64}\Big)^{64cd},\end{aligned} \tag{53}$$

where the last inequality follows from Corollary 7.3 below. (See Lemma 7.2 and Corollary 7.3 immediately following the present proof.) Hence

$$\begin{aligned}\mathbf{P}\Big(A \text{ avoids } B, S \to_m S'\Big) &\le \sum_{B' \subset S'} \mathbf{P}(B \to_m B') m e^{-ckd} \\ &\le 2^{dk} m e^{-ckd} \max_{B'} \mathbf{P}(B \to_m B'),\end{aligned}$$

where the 2^{dk} is an upper bound on the number of subsets $B' \subset S'$. But for any B' we have

$$\mathbf{P}(B \to_m B') \le \sum_{\hat{S}:\hat{S}\supset B'} \mathbf{P}(S \to_m \hat{S}) \le 2^{dk} \tbinom{2^d}{|S|}^{-1} (1 + \lambda(|S|)).$$

It follows that

$$\mathbf{P}\Big(A \text{ avoids } B, S \to_m S'\Big) \le 4^{dk} m e^{-cdk} \tbinom{2^d}{|S|}^{-1} (1 + \lambda(|S|)) \tag{54}$$

$$= [4e^{-c}]^d \lceil \beta \log \alpha\, c \rceil d^5 \tbinom{2^d}{|S|}^{-1} (1 + \lambda(|S|)) \tag{55}$$

$$\le \alpha^{-d} \tbinom{2^d}{|S|}^{-1} (1 + \lambda(|S|)), \tag{56}$$

where the second inequality follows from the definition of c. Finally, since $\mathbf{P}(S \to_m S') \ge \binom{2^d}{|S|}^{-1}(1 - \lambda(|S|))$, we get $\mathbf{P}(A \text{ avoids } B \,|\, S \to_m S') \le \alpha^{-d} \frac{1+\lambda(|S|)}{1-\lambda(|S|)}$. □

Lemma 7.1 used a corollary to the following lemma.

LEMMA 7.2. *Let $\{X_n : n \ge 0\}$ be the zigzag shuffle. Let $Z = Z(A, B, 1)$ be the number of cards that are antisocial in the first round. Define $\mathcal{F}_B = \sigma(X_1(B), \dots, X_d(B))$. Let $p = 1 - \frac{|B|}{2^d}$ and let $k = |A|$. For $\theta \ge 0$ define $\Phi_p(\theta) = 1 - p + pe^\theta$. Then for all $\theta \ge 0$ we have*

$$\mathbf{E}\Big(e^{\theta Z} \,\Big|\, \mathcal{F}_B\Big) \le \Phi_p(\theta)^k. \tag{57}$$

Proof. We verify this by induction on d. If $d = 1$, then the LHS of (57) is 1 if $p < 1$, and $e^{\theta k}$ otherwise, so (57) holds. Now suppose that $d > 1$. Let A' be the set of cards in A not adjacent to B in direction 1, and let $k' = |A'|$. Let l be half the number of cards in A' adjacent to another card in A' in direction 1. (Note that l is an integer.) Let k_0 and k_1 be the number of cards in A' that end up with a leading 0 and 1, respectively, after the first step of the round (i.e., after the edges in direction 1 ring). Of those in the first group, let Z_0 be the number that are antisocial, with a similar definition for Z_1. Note that given $\mathcal{F}_B$, the random variables k_0 and k_1 are both distributed like $W + l$, where $W \sim \text{Binomial}(k' - 2l, \frac{1}{2})$, and note that $Z = Z_0 + Z_1$. By induction, we have

$$\begin{aligned}\mathbf{E}\Big(e^{\theta Z} \,\Big|\, \mathcal{F}_B, X_1(A)\Big) &= \mathbf{E}\Big(e^{\theta Z_0} \,\Big|\, \mathcal{F}_B, X_1(A)\Big)\mathbf{E}\Big(e^{\theta Z_1} \,\Big|\, \mathcal{F}_B, X_1(A)\Big) \\ &\le \Phi_{p_0}(\theta)^{k_0} \Phi_{p_1}(\theta)^{k_1},\end{aligned}$$

where p_0 is the fraction of locations of the part of the hypercube with a leading 0 not occupied by a card in B after the first step, with a similar definition for p_1. It follows that $\mathbf{E}(e^{\theta Z} \,|\, \mathcal{F}_B, k_0, k_1) \le \Phi_{p_0}(\theta)^{k_0} \Phi_{p_1}(\theta)^{k_1}$. Hence

$$\begin{aligned}\mathbf{E}\Big(e^{\theta Z} \,\Big|\, \mathcal{F}_B\Big) &\le \sum_{i=0}^{k'-2l} (\tfrac{1}{2})^{k'-2l} \binom{k'-2l}{i} \Phi_{p_0}^i(\theta) \Phi_{p_1}^{k'-2l-i}(\theta) \Phi_{p_0}^l(\theta) \Phi_{p_1}^l(\theta) \\ &= \Big[\tfrac{1}{2}\Phi_{p_0}(\theta) + \tfrac{1}{2}\Phi_{p_1}(\theta)\Big]^{k'-2l} \Phi_{p_0}^l(\theta) \Phi_{p_1}^l(\theta) \\ &\le \Big[\tfrac{1}{2}\Phi_{p_0}(\theta) + \tfrac{1}{2}\Phi_{p_1}(\theta)\Big]^{k'} = \Phi_p(\theta)^{k'},\end{aligned}$$

where the last inequality follows from the AM-GM inequality and the final equality holds because $p = \frac{1}{2}(p_0 + p_1)$. This yields the lemma because $k' \le k$. ∎

Lemma 7.2 easily gives the following large deviation inequality, which was used in the proof of Lemma 7.1.

COROLLARY 7.3. *Suppose that $p \le 3/4$. Then*

$$\mathbf{P}\Big(Z > \tfrac{7}{8}k \,\Big|\, \mathcal{F}_B\Big) < e^{-k/64}.$$

Proof. We have

$$\mathbf{E}(e^{\theta(Z-pk)} \mid \mathcal{F}_B) = e^{-pk\theta}\mathbf{E}(e^{\theta Z} \mid \mathcal{F}_B) \tag{58}$$

$$\le \Big[(1-p)e^{-p\theta} + pe^{\theta(1-p)}\Big]^k, \tag{59}$$

by Lemma 7.2. The quantity inside the square brackets is $\mathbf{E}(e^{\theta(Y-p)})$, for a Bernoulli($p$) random variable Y. The inequality $\mathbf{E}(e^W) \le e^{\mathrm{var}(W)}$, valid when $\mathbf{E}(W) = 0$ and $W \le 1$ (see, e.g., [10]), implies that the quantity (59) is at most $\exp(\frac{1}{4}\theta^2 k)$ if $\theta \le 1$. Letting $\theta = \frac{1}{4}$ gives

$$\mathbf{E}\Big(\exp[\tfrac{1}{4}(Z-pk)] \,\Big|\, \mathcal{F}_B\Big) \le e^{k/64}, \tag{60}$$

and hence

$$\mathbf{P}\Big(Z > \tfrac{7}{8}k \,\Big|\, \mathcal{F}_B\Big) = \mathbf{P}\Big(\exp[\tfrac{1}{4}(Z-pk)] > \exp\Big[\tfrac{7k}{32} - \tfrac{pk}{4}\Big] \mid \mathcal{F}_B\Big) \tag{61}$$

$$\le \exp\Big[-\tfrac{7k}{32} + \tfrac{pk}{4}\Big] \exp\Big[\tfrac{k}{64}\Big], \tag{62}$$

by Markov's inequality. Finally, since $p \le 3/4$, the quantity (62) is at most $e^{-k/64}$. ∎

8. Appendix B. The purpose of this section is to prove Corollary 8.5, which is used to bound the root profile. If K is the transition kernel for a Markov chain on the state space V, we will consider K as an operator acting on the space of functions $f : V \to \mathbf{R}$ by

$$Kf(x) = \sum_{y \in V} K(x,y) f(y). \tag{63}$$

We will need the following lemma, which is well known.

LEMMA 8.1. *Let K be a doubly stochastic transition kernel and define $\hat{K} = KK^t$. For any function $g : V \to [0,1]$ and $n \ge 1$ we have*

$$||K^t g||_2^2 \le \langle g, g\rangle^{1-\frac{1}{n}} \langle \hat{K}^n g, g\rangle^{\frac{1}{n}}.$$

Proof. Since $\hat{K}$ is symmetric it is diagonalizable. Thus we can write $g = \sum_i \alpha_i g^i$, where the g^i are orthonormal eigenfunctions of $\hat{K}$ with corresponding eigenvalues λ_i. We have

$$\frac{||K^t g||_2^2}{\langle g, g\rangle} = \frac{\langle \hat{K} g, g\rangle}{\langle g, g\rangle} \tag{64}$$

$$= \frac{\sum_i \alpha_i^2 \lambda_i}{\sum_i \alpha_i^2} \tag{65}$$

$$\le \Big(\frac{\sum_i \alpha_i^2 \lambda_i^n}{\sum_i \alpha_i^2}\Big)^{1/n} = \Big(\frac{\langle \hat{K}^n g, g\rangle}{\langle g, g\rangle}\Big)^{1/n}, \tag{66}$$

by Jensen's inequality. Multiplying both sides by $\langle g, g\rangle$ yields the lemma. □

We will also need the following lemma, which was proved by Keith Ball.

LEMMA 8.2. *Let X be a random variable taking values in $[0,1]$ and suppose that $\mathbf{E}(X) = \mu \leq \frac{1}{2}$. Then for any $p > 1$ we have*

$$\frac{\mathbf{E}(X^p)}{\mu^p} - 1 \leq (\mu^{1-p} - 1)\mathbf{E}\Big|\frac{X-\mu}{\mu}\Big|. \tag{67}$$

Proof. Let $l = \frac{1}{2}\mathbf{E}(|X-\mu|)$. For a given value of l, the LHS of (67) is maximized when X is concentrated on the three values 0, μ, and 1 (because it is a convex function of X). Let p_0, p_μ, and p_1 be the respective probabilities. Then $l = p_1(1-\mu) = p_0\mu$, and hence $p_\mu = 1 - p_0 - p_1 = 1 - \frac{l}{\mu(1-\mu)}$. It follows that

$$\begin{aligned}\frac{\mathbf{E}(X^p)}{\mu^p} - 1 &= \frac{p_1 + p_\mu\mu^p}{\mu^p} - 1 \\ &= l\Big[\frac{1}{\mu^p(1-\mu)} - \frac{1}{\mu(1-\mu)}\Big] \\ &\leq \frac{2l}{\mu}(\mu^{1-p} - 1),\end{aligned}$$

since $1 - \mu \geq \frac{1}{2}$, and the proof is complete. □

Fix $d_\star \leq d$. Recall that the $d_\star$-truncated Thorp shuffle is the Markov chain with transition kernel $K_\star = K_1 \cdots K_{d_\star}$. Let V denote the state space of this chain. We will need the following technical lemma.

LEMMA 8.3. *Fix $f : V \to [0,1]$ and suppose that $||f||_1 \leq 6^{-cdd_\star^6}$. Then*

$$||K_\star^t f||_2^2 \leq ||f||_1^{1+1/cdd_\star^5},$$

where c is the constant appearing in Corollary 3.2.

Proof. We will prove the lemma by induction on $d_\star$. Suppose that $d_\star = 1$. Then the truncated Thorp shuffle makes the distribution uniform over V in one step. Thus,

$$||K_\star^t f||_2^2 = \sum_{x\in V} ||f||_1^2 \frac{1}{|V|} \tag{68}$$

$$= ||f||_1^2 \tag{69}$$

$$\leq ||f||_1^p, \tag{70}$$

for any $p \in [1,2]$, since $||f||_1 \leq 1$. Suppose now that $d_\star \geq 2$ and assume that the result holds for $d_\star - 1$. Define $\mathcal{L}_\star$ as the set of vertices in the cube whose $d_\star^{th}$ coordinate is 0. Let $\mathcal{B}$ denote the collection of subsets b of $\{1,\ldots,2^d\}$ such that $X(b) = \mathcal{L}_\star$ for some $X \in V$ (i.e., there is a configuration $X \in V$ such that the set of cards occupying $\mathcal{L}_\star$ is b). For $b \in \mathcal{B}$, define $V_b = \{X \in V : X(b) = \mathcal{L}_\star\}$. Let $r = ||f||_1$ and for $\Lambda \subset \mathcal{B}$, define

$$V_\Lambda = \cup_{b\in\Lambda} V_b.$$

Let

$$H = \left\{ b \in \mathcal{B} : \frac{||f\mathbf{1}_{V_b}||_1}{||f||_1} \geq \frac{r^{-1/d_\star}}{|\mathcal{B}|} \right\}.$$

Since $\sum_{b\in\mathcal{B}} \frac{||f\mathbf{1}_{V_b}||_1}{||f||_1} = \frac{||f||_1}{||f||_1} = 1$, Markov's inequality implies that

$$\frac{|H|}{|\mathcal{B}|} \leq r^{1/d_\star}. \tag{71}$$

Let $A = V_H$ and let $f_1 = f\mathbf{1}_A$ and $f_2 = f\mathbf{1}_{A^c}$. Then

$$||K_\star^t f||_2^2 = ||K_\star^t f_1 + K_\star^t f_2||_2^2 \leq 2||K_\star^t f_1||_2^2 + 2||K_\star^t f_2||_2^2. \tag{72}$$

We will bound each term on the RHS separately. First, consider $||K_\star^t f_1||_2^2$. Let $\hat{K}$ be the transition kernel for the $d_\star$-truncated zigzag shuffle, i.e, $\hat{K} = K_1 \cdots K_{d_\star} \cdots K_1$. Let $n = cd(d_\star - 1)^4$. Using Corollary 3.2 (with $k = 1$) and combining this with (71) gives $\hat{K}^n(x, V_H) \leq \exp(\exp(-1))r^{1/d_\star}$ for all x. Hence

$$\langle \hat{K}^n f_1, \mathbf{1}_A\rangle = |V|^{-1} \sum_x f_1(x)\hat{K}^n(x, V_H) \tag{73}$$

$$\leq ||f_1||_1 \exp(\exp(-1))r^{1/d_\star}. \tag{74}$$

Finally, Lemma 8.1 gives

$$||K_\star^t f_1||_2^2 \leq \langle f_1, f_1\rangle^{1-1/n}\langle \hat{K}^n f_1, f_1\rangle^{1/n} \tag{75}$$

$$\leq \langle f_1, f_1\rangle^{1-1/n}\langle \hat{K}^n f_1, \mathbf{1}_A\rangle^{1/n}, \tag{76}$$

where the second inequality holds because $f_1 \leq \mathbf{1}_A$. Putting this all together, we get

$$||K_\star^t f_1||_2^2 \leq \langle f_1, f_1\rangle^{1-1/n}\left[||f_1||_1(\exp(\exp(-1)))r^{1/d_\star}\right]^{1/n} \tag{77}$$

$$\leq 2\left(\frac{\langle f_1, f_1\rangle}{||f_1||_1}\right)^{1-1/n} \times ||f_1||_1 \times r^{1/d_\star n}, \tag{78}$$

since $\exp(\frac{1}{n}\exp(-1)) \leq 2$ for all n. Since $n = cd(d_\star - 1)^4$, and $\frac{\langle f_1, f_1\rangle}{||f_1||_1} \leq 1$, we have

$$||K_\star^t f_1||_2^2 \leq 2r^{1/cdd_\star(d_\star-1)^4}||f||_1. \tag{79}$$

Next we bound $||K_\star^t f_2||_2^2$. Since $K_{d_\star}$ is symmetric it contracts l^2. Hence

$$||K_\star^t f_2||_2^2 \leq ||K_{(d_\star-1)} \cdots K_1 f_2||_2^2 \tag{80}$$

$$= \sum_{b\in\mathcal{B}} ||K_{(d_\star-1)} \cdots K_1 f_2 \mathbf{1}_{V_b}||_2^2. \tag{81}$$

Note that $K_1 \cdots K_{(d_\star-1)}$ is just the transition kernel for a $(d_\star - 1)$-truncated Thorp shuffle and that the V_b are communicating classes for this process. Thus, we can use the induction hypothesis to bound each $||K_{(d_\star-1)} \cdots K_1 f_2 \mathbf{1}_{V_b}||_2^2$, provided that the corresponding normalized l^1 norm $\frac{||f_2\mathbf{1}_{V_b}||_1}{||\mathbf{1}_{V_b}||_1}$ is sufficiently small. Define $r_b := \frac{||f_2\mathbf{1}_{V_b}||_1}{||\mathbf{1}_{V_b}||_1}$. We claim that for every $b \in \mathcal{B}$ we have $r_b \leq r^{\frac{d_\star-1}{d_\star}}$. To see this, note that if $b \in H$, then $||f_2\mathbf{1}_{V_b}||_1 = 0$ and the claim holds trivially, so assume $b \notin H$. Then

$$\frac{||f_2\mathbf{1}_{V_b}||_1}{||\mathbf{1}_{V_b}||_1} = ||f_2\mathbf{1}_{V_b}||_1|\mathcal{B}| \tag{82}$$

$$\leq ||f\mathbf{1}_{V_b}||_1|\mathcal{B}| \tag{83}$$

$$\leq r^{\frac{-1}{d_\star}}||f||_1 = r^{\frac{d_\star-1}{d_\star}}, \tag{84}$$

where the first equality holds because $||\mathbf{1}_{V_b}||_1 = |\mathcal{B}|^{-1}$, the second inequality holds because $b \notin H$ (and by the definition of H), and the last equality holds because $||f||_1 = r$. It follows that

$$r_b \leq r^{\frac{d_\star - 1}{d_\star}} \leq 6^{-c(d_\star - 1)d_\star{}^5} \leq 6^{-c(d_\star-1)^6}. \tag{85}$$

Since r_b is the norm of $f_2\mathbf{1}_{V_b}$ when restricted to the smaller state space accessible via $K_1 \cdots K_{d_\star - 1}$, the induction hypothesis gives

$$||K_{(d_\star-1)} \cdots K_1 f_2 \mathbf{1}_{V_b}||_2^2 \leq r_b^{1/cd(d_\star-1)^5} ||f_2\mathbf{1}_{V_b}||_1 \tag{86}$$

$$\leq r^{1/cdd_\star(d_\star-1)^4} ||f_2\mathbf{1}_{V_b}||_1, \tag{87}$$

where the second inequality follows from the first inequality in (85). Combining this with (81) and using the fact that $f_2 \leq f$ gives

$$||K_\star^t f_2||_2^2 \leq r^{1/cdd_\star(d_\star-1)^4} ||f||_1. \tag{88}$$

We are now ready to bound $||K^t f||_2^2$. Combining (88), (79), and (72), we get

$$||K_\star^t f||_2^2 \leq \left(6r^{1/cdd_\star(d_\star-1)^4}\right)||f||_1. \tag{89}$$

Since $(k-1)^{-4} - k^{-4} \geq k^{-5}$ for integers $k \geq 2$, the quantity (89) is at most

$$6r^{1/cdd_\star^5 + 1/cdd_\star^6}||f||_1 \leq r^{1/cdd_\star^5}||f||_1,$$

since $r \leq 6^{-cdd_\star^6}$. This concludes the proof. □

Corollary 8.5 is a consequence of the following lemma, which extends Lemma 8.3 by removing the assumption on $||f||_1$.

LEMMA 8.4. *Let $C = \lceil 2^{15}c^2 15 \log 2 \log 6 \rceil$. Then for any $f : V \to [0,1]$ we have*

$$||K_\star^t f||_2^2 \leq ||f||_1^{1+1/Cd^2 d_\star^{12}}.$$

Proof. Again, our proof will be by induction on $d_\star$. The base case $d_\star = 1$ can be handled identically as in the proof of Lemma 8.3, so assume that $d_\star \geq 2$ and suppose that the result holds for $d_\star - 1$. Define $r = ||f||_1$. We may assume that $r > 6^{-cdd_\star^6}$, since otherwise we can invoke Lemma 8.3.

We can also assume w.l.o.g. that $r \leq \frac{1}{2}$. Otherwise, let $h = 1 - f$, and suppose that the result holds for h, i.e., for $q = 1/Cd^2 d_\star^{12}$ we have

$$||K_\star^t h||_2^2 \leq ||h||_1^{1+q},$$

or equivalently,

$$||h||_1 - ||K_\star^t h||_2^2 \geq \left[1 - ||h||_1^q\right]||h||_1. \tag{90}$$

Note that

$$||K_\star^t h||_2^2 = \langle K_\star^t(1-f), K_\star^t(1-f)\rangle \tag{91}$$

$$= \langle K_\star^t \mathbf{1}, K_\star^t \mathbf{1}\rangle - 2\langle K_\star^t \mathbf{1}, K_\star^t f\rangle + \langle K_\star^t f, K_\star^t f\rangle \tag{92}$$

$$= 1 - 2||f||_1 + ||K_\star^t f||_2^2 \tag{93}$$

$$= ||h||_1 - ||f||_1 + ||K_\star^t f||_2^2, \tag{94}$$

where the third equality holds because $K_\star^t$ is doubly stochastic and hence $K_\star^t \mathbf{1} = \mathbf{1}$. Thus

$$||h||_1 - ||K^t h||_2^2 = ||f||_1 - ||K^t f||_2^2. \tag{95}$$

Define $u : [0,1] \to \mathbf{R}$ by

$$u(x) = (1 - x^q)x = \frac{x(1-x)}{1 + x^q + \cdots + x^{1-q}}, \tag{96}$$

so the RHS of (90) is $u(||h||_1)$. Since the numerator on the RHS of (96) is symmetric about $\frac{1}{2}$ and the denominator is increasing, we have $u(x) \geq u(1-x)$ if $x \leq \frac{1}{2}$. This, combined with (95), shows that (90) is still true if we replace the h by f. Thus we can assume henceforth that $r \leq \frac{1}{2}$.

Let $\mathcal{B}$ and V_b be defined as in the proof of Lemma 8.3. Then

$$||K_\star^t f||_2^2 \leq ||K_{(d_\star - 1)} \cdots K_1 f||_2^2 \tag{97}$$

$$= \sum_{b \in \mathcal{B}} ||K_{(d_\star - 1)} \cdots K_1 f \mathbf{1}_{V_b}||_2^2. \tag{98}$$

For $b \in \mathcal{B}$, define $r_b = \frac{||f\mathbf{1}_{V_b}||_1}{||\mathbf{1}_{V_b}||_1} = ||f\mathbf{1}_{V_b}||_1 \, |\mathcal{B}|$. Then induction gives

$$||K_{(d_\star - 1)} \cdots K_1 f \mathbf{1}_{V_b}||_2^2 \leq r_b^{1/Cd^2(d_\star - 1)^{12}} ||f\mathbf{1}_{V_b}||_1.$$

Combining this with (98) gives

$$||K_\star^t f||_2^2 \leq \sum_{b \in \mathcal{B}} r_b^{1/Cd^2(d_\star - 1)^{12}} ||f\mathbf{1}_{V_b}||_1 = |\mathcal{B}|^{-1} \sum_{b \in \mathcal{B}} r_b^{1 + 1/Cd^2(d_\star - 1)^{12}}.$$

Thus, unless

$$|\mathcal{B}|^{-1} \sum_{b \in \mathcal{B}} r_b^{1 + 1/Cd^2(d_\star - 1)^{12}} \geq r^{1 + 1/Cd^2 d_\star^{12}}, \tag{99}$$

the result is immediate. So assume that (99) holds. For $b \in \mathcal{B}$, define $w_b = \frac{||f\mathbf{1}_{V_b}||_1}{||f||_1}$. Note that $\sum_{b \in \mathcal{B}} w_b = 1$. Let U be chosen uniformly at random from $\mathcal{B}$, and let $p = 1 + 1/Cd^2(d_\star - 1)^{12}$. Dividing both sides of (99) by r^p gives

$$\frac{\mathbf{E}(r_U^p)}{r^p} \geq r^{1/Cd^2 d_\star^{12} - 1/Cd^2(d_\star - 1)^{12}}. \tag{100}$$

Using the inequality $k^{-12} - (k-1)^{-12} \leq -k^{-13}$, valid for integers $k \geq 2$, and subtracting 1 from both sides of (100) gives

$$\frac{\mathbf{E}(r_U^p)}{r^p} - 1 \geq r^{-1/Cd^2 d_\star^{13}} - 1. \tag{101}$$

Let π be the uniform probability measure on $\mathcal{B}$ and let ν be the measure on $\mathcal{B}$ defined by the w_b. Define

$$\|\pi - \nu\|_{TV} = \tfrac{1}{2} \sum_{b \in \mathcal{B}} \Big| w_b - |\mathcal{B}|^{-1} \Big| = \tfrac{1}{2} \mathbf{E} \Big| \frac{r_U - r}{r} \Big|.$$

Note that $\mathbf{E}(r_U) = |\mathcal{B}|^{-1} \sum_{b \in \mathcal{B}} r_b = r$. Plugging $X = r_U$ and $\mu = r$ into Lemma 8.2 and combining with (101) gives

$$2\|\pi - \nu\|_{TV} \geq \frac{r^{-1/Cd^2 d_\star^{13}} - 1}{r^{-1/Cd^2 (d_\star - 1)^{12}} - 1} \tag{102}$$

$$= \frac{\exp\Big(\frac{-\log r}{Cd^2 d_\star^{13}}\Big) - 1}{\exp\Big(\frac{-\log r}{Cd^2 (d_\star - 1)^{12}}\Big) - 1}. \tag{103}$$

Since $r > 6^{-cdd_\star^6}$, the quantities in the exponents in (103) are in $(0, \frac{1}{2}]$. (Recall that C is much larger than c.) Hence, the fact that $\frac{e^t - 1}{t} \in [1, 2]$ whenever $t \in (0, \frac{1}{2}]$ implies that the quantity in (103) is at least

$$\frac{(d_\star - 1)^{12}}{2 d_\star^{13}} \geq \frac{d_\star^{-1}}{2^{13}},$$

where the inequality holds because $d_\star \geq 2$ and hence $\frac{d_\star - 1}{d_\star} \geq \frac{1}{2}$. It follows that $\|\pi - \nu\|_{TV} \geq 2^{-14} d_\star^{-1}$. Note that

$$2\|\pi - \nu\|_{TV} = \sum_{b \in \mathcal{B}} \max(\nu(b), \pi(b)) - \min(\nu(b), \pi(b)), \tag{104}$$

$$2 = \sum_{b \in \mathcal{B}} \max(\nu(b), \pi(b)) + \min(\nu(b), \pi(b)). \tag{105}$$

Subtracting the first equation from the second and dividing by 2 gives

$$1 - \|\pi - \nu\|_{TV} = \sum_{b \in \mathcal{B}} \min(\nu(b), \pi(b)). \tag{106}$$

Let $\hat{K}$ be the transition kernel for the $d_\star$-truncated zigzag shuffle. Note that

$$\langle f, \hat{K}^n f \rangle = \sum_{b \in \mathcal{B}} \langle f \mathbf{1}_{V_b}, (\hat{K}^n f) \mathbf{1}_{V_b} \rangle \tag{107}$$

$$\leq \sum_{b \in \mathcal{B}} \min\Big(||f \mathbf{1}_{V_b}||_1, ||(\hat{K}^n f) \mathbf{1}_{V_b}||_1 \Big) \tag{108}$$

$$= ||f||_1 \sum_{b \in \mathcal{B}} \min\Big(w_b, \frac{||(\hat{K}^n f) \mathbf{1}_{V_b}||_1}{||f||_1} \Big), \tag{109}$$

where the inequality holds because $f \mathbf{1}_{V_b} \leq 1$ and $(\hat{K}^n f) \mathbf{1}_{V_b} \leq 1$. Let $n = 15cdd_\star^5 \log 2$. Corollary 3.2 implies that $\frac{||(\hat{K}^n f) \mathbf{1}_{V_b}||_1}{||f||_1} \leq \alpha(d_\star) |\mathcal{B}|^{-1}$, where $\alpha(k) := \exp(2^{-15k})$. Hence,

$$\langle f, \hat{K}^n f \rangle \leq ||f||_1 \, \alpha(d_\star) \sum_{b \in \mathcal{B}} \min\big(w_b, \tfrac{1}{|\mathcal{B}|} \big) \tag{110}$$

$$= ||f||_1 \, \alpha(d_\star) (1 - \|\nu - \pi\|_{TV}) \tag{111}$$

$$\leq ||f||_1 \, \alpha(d_\star) \left[1 - 2^{-14} d_\star^{-1} \right]. \tag{112}$$

Hence Lemma 8.1 gives

$$||K_\star^t f||_2^2 \le \langle f,f\rangle^{1-1/n}\left[||f||_1\,\alpha(d_\star)(1-2^{-14}d_\star^{-1})\right]^{1/n} \tag{113}$$

$$= \Big(\frac{\langle f,f\rangle}{||f||_1}\Big)^{1-1/n} \times ||f||_1 \times \alpha(d_\star)^{1/n} \times (1-2^{-14}d_\star^{-1})^{1/n} \tag{114}$$

$$\le ||f||_1 \exp\Big(\frac{1}{n}\Big[2^{-15d_\star} - d_\star^{-1}2^{-14}\Big]\Big) \tag{115}$$

$$\le ||f||_1 \exp\Big(-1/2^{15}cdd_\star^6 15\log 2\Big), \tag{116}$$

since $\frac{\langle f,f\rangle}{||f||_1} \le 1$ and $2^{-15k} \le 2^{-15}k^{-1}$ for all positive integers k. Finally, since $r > 6^{-cdd_\star^6} = \exp(-cdd_\star^6 \log 6)$, we have $r^{1/Cd^2d_\star^{12}} \ge \exp(-1/2^{15}cdd_\star^6 15\log 2)$. (Recall that $C \ge 2^{15}c^2 15\log 2\log 6$.) It follows that $||K_\star^t f||_2^2 \le r^{1/Cd^2d_\star^{12}}||f||_1$. This completes the proof. ∎

To bound the root profile, we used the following corollary.

COROLLARY 8.5. *Fix $S \subset V$ and let*

$$x = \frac{|S|}{(2^d)!}.$$

Let $\{p(x,y)\}$ be the transition probabilities for a round of the Thorp shuffle. Then there is a universal constant $C > 0$ such that

$$||p(S,\cdot)||_2^2 \le x^{1+C/d^{14}}.$$

Proof. Let $f = \mathbf{1}_S$ and $d_\star = d$ and apply Lemma 8.4. (Note that if K is the transition kernel for a round of the Thorp shuffle, then $p(S,\cdot) = K^t f$.) ∎

Acknowledgments. I am grateful to K. Ball, T. Coulhon, E. Mossel, C. Nair, Y. Peres, A. Sinclair, D. Wilson, and P. Winkler for invaluable discussions. Keith Ball proved Lemma 8.2. I want to thank the referees for carefully reading the paper and suggesting some significant improvements.

I am grateful to Laurent Saloff-Coste and Jessica Zuniga for pointing out an error in an earlier version of the paper.

I also want to thank Christian Borgs and Jennifer Chayes for giving me the opportunity to spend a year at the Theory Group of Microsoft Research, where I did much of this research.

REFERENCES

[1] D. ALDOUS AND P. DIACONIS, *Shuffling cards and stopping times*, Amer. Math. Monthly, 93 (1986), pp. 333–348.
[2] D. BAYER AND P. DIACONIS, *Tracing the dovetail shuffle to its lair*, Ann. Appl. Probab., 2 (1992), pp. 294–313.
[3] P. DIACONIS, *Personal communication*, 2000.
[4] P. DIACONIS, *Group representations in Probability and Statistics*, Institute of Mathematical Statistics Lecture Notes—Monograph Series 11, Institute of Mathematical Statistics, Hayward, CA, 1988.
[5] P. DIACONIS AND J. FILL, *Strong stationary times via a new form of duality*, Ann. Probab., 18 (1990), pp. 1483–1522.
[6] D. KNUTH, *Searching and Sorting*, The Art of Computer Programming III, Addison-Wesley, Reading, MA, 1973.
[7] B. MORRIS, *The mixing time for simple exclusion*, Ann. Appl. Probab., 2 (2006), pp. 615–635.

[8] B. Morris and Y. Peres, *Evolving sets, mixing, and heat kernel bounds*, Probab. Theory Related Fields, 133 (2005), pp. 245–266.
[9] L. Saloff-Coste, *Random walks on finite groups*, in Probability on Discrete Structures, H. Kesten, ed., Springer, Berlin, 2004, pp. 263–346.
[10] W. Steiger, *A best possible Kolmogoroff-type inequality for martingales and a characteristic property*, Ann. Math. Statist., 40 (1969), pp. 764–769.
[11] E. Thorp, *Nonrandom shuffling with applications to the game of Faro*, J. Amer. Statist. Assoc., 68 (1973), pp. 842–847.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 505–522

EVERY MONOTONE GRAPH PROPERTY IS TESTABLE*

NOGA ALON† AND ASAF SHAPIRA‡

Abstract. A graph property is called *monotone* if it is closed under removal of edges and vertices. Many monotone graph properties are some of the most well-studied properties in graph theory, and the abstract family of all monotone graph properties was also extensively studied. Our main result in this paper is that any monotone graph property can be tested with one-sided error, and with query complexity depending only on ϵ. This result unifies several previous results in the area of property testing and also implies the testability of well-studied graph properties that were previously not known to be testable. At the heart of the proof is an application of a variant of Szemerédi's regularity lemma. The main ideas behind this application may be useful in characterizing all testable graph properties and in generally studying graph property testing. As a byproduct of our techniques we also obtain additional results in graph theory and property testing, which are of independent interest. One of these results is that the query complexity of testing testable graph properties with one-sided error may be arbitrarily large. Another result, which significantly extends previous results in extremal graph theory, is that for any monotone graph property $\mathcal{P}$, any graph that is ϵ-far from satisfying $\mathcal{P}$ contains a subgraph of size depending on ϵ only, which does not satisfy $\mathcal{P}$. Finally, we prove the following compactness statement: If a graph G is ϵ-far from satisfying a (possibly infinite) set of monotone graph properties $\mathcal{P}$, then it is at least $\delta_{\mathcal{P}}(\epsilon)$-far from satisfying one of the properties.

Key words. property testing, monotone properties, regularity lemma, graphs

AMS subject classifications. 05D99, 05C85, 68W20, 68W25

DOI. 10.1137/050633445

1. Introduction.

1.1. Definitions and background. All graphs considered here are finite and undirected and have neither loops nor parallel edges. Let $\mathcal{P}$ be a property of graphs, namely, a family of graphs closed under isomorphism. All graph properties discussed in this paper are assumed to be decidable; that is, we disregard properties for which it is not possible to tell whether a given graph satisfies them. A graph G with n vertices is said to be *ϵ-far from satisfying $\mathcal{P}$* if one must add or delete at least ϵn^2 edges in order to turn G into a graph satisfying $\mathcal{P}$. A *tester* for $\mathcal{P}$ is a randomized algorithm which, given the quantity n and the ability to query whether a desired pair of vertices of an input graph G with n vertices are adjacent or not, distinguishes with high probability (say, 2/3) between the case of G satisfying $\mathcal{P}$ and the case of G being ϵ-far from satisfying $\mathcal{P}$. One of the striking results in the area of property testing is that many natural graph properties have a tester, whose total number of queries is bounded only by a function of ϵ, which is independent of the size of the input graph.

*Received by the editors June 11, 2005; accepted for publication (in revised form) July 11, 2006; published electronically May 23, 2008. A preliminary version of this paper appeared in the Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2005, pp. 128–137.

http://www.siam.org/journals/sicomp/38-2/63344.html

†Schools of Mathematics and Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel (nogaa@tau.ac.il). This author's research was supported in part by a USA Israeli BSF grant, by a grant from the Israel Science Foundation, and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

‡School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel (asafico@tau.ac.il). This work forms part of the author's Ph.D. thesis. This author's research was supported in part by a Charles Clore Foundation Fellowship and an IBM Ph.D. Fellowship.

A property having such a tester is called *testable*. Note, that if the number of queries performed by the tester is bounded by a function of ϵ only, then so is its running time. A tester is said to have *one-sided error* if whenever G satisfies $\mathcal{P}$, the algorithm declares that this is the case with probability 1. Throughout the paper, we assume that a tester first samples a set of vertices S, queries all the pairs $(i,j) \in S$, and then accepts or rejects by considering the graph spanned by the set. As observed in [3] and formally proved in [23], this can be assumed with no loss of generality, as this assumption at most squares the query complexity (and we will not care about such factors in this paper).

The general notion of property testing was first formulated by Rubinfeld and Sudan [33], who were motivated mainly by its connection to the study of program checking. The study of the notion of testability for combinatorial structures, and mainly for labeled graphs, was introduced in the seminal paper of Goldreich, Goldwasser, and Ron [22], who showed that several natural graph properties are testable. In the wake of [22], many other graph properties were shown to be testable, while others were shown to be nontestable. See [18], [32], and [31] for additional results and references on graph property testing as well as on testing properties of other combinatorial structures.

1.2. Related work. The most interesting results in property testing are those that show that large families of problems are testable. The main result of [22] states that a certain abstract graph partition problem, which includes as a special case k-colorability, having a large cut and having a large clique, is testable. The authors of [23] gave a characterization of the partition problems discussed in [22] that are testable with one-sided error. In [3], a logical characterization of a family of testable graph properties was obtained. According to this characterization, every first order graph property of type $\exists\forall$ is testable, while there are first order graph properties of type $\forall\exists$ that are not testable. These results were extended in [17].

There are also several general testability and nontestability results in other areas besides testing graph properties. In [4] it is proved that every regular language is testable. This result was extended to any read-once branching program in [29]. On the other hand, it was proved in [20] that there are read-twice branching programs that are not testable. The main result of [6] states that any constraint satisfaction problem is testable.

With this abundance of general testability results, a natural question is what makes a combinatorial property testable. As graphs are the most well-studied combinatorial structures in the theory of computation, it is natural to consider the problem of characterizing the testable graph properties as the most important open problem in the area of property testing. Regretfully, though, finding such a characterization seems to be a very challenging endeavor, which is still open. Therefore, a natural line of research is to find large families of testable graph properties.

1.3. The main new result. Our main goal in this paper is to show that all the graph properties that belong to a large, natural, and well-studied family of graph properties are testable. In fact, we even show that these properties are testable with one-sided error. A graph property $\mathcal{P}$ is said to be *monotone* if it is closed under removal of edges and vertices. In other words, if a graph G does not satisfy $\mathcal{P}$, then any graph that contains G as a (not necessarily induced) subgraph does not satisfy $\mathcal{P}$ as well. Various monotone graph properties were extensively studied in graph theory. As examples of monotone properties one can consider the property of having a homomorphism to a fixed graph H (which includes as a special case the

property of being k-colorable; see Definition 2.2) and the property of not containing a (not necessarily induced) copy of some fixed graph H. Another set of well-studied monotone properties are those defined by having a *fractional chromatic number*, *vector chromatic number*, and *Lovász theta function* bounded by some constant c, which need not be an integer (see [25] and [27]). Another monotone property is being $(k, \mathcal{H})$-*Ramsey*: For a (possibly infinite) family of graphs $\mathcal{H}$, a graph is said to be $(k, \mathcal{H})$-Ramsey if one can color its edges using k colors, such that no color class contains a copy of a graph $H \in \mathcal{H}$. This property is the main focus of Ramsey theory; see [24] and its references. As another example, one can consider the property of being $(k, \mathcal{H}, f)$-multicolorable; for a (possibly infinite) family of graphs $\mathcal{H}$ and a function f from $\mathcal{H}$ to the positive integers, a graph is said to be $(k, \mathcal{H}, f)$-multicolorable if one can color its edges using k colors, such that every copy of a graph $H \in \mathcal{H}$ receives at least $f(H)$ colors. See [16], [14], and their references for a discussion of some special cases. The abstract family of monotone graph properties has also been extensively studied in graph theory. See [21], [13], [11], and their references. Our main result is the following.

THEOREM 1 (main result). *Every monotone graph property is testable with one-sided error.*

We stress that we actually prove a slightly weaker statement than the one given above, as the monotone property has to satisfy some technical conditions (which cannot be avoided). However, as the cases where the actual result is weaker than what is stated in Theorem 1 deal with extremely unnatural properties, and even in these cases the actual result is roughly the same, we postpone the precise statement to section 5 (see Theorem 6). Another important note is that in [23], Goldreich and Trevisan define a monotone graph property to be one that is closed under removal of edges, and not necessarily under removal of vertices. They show that there are such properties that are not testable even with two-sided error. In fact, their result is stronger as the property they define belongs to NP and requires query complexity $\Omega(n^2)$. This means that Theorem 1 cannot be extended, in a strong sense, to properties that are only closed under removal of edges.

As we have mentioned above, having a homomorphism to a fixed graph H, k-colorability, and the property of not containing a copy of a fixed graph H are monotone properties and are thus testable with one-sided error by Theorem 1. These properties were known to be testable before, and as Theorem 1 applies to general monotone properties, the bounds it supplies for these properties are inferior compared to the ones proved by the ad hoc arguments (see [5], [22], [23], and [7]). In Theorem 4 we prove that this is unavoidable. The main importance of Theorem 1 thus lies in its generality. However, as described in the beginning of this subsection, there are additional natural and well-studied monotone graph properties that prior to this work were not known to be testable, and we may thus use Theorem 1 to conclude that these properties are testable with one-sided error. We also believe that Theorem 1 and its proof may be an important step toward a combinatorial characterization of the graph properties that are testable with one-sided error. Another important aspect of Theorem 1 is that it can be used to prove general results on graph property testing. Two examples are Theorems 4 and 5, which we describe in the next subsection. Another result appears in a related subsequent paper [8] and is discussed in section 5. We believe that Theorem 1 will be useful for proving other consequences as well. See section 7 for more details and possible natural lines of research suggested by the results of this paper.

1.4. Techniques and additional results. The first technical ingredient in the proof of Theorem 1 is the proof of an (almost) equivalent formulation of it. For a (possibly infinite) family of graphs $\mathcal{F}$ we say that a graph is $\mathcal{F}$-free if it contains no member from $\mathcal{F}$ as a (not necessarily induced) subgraph. Clearly, being $\mathcal{F}$-free is a monotone property. It is well known (see, e.g., [2]) that for any *finite* family of graphs $\mathcal{F}$, the property of being $\mathcal{F}$-free is testable. This follows from a standard application of Szemerédi's regularity lemma. As we discuss in section 2, this lemma is inadequate for obtaining a similar result for infinite families of graphs. The main technical step in the proof of Theorem 1 is the following theorem, which is the main technical contribution of this paper.

THEOREM 2. *For every (possibly infinite) family of graphs $\mathcal{F}$, there are functions $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$ with the following properties: If G is a graph on $n \geq N_{\mathcal{F}}(\epsilon)$ vertices which is ϵ-far from being $\mathcal{F}$-free, then a random subset of $Q_{\mathcal{F}}(\epsilon)$ vertices of G spans a member of $\mathcal{F}$ with probability at least $2/3$.*

Note that Theorem 2 immediately implies that for every family of graphs $\mathcal{F}$, the property of being $\mathcal{F}$-free is testable. In order to prove Theorem 2 we apply a strong version of the regularity lemma, proved by Alon et al. [3]. We believe that our application of this lemma may be useful for attacking other problems. As a byproduct of our argument we obtain the following graph theoretic result.

THEOREM 3. *For every monotone graph property $\mathcal{P}$, there is a function $W_{\mathcal{P}}(\epsilon)$ with the following property: If G is ϵ-far from satisfying $\mathcal{P}$, then G contains a subgraph of size at most $W_{\mathcal{P}}(\epsilon)$ which does not satisfy $\mathcal{P}$.*

The above theorem significantly extends a result of Rödl and Duke [30], conjectured by Erdős, which asserts that the above statement holds for the k-colorability property. Theorem 3 applies to any monotone property and, in particular, to all the properties discussed in the beginning of the previous subsection.

As will become evident from the proof of Theorem 1 (which is based on Theorem 2), the upper bounds for testing a monotone property depend on the property being tested. In other words, what we prove is that for every property $\mathcal{P}$, there is a function $Q_{\mathcal{P}}(\epsilon)$ such that $\mathcal{P}$ can be tested with query complexity $Q_{\mathcal{P}}(\epsilon)$. A natural question one may ask is if the dependency on the specific property being tested can be removed. We rule out this possibility by proving the following.

THEOREM 4. *For any function $Q : (0,1) \mapsto \mathbb{N}$, there is a monotone graph property $\mathcal{P}$ which has no one-sided error property tester with query complexity bounded by $o(Q(\epsilon))$.*

Prior to this work, the best lower bound proved for testing a testable graph property with one-sided error was obtained in [1], where it is shown that for every nonbipartite graph H, the query complexity of testing whether a graph does not contain a copy of H is at least $(1/\epsilon)^{\Omega(\log 1/\epsilon)}$. The fact that for every H this property is testable with one-sided error follows from [2] and [3] and also as a special case from Theorem 1. As by Theorem 1 every monotone graph property is testable with one-sided error, Theorem 4 establishes that the one-sided error query complexity of testing testable graph properties, even those that are testable with one-sided error, may be *arbitrarily large*.

Our next result can be considered a compactness-type result in property testing. Suppose $\mathcal{P}_1, \ldots, \mathcal{P}_k$ are k graph properties that are closed under removal of edges. It is clear that if a graph G is ϵ-far from satisfying these k properties then it is at least ϵ/k-far from satisfying at least one of them. However, it is not clear that there is a fixed $\delta > 0$ such that even if $k \to \infty$, G must be δ-far from satisfying one of these properties. By using Theorem 2 we can prove that if these properties are monotone

then such a δ exists. We also show that in general there is no such δ.

THEOREM 5. *For any (possibly infinite) set of monotone graph properties $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \dots\}$, there is a function $\delta_{\mathcal{P}} : (0,1) \mapsto (0,1)$ with the following property: If a graph G is ϵ-far from satisfying all the properties of $\mathcal{P}$, then for some i, the graph G is $\delta_{\mathcal{P}}(\epsilon)$-far from satisfying $\mathcal{P}_i$. Furthermore, there are properties $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \dots\}$, which are closed under removal of edges, for which no such $\delta_{\mathcal{P}}$ exists.*

1.5. Recent results. By applying the techniques of this paper along with several additional ideas we have managed to extend Theorem 1 by showing that any hereditary graph property is testable with one-sided error (a graph property is hereditary if it is closed under removal of vertices, and not necessarily under removal of edges). Besides implying that many additional graph properties are testable, we can also use this result to obtain a precise characterization of the graph properties, which can be tested with one-sided error by testers with a certain natural restriction (essentially all the testers that have been designed thus far in the literature satisfy this restriction). These results, which appear in a subsequent paper [9], demonstrate the relevance of the techniques developed in this paper to the problem of characterizing the testable graph properties. Also, in a joint work with Benny Sudakov [10], we have obtained approximation algorithms for the edit distance of a given graph from satisfying an arbitrary monotone graph property. We also obtained nearly matching hardness of approximation results. Some of the results of [10] also apply the main technique developed in this paper. In a recent paper, Avart, Rödl, and Schacht [12] extended our main result to the case of 3-uniform hypergraphs, which applies the main ideas of this paper along with the hypergraph regularity lemma. Finally, Lovász and Szegedy [28] recently gave an alterative proof of the main result of [9] on testing hereditary properties using the notion of convergent graph sequences.

1.6. Organization. The rest of the paper is organized as follows. In section 2 we introduce the basic notions of regularity and state the regularity lemmas that we use and some of their standard consequences. We also (do our best to) explain why the standard regularity lemma and its applications seem inadequate for proving Theorem 2. In section 3 we give a high level description of the proof of Theorem 2 as well as the main ideas behind it. The full proof of Theorem 2 appears in section 4. In section 5 we give the precise statement of Theorem 1 and use Theorem 2 in order to prove it. In section 7, we describe several possible extensions and open problems that this paper suggests. The proofs of Theorems 3 and 5 appear in section 4 and the proof of Theorem 4 appears in section 6. Throughout the paper, whenever we relate, for example, to a function $f_{3.1}$, we mean the function f defined in Lemma/Claim/Theorem 3.1.

2. Regularity lemmas: Definitions, statements, and applications. In this section we discuss the basic notions of regularity and some of the basic applications of regular partitions and state the regularity lemmas that we use in the proof of Theorem 2. For a comprehensive survey on the regularity lemma the reader is referred to [26]. We start with some basic definitions. For every two nonempty disjoint vertex sets A and B of a graph G, we define $e(A,B)$ to be the number of edges of G between A and B. The *edge density* of the pair is defined by $d(A,B) = e(A,B)/|A||B|$.

DEFINITION 2.1 (γ-regular pair). *A pair (A,B) is* γ-regular *if for any two subsets $A' \subseteq A$ and $B' \subseteq B$, satisfying $|A'| \geq \gamma|A|$ and $|B'| \geq \gamma|B|$, the inequality $|d(A',B') - d(A,B)| \leq \gamma$ holds.*

Note that a sufficiently large random bipartite graph, where each edge is chosen independently with probability d, is very likely to be a γ-regular pair with density

roughly d for any $\gamma > 0$. Thus, in some sense, the smaller γ is, the closer a γ-regular pair is to looking like a random bipartite graph. For this reason, the reader who is unfamiliar with the regularity lemma and its applications should try to compare the statements given in this section to analogous statements about random graphs. Throughout the paper we will make extensive use of the notion of graph homomorphism, which we now formally define.

DEFINITION 2.2 (homomorphism). *A homomorphism from a graph F to a graph K is a mapping $\varphi : V(F) \mapsto V(K)$ that maps edges to edges; namely, $(v,u) \in E(F)$ implies $(\varphi(v), \varphi(u)) \in E(K)$.*

Observe that a graph F has a homomorphism into the complete graph of size k if and only if F is k-colorable. In what follows, $F \mapsto K$ denotes the fact that there is a homomorphism from F to K. Let F be a graph on f vertices and K a graph on k vertices, and suppose $F \mapsto K$. Let G be a graph obtained by taking a copy of K, replacing every vertex with a sufficiently large independent set, and replacing every edge with a random bipartite graph of edge density d. It is easy to show that with high probability, G contains many copies of F. The following lemma shows that in order to infer that G contains many copies of F, it is enough to replace every edge with a "regular enough" pair. Intuitively, the larger f and k are, and the sparser the regular pairs are, the more regular we need each pair to be, because we need the graph to be "closer" to a random graph. This is formulated in Lemma 2.3 below. Several versions of this lemma were previously proved in papers using the regularity lemma. See, e.g., [26]. The reader should think of the mapping φ in the statement of the lemma as defining the homomorphism from F to the (implicit) graph K.

LEMMA 2.3. *For every real $0 < \eta < 1$, and integers $k, f \geq 1$ there exist $\gamma = \gamma_{2.3}(\eta, k, f)$, $\delta = \delta_{2.3}(\eta, k, f)$, and $M = M_{2.3}(\eta, k, f)$ with the following property. Let F be any graph on f vertices, and let $U_1, \ldots, U_k$ be k pairwise disjoint sets of vertices in a graph G, where $|U_1| = \cdots = |U_k| = m \geq M$. Suppose there is a mapping $\varphi : V(F) \mapsto \{1, \ldots, k\}$ such that the following holds: If (i,j) is an edge of F then $(U_{\varphi(i)}, U_{\varphi(j)})$ is γ-regular with density at least η. Then, the sets $U_1, \ldots, U_k$ span at least δm^f copies of F.*

COMMENT 2.4. *Note that the functions $\gamma_{2.3}(\eta, k, f)$ and $\delta_{2.3}(\eta, k, f)$ may and will be assumed to be monotone nonincreasing in k and f. Similarly, we will assume that the function $M_{2.3}(\eta, k, f)$ is monotone nondecreasing in k and f. Also, for ease of future definitions (particularly the one given in (4)) we set $\gamma_{2.3}(\eta, k, 0) = \delta_{2.3}(\eta, k, 0) = M_{2.3}(\eta, k, 0) = 1$ for any $k \geq 1$ and $0 < \eta < 1$.*

A partition $\mathcal{A} = \{V_i \mid 1 \leq i \leq k\}$ of the vertex set of a graph is called an *equipartition* if $|V_i|$ and $|V_j|$ differ by no more than 1 for all $1 \leq i < j \leq k$ (so, in particular, each V_i has one of two possible sizes). When we refer to the size of such an equipartition, we mean the number of partition classes of the equipartition (k above). The regularity lemma of Szemerédi can be formulated as follows.

LEMMA 2.5 (see [34]). *For every m and $\gamma > 0$ there exists a number $T = T_{2.5}(m, \gamma)$ with the following property: Any graph G on $n \geq T$ vertices has an equipartition $\mathcal{A} = \{V_i \mid 1 \leq i \leq k\}$ of $V(G)$ with $m \leq k \leq T$, for which all pairs (V_i, V_j), but at most $\gamma\binom{k}{2}$ of them, are γ-regular.*

The original formulation of the lemma also allows for an exceptional set with up to γn vertices outside of this equipartition, but one can first apply the original formulation with a somewhat smaller parameter instead of γ and then evenly distribute the exceptional vertices among the sets of the partition to obtain this formulation. $T_{2.5}(m, \gamma)$ may and is assumed to be monotone nondecreasing in m and monotone

nonincreasing in γ.

A standard application of Lemmas 2.3 and 2.5 shows that for any *finite* set of graphs $\mathcal{F}$, the property of not containing a member of $\mathcal{F}$, that is, being $\mathcal{F}$-free, is testable. We first use Lemma 2.3 by setting f and k to be the size of the largest graph in $\mathcal{F}$ and letting $\eta = \epsilon$. Lemma 2.3 gives a $\gamma_{2.3}$, which tells us how regular an equipartition should be (that is, how small should γ be) in order to find many copies of a member of $\mathcal{F}$ in it, assuming the input graph is ϵ-far from being $\mathcal{F}$-free. We then apply Lemma 2.5, with $\gamma = \gamma_{2.3}$. The main difficulty with applying this strategy when $\mathcal{F}$ is infinite is that we do not know a priori the size of the member of $\mathcal{F}$ that we will eventually find in the equipartition that Lemma 2.5 returns. After finding $F \in \mathcal{F}$ in an equipartition, we may find out that F is too large for Lemma 2.3 to be applied, because Lemma 2.5 was not used with a small enough γ. One may then try to find a new equipartition based on the size of F. However, that requires using a smaller γ, and thus the new equipartition may be larger (that is, contain more partition classes) and thus contain only larger members of $\mathcal{F}$. Hence, even the new γ is not good enough in order to apply Lemma 2.3. This leads to a circular definition of constants, which seems unbreakable. Our main tool in the proof of Theorem 2 is Lemma 2.7 below, proved in [3] for a different reason, which enables us to break this circular chain of definitions. This lemma can be considered a variant of the standard regularity lemma, where one can use a function that defines γ as a function of the size of the equipartition[1] rather than having to use a fixed γ as in Lemma 2.5. To state the lemma we need the following definition.

DEFINITION 2.6 (the function $W_{\mathcal{E},m}$). *Let $\mathcal{E}(r) : \mathbb{N} \mapsto (0,1)$ be an arbitrary monotone nonincreasing function. Let also m be an arbitrary positive integer. We define the function $W_{\mathcal{E},m} : \mathbb{N} \mapsto (0,1)$ inductively as follows: $W_{\mathcal{E},m}(1) = T_{2.5}(m, \mathcal{E}(0))$. For any integer $i > 1$ put $R = W_{\mathcal{E},m}(i-1)$ and define*

$$W_{\mathcal{E},m}(i) = T_{2.5}(R, \mathcal{E}(R)/R^2). \tag{1}$$

LEMMA 2.7 (see [3]). *For every integer m and monotone nonincreasing function $\mathcal{E}(r) : \mathbb{N} \mapsto (0,1)$ define*

$$S = S_{2.7}(m, \mathcal{E}) = W_{\mathcal{E},m}(100/\mathcal{E}(0)^4).$$

For any graph G on $n \geq S$ vertices, there exist an equipartition $\mathcal{A} = \{V_i \mid 1 \leq i \leq k\}$ of $V(G)$ and an induced subgraph U of G, with an equipartition $\mathcal{B} = \{U_i \mid 1 \leq i \leq k\}$ of the vertices of U, that satisfy the following:

1. *$m \leq k \leq S$.*
2. *$U_i \subseteq V_i$ for all $i \geq 1$, and $|U_i| \geq n/S$.*
3. *In the equipartition $\mathcal{B}$,* all *pairs are $\mathcal{E}(k)$-regular.*
4. *All but at most $\mathcal{E}(0)\binom{k}{2}$ of the pairs $1 \leq i < j \leq k$ are such that $|d(V_i, V_j) - d(U_i, U_j)| < \mathcal{E}(0)$.*

COMMENT 2.8. *For technical reasons (see the proof in* [3]*), Lemma* 2.7 *requires that for any $r > 0$ the function $\mathcal{E}(r)$ will satisfy*

$$\mathcal{E}(r) \leq \min\{\mathcal{E}(0)/4, 1/4r^2\}. \tag{2}$$

One of the difficulties in the proof of Theorem 2 is in showing that all the constants that are used in the course of the proof can be upper bounded by functions depending on ϵ only. The following observation will thus be useful.

[1] This is a simplification of the actual statement; see item (3) in the statement of Lemma 2.7.

PROPOSITION 2.9. *If m is bounded by a function of ϵ only and $\mathcal{E}(r)$ satisfies (2), then the integer $S = S_{2.7}(m, \mathcal{E})$ can be upper bounded by a function of ϵ only.*[2]

The dependency of the function $T_{2.5}(m, \gamma)$ on γ is a tower of exponents of height polynomial in $1/\gamma$ (see the proof in [26]). Thus, even for moderate functions $\mathcal{E}$ the integer S has a huge dependency on ϵ, which is a tower of towers of exponents of height polynomial in $1/\epsilon$.

3. Overview of the proof of Theorem 2. Though we believe that the proof of Theorem 2 is not harder than several other proofs applying the regularity lemma, we could not avoid the usage of a hefty number of constants that may hide the main ideas of the proof. We thus give in this section a general overview of the proof and the way we overcome the difficulties described in section 2. The complete proof is given in section 4.

For an equipartition of a graph G, let the *regularity graph* of G, denoted $R = R(G)$, be the following graph: We first use Lemma 2.5 in order to obtain the equipartition satisfying the assertions of the lemma. Let k be the size of the equipartition. Then, R is a graph on k vertices, where vertices i and j are connected if and only if (V_i, V_j) is a dense regular pair (with the appropriate parameters). In some sense, the regularity graph is an approximation of the original graph, up to γn^2 modifications. One of the main (implicit) implications of the regularity lemma is the following: Suppose we consider two graphs to be *similar* if their regularity graphs are identical. It thus follows from Lemma 2.5 that for every $\gamma > 0$, the number of graphs that are pairwise nonsimilar is bounded by a function of γ only ($2^{\binom{T}{2}}$, where $T = T_{2.5}(1/\gamma, \gamma)$). Namely, up to γn^2 modifications, all the graphs can be approximated using a set of equipartitions of size bounded by a function of γ only. The reader is referred to [15], where this interpretation of the regularity lemma is also (implicitly) used. This leads us to the key definitions of the proof of Theorem 2. The reader should think of the graphs R considered below as the set of regularity graphs discussed above, and the parameter r as representing the size of R.

DEFINITION 3.1 (the family $\mathcal{F}_r$). *For any (possibly infinite) family of graphs $\mathcal{F}$, and any integer r, let $\mathcal{F}_r$ be the following set of graphs: A graph R belongs to $\mathcal{F}_r$ if it has at most r vertices and there is at least one $F \in \mathcal{F}$ such that $F \mapsto R$.*

Practicing definitions, observe that if $\mathcal{F}$ is the family of odd cycles, then $\mathcal{F}_r$ is precisely the family of nonbipartite graphs of size at most r. In the proof of Theorem 2, the set $\mathcal{F}_r$, defined above, will represent a subset of the regularity graphs of size at most r—namely, those R for which there is at least one $F \in \mathcal{F}$ such that $F \mapsto R$. As r will be a function of ϵ only, and thus finite, we can take the maximum over all the graphs $R \in \mathcal{F}_r$ of the size of the smallest $F \in \mathcal{F}$ such that $F \mapsto R$. We thus define the following.

DEFINITION 3.2 (the function $\Psi_{\mathcal{F}}$). *For any family of graphs $\mathcal{F}$ and integer r for which $\mathcal{F}_r \neq \emptyset$, define*

$$\Psi_{\mathcal{F}}(r) = \max_{R \in \mathcal{F}_r} \min_{\{F \in \mathcal{F} : F \mapsto R\}} |V(F)|. \tag{3}$$

Define $\Psi_{\mathcal{F}}(r) = 0$ if $\mathcal{F}_r = \emptyset$. Therefore, $\Psi_{\mathcal{F}}(r)$ is monotone nondecreasing in r.

Practicing definitions again, note that if $\mathcal{F}$ is the family of odd cycles, then $\Psi_{\mathcal{F}}(r) = r$ when r is odd, and $\Psi_{\mathcal{F}}(r) = r - 1$ when r is even. The "right" way to

[2] In our application of Lemma 2.7 the function $\mathcal{E}$ will (implicitly) depend on ϵ. For example, it will be convenient to set $\mathcal{E}(0) = \epsilon$. However, note that even in this case $S_{2.7}(m, \mathcal{E})$ can be upper bounded by a function of ϵ only.

think of the function $\Psi_{\mathcal{F}}$ is the following: Let R be a graph of size at most r and suppose we are guaranteed that there is a graph $F' \in \mathcal{F}$ such that $F' \mapsto R$ (thus $R \in \mathcal{F}_r$). Then by this information only and *without* having to know the structure of R itself, the definition of $\Psi_{\mathcal{F}}$ implies that there is a graph $F \in \mathcal{F}$ of size at most $\Psi_{\mathcal{F}}(r)$, such that $F \mapsto R$.

The function $\Psi_{\mathcal{F}}$ has a critical role in the proof of Theorem 2. The first usage of this function is that, as by Lemma 2.5 we can upper bound the size of the regularity graph R, we can also upper bound the size of the smallest graph $F \in \mathcal{F}$ for which $F \mapsto R$. A second important property of $\Psi_{\mathcal{F}}$ is discussed in section 5. A natural question one may ask is whether there is a function Ψ that can upper bound $\Psi_{\mathcal{F}}$ for all families $\mathcal{F}$. As it turns out, this is impossible; namely, the dependency on the specific family $\mathcal{F}$ is unavoidable. See the discussion following the proof of Theorem 4 in section 6. As we have mentioned in the previous section, the main difficulty that prevents one from proving Theorem 2 using Lemma 2.3 is that one does not know a priori the size of the graph that one may expect to find in the equipartition. This leads us to define the following function, where $0 < \epsilon < 1$ is an arbitrary real.

$$\mathcal{E}'(r) = \begin{cases} \epsilon/8, & r = 0, \\ \gamma_{2.3}(\epsilon/8,\ r,\ \Psi_{\mathcal{F}}(r)), & r \geq 1. \end{cases} \tag{4}$$

In simple words, given r, which will represent the size of the equipartition and thus also the size of the regularity graph which it defines, $\mathcal{E}'(r)$ returns "how regular" this equipartition should be in order to allow one to find many copies of the *largest* graph one may possibly have to work with. Note that we obtain the upper bound on the size of this largest possible graph by invoking $\Psi_{\mathcal{F}}(r)$. As for different families of graphs $\mathcal{F}$, the function $\Psi_{\mathcal{F}}(r)$ may behave differently; $\mathcal{E}'(r)$ may also behave differently for different families $\mathcal{F}$, as it is defined in terms of $\Psi_{\mathcal{F}}(r)$. However, and this is one of the key points of the proof, as we are fixing the family of graphs $\mathcal{F}$, the function $\mathcal{E}'(r)$ depends only on r.

Given the above definitions we apply Lemma 2.7 with a slight modification of $\mathcal{E}'(r)$ in order to obtain an equipartition of G. We then throw away edges that reside inside the sets V_i and between (V_i, V_j), whose edge density differs significantly from that of (U_i, U_j). We then argue that we thus throw away less than ϵn^2 edges. As G is by assumption ϵ-far from not containing a member of $\mathcal{F}$, the new graph still contains a copy of $F \in \mathcal{F}$. By the definition of the new graph, it thus means that there is a (natural) homomorphism from F to the regularity graph of G. We then arrive at the main step of the proof, where we use the key property of Lemma 2.7, item (3), and the definition of $\mathcal{E}'(r)$ to get that the sets U_i are regular enough to let us use Lemma 2.3 on them and to infer that they span many copies of F. It thus follows that a large enough sample of vertices spans a copy of F with high probability. The complete details appear in section 4.

4. Proofs of Theorems 2, 3, and 5. We start with the proof of Theorem 2. We assume the reader is familiar with the overview of its proof given in section 3.

Proof of Theorem 2. Fix any family of graphs $\mathcal{F}$. Our goal is to show the existence of functions $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$ with the following properties: If a graph G on $n \geq N_{\mathcal{F}}(\epsilon)$ vertices is ϵ-far from being $\mathcal{F}$-free, then a random subset of $Q_{\mathcal{F}}(\epsilon)$ vertices of $V(G)$ spans a member of $\mathcal{F}$ with probability at least $2/3$. For the rest of the proof, let $\mathcal{E}'(r) : N \mapsto (0, 1)$ be as defined in (4). In order to apply Lemma 2.7, we need to define a function $\mathcal{E}$, based on $\mathcal{E}'$, which will satisfy the technical condition (2)

in Comment 2.8. We thus set $\mathcal{E}(0) = \mathcal{E}'(0)$ $(= \epsilon/8)$ and define for any $r > 0$

$$\mathcal{E}(r) = \min\{\mathcal{E}'(r),\ \mathcal{E}(0)/4,\ 1/4r^2\}. \tag{5}$$

For the rest of the proof set

$$S(\epsilon) = S_{2.7}(8/\epsilon, \mathcal{E}).$$

We may indeed define $S(\epsilon)$ using $\mathcal{E}$ as it satisfies (2). Furthermore, as we define $S(\epsilon)$ using $m = 8/\epsilon$ we get by Proposition 2.9 that $S(\epsilon)$ is indeed a function of ϵ only. We now set

$$N = N_{\mathcal{F}}(\epsilon) = S(\epsilon) \cdot M_{2.3}(\epsilon/8, S(\epsilon), \Psi_{\mathcal{F}}(S(\epsilon))) \tag{6}$$

to be an integer bounded by a function of ϵ as well. We postpone the definition of $Q_{\mathcal{F}}(\epsilon)$ till the end of the proof.

Given a graph G on n vertices, with $n \geq N \geq S(\epsilon)$, we can use Lemma 2.7 with $m = 8/\epsilon$ and $\mathcal{E}(r)$ as defined in (5) in order to obtain an equipartition of $V(G)$ into $8/\epsilon \leq k \leq S(\epsilon)$ clusters $V_1, \ldots, V_k$ (this is possible by item 1 in Lemma 2.7). By item 2 of Lemma 2.7, for every $1 \leq i \leq k$ we have sets $U_i \subseteq V_i$ each of size at least $n/S(\epsilon)$. Remove from G the following edges according to the following order:

1. Any edge (u, v) for which both u and v belong to the same cluster V_i. As each of the clusters contains at most $n/k + 1$ vertices, the total number of edges removed is at most $k(n/k)^2$. As $k \geq 8/\epsilon$ we have $k(n/k)^2 < \frac{\epsilon}{8}n^2$.
2. If for some $i < j$ we have $|d(V_i, V_j) - d(U_i, U_j)| > \frac{\epsilon}{8} = \mathcal{E}(0)$, remove all the edges connecting vertices that belong to V_i to vertices that belong to V_j. By item 4 of Lemma 2.7, there are at most $\frac{\epsilon}{8}k^2$ such pairs i, j. As V_i and V_j contain at most $(n/k+1)$ vertices, we remove at most $\frac{\epsilon}{8}k^2 \cdot (n/k+1)^2 \leq \frac{\epsilon}{7}n^2$ edges in this step.
3. If for some $i < j$ we have $d(U_i, U_j) < \frac{\epsilon}{8}$, remove all the edges connecting vertices that belong to V_i to vertices that belong to V_j. As we have already removed in the previous step all the edges between pairs (V_i, V_j) for which $|d(V_i, V_j) - d(U_i, U_j)| > \frac{\epsilon}{8}$, we may conclude that if $d(U_i, U_j) < \frac{\epsilon}{8}$ then we also have $d(V_i, V_j) < \frac{\epsilon}{8} + \mathcal{E}(0) = \frac{\epsilon}{4}$. As V_i and V_j contain at most $(n/k + 1)$ vertices, we thus remove at most $k^2 \cdot \frac{\epsilon}{4}(n/k + 1)^2 \leq \frac{\epsilon}{3}n^2$ edges.

Call the graph obtained after removing the above edges G', and observe that G' is obtained from G by removing less than ϵn^2 edges. By item 3 of Lemma 2.7, in G all the pairs (U_i, U_j) are $\mathcal{E}(k)$-regular. Thus, by the third step of obtaining G' we get the following property.

PROPOSITION 4.1. *If $v_i \in V_i$ is connected to $v_j \in V_j$ in G', then (U_i, U_j) is an $\mathcal{E}(k)$-regular pair with density at least $\frac{\epsilon}{8}$ in G.*

Consider a graph R on k vertices $r_1, \ldots, r_k$, where vertices r_i and r_j are connected if and only if (U_i, U_j) is an $\mathcal{E}(k)$-regular pair in G with density at least $\frac{\epsilon}{8}$. This is the regularity graph, which we have mentioned in section 3, of the graph induced by the sets $U_1, \ldots, U_k$. As G is by assumption ϵ-far from being $\mathcal{F}$-free, and G' is obtained from G by removing less than ϵn^2 edges, G' must contain a copy of a graph $F' \in \mathcal{F}$. Let R_i contain all the vertices of F' that belong to cluster V_i and note that by Proposition 4.1, there is a natural homomorphism $\varphi : V(F') \mapsto V(R)$ which maps all the vertices of $R_i \subseteq V(F')$ to r_i. As $|V(R)| = k$ and F' is a graph in $\mathcal{F}$ such that $F' \mapsto R$, we conclude that $R \in \mathcal{F}_k$ (recall Definition 3.1). Therefore, there is a graph $F \in \mathcal{F}$ of size at most $\Psi_{\mathcal{F}}(k)$ such that $V(F) \mapsto V(R)$ (recall Definition 3.2). Let

$\varphi : V(F) \mapsto V(R)$ be the homomorphism mapping the vertices of F to the vertices of R. By definition, we have that whenever (i,j) is an edge of F their image $(\varphi(i),\varphi(j))$ is an edge of R. Furthermore, by definition of R we know that if $(\varphi(i),\varphi(j))$ is an edge of R then $(U_{\varphi(i)}, U_{\varphi(j)})$ is an $\mathcal{E}(k)$-regular pair with density at least $\frac{\epsilon}{8}$.

We have thus arrived at the following situation: We have k clusters of vertices $U_1,\ldots,U_k$ of the same size. We also have a graph F of size at most $\Psi_{\mathcal{F}}(k)$, and a mapping $\varphi : V(H) \mapsto \{1,\ldots,k\}$ that satisfies the condition; if $(i,j) \in E(F)$ then $(U_{\varphi(i)}, U_{\varphi(j)})$ is an $\mathcal{E}(k)$-regular pair with density $\epsilon/8$. This, together with the definition of $\mathcal{E}(k)$, implies that we can use Lemma 2.3 on the graph U spanned by $U_1,\ldots,U_k$. Let $f \leq \Psi_{\mathcal{F}}(k)$ denote the size of F. Item 4 in Lemma 2.7 states that each U_i contains at least $n/S(\epsilon)$ vertices. Also, by (6), and by the monotonicity properties of $M_{2.3}$ discussed in Comment 2.4, we have for any $1 \leq i \leq k$

$$|U_i| \geq n/S(\epsilon) \geq M_{2.3}(\epsilon/8, S(\epsilon), \Psi_{\mathcal{F}}(S(\epsilon))) \geq M_{2.3}(\epsilon/8, k, \Psi_{\mathcal{F}}(k)).$$

Therefore, we may apply Lemma 2.3 on the sets $U_1,\ldots,U_k$ to conclude that U spans at least

$$\delta \prod_{i=1}^{f} |U_i| \geq \delta(n/S(\epsilon))^f \geq \delta n^f / S(\epsilon)^{\Psi_{\mathcal{F}}(k)} \geq \delta n^f / S(\epsilon)^{\Psi_{\mathcal{F}}(S(\epsilon))} \tag{7}$$

copies of F, where $\delta = \delta_{2.3}(\epsilon/8, k, \Psi_{\mathcal{F}}(k))$. By Comment 2.4, the function $\delta_{2.3}(\eta,k,f)$ is monotone nonincreasing in k and f. Also, $\Psi_{\mathcal{F}}(k)$ is monotone nondecreasing in k. Hence, as $k \leq S(\epsilon)$ we have that $\delta \geq \delta_{2.3}(\epsilon/8, S(\epsilon), \Psi_{\mathcal{F}}(S(\epsilon)))$, and, in particular, $1/\delta$ is upper bounded by a function of ϵ only. As U is a subgraph of G, we may conclude that G contains at least as many copies of F as (7). Thus, if we independently sample $2S(\epsilon)^{\Psi_{\mathcal{F}}(S(\epsilon))}/\delta$ sets of $\Psi_{\mathcal{F}}(S(\epsilon))$ $(\geq f)$ vertices (which is a total of $2\Psi_{\mathcal{F}}(S(\epsilon)) \cdot S(\epsilon)^{\Psi_{\mathcal{F}}(S(\epsilon))}/\delta$ vertices) we have probability at least $2/3$ of finding a copy of $F \in \mathcal{F}$.

We can now give the formal definition of $Q_{\mathcal{F}}(\epsilon)$. Given a family of graphs $\mathcal{F}$ let $\Psi_{\mathcal{F}}(r)$ be the function from Definition 3.2. We note that the only place where $Q_{\mathcal{F}}(\epsilon)$ depends on $\mathcal{F}$ is in the function $\Psi_{\mathcal{F}}(r)$. Using $\Psi_{\mathcal{F}}(r)$ define the function $\mathcal{E}(r)$ as in (5). Given $\epsilon > 0$ define the function $W_{\mathcal{E},8/\epsilon}$ as in Definition 2.6 and put $S(\epsilon) = W_{\mathcal{E},8/\epsilon}(100/(\epsilon/8)^4)$. Finally, we can set

$$Q_{\mathcal{F}}(\epsilon) = \frac{2\Psi_{\mathcal{F}}(S(\epsilon)) \cdot S(\epsilon)^{\Psi_{\mathcal{F}}(S(\epsilon))}}{\delta_{2.3}(\epsilon/8, S(\epsilon), \Psi_{\mathcal{F}}(S(\epsilon)))} \tag{8}$$

to be a function of ϵ only. This completes the proof of the theorem. □

From the definition of $\mathcal{E}'(r)$ in (4) it is clear that if the function $\Psi_{\mathcal{F}}(r)$ is recursive, then so is $\mathcal{E}'(r)$ and therefore also $\mathcal{E}(r)$ (for this we also need the fact that $\gamma_{2.3}(\eta,k,f)$ is recursive, which follows from the standard proofs of Lemma 2.3; see [26]). In this case the function $W_{\mathcal{E},m}(i)$ is also recursive (see Definition 2.6), and therefore so is the function $S_{2.7}(8/\epsilon, \mathcal{E})$. Finally, this means that the integer $S(\epsilon)$, used in the above proof, can also be computed. Now, given $S(\epsilon)$ and the fact that $\Psi_{\mathcal{F}}(r)$ is recursive, one can use (6) and (8) as well as the fact that $\delta_{2.3}(\eta,k,f)$ and $M_{2.3}(\eta,k,f)$ are recursive (see the proof in [26]) in order to compute $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$.

We finish this section with the proofs of Theorems 3 and 5.

Proof of Theorem 3. We claim that we can set $W_{\mathcal{P}}(\epsilon) = \max\{N_{\mathcal{F}}(\epsilon), Q_{\mathcal{F}}(\epsilon)\}$ with $\mathcal{F} = \mathcal{F}_{\mathcal{P}}$ as in the proof of Theorem 1, and $N_{\mathcal{F}}(\epsilon)$, $Q_{\mathcal{F}}(\epsilon)$ are the functions from Theorem 2. Indeed, If G is ϵ-far from satisfying $\mathcal{P}$, and G has less than $N_{\mathcal{F}}(\epsilon)$

vertices, we can take G itself to be a subgraph of G not satisfying $\mathcal{P}$. Suppose now that G has more than $N_{\mathcal{F}}(\epsilon)$ vertices. As G is also ϵ-far from being $\mathcal{F}$-free, we get from Theorem 2 that G contains a subgraph (in fact, many) of size $Q_{\mathcal{F}}(\epsilon)$, which is not $\mathcal{F}$-free and therefore does not satisfy $\mathcal{P}$. □

Proof of Theorem 5. For each of the monotone properties $\mathcal{P}_i$, let $\mathcal{F}_i$ be the family of graphs, which do not satisfy $\mathcal{P}_i$, and let $\mathcal{F} = \mathcal{F}_1 \bigcup \mathcal{F}_2 \bigcup \mathcal{F}_3 \bigcup \dots$. Clearly, a graph G satisfies all the properties of $\mathcal{P}$ if and only if it is $\mathcal{F}$-free. Consider a graph G, which is ϵ-far from satisfying all the properties of $\mathcal{P}$. In this case G is also ϵ-far from being $\mathcal{F}$-free. The proof of Theorem 2 establishes that there is a graph $F \in \mathcal{F}$ of size at most $f = f_{\mathcal{F}}(\epsilon)$ such that G contains $\delta_{\mathcal{F}}(\epsilon)n^f$ copies of F. Note that removing an edge from G destroys at most $\binom{n}{f-2} \leq n^{f-2}$ copies of F. Thus, one must remove at least $\delta_{\mathcal{F}}(\epsilon)n^2$ edges from G in order to make it F-free. Let i be such that $F \in \mathcal{F}_i$. We may now infer that G is $\delta_{\mathcal{F}}(\epsilon)$-far from satisfying $\mathcal{P}_i$. Finally, note that as $\mathcal{F}$ is determined by $\mathcal{P}$, we can also say that G is $\delta_{\mathcal{P}}(\epsilon)$-far from satisfying $\mathcal{P}_i$.

To show that in case the properties $\mathcal{P}_i$ are just closed under removal of edges the above does not hold, consider the following: For any integer n, let $H_1, H_2, \dots$ be some ordering of the graphs on n vertices, which contain precisely $n^{3/2}$ edges. A graph of size n is said to satisfy property $\mathcal{P}_i$ if it contains no copy of H_i. Clearly, any property $\mathcal{P}_i$ is closed under removal of edges but not necessarily under removal of vertices. Observe that any graph with at least $n^{3/2}$ edges does not satisfy one of the properties $\mathcal{P}_i$. Therefore, any graph G of size n which contains $2\epsilon n^2$ edges is ϵ-far from satisfying all the properties $\mathcal{P}_i$. We claim that any such G is not $\frac{\log n}{\sqrt{n}}$-far from satisfying any one of these properties. To this end, it is enough to show that for any graph H_i, we can remove at most $n^{3/2} \log n$ edges from G and thus make it H_i-free. To see this, note that as G and H_i are both of size n, G spans at most $n!$ copies of H_i. As H_i contains $n^{3/2}$ edges a randomly chosen edge of G is spanned by H_i with probability at least $n^{3/2}/\binom{n}{2} > 1/\sqrt{n}$. Thus, if we remove from G a set of $n^{3/2} \log n$ edges, where each edge is randomly and uniformly chosen from the edges of G (with repetitions), the probability that none of the edges of one of the copies of H_i in G were removed is at most $(1 - 1/\sqrt{n})^{n^{3/2} \log n} < 1/n!$. By the union bound, the probability that for *some* copy of H_i in G, none of its edges were removed is strictly smaller than 1. Thus, there exists a choice of $n^{3/2} \log n$ edges, whose removal from G makes the graph H_i-free. □

5. Proof of Theorem 1. For a monotone graph property $\mathcal{P}$, define $\mathcal{F} = \mathcal{F}_{\mathcal{P}}$ to be the set of graphs which are minimal with respect to not satisfying property $\mathcal{P}$. In other words, a graph F belongs to $\mathcal{F}$ if it does not satisfy $\mathcal{P}$, but any graph obtained from F by removing an edge or a vertex satisfies $\mathcal{P}$. Thus, for example, if $\mathcal{P}$ is the property of being 2-colorable, then $\mathcal{F}$ is the set of odd cycles. Clearly, a graph satisfies $\mathcal{P}$ if and only if it contains no member of $\mathcal{F}$ as a (not necessarily induced) subgraph.

As we have mentioned in section 1, we will prove a slightly different version of Theorem 1. In order to precisely restate Theorem 1 we need two definitions. Note that in defining a tester in section 1, we did not mention whether the error parameter ϵ is given as part of the input, or whether the tester is designed to distinguish between graphs that satisfy $\mathcal{P}$ from those that are ϵ-far from satisfying it, when ϵ is a known fixed constant. In fact, the literature about property testing is not clear about this issue as in some papers ϵ is assumed to be a part of the input while in others it is not. We define a property to be *uniformly* testable if there is a tester for it that receives ϵ as part of the input. We define a property to be *nonuniformly* testable if for every

fixed ϵ, there is a tester that can distinguish between graphs that satisfy $\mathcal{P}$ from those ϵ-far from satisfying it (which may not work properly for other values of ϵ). We are now ready to restate Theorem 1.

THEOREM 6 (Theorem 1 restated). *Every monotone graph property $\mathcal{P}$ is nonuniformly testable with one-sided error. Moreover, if the function $\Psi_{\mathcal{F}}$ is recursive (where $\mathcal{F} = \mathcal{F}_{\mathcal{P}}$) then $\mathcal{P}$ is also uniformly testable with one-sided error.*

We stress that all reasonable graph properties $\mathcal{P}$, in particular, those that were discussed in section 1, are such that $\Psi_{\mathcal{F}}$ is recursive (a function is recursive if there is an algorithm that computes it in finite time). In particular, all the monotone properties mentioned in section 1 are uniformly testable with one-sided error. We thus bother to define uniformly and nonuniformly testing as well as discuss $\Psi_{\mathcal{F}}$ because it has the following interesting property: Not only is it sufficient to require $\Psi_{\mathcal{F}}$ to be recursive in order to infer that $\mathcal{P}$ can be tested uniformly with one-sided error, but this is also *necessary*. In other words, the recursiveness of $\Psi_{\mathcal{F}}$ determines whether $\mathcal{P}$ can be tested uniformly.[3] This is somewhat surprising as $\Psi_{\mathcal{F}}$ has little to do with property testing. Using this necessary condition, it is possible to show that there are graph properties that can be *nonuniformly* tested with one-sided error but cannot be *uniformly* tested, even with two-sided error. In fact, there are such graph properties, which are monotone and belong to $coNP$. The proofs of the necessity of $\Psi_{\mathcal{F}}$ being recursive in order to obtain a uniform tester, as well as the existence of a property that cannot be tested uniformly, are rather involved and significantly deviate from the main topic of this paper. Hence, we refrain from describing them here. These results will appear in a subsequent paper [8].

Proof of Theorem 6. Let $\mathcal{F} = \mathcal{F}_{\mathcal{P}}$ be as defined above, and let $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$ be the functions of Theorem 2. As satisfying $\mathcal{P}$ is equivalent to being $\mathcal{F}$-free, we focus on testing the property of being $\mathcal{F}$-free. We first show that every monotone property is nonuniformly testable. In this case we may design a tester for every given error parameter ϵ (but one that can handle *any* graph as an input). In this case, for every fixed ϵ, the tester knows the values of $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$ in advance (i.e., they are part of its description). If the size of the input graph is less than $N_{\mathcal{F}}(\epsilon)$, the algorithm queries about all edges of the graph and accepts if and only if the graph is $\mathcal{F}$-free (obviously, in this case the algorithm always answers correctly). If the size of the input graph is larger than $N_{\mathcal{F}}(\epsilon)$, it samples $Q_{\mathcal{F}}(\epsilon)$ random vertices and accepts if and only if the graph spanned by this set of vertices is $\mathcal{F}$-free. Clearly, if G is $\mathcal{F}$-free the algorithm declares that this is the case with probability 1. On the other hand, if it is ϵ-far from being $\mathcal{F}$-free then by Theorem 2 the sample of size $Q_{\mathcal{F}}(\epsilon)$ will contain $F \in \mathcal{F}$ with probability at least 2/3, and thus the algorithm will reject the input with this probability. In any case, the query complexity, which is $\max\{N_{\mathcal{F}}(\epsilon), Q_{\mathcal{F}}(\epsilon)\}$, is bounded by a function of ϵ only.

We now turn to uniform testers. In this case, we can imitate the proof of the case where ϵ is given in advance, which was described above. The only technical obstacle that may prevent us from carrying out the same testing algorithm is that the algorithm should be able to compute $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$. As the details of the proof of Theorem 2 reveal (see the discussion following the proof of Theorem 2 in section 4), the only step in computing $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$ which is not well defined (i.e., that depends on $\mathcal{F}$) is the computation of the function $\Psi_{\mathcal{F}}(r)$ (see Definition 3.2). In other words, if $\Psi_{\mathcal{F}}$ is recursive, then so are $N_{\mathcal{F}}(\epsilon)$ and $Q_{\mathcal{F}}(\epsilon)$. We thus get that if $\Psi_{\mathcal{F}}$ is

[3] This is in fact a simplification of the actual result that we can show. See [8] for the precise statement.

recursive, we can uniformly test the property of being $\mathcal{F}$-free. □

6. Proof of Theorem 4. In this section we describe the proof of Theorem 4. We remind the reader that we denote by $F \mapsto K$ the fact that there is a homomorphism from F to K (see Definition 2.2). In what follows, an s-blowup of a graph K is the graph obtained from K by replacing every vertex $v_i \in V(K)$ with an independent set I_i, of size s, and replacing every edge $(v_i, v_j) \in E(K)$ with a complete bipartite graph whose partition classes are I_i and I_j. It is easy to see that a blowup of K is far from being K-free (K-free is the property of not containing a copy of K). It is also easy to see that if $F \mapsto K$, then a blowup of K is far from being F-free (see [1, Lemma 3.3]). However, in this case the farness of the blowup from being F-free is a function of the size of F. As it turns out, for the proof of Theorem 4 we need a stronger assertion where the farness is only a function of k. This is given in Lemma 6.1 below, which is proved in [8].

LEMMA 6.1 (see [8]). *Let F be a graph on f vertices with at least one edge, let K be a graph on k vertices, and suppose $F \mapsto K$ (thus, $k \geq 2$). Then, for every sufficiently large $n \geq n(f)$, an n/k-blowup of K, is $\frac{1}{2k^2}$-far from being F-free.*

For the proof of Theorem 4 we also need the following simple observation. Recall that by the results of [3] and [23] we may assume that a tester first picks a set of vertices S, and then accepts or rejects according to the graph induced by S. We also comment on the following subtle point: when defining a testable graph property we require the query complexity to be *upper bounded* by a function of ϵ. This, however, does not imply that it is *independent* of n. Therefore, when considering arbitrary testers we have to consider testers whose query complexity and decisions may depend on the size of the input, which we denote by n.

CLAIM 6.2. *Let $\mathcal{F}$ be a family of graphs, such that no $F \in \mathcal{F}$ has isolated vertices and let T be a one-sided error tester for the property of being $\mathcal{F}$-free with query complexity $Q(\epsilon, n)$. If for some $\epsilon_0 > 0$ and n, after T samples a set of vertices S of size $Q(\epsilon_0, n)$, the graph induced by S is $\mathcal{F}$-free, then T must accept the input.*

Proof. Fix any n and $\epsilon_0 > 0$ and suppose that when we execute T on a graph G of size n with $\epsilon = \epsilon_0$, and the sample of vertices S spans a graph H that is $\mathcal{F}$-free, the algorithm still rejects the input. Consider a graph G' of size n, which contains a graph isomorphic to H on some of its $|S|$ vertices. Other than the edges of the copy of H, the graph G' has no other edges. As $\mathcal{F}$ contains no graph with isolated vertices and H is $\mathcal{F}$-free we infer that G' is $\mathcal{F}$-free. Suppose we execute T on G' with $\epsilon = \epsilon_0$. As G and G' are of the same size n, when given G' as input the algorithm samples a set of vertices of size $Q(\epsilon_0, n) = |V(H)|$. As we assume that when given H the algorithm rejects, we get that there is a nonzero probability that T will reject G', contradicting the assumption that it has one-sided error. □

As our goal is to prove a lower bound on the query complexity we may and will assume that Q is monotone nonincreasing (hence, monotone nondecreasing in $1/\epsilon$). For every such function Q we will define a property $\mathcal{P} = \mathcal{P}(Q)$ needed in order to prove Theorem 4. These properties can be thought of as *sparse bipartiteness* as they will be defined in terms of not containing a certain subset of the set of odd cycles.

Let $Q : (0,1) \mapsto \mathbb{N}$ be an arbitrary monotone nonincreasing function. For such a function, let Q^i be the following i times iterated version of Q. We put $Q^1(x) = Q(x)$ and for any $i \geq 1$ define

$$Q^{i+1}(x) = 2Q\left(\frac{1}{2(Q^i(x)+2)^2}\right) + 1. \tag{9}$$

Define $I(Q) = \{Q^i(1/2) : i \in N\}$ and note that $I(Q)$ contains only odd integers. For a function as above, let $C(Q) = \{C_i : i \in I(Q)\}$; that is, $C(Q)$ is the set of odd cycles whose lengths are the integers of the set $I(Q)$. Finally, let $\mathcal{P} = \mathcal{P}(Q)$ denote the property of not containing any of the odd cycles of $C(Q)$ as a (not necessarily induced) subgraph.

Proof of Theorem 4. Given a monotone nonincreasing function Q, let $\mathcal{P} = \mathcal{P}(Q)$ be the property defined above. We show that for any positive integer k for which $k-2 \in I(Q)$, any one-sided error tester that distinguishes between graphs of sufficiently large n that satisfy $\mathcal{P}$ from those that are $\frac{1}{2k^2}$-far from satisfying it has query complexity at least $Q(1/2k^2)$. As Q is by assumption monotone nonincreasing, $I(Q)$ contains infinitely many integers. Hence, for infinitely many values of ϵ, and for all large enough n, the query complexity of such a one-sided error tester is at least $Q(\epsilon)$. Note also that the set of these ϵ's approaches zero.

Fix any integer k for which $k-2 \in I(Q)$ and assume $k-2 = Q^i(1/2)$. As $I(Q)$ contains only odd integers, k is also odd. Define $\ell = Q^{i+1}(1/2)$ and recall that by (9), we have $\ell = 2Q(1/2k^2) + 1$. As it is clear that there is a homomorphism from C_ℓ to C_k, we get by Lemma 6.1 that for any $n \geq N(\ell)$, an n/k-blowup of C_k is $\frac{1}{2k^2}$-far from being C_ℓ-free. Denote such a blowup by G. As by definition $C_\ell \in C(Q)$, the graph G is also $\frac{1}{2k^2}$-far from satisfying $\mathcal{P}$. Also, as $k-2$ is odd, G contains no copy of C_{k-2}. In particular, G contains no member of $C(Q)$ of length less than ℓ. As property $\mathcal{P}$ is determined in terms of not containing a certain set of odd cycles, none of which has isolated vertices, we get from Claim 6.2 that a one-sided error tester must find a copy of a graph not satisfying $\mathcal{P}$ in order to determine that it does not satisfy $\mathcal{P}$. Therefore, for any $n \geq N(\ell)$ the query complexity of any tester for distinguishing between graphs of size n satisfying $\mathcal{P}$ from graphs of size n that are $\frac{1}{2k^2}$-far from satisfying it is at least ℓ. As $\ell = 2Q(1/2k^2) + 1 \geq Q(1/2k^2)$ the proof is complete. □

An immediate consequence of Theorem 4 is that there is no function $Q(\epsilon)$ that upper bounds the query complexity $Q_{\mathcal{F}}(\epsilon)$ of testing the property of being $\mathcal{F}$-free for all families of graphs, $\mathcal{F}$. In other words, the dependence on the specific family of graph is unavoidable. By the same reasoning, the dependence on $\mathcal{P}$ in Theorem 3 is also unavoidable. As we have commented after the proof of Theorem 2 in section 4, the only dependence of the function $Q_{\mathcal{F}}(\epsilon)$ defined in the proof of Theorem 1 (see (8)) on $\mathcal{P}$ is due to the function $\Psi_{\mathcal{F}}$ from Definition 3.2 (where $\mathcal{F} = \mathcal{F}_{\mathcal{P}}$ is the set of minimal graphs with respect to not satisfying $\mathcal{F}$). This implies that the function $\Psi_{\mathcal{F}}$ must depend on $\mathcal{F}$ and thus also on $\mathcal{P}$, as otherwise we could obtain an upper bound on $Q_{\mathcal{F}}(\epsilon)$ which would apply to all families of graphs, thus contradicting Theorem 4. We conjecture that Theorem 4 can be extended to two-sided error testers; see section 7.

As we have commented at the beginning of this section, the proof of Theorem 4 heavily relies on the fact that the farness of the graph considered in Lemma 6.1 from being F-free is only a function of k. From the proof of Theorem 4 it should indeed be clear that if this farness had been a function of the size of F, then the length of each cycle of the family would have depended on its own size, which would result in a cycle of definitions.

7. Concluding remarks and open problems. Besides proving that a large family of graph properties are all testable, and that specific properties that were previously not known to be testable are in fact testable, another important aspect of Theorem 1 is that it can be used to prove general results on testing graph properties. Two such results are Theorems 4 and 5. Another result, discussed in section 5, is that there are graph properties that can be nonuniformly tested but cannot be uniformly

tested [8]. We believe that Theorem 1 will be useful for proving other results as well.

Our main result gives that the natural family of monotone graph properties are all testable with one-sided error. This gives rise to several questions. For example, one can study the relation between testing with one-sided and two-sided error by considering how large the gap between the query complexity of testing a monotone graph property with one-sided and two-sided error can be. Specifically, it will be interesting to investigate whether there is a monotone property for which there is a superpolynomial gap between the two tasks. It will also be interesting to strengthen Theorem 4 by proving that for any function $Q : (0, 1) \mapsto \mathbb{N}$, there is a monotone graph property that cannot be tested with $o(Q(\epsilon))$ queries, even with two-sided error. Currently, the best lower bound on the *two-sided* error query complexity of a monotone graph property is a $(1/\epsilon)^{\Omega(\log 1/\epsilon)}$ lower bound for testing the property of not containing a copy of a graph H for any nonbipartite H [7].

A particularly interesting problem to study regarding the family of monotone graph properties is to obtain a characterization of the monotone properties, which are testable with $poly(1/\epsilon)$ queries. For some properties, such as k-colorability, it is known that $poly(1/\epsilon)$ queries suffice (see [22] and [5]). For others, such as being H-free for any nonbipartite H, it is known that $poly(1/\epsilon)$ are not sufficient (see [1] and [7]).

Even a special case of this problem seems hard to resolve. While it is known that the property of not containing an odd cycle, namely, being bipartite, can be tested with $\tilde{O}(1/\epsilon)$ queries (see [5]), Theorem 4 establishes that testing the property of being $\mathcal{F}$-free, where $\mathcal{F}$ is a subset of the family of odd cycles, may be arbitrarily hard (at least with one-sided error). It is interesting to check if one can at least characterize the families of odd cycles $\mathcal{F}$, for which one can test the property of being $\mathcal{F}$-free with $poly(1/\epsilon)$ queries.

As was mentioned in the introduction, a result of Goldreich and Trevisan [23] rules out the possibility of extending Theorem 2 to graph properties that are only closed under removal of edges. It seems interesting to bridge the gap between their result and the main result of this paper by characterizing the testable graph properties that are closed under edge removal.

Two graph properties $\mathcal{P}_1$ and $\mathcal{P}_2$ are defined in [3] to be *indistinguishable* if for every $\epsilon > 0$ and large enough n, any graph on n vertices satisfying one property is never ϵ-far from satisfying the other. It is shown in [3] that in this case, $\mathcal{P}_1$ is testable if and only if $\mathcal{P}_2$ is testable. It is first proved in [3] that certain colorability properties are testable with one-sided error. It is then shown that every first order graph property of type $\exists\forall$ is indistinguishable from some colorability property, thus obtaining that these properties are also testable. It would be interesting to characterize (combinatorially, logically, or by other means) the graph properties that are indistinguishable from some monotone property. By Theorem 1, this will immediately imply that these properties are testable, possibly with two-sided error.

Fischer and Newman [19] have recently shown that if a graph property $\mathcal{P}$ is testable, then it is also estimable; that is, it is possible to estimate how far a given graph is from satisfying $\mathcal{P}$, within an error $\delta > 0$ in time depending only on δ. Combining Theorem 1 and the result of [19] gives that any monotone property is estimable. We further note that this result (in fact, a stronger one) follows directly from the main result of [10], which was obtained independently of [19].

The proof of Theorem 5 gives weak lower bounds for the function $\delta_{\mathcal{P}}(\epsilon)$. It may be interesting to check if this dependency can be linear or polynomial for some natural

families $\mathcal{P}$.

Acknowledgments. We would like to thank Eldar Fischer and Oded Goldreich for their very helpful suggestions regarding the presentation of our results.

REFERENCES

[1] N. Alon, *Testing subgraphs in large graphs*, Random Structures Algorithms, 21 (2002), pp. 359–370.
[2] N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster, *The algorithmic aspects of the regularity lemma*, J. Algorithms, 16 (1994), pp. 80–109.
[3] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, Combinatorica, 20 (2000), pp. 451–476.
[4] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy, *Regular languages are testable with a constant number of queries*, SIAM J. Comput., 30 (2001), pp. 1842–1862.
[5] N. Alon and M. Krivelevich, *Testing k-colorability*, SIAM J. Discrete Math., 15 (2002), pp. 211–227.
[6] N. Alon and A. Shapira, *Testing satisfiability*, J. Algorithms, 47 (2003), pp. 87–103.
[7] N. Alon and A. Shapira, *Testing subgraphs in directed graphs*, J. Comput. System. Sci., 69 (2004), pp. 353–382.
[8] N. Alon and A. Shapira, *A Separation Theorem in Property-Testing*, manuscript.
[9] N. Alon and A. Shapira, *A characterization of the (natural) graph properties testable with one-sided error*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 429–438.
[10] N. Alon, A. Shapira, and B. Sudakov, *Additive approximation for edge-deletion problems*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 419–428.
[11] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., Wiley, New York, 2000.
[12] C. Avart, V. Rödl, and M. Schacht, *Every monotone 3-graph property is testable*, SIAM J. Discrete Math., 21 (2007), pp. 73–92.
[13] J. Balogh, B. Bollobás, and D. Weinreich, *Measures on monotone properties of graphs*, Discrete Appl. Math., 116 (2002), pp. 17–36.
[14] D. Eichhorn and D. Mubayi, *Edge-coloring cliques with many colors on subcliques*, Combinatorica, 20 (2000), pp. 441–444.
[15] P. Erdős, P. Frankl, and V. Rödl, *The asymptotic number of graphs not containing a fixed subgraph and a problem for hypergraphs having no exponent*, Graphs Combin., 2 (1986), pp. 113–121.
[16] P. Erdős and A. Gyárfás, *A variant of the classical Ramsey problem*, Combinatorica, 17 (1997), pp. 459–467.
[17] E. Fischer, *Testing graphs for colorability properties*, in Proceedings of the 12th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 873–882.
[18] E. Fischer, *The art of uninformed decisions: A primer to property testing*, The Computational Complexity Column of The Bulletin of the European Association for Theoretical Computer Science, 75 (2001), pp. 97–126.
[19] E. Fischer and I. Newman, *Testing versus estimation of graph properties*, in Proceedings of the 37th ACM Symposium on Theory of Computing, 2005, pp. 138–146.
[20] E. Fischer, I. Newman, and J. Sgall, *Functions that have read-twice constant width branching programs are not necessarily testable*, Random Structures Algorithms, 24 (2004), pp. 175–193.
[21] E. Friedgut and G. Kalai, *Every monotone graph property has a sharp threshold*, Proc. Amer. Math. Soc., 124 (1996), pp. 2993–3002.
[22] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.
[23] O. Goldreich and L. Trevisan, *Three theorems regarding testing graph properties*, Random Structures Algorithms, 23 (2003), pp. 23–57.
[24] R. L. Graham, B. L. Rothschild, and J. H. Spencer, *Ramsey Theory*, 2nd ed., Wiley, New York, 1990.
[25] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming*, J. ACM, 45 (1998), pp. 246–265.

[26] J. Komlós and M. Simonovits, *Szemerédi's regularity lemma and its applications in graph theory*, in Combinatorics, Paul Erdős Is Eighty, Vol. II, D. Miklós, V. T. Sós, and T. Szőnyi, eds., János Bolyai Math. Soc., Budapest, 1996, pp. 295–352.
[27] L. Lovász, *On the Shannon capacity of a graph*, IEEE Trans. Inform. Theory, 25 (1979), pp. 1–7.
[28] L. Lovász and B. Szegedy, *Graph Limits and Testing Hereditary Graph Properties*, manuscript, 2005.
[29] I. Newman, *Testing of functions that have small width branching programs*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 251–258.
[30] V. Rödl and R. Duke, *On graphs with small subgraphs of large chromatic number*, Graphs Combin., 1 (1985), pp. 91–96.
[31] D. Ron, *Property testing*, in Handbook of Randomized Computing, Vol. II, P. M. Pardalos, S. Rajasekaran, J. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 597–649.
[32] R. Rubinfeld, *Sublinear Time Algorithms*, manuscript.
[33] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[34] E. Szemerédi, *Regular partitions of graphs*, in Problèmes combinatoires et théorie des graphes, Colloq. Internat. CNRS 260, J. C. Bermond, J. C. Fournier, M. Las Vergnas, and D. Sotteau, eds., CNRS, Paris, 1978, pp. 399–401.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 523–550

THE ROUND COMPLEXITY OF TWO-PARTY RANDOM SELECTION*

SAURABH SANGHVI† AND SALIL VADHAN†

Abstract. We study the round complexity of two-party protocols for generating a random n-bit string such that the output is guaranteed to have bounded "bias," even if one of the two parties deviates from the protocol (possibly using unlimited computational resources). Specifically, we require that the output's statistical difference from the uniform distribution on $\{0,1\}^n$ is bounded by a constant less than 1. We present a protocol for the above problem that has $2\log^* n + O(1)$ rounds, improving a previous $2n$-round protocol of Goldreich, Goldwasser, and Linial (FOCS '91). Like the GGL Protocol, our protocol actually provides a stronger guarantee, ensuring that the output lands in any set $T \subseteq \{0,1\}^n$ of density μ with probability at most $O(\sqrt{\mu + \delta})$, where δ may be an arbitrarily small constant. We then prove a nearly matching lower bound, showing that any protocol guaranteeing bounded statistical difference requires at least $\log^* n - \log^* \log^* n - O(1)$ rounds. We also prove several results for the case when the output's bias is measured by the maximum *multiplicative* factor by which a party can increase the probability of a set $T \subseteq \{0,1\}^n$.

Key words. cryptography, distributed computing, coin flipping

AMS subject classifications. 68Q10, 68Q85

DOI. 10.1137/050641715

1. Introduction.

One of the most basic protocol problems in cryptography and distributed computing is that of *random selection*, in which several mutually distrusting parties aim to generate an n-bit random string jointly. The goal is to design a protocol such that even if a party cheats, the outcome will still not be too "biased." (There are many different choices for how to measure the "bias" of the output; the one we use will be specified later.) Random selection protocols can dramatically simplify the design of protocols for other tasks via the following common methodology: first, design a protocol in a model where truly random strings are provided by a trusted third party (generally a much easier task), and then use the random selection protocol to eliminate the trusted third party. Specific applications of this paradigm often require random selection protocols with specific additional properties (such as "simulatability"), but the basic requirement of bounded "bias" in the face of adversarial behavior is always present in some form and thus merits study on its own.

Because of their wide applicability, there is a large literature on random selection protocols, both in the computational setting, where cheating parties are restricted to polynomial time (starting with Blum's "coin flipping by telephone" [Blu82]), and in the information-theoretic setting, where security is provided even against computationally unbounded adversaries. There has also been a significant amount of recent work on random selection in the quantum setting, where the communication

*Received by the editors October 2, 2005; accepted for publication (in revised form) October 3, 2006; published electronically May 23, 2008. Preliminary versions of this work appeared in the first author's undergraduate thesis [San04] and in the conference paper [SV05].

http://www.siam.org/journals/sicomp/38-2/64171.html

†School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (ssanghvi@post.harvard.edu, salil@eecs.harvard.edu). The work of the first author was supported in part by a grant from the Harvard College Research Program and NSF grant CCR-0133096. The work of the second author was done in part while he was a Fellow at the Radcliffe Institute for Advanced Study at Harvard University. The work of the second author was supported by NSF grant CNS-0430336, ONR grant N00014-04-1-0478, and a Sloan Research Fellowship.

consists of quantum bits (qubits) and security is provided against a computationally unbounded quantum adversary; see [Amb04] and the references therein.

In this paper, we focus on *two-party* protocols in the (classical) *information-theoretic setting* (also known as the "full information model"). In addition to its stronger security guarantees, the information-theoretic setting has the advantage that protocols typically do not require complexity-theoretic assumptions (such as the existence of one-way functions). Various such random selection protocols have been used to construct perfectly hiding bit-commitment schemes [NOVY98], to convert honest-verifier zero-knowledge proofs into general zero-knowledge proofs [Dam94, DGW94, GSV98], to construct oblivious transfer protocols in the bounded storage model [CCM98, DHRS04], and to perform general fault-tolerant computation [GGL98]. There has also been substantial work in the k-party case for $k \geq 3$, where the goal is to tolerate coalitions of a minority of cheating players. This body of work includes the well-studied "collective coin-flipping" problem [BL89, Sak89, AN93, BN00, ORV94, RZ01, Fei99] (closely related to the "leader-election" problem) and again the use of random selection as a tool for general fault-tolerant computation [GGL98].

In most of the lines of work mentioned above (computational and information-theoretic, two-party and k-party), the *round complexity* has been a major parameter of interest. For some forms of random selection and their applications, constant-round protocols have been found (e.g., [DGW94, GSV98] improving [Dam94], [DHRS04] improving [CCM98], and [Lin01, KO04] improving [Blu82, Yao86]), but for others the best known protocols have a nonconstant number of rounds, e.g., [Cle86, NOVY98, GGL98, RZ01]. Lower bounds on round complexity, however, have proven much more difficult to obtain. In the computational setting, Cleve [Cle86] proved that for two-party random selection protocols, the number of rounds must grow linearly as the bias of the output tends to zero. (See also [CI93].) Ambainis [Amb04] gave a similar kind of result for two-party quantum protocols for leader election, a.k.a. weak coin flipping. As far as we know, all other previous round complexity lower bounds impose additional constraints on the protocol (beyond the basic security guarantee of bounded bias). For example, in the computational setting, it has been recently shown that five rounds are necessary and sufficient for random selection protocols satisfying a certain "black-box simulation" condition [KO04]. In the information-theoretic setting, a long line of work on the collective coin-flipping problem has culminated in the $(\log^* n + O(1))$-round protocol of Russell and Zuckerman [RZ01] (see also Feige [Fei99]), but the only known lower bound (of $\Omega(\log^* n)$ rounds), due to Russell, Saks, and Zuckerman [RSZ02], is restricted to protocols where each party can communicate only a small number of bits per round. Without this restriction, it is not even known how to prove that one round is impossible.

The problem and main results. As mentioned above, previous works on random selection have considered a number of different measures of the bias of the output, typically motivated by particular applications. Here we focus on what we consider to be the most natural measure—the statistical difference (i.e., total variation distance) of the output from the uniform distribution. The *statistical difference* between two random variables X and Y taking values in a universe $\mathcal{U}$ is defined to be $\max_{S \subseteq \mathcal{U}} |\Pr[X \in S] - \Pr[Y \in S]|$. We call the maximum statistical difference of the output from uniform when a player is honest (but when the other may deviate arbitrarily from the protocol) that player's *statistical guarantee*. We seek a two-party protocol that produces an output in $\{0,1\}^n$ such that both players' statistical guarantees are constant (i.e., bounded away from 1). Equivalently (see Lemma 2.4), we

want to satisfy the following criterion.

Statistical Criterion: There are fixed constants $\mu > 0$ and $\epsilon > 0$ such that for every n and every subset $T \subseteq \{0,1\}^n$ of density at most μ, the probability that the output lands in T is at most $1-\epsilon$, even if one party deviates arbitrarily from the specified protocol.

In addition to being a natural choice, this criterion is closely related to others considered in the literature. In particular, the standard criterion for the "collective coin-flipping" problem is that output bit $B \in \{0,1\}$ satisfies $\max\{\Pr[B=0], \Pr[B=1]\} < p$, where p is a constant less than 1; this is equivalent to B's statistical difference from uniform being bounded away from 1. (Here we see that the problem we consider is in some sense "dual" to collective coin flipping—we restrict ourselves to two players, but the output comes from a large set, whereas in collective coin flipping there are many players, but the output has only two possibilities.)

Of course, the first question is whether or not the Statistical Criterion can be met at all, regardless of round complexity. Indeed, being able to tolerate computationally unbounded cheating strategies is a strong requirement. In fact, when $n = 1$ (i.e., the output is a single bit), it turns out that one of the two parties can always force the outcome to be constant [Sak89]. This implies that the Statistical Criterion is impossible to meet for $\mu = 1/2$. Surprisingly, the criterion is achievable, however, for some smaller constant $\mu > 0$. This is implied by the following result of Goldreich, Goldwasser, and Linial [GGL98].

THEOREM 1.1 (see [GGL98]). *For every n, there is a two-party protocol producing output in $\{0,1\}^n$ such that, as long as one party plays honestly, the probability that the output lands in any set $T \subseteq \{0,1\}^n$ of density μ is at most $p = O(\sqrt{\mu})$. The protocol has $2n$ rounds.*

Notice that for sufficiently small μ, the probability p is indeed a constant less than 1. This implies that the Statistical Criterion is achievable with a *linear* number of rounds. Our goal is to determine the minimal round complexity of this problem.

First, we give a protocol achieving the Statistical Criterion with substantially fewer rounds than the above.

THEOREM 1.2. *For every constant $\delta > 0$, there is an efficient two-party protocol producing output in $\{0,1\}^n$ with $2\log^* n + O(1)$ rounds such that, as long as one party plays honestly, the probability that the output lands in any set T of density μ is at most $p = O(\sqrt{\mu + \delta})$.*

Our protocol is inspired by the $\log^* n$-round protocols for leader election [RZ01, Fei99] and Lautemann's proof that **BPP** is contained in the polynomial hierarchy [Lau83]. Specifically, we exhibit a two-round protocol that reduces the universe of size $N = 2^n$ to a universe of size polylog(N), while approximately preserving the density of the set T with high probability. Repeating this protocol $\log^* n$ times reduces the universe size to a constant, after which point we apply the GGL Protocol.

Second, we prove a lower bound that matches the above up to a factor of $2 + o(1)$.

THEOREM 1.3. *Any two-party protocol producing output in $\{0,1\}^n$ that satisfies the Statistical Criterion must have at least $\log^* n - \log^* \log^* n - O(1)$ rounds.*

Our proof of this theorem is a technically intricate induction on the game tree of the protocol. Roughly speaking, we associate with each node z of the game tree a collection $\mathcal{H}$ of very small sets such that if the protocol is started at z and R is a random subset of the universe of density $o(1)$, then one of the players X can force the outcome of the protocol to land in $R \cup S$ with probability $1 - o(1)$ for any $S \in \mathcal{H}$. The challenge is to keep the size of the sets in the collections $\mathcal{H}$ small as we induct up the game tree (so that they remain of density $o(1)$ when z is the root, which yields the

desired lower bound). In particular, a node can have an arbitrary number of children, and so we cannot afford to take unions of sets S occurring across all children. The key idea that allows us to keep the sets small is the following: We consider two cases: If we have a collection of sets that contains a large disjoint subcollection, then the random set R will contain one of the sets with high probability, and so we do not need to carry the set through the recursion. On the other hand, if the collection of sets has no large disjoint subcollection, then we show how we can use this fact to construct a successful strategy for the *other* player (based on how we inductively construct the collections $\mathcal{H}$).

We stress that our lower bound does not impose any additional constraint on the protocol, such as the number of bits sent per round. Thus, we hope that our techniques can help in establishing unrestricted lower bounds on round complexity for other problems, in particular for the collective coin-flipping (and leader-election) problem.

Results on multiplicative guarantees. A different measure of the quality of random selection protocol is a *multiplicative guarantee* ρ, whereby we require that even if one player cheats, the probability that the outcome lands in any set T of density μ is at most $\rho \cdot \mu$. The goal, naturally, is for ρ to be as small as possible (ideally a constant independent of n). Previous protocols, e.g., the one in [DGW94], have given a multiplicative guarantee to one player, while the other has a statistical guarantee (i.e., a bound on the output's statistical difference from uniform if the other cheats). Our observations and results on multiplicative guarantees are the following:

- If both parties have multiplicative guarantees ρ_A and ρ_B, then an argument of [GGL98] implies $\rho_A \cdot \rho_B \geq 2^n$, regardless of the number of rounds. On the other hand, for any desired ρ_A, there is a simple two-round protocol with multiplicative guarantees of ρ_A and $2^n/\rho_A$ for the two players.
- If one party has a multiplicative guarantee ρ and the other has a statistical guarantee ε, then $\varepsilon \geq 1/\rho - 1/2^n$. This explains inverse relationships in existing protocols of [DGW94] (where $\varepsilon = 1/\mathrm{poly}(n)$ and $\rho = \mathrm{poly}(n)$) and [GSV98] (where $\varepsilon = \mathrm{poly}(n) \cdot 2^{-k}$ and $\rho = 2^k$ for any k).[1]
- There is a protocol with $2\log^* n + O(1)$ rounds that provides a constant statistical guarantee to one player and a $1+\delta$ multiplicative guarantee to the other, for an arbitrarily small constant δ. Theorem 1.3 implies that this round complexity is tight up to a constant factor, because a constant multiplicative guarantee implies a constant statistical guarantee.

Notation for logarithms. As in other work [RZ01], for the purposes of this paper, we define $\log_b^{(k)} n$ to be k base-b iterated logarithms of n, with 1 being a minimum value:

$$\log_b^{(k)} n = \begin{cases} 1 & : \text{ if } \log_b^{(k-1)} n < b, \\ \log_b \left(\log_b^{(k-1)} n \right) & : \text{ otherwise,} \end{cases}$$

with $\log_b^{(0)} n = n$. Moreover, for $n \geq 1$, we define $\log_b^* n$ to be the least natural number k such that $\log_b^{(k)} n = 1$. Throughout the paper, we take the base of the logarithms to be $b = 2$ unless otherwise specified.

[1] Actually, the protocol of [GSV98] does not provide a multiplicative guarantee of 2^k but rather ensures that the probability that the output lands in any set T of density μ is at most $2^k \cdot \mu + o(1)$. Our lower bound also applies to this more general type of guarantee.

2. Defining random selection protocols. Although we introduced the problem for a universe $\{0,1\}^n$, for the rest of the paper we assume we have an arbitrary universe $\mathcal{U}$. We can formally characterize a random selection protocol as follows.

DEFINITION 2.1. *A* random selection protocol $\Pi = (A, B, f)$ *over a universe* $\mathcal{U}$ *consists of a pair of programs* A *and* B *and a function* f *such that we have the following:*

- *Both* A *(Alice) and* B *(Bob) alternately output strings ("messages")* m_i *of arbitrary length that are a function of the conversation thus far and their sequences of random coin tosses* r_A *and* r_B*, respectively. That is,* $m_1 = A(r_A)$*,* $m_2 = B(r_B, m_1)$*,* $m_3 = A(r_A, m_1 m_2)$*, etc.*
- *The* conversation *between Alice and Bob is the transcript* $(A, B) = (m_1, m_2, \ldots, m_r)$*, where* r *is a parameter defining the number of messages (also called the number of "rounds" or "turns") of the protocol.*
- *The* output *of the protocol is* $f((m_1, m_2, \ldots, m_r))$*, which is some element of* $\mathcal{U}$*.*

We are interested in the behavior of the protocol when one of these programs is replaced with an arbitrary "cheating" program A^* or B^*, which may send its messages as an arbitrary function of the conversation and input length. If the cheating program "aborts" or sends an irregular message (too long, ill-formed, etc.), the protocol can assume it has sent the empty string.

Although the formulation we have provided assumes a protocol operates over a single fixed universe, in general we will be interested in studying asymptotic behavior of protocols as the universe size increases. Thus, we define a *random selection protocol ensemble* to be a sequence $(\Pi^{(1)}, \Pi^{(2)}, \ldots)$ where each $\Pi^{(N)}$ is a protocol over $\mathcal{U} = \{1, \ldots, N\}$. From now on, we blur the distinction between random selection protocols over a fixed universe and random selection protocol ensembles.

Two additional desirable properties of random selection protocols are the following: (a) the output is uniformly distributed in $\mathcal{U}$ assuming honest players; (b) in a protocol ensemble, honest strategies can be computed in time polynomial in the output length, $\log N$. Our protocols will have these properties, but our lower bounds will apply even to protocols without them.

We now introduce a formalism that will be essential in the proofs in this paper.

DEFINITION 2.2. *Given a protocol* Π *over universe* $\mathcal{U}$*, define the* game tree T *as follows:*

- *A set of nodes* V*, each representing a partial transcript of messages,* $(m_1, \ldots, m_i)$*.*
- *A set of edges* E*, defined by* $(u, v) \in E$ *if and only if* $u = (m_1, \ldots, m_i)$ *and (abusing notation)* $v = (u, m_{i+1})$*, for some message* m_{i+1}*. That is,* u *has* v *as a* child *if* v *is a potential protocol state one message after state* u*. Note that this makes* T *a tree, rooted at the empty transcript.*
- *For each node* z*, a distribution* $\mathcal{D}_z$ *over the children* z_i *whereby* A *or* B *chooses the next message (where the children are all nodes* z_i *such that* $(z, z_i) \in E$*).*
- *For every leaf* $z = (m_1, \ldots, m_r)$*, a label equal to* $f((m_1, \ldots, m_r))$*, the output of the protocol ending at node* z*.*

One can verify that this formalism produces an equivalent specification to Definition 2.1 of a random selection protocol.

Just as any node of a tree can be viewed as the root of another tree, any node of a protocol's game tree induces its own random selection protocol starting from that state. We simply fix the messages leading to that node and have the players choose the remaining messages as in the original protocol. This observation is one of the

main reasons that the abstraction of a random selection protocol as a tree will prove useful.

Evaluating a random selection protocol. We evaluate random selection protocols with metrics measuring how "close" the output is to the uniform distribution on $\mathcal{U}$. The primary metric we use is the following.

DEFINITION 2.3. *The* statistical difference from uniform *of a distribution X over universe $\mathcal{U}$ is defined to be*

$$\max_T \left| \Pr_{x \leftarrow X}[x \in T] - \mu(T) \right| = \max_T \left(\Pr_{x \leftarrow X}[x \in T] - \mu(T) \right),$$

where $T \subseteq \mathcal{U}$ and $\mu(T)$ is the density of T in $\mathcal{U}$, $|T|/|\mathcal{U}|$.

It can be verified that this distance is in the interval $[0, 1 - 1/N]$, where N is the size of the universe $\mathcal{U}$. A statistical difference of 0 implies that X is uniform, and $1 - 1/N$ implies X is concentrated on a single point. It is equal to one-half of the ℓ_1 distance between X and the uniform distribution, where we view each as a vector in $[0, 1]^N$.

We will want to avoid output distributions X whose statistical difference from uniform is very close to 1. The following lemma demonstrates that this (undesirable) property is equivalent to X landing in a small set with high probability.

LEMMA 2.4. *If X has statistical difference at least $1 - \epsilon$ from uniform, then there exists a set T such that $\mu(T) \leq \epsilon$ and $\Pr_{x \leftarrow X}[x \in T] \geq 1 - \epsilon$. Conversely, if there exists such a set T, then X has statistical difference at least $1 - 2\epsilon$ from uniform.*

Proof. If X has statistical difference at least $1 - \epsilon$ from uniform, then there exists a set T such that $\Pr[x \in T] - \mu(T) \geq 1 - \epsilon$. Since $\Pr[x \in T] \leq 1$, we can conclude that $\mu(T) \leq \epsilon$, and since $\mu(T) \geq 0$, we can conclude that $\Pr[x \in T] \geq 1 - \epsilon$. The second statement follows directly from the definition of statistical difference. ∎

Given these metrics, we can define the following.

DEFINITION 2.5. *Let $\Pi = (A, B, f)$ be a random selection protocol. The* statistical guarantee *for Alice playing honest strategy A in Π, denoted ϵ_A, is the maximum over all B^* of the statistical difference between the distribution of $f((A, B^*))$ and the uniform distribution over $\mathcal{U}$. The guarantee for Bob is defined analogously.*

Intuitively, the guarantee of a protocol for a player bounds the damage that the opponent can effect on the distribution by deviating from the protocol. Unfortunately, the terminology here is a bit counterintuitive—the lower the number, the better the guarantee. We will try to avoid confusion by saying a guarantee is "at best x," rather than "at least x."

Armed with this notion of a guarantee, we can state the following important equivalence, following directly from Lemma 2.4.

PROPOSITION 2.6. *The Statistical Criterion is equivalent to both of the statistical guarantees of a protocol being bounded away from* 1.

Later on, we will prove the following proposition, which lower bounds the ability of any protocol to provide strong statistical guarantees to both players simultaneously.

PROPOSITION 2.7. *In any random selection protocol Π over universe $\mathcal{U}$ achieving statistical guarantees ϵ_A and ϵ_B, $\epsilon_A + \epsilon_B \geq 1 - 1/N$, where $N = |\mathcal{U}|$.*

In addition to statistical guarantees, we also consider multiplicative guarantees, which come from bounds on multiplicative difference.

DEFINITION 2.8. *The* multiplicative ratio *of a distribution X is*

$$\max_T \Pr_{x \leftarrow X}[x \in T]/\mu(T),$$

where T ranges over all nonempty subsets of $\mathcal{U}$. Similarly to Definition 2.5*, for a random selection protocol $\Pi = (A, B, f)$, we define the* multiplicative guarantee *ρ_A (resp., ρ_B) for Alice (resp., Bob) by taking the maximum multiplicative ratio over all cheating strategies B^* (resp., A^*).*

The multiplicative ratio is always a rational in $[1, N]$, where 1 implies the uniform distribution, and N implies one element is chosen with probability 1. The multiplicative ratio of a distribution is actually equal to the factor by which a single element's probability of being the protocol's output can be increased from uniform. Formally, we have the following lemma.

LEMMA 2.9. *The multiplicative ratio of a distribution X is equal to*

$$\max_{s \in \mathcal{U}} \left(N \cdot \Pr_{x \leftarrow X}[x = s] \right).$$

Later on, we will prove the following result from [GGL98].

THEOREM 2.10 (see [GGL98]). *For any protocol Π, we have $\rho_A \cdot \rho_B \geq N$.*

When the universe is $\{0,1\}^n$ (i.e., $N = 2^n$), this theorem implies that one player can always increase the probability that a single element is chosen by an exponential factor—namely, $2^{n/2}$.

While both measures bound the deviation of a distribution from uniform, multiplicative ratio tends to focus on the concentration of probability into small sets (indeed, by Lemma 2.9, sets of size 1), while statistical difference will prove more useful when considering larger subsets (e.g., a constant fraction of the universe).

This said, we can prove some basic relationships between the two metrics that will prove useful.

LEMMA 2.11. *Let X be an arbitrary distribution over universe $\mathcal{U}$, with $N = |\mathcal{U}|$. Denote by ϵ the statistical difference of X from uniform and by ρ the multiplicative difference from uniform. Then*

1. $\rho \leq N\epsilon + 1$,
2. $\epsilon \leq 1 - 1/\rho$.

Part 2 of Lemma 2.11 implies that a distribution with a constant multiplicative ratio will have a constant statistical difference, though the converse is not necessarily true. Put another way, a strong multiplicative guarantee is harder to achieve than a strong statistical guarantee.

Proof of Lemma 2.11. By Lemma 2.9, we have $\rho = N \cdot \max_{s \in \mathcal{U}} \Pr_{x \leftarrow X}[x = s]$. But by the definition of statistical difference, we have for any $x \in \mathcal{U}$, $\Pr_{x \leftarrow X}[x = s] \leq \epsilon + 1/N$, by setting $T = \{s\}$. Part 1 of the lemma follows.

For part 2, it suffices to show that for all T, $\Pr_{x \leftarrow X}[x \in T] - \mu(T) \leq 1 - 1/\rho$. If $\mu(T) \geq 1/\rho$, then this certainly holds. Otherwise, $\Pr_{x \leftarrow X}[x \in T] \leq |T| \cdot (\rho/N) = \mu(T)\rho$, which implies that $\Pr_{x \leftarrow X}[x \in T] - \mu(T) \leq \Pr_{x \leftarrow X}[x \in T](1 - 1/\rho) \leq 1 - 1/\rho$. ∎

3. The Iterated Random Shift Protocol. In this section, we describe the main protocol of this paper, the Iterated Random Shift Protocol, and prove its main properties. That is, we show that for any constant δ, Iterated Random Shift is a $(2\log^* N + O(1))$-round protocol where the probability that the output falls in a set of density μ is at most $O(\sqrt{\mu + \delta})$. It follows that the protocol satisfies the Statistical Criterion given above.

3.1. The GGL Protocol. We begin by briefly describing the $2\log N$-round protocol satisfying the Statistical Criterion given by Goldreich, Goldwasser, and Linial [GGL98], which we will use in the construction of our protocol.

The GGL Protocol (idealized). Let $\mathcal{U}$ be a universe of size N, where N is a power of 2.

1. Alice randomly divides $\mathcal{U}$ into two equal-sized halves and sends this partition to Bob.
2. Bob randomly selects one of the halves, randomly divides it, and sends the resulting partition to Alice.
3. This process continues until one element remains: this element is the output of the protocol.

Actually, to obtain a protocol with computation time polylogarthmic in N, the authors use pairwise independent partitions of the universe. They first prove that this protocol achieves that, as long as one party plays honestly, the probability that the output lands in any set $T \subseteq \mathcal{U}$ of density μ is at most $p = O(\mu^{1/4})$.

They go on to improve this result by using a slightly different protocol to achieve Theorem 1.1 (Theorem 23 in [GGL98]), which improves the bound to $p = O(\sqrt{\mu})$.

3.2. The Random Shift Protocol. The Iterated Random Shift Protocol is inspired by the $\log^* n$-round protocols for leader election [RZ01, Fei99] and Lautemann's proof that **BPP** is contained in the polynomial hierarchy [Lau83]. It is built by iteration of the following two-round protocol, which we will call the Random Shift Protocol.

The Random Shift Protocol. Given a universe $\mathcal{U}$ of size N and $m \in \mathbb{N}$,

1. Alice uniformly randomly selects and sends a sequence of elements $a_1, \ldots, a_m \in \mathcal{U}$;
2. Bob uniformly randomly selects and sends a sequence of elements $b_1, \ldots, b_m \in \mathcal{U}$;
3. output the sequence $(a_i + b_j)_{1 \le i,j \le m}$, where $+$ is a group operation over $\mathcal{U}$.

Note that the Random Shift Protocol is not, strictly speaking, a random selection protocol over $\mathcal{U}$: its output is a *sequence* of elements from the universe. In using it, we will typically choose the parameter m so that the number of output elements, m^2, is much smaller than N (e.g., $m = \text{polylog}(N)$) and recursively use our random selection protocol to select one of the m^2 output elements. To show that this approach yields a protocol with bounded statistical guarantees, we argue that even if one of the players cheats, any subset T of the universe is unlikely to appear in much more than a $\mu(T)$ fraction of the outputs of the Random Shift Protocol. This is formalized by the following lemma.

LEMMA 3.1. *Let T be an arbitrary subset of $\mathcal{U}$. Let $\mu(T) = |T|/N$, and let $\mu'(T)$ denote the density of T in the sequence output by the Random Shift Protocol: $\mu'(T) = \#\{(i,j) : a_i + b_j \in T\}/m^2$. Then as long as one player plays honestly (i.e., chooses elements uniformly at random) and $m \ge (1/2\delta^2) \cdot \log(N/\epsilon)$, we have*

$$\Pr[\mu'(T) \ge \mu(T) + \delta] \le \epsilon.$$

That is, when one player is honest, the sequence $(a_i + b_j)_{ij}$ will be sufficiently random so that it is very unlikely that the density of T in the output sequence will increase substantially.

Proof. Suppose Alice plays honestly and chooses her elements $a_1, \ldots, a_m$ uniformly at random from $\mathcal{U}$. The lemma certainly holds a fortiori for an honest Alice, as a cheating Bob can see what elements Alice has selected.

For each element $b \in \mathcal{U}$, define the random variables

$$X_i^{(b)} = \begin{cases} 1 & \text{if } (a_i + b) \in T, \\ 0 & \text{otherwise.} \end{cases}$$

Then define $X^{(b)} = (1/m)\sum_{i=1}^{m} X_i^{(b)}$. Notice that $\mathrm{E}[X_i^{(b)}] = \mu(T)$ and, for each b, the random variables $X_1^{(b)}, \ldots, X_m^{(b)}$ are mutually independent.

By a Chernoff bound, we may conclude the following for any δ:

$$\Pr[X^{(b)} \geq \mu(T) + \delta] \leq e^{-2\delta^2 m} \leq 2^{-2\delta^2 m} \leq \epsilon/N.$$

Using a union bound, we conclude that

$$\Pr[\exists b \in \mathcal{U} \text{ such that } X^{(b)} \geq \mu(T) + \delta] \leq N \cdot \epsilon/N = \epsilon.$$

But if for all b we have $X^{(b)} < \mu(T) + \delta$, then no matter what elements $b_1, \ldots, b_m$ Bob chooses, we have

$$\mu'(T) = (1/m)\sum_{j=1}^{m} X^{(b_j)} < \mu(T) + \delta.$$

It follows that $\Pr[\mu'(T) \geq \mu(T) + \delta] \leq \epsilon$ as desired. □

Remark 3.2. We note that the number of elements sent by Bob need only be $(1/2\delta^2) \cdot \log(1/\epsilon)$ (i.e., the $\log N$ factor can be eliminated), since there is no need to do a union bound as in the above proof when proving Bob's guarantee. However, the symmetry of the protocol as presented above has the advantage that it can actually be implemented in *one* round in a model of simultaneous communication (where honest parties can send messages at the same time, but a cheating party may wait to see the other party's message before sending its own message), as is typically used in many-party protocols (e.g., leader election and collective coin flipping). This reduces the round complexity of our Iterated Random Shift Protocol below to $\log^* N + O(1)$ in the simultaneous communication model. It is interesting to know whether our lower bound of $\log^* N - \log^* \log^* N - O(1)$ rounds (in section 4.2) can be extended to the simultaneous communication model (without paying the factor of 2 in the trivial reduction to our nonsimultaneous model), since we would then have bounds in that model that are tight up to a factor of $1 + o(1)$.

3.3. The Iterated Random Shift Protocol. We now describe our Iterated Random Shift Protocol satisfying Theorem 1.2, which consists of recursively applying the Random Shift Protocol until the universe size is small (say, less than a fixed constant), after which we apply the GGL Protocol from [GGL98] discussed in section 3.1. Formally, we have the following.

The Iterated Random Shift Protocol. Given a universe $\mathcal{U}$ of size N and $M \in \mathbb{N}$ being a sufficiently large "cutoff parameter" that is a power of 2, we have the following three cases:

1. If $N > M^2$, then letting $m = \max\{M, \lceil \log^3 N \rceil\}$, execute the following:
   (a) Alice uniformly randomly selects and sends a sequence of elements $a_1, \ldots, a_m \in \mathcal{U}$.
   (b) Bob uniformly randomly selects and sends a sequence of elements $b_1, \ldots, b_m \in \mathcal{U}$.
   (c) Recursively execute the protocol on universe $\mathcal{U}' = [m] \times [m]$ to obtain result (i, j) and output $a_i + b_j$.
2. If $N = M^2$, run the GGL Protocol on $\mathcal{U}$ and output its result.
3. If $N < M^2$, recursively use the protocol on universe $\mathcal{U}' = \mathcal{U} \times [M^2]$ to obtain result (x, y) and output x.

First, we observe that, assuming M is chosen sufficiently large, the recursion will always terminate with an application of case 2 (the GGL Protocol): When we run case 1, we have $|\mathcal{U}'| = m^2 = \max\{M^2, \lceil \log^3 N \rceil\} \leq \max\{M^2, N-1\}$ for sufficiently large N, and so eventually the universe size will equal M^2, provided M is large enough. Case 3 will be executed at most once and is present only to avoid running the GGL Protocol immediately if the universe size is not a power of 2.

Observe that the output of this protocol is uniform if both players are playing honestly—by symmetry, all elements of the universe are equally likely to be selected. We note also that, by inspection, the honest players' strategies and the output of the protocol can certainly be computed in polynomial time.

We now analyze the round complexity of the Iterated Random Shift Protocol.

PROPOSITION 3.3. *For all sufficiently large M and all N, the Iterated Random Shift Protocol over a universe $\mathcal{U}$ of size N with parameter M takes $2\log^* N + O(\log M)$ rounds. Moreover, the strategies of the players are computable in time $\mathrm{poly}(\log N, \log M)$.*

Proof. Each application of the Random Shift Protocol (except for the last) reduces the universe size from N to $\lceil \log^3 N \rceil^2 < \log^7 N$ for sufficiently large N and takes two rounds. A lemma proven in [RZ01] states that if $f(n) \leq \log^a n$ for some constant a, then $f^{(\log^* n)}(n) \leq b$ for some constant b depending only on a, where $f^{(k)}$ represents k repeated applications of f. This implies that if M is sufficiently large and the initial universe size is $N \geq M^2$, the Random Shift Protocol is applied at most $\log^* N$ times. (If $N < M^2$, then we apply the Random Shift Protocol at most $\log^*(NM^2) = \log^* N + O(\log M)$ times.) By Theorem 1.1, the GGL Protocol on a universe of size at most M^2 takes at most $4\log M$ rounds.

As for the efficiency of the protocol, note that the players need only generate in each round a number of random bits that is polylogarithmic in the size of the universe. ∎

3.4. The statistical guarantees of the Iterated Random Shift Protocol.

THEOREM 3.4. *If $M \geq 1/\delta^3$, then for any set $T \subseteq \mathcal{U}$, the probability that the output of the Iterated Random Shift Protocol lands in T is $O(\sqrt{\mu + \delta})$, where $\mu = \mu(T)$, assuming at least one player plays honestly.*

COROLLARY 3.5. *For a sufficiently large constant M, the Iterated Random Shift Protocol satisfies the Statistical Criterion. Equivalently, there exists a constant $\epsilon > 0$ such that the Iterated Random Shift Protocol achieves $\max\{\epsilon_A, \epsilon_B\} \leq 1 - \epsilon$.*

Observe that Theorem 3.4 is much stronger than what we need to show Corollary 3.5. Using Theorem 3.4, we know that when one player is honest, for any "small" set T, the probability that the output falls in T is close to zero. The Statistical Criterion requires only that this probability is not arbitrarily close to 1.

Proof of Theorem 3.4. The key idea is that in the ith application of the Random Shift Protocol, we can bound the increase in density of any particular set T by at most some small δ_i (with high probability) and these δ_i's can be chosen so that $\sum_i \delta_i \leq \delta$. The Iterated Random Shift Protocol concludes by applying the GGL Protocol to this small universe, and then Theorem 1.1 gives us the result.

We first note that the modification of the protocol in case $N < M^2$, selecting from $\mathcal{U} \times [M^2]$ and taking the first component, does not affect the property claimed in the theorem (because the density of $T \times [M^2]$ in $\mathcal{U} \times [M^2]$ equals the density of T in $\mathcal{U}$). Thus we assume that $N \geq M^2$, and let $N_0, N_1, \ldots, N_{k^*}$ be the universe sizes in an execution of the Iterated Random Shift Protocol, where k^* is the first value of k such that $N_k = M^2$. That is,

$$N_0 = N,$$
$$N_k = m_k^2 \quad \text{for } m_k = \max\{M, \lceil \log^3 N_{k-1} \rceil\}.$$

Note that for sufficiently large M, the sequence of N_i's is strictly decreasing, and there exists a finite k^* such that $N_{k^*} = M^2$.

Given a subset $T \subseteq \mathcal{U}$, we track how T evolves through an execution of the Iterated Random Shift Protocol using the following notation for $k = 0, \ldots, k^*$:

$$\begin{aligned} &\mathcal{U}_0 = \mathcal{U}, && \mathcal{U}_k = [m_k] \times [m_k], \\ &T_0 = T, && T_k = \{(i,j) \in \mathcal{U}_k : (a_i + b_j) \in T_{k-1},\ 1 \le i, j \le m_k\}, \\ &\mu(T_k) = |T_k|/|\mathcal{U}_k|, \end{aligned}$$

where in the definition of T_k, (a_i) and (b_j) are the sequences of elements of $\mathcal{U}_{k-1}$ chosen by Alice and Bob in the kth application of the Random Shift Protocol, and $+$ is the group operation over $\mathcal{U}_{k-1}$ used in the protocol.

Intuitively, $\mathcal{U}_k$ is the remaining universe (of size N_k) after k iterations, and T_k represents the portion of the remaining universe such that choosing $(i,j) \in T_k$ will lead to an element of T being the output of the whole protocol. We call $\mu(T_k)$ the "effective density" of T in the kth iteration.

CLAIM 3.6. *There is a finite constant C independent of N and M such that we have*

$$\Pr\left[\mu(T_{k^*}) \ge \mu(T) + C \cdot M^{-1/3}\right] \le C \cdot 2^{-M^{1/3}},$$

provided at least one party plays honestly.

Proof. Recall that in the kth iteration, we are applying the Random Shift Protocol with parameter $m = m_k = \max\{M, \lceil \log^3 N_{k-1} \rceil\}$. Define $\epsilon_k = 2^{-m_k^{1/3}}$, and $\delta_k = 1/m_k^{1/3}$. Notice that $m_k \ge (1/2\delta_k^2) \cdot \log(N_{k-1}/\epsilon_k)$.

Using Lemma 3.1 repeatedly in an induction and using a union bound, we have that for any k,

$$\Pr\left[\mu(T_k) \ge \mu(T) + \sum_{i=1}^{k} \delta_i\right] \le \sum_{i=1}^{k} \epsilon_i.$$

Since the N_k's are decreasing exponentially fast, we have

$$\begin{aligned} \sum_{i=0}^{k^*} \delta_i &= O(\delta_{k^*}) \\ &= O(1/m_{k^*}^{1/3}) \\ &= O(1/M^{1/3}). \end{aligned}$$

Similarly,

$$\sum_{i=1}^{k^*} \epsilon_i = O(2^{-m_{k^*}^{1/3}}) = O(2^{-M^{1/3}}).$$

This completes the proof. ∎

Applying Claim 3.6 and using Theorem 1.1, we deduce that the probability that the output lands in T is at most

$$O\left(\sqrt{\mu(T) + C \cdot M^{-1/3}}\right) + C \cdot 2^{-M^{1/3}} = O\left(\sqrt{\mu(T) + M^{-1/3}}\right) = O\left(\sqrt{\mu(T) + \delta}\right),$$

using $M \geq 1/\delta^3$. Theorem 3.4 is proven. □

Recalling Proposition 3.3, and taking M to be a sufficiently large constant, we obtain a protocol with $2\log^* N + O(1)$ rounds satisfying the Statistical Criterion, thereby proving Theorem 1.2. More generally, we obtain a protocol of $2\log^* N + O(\log(1/\delta))$ rounds such that the output lands in a sets of density μ with probability at most $O(\sqrt{\mu + \delta})$. Note that we can take δ to be a slowly vanishing function of N and still have $O(\log^* N)$ rounds.

3.5. The multiplicative guarantees of the Iterated Random Shift Protocol. In this section, we discuss the multiplicative guarantees provided by the Iterated Random Shift Protocol. Later, we will see how lower bounds require that one of the players (in this case, Alice) receives a very poor multiplicative guarantee; however, we will see that Bob receives a very strong guarantee. In this way, we can say something about the ability of a protocol to provide a strong multiplicative guarantee to one player, while providing a strong statistical guarantee to the other. Specifically, we establish the following theorem.

THEOREM 3.7. *There exist constants $\epsilon < 1$ and ρ such that the Iterated Random Shift Protocol with the cutoff parameter M taken to be a sufficiently large constant achieves guarantees $\rho_B \leq \rho$ and $\epsilon_A \leq \epsilon$.*

This is the first protocol achieving constant statistical and multiplicative guarantees that we know of, and later we will prove Theorem 1.3, which, together with Lemma 2.11, implies that it has optimal round complexity (up to a factor of $2+o(1)$). (See Corollary 5.2.)

Given Corollary 3.5, to prove Theorem 3.7, it suffices to show the following.

PROPOSITION 3.8. *Let Π be a Iterated Random Shift Protocol defined with constant cutoff parameter M. Then Π provides a constant multiplicative guarantee to Bob: there exists constant ρ such that, as long as Bob plays honestly, the output of the Iterated Random Shift Protocol will fall in a set T with probability at most $M^2 \cdot \mu(T)$, for any set T.*

Proof of Proposition 3.8. Fix an arbitrary set $T \subseteq \mathcal{U}$. We use the notation from the proof of Theorem 1.2; in particular, $\mathcal{U}_k$ is the remaining universe after k iterations, and T_k is the set of elements of $\mathcal{U}_k$ corresponding to elements of T. The following is the key lemma.

LEMMA 3.9. *Assuming Bob plays honestly,* $\mathrm{E}[\mu(T_k)] = \mathrm{E}[\mu(T_{k-1})]$ *for all* $k = 1, \ldots, k^*$.

Proof. Consider the Random Shift Protocol. Let $a_1, \ldots, a_m$ be given. Then if $b_1, \ldots, b_m$ are chosen uniformly at random, it follows that for each i, j, the element $a_i + b_j$ is uniform over $\mathcal{U}$ (since $+$ is a group operation), and thus $\Pr[a_i + b_j \in T] = \mu(T)$. By linearity of expectations, we can conclude that $\mathrm{E}[\#(a_i + b_j) \in T] = \mu(T) \cdot m^2$ (where m^2 is the size of the new universe), and thus $\mathrm{E}[\mu(T')] = \mu(T)$, where $\mu(T')$ is the residual density of T in the resulting universe.

Applying this logic within the Iterated Random Shift Protocol, the lemma is proven (since for given $\mu(T_{k-1})$, we know $\mathrm{E}[\mu(T_k)] = \mu(T_{k-1})$). □

By induction, we then have that for all k, $\mathrm{E}[\mu(T_k)] = \mu(T)$. In particular, this is true for $k = k^*$. We can then derive

$$\begin{aligned}\mu(T) &= \mathrm{E}[\mu(T_{k^*})] \\ &\geq (1/M^2) \cdot \mathrm{E}[|T_{k^*}|] \\ &\geq (1/M^2) \cdot \Pr[|T_{k^*}| > 0].\end{aligned}$$

Since if $|T_{k^*}| = 0$, the protocol's output cannot possibly fall in T, we conclude that the probability the output falls in T is at most $M^2 \cdot \mu(T)$. This proves the proposition and thus Theorem 3.7. □

As an aside, notice that by using Lemma 2.11, Proposition 3.8 allows us to conclude one half of Theorem 1.2: the Iterated Random Shift Protocol provides a constant statistical guarantee to Bob.

We can conclude that the Iterated Random Shift Protocol has the following properties:

- It has only $2\log^* N + O(1)$ rounds.
- It provides both Alice and Bob with constant statistical guarantees (equivalently, it satisfies the Statistical Criterion).
- It provides Bob with a constant multiplicative guarantee.

Notice that in the above proof, we never used the multiplicative guarantee properties of the GGL Protocol—we simply relied on the initial recursions of Random Shift to provide the strong guarantee to Bob.

In fact, by changing the protocol used when the universe size becomes of size M^2 in the definition of the Iterated Random Shift Protocol, we can improve even further the multiplicative guarantee given to Bob. The current protocol implies only that Bob gets *some* constant multiplicative guarantee. Specifically, consider the following simple two-round protocol.

The Random Set Protocol. Given universe $\mathcal{U}$ of size N and parameter K,

1. Alice selects a subset S of $\mathcal{U}$ of size K, uniformly at random, and sends S to Bob;
2. Bob selects an element $x \in S$, uniformly at random;
3. the output is x.

It is straightforward to prove the following.

PROPOSITION 3.10. *For all positive integers $N \geq K$, the Random Set Protocol provides multiplicative guarantees $\rho_A = K$ and $\rho_B = N/K$.*

Thus, by using the Random Set Protocol on the universe of size M^2 with parameter $K = M^2/(1+\gamma)$ instead of GGL, Bob can achieve a multiplicative guarantee $1+\gamma$, while still keeping Alice's statistical guarantee constant (when γ is constant—if the residual density of Bob's target set T is smaller than $1 - 1/(1+\gamma)$, there is a nonzero probability that Alice will choose a set S disjoint from T).

In the next section, we will prove that the Iterated Random Shift Protocol has optimal round complexity, up to a factor of $2 + o(1)$, among protocols achieving the Statistical Criterion.

4. Lower bounds on statistical guarantees.

4.1. Tradeoffs between statistical guarantees. As a warmup to our main lower bound, in this section we present a tradeoff between the statistical guarantees ϵ_A and ϵ_B of Alice and Bob, respectively.

PROPOSITION 4.1 (Proposition 2.7, restated). *In any random selection protocol* Π *over universe* $\mathcal{U}$ *achieving statistical guarantees* ϵ_A *and* ϵ_B, $\epsilon_A + \epsilon_B \geq 1 - 1/N$, *where* $N = |\mathcal{U}|$.

COROLLARY 4.2. *In any random selection protocol* Π, $\max\{\epsilon_A, \epsilon_B\} \geq 1/2 - 1/(2N)$.

Proof. Suppose we have a protocol (A, B, f), where $\epsilon_A + \epsilon_B < 1 - 1/N$. Then we can partition the universe into two sets, S and $\mathcal{U} \setminus S$, where $|S| > \epsilon_A N$ and $|\mathcal{U} \setminus S| > \epsilon_B N$. Now, the argument follows logic similar to the impossibility result of Saks [Sak89] for collective coin flipping when at least half of the players are dishonest (where we think of outcomes in S as "heads" and $\mathcal{U} \setminus S$ as "tails").

Specifically, we view the protocol as a game where Alice wins if the output lands in S and Bob wins if the output lands in $\mathcal{U} \setminus S$. A well-known result in game theory is Zermelo's theorem: it implies that one of the players will have a winning strategy (one that wins regardless of how the other player plays). The basic reasoning is *backwards induction* on the game tree: every leaf node can be labeled A-WIN or B-WIN, depending on whether the output is in S or $\mathcal{U} \setminus S$, respectively, and then we can inductively label the remaining nodes depending on whether there exists a winning child for the current player to select. If there is, the current player can choose that child and will thus have a winning strategy from the current node. If there is not, then the opposing player can certainly win from the current node, as he or she has a winning strategy from all the children of the node.

This result implies one of the following:

- There exists strategy A^* where $\Pr[f((A^*, B^*)) \in S] = 1$, for any B^*. Taking $B^* = B$ (Bob's honest strategy), we have $\Pr[f((A^*, B)) \in S] - \mu(S) = 1 - \mu(S) > \epsilon_B$. This contradicts the guarantee of ϵ_B for Bob.
- There exists strategy B^* where $\Pr[f((A^*, B^*)) \in \mathcal{U} \setminus S] = 1$, for any A^*. This similarly contradicts guarantee ϵ_A. ∎

The main intuition behind the above proof is that, at every stage, either there exists a move that is good for the current player or all moves are good for the other player. In either case, the result is good for one of the two players. All that is needed is a way to make sure that every node on the bottom level can be defined as "winning" for someone and that this notion can propagate up the tree. As we will see, this type of reasoning will figure strongly in the proof of our main lower bound. There, the primary challenge will be to handle the cases when some nodes do not appear to be "winning" for either player.

4.2. The main lower bound. In this section, we prove Theorem 1.3, giving a lower bound on round complexity matching the Iterated Random Shift Protocol up to a factor of $2 + o(1)$.

THEOREM 4.3 (Theorem 1.3, strengthened). *For any* $\epsilon, \mu > 0$ *and* $N \in \mathbb{N}$, *any random selection protocol on a universe of size* N *satisfying the Statistical Criterion with parameters* ϵ *and* μ *requires at least* $\log^* N - \log^*(\max\{\log^* N, 1/\epsilon, 1/\mu\}) - O(1)$ *rounds.*

COROLLARY 4.4. *For every constant* $\delta > 0$, *there exists a constant* C *such that if a protocol* Π *achieves* $\epsilon_A, \epsilon_B \leq 1 - \delta$, *then* Π *has at least* $\log^* N - \log^* \log^* N - C$ *rounds.*

To prove this theorem, we must show that in a protocol with "few" rounds, one of the two players will be able to find a set of small size that will contain the output with high probability. We will refer to such a set (into which the cheating player is trying to make the output fall) as a *cheating set*. The proof will rely to some degree on the

probabilistic method: we will show the existence of such a cheating set by choosing it *randomly*, at least in part. The distribution on sets we will use is the following.

DEFINITION 4.5. *For a universe $\mathcal{U}$ and a parameter $\mu \in [0,1]$, we define* "a random subset of $\mathcal{U}$ of expected density μ" *to be a set $S \subseteq \mathcal{U}$ obtained by including each element $x \in \mathcal{U}$ in S independently with probability μ.*

Notice that the expected density of sets S chosen in this way is μ (and with high probability the density will not deviate significantly from μ).

We now can state the main helper theorem that will allow us to prove Theorem 4.3.

THEOREM 4.6. *There exists a function h such that for any $\mu, \epsilon > 0$, $r \in \mathbb{N}$, and protocol Π with r rounds, one of the following three cases holds:*

1. *(*A-EASY-WIN*) When R is a randomly chosen set of expected density μ, and Alice plays a strategy maximizing the probability that the output of the protocol falls in R assuming that Bob plays honestly, she will succeed with probability $1-\epsilon$, on average over all possible R. Formally,*

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}[\Pi(A^*,B)\in R]\right\}\right] \geq 1-\epsilon.$$

2. *(*B-EASY-WIN*)*

$$\mathop{\mathrm{E}}_{R}\left[\max_{B^*}\left\{\Pr_{A}[\Pi(A,B^*)\in R]\right\}\right] \geq 1-\epsilon.$$

3. *(*DIFFICULT-WIN-WIN*) When R is a randomly chosen set of expected density μ, both Alice and Bob can force the output into R plus an additional $h(r,\epsilon,\mu)$ elements with high probability. That is, the following two conditions hold:*
   (a) *$\exists\, T$, $|T| \leq h(r,\epsilon,\mu)$, such that*

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}[\Pi(A^*,B)\in R\cup T]\right\}\right] \geq 1-\epsilon.$$

   (b) *$\exists\, S$, $|S| \leq h(r,\epsilon,\mu)$, such that*

$$\mathop{\mathrm{E}}_{R}\left[\max_{B^*}\left\{\Pr_{A}[\Pi(A,B^*)\in R\cup S]\right\}\right] \geq 1-\epsilon.$$

Moreover, h does not grow too fast in r. Specifically, there is a constant C such that $h(r,\epsilon,\mu) \leq \mu N$ for all $r \leq \log^* N - \log^*(\max\{\log^* N, 1/\epsilon, 1/\mu\}) - C$.

Putting the three conditions together, this theorem says that either one player can make the output fall into a random set of a certain expected density with high probability (EASY-WIN), or *both* players can make the output fall into a set consisting of a randomly chosen set of a certain expected density and a certain bounded number of (nonrandom) elements (DIFFICULT-WIN-WIN).

Proof of Theorem 4.3. Let μ and ϵ be given and set $\mu' = \mu/4$, $\epsilon' = \epsilon/4$. By Theorem 4.6, we know that one of the players can force the output into a set $R \cup X$, where $|X| = h(r,\epsilon',\mu')$, with probability $1-\epsilon'$ in expectation over selecting R of expected density μ', for any protocol using r rounds. Suppose, without loss of generality, that the cheating player is Alice, playing with strategy A^*. Then we have

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}[\Pi(A^*,B)\in R\cup X]\right\}\right] \geq 1-\epsilon',$$

where R is a random set of expected density μ'. By a Chernoff bound, we have that

$$\Pr_{R}[\mu(R) \geq 2\mu'] \leq e^{-2(\mu')^2 N} = e^{-\mu^2 N/8} \leq \epsilon',$$

where we may assume the last inequality holds because otherwise

$$\log^* N - \log^*(\max\{\log^* N, 1/\epsilon, 1/\mu\}) - C \leq 0$$

for a constant C and the lower bound to be proven is trivial. But then it follows that

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}[\Pi(A^*, B) \in R \cup X]\right\} \cdot I(\mu(R) < 2\mu')\right] \geq 1 - 2\epsilon',$$

where $I(\mu(R) < 2\mu')$ is the indicator random variable for the event $\mu(R) < 2\mu'$. By averaging, we can find a particular set R^*, $\mu(R^*) < 2\mu'$, such that

$$\max_{A^*}\left\{\Pr_{B}[\Pi(A^*, B) \in R^* \cup X]\right\} \geq 1 - 2\epsilon' > 1 - \epsilon.$$

Assuming for contradiction that

$$\begin{aligned} r &\leq \log^* N - \log^*(\max\{\log^* N, 1/\epsilon', 1/\mu'\}) - C \\ &= \log^* N - \log^*(\max\{\log^* N, 1/\epsilon, 1/\mu\}) - O(1), \end{aligned}$$

we have $h(r, \epsilon', \mu')/N < \mu' N$, and so $|R^* \cup X| \leq 3\mu' N < \mu N$, violating the Statistical Criterion. □

4.2.1. Proof outline. In this section, we give an overview of the proof of Theorem 4.6. A detailed implementation is contained in section 4.2.2. A pictorial depiction of the proof for protocols with up to three rounds can be found in [San05].

Proving Theorem 4.6 will require an intricate analysis of the game tree using backwards induction. Like the proof of Proposition 4.1, we will show how to "label" the nodes of the game tree, where each label corresponds to a power of a player to force a particular outcome.

The labels. We will use three labels, corresponding precisely to the three cases of a protocol in Theorem 4.6. Specifically, the labels for a node on level k of the game tree will correspond to the following (where the leaves are at level 0):

- A-EASY-WIN: Alice could from that point choose a cheating set of small (say, constant) density at random and "win"—that is, make the output fall in that set with high probability.
- B-EASY-WIN: Bob could choose a cheating set at random and win.
- DIFFICULT-WIN-WIN: Neither player can win easily by choosing a totally random set, but *both* can win by choosing a set partly at random but also including a small (e.g., constant or very slowly growing) number of nonrandom elements (what we call a "helper set").

A node z labeled as DIFFICULT-WIN-WIN will have two collections of sets associated with it: $A\text{-}\mathcal{H}_z$ and $B\text{-}\mathcal{H}_z$. If the node is on level k of the game tree and it is Alice's turn to act, then $A\text{-}\mathcal{H}_z$ consists of sets of size s_{k-1} and $B\text{-}\mathcal{H}_z$ consists of sets of size s_k, where $s_1, \ldots, s_r$ will be an ascending sequence of appropriately defined constants (where r is the number of rounds of the protocol).[2] Each set $H \in A\text{-}\mathcal{H}_z$ is a set Alice can use as a "helper"—after choosing a cheating set at random and then adding the helper set, Alice can win from the given node with high probability. Similarly every set in $B\text{-}\mathcal{H}_z$ can be used by Bob as a "helper."

[2] In the actual proof, they will be very slowly growing functions of N, but for this outline one may think of them as constants.

Our main challenge is to show that every node can be given a label as above. Once we do that, Theorem 4.6 and thus Theorem 4.3 would follow readily. As a start, notice that the leaves of the tree (the base case of our induction) can certainly be labeled DIFFICULT-WIN-WIN, simply by setting $A\text{-}\mathcal{H}_z = B\text{-}\mathcal{H}_z = \{\{x\}\}$, where x is the output of the protocol at leaf z.

Piths. Before demonstrating how the internal nodes will be labeled, we define the following key concept.

DEFINITION 4.7. *Given a collection $\mathcal{H}$ of nonempty subsets of a universe $\mathcal{U}$, the* s-pith *of $\mathcal{H}$ is the collection of all sets $S \subseteq \mathcal{U}$, $|S| \leq s$, that intersect every $H \in \mathcal{H}$ (that is, $S \cap H \neq \emptyset$). We call each such S in the pith an* intersect-set *of $\mathcal{H}$.*

Three combinatorial facts about piths and collections of disjoint sets will prove useful.

FACT 4.8. *Suppose $\mathcal{H}$ consists of nonempty sets of size at most s. Then for any s', either $\mathcal{H}$ has a disjoint subcollection of size at least s'/s or it has a nonempty s'-pith.*

Proof. Take a maximal disjoint subcollection $\mathcal{P}$ of $\mathcal{H}$. If it is not of size at least s'/s, then the union of all sets in $\mathcal{P}$ will be a set of size at most s' intersecting every set in $\mathcal{H}$ (because $\mathcal{P}$ is maximal). □

FACT 4.9. *Suppose $\mathcal{H}$ consists of m disjoint sets of size at most s and $m \geq (1/\mu)^s \cdot \ln(1/\epsilon)$. Then the probability that a random set R of expected density μ will encompass a set in $\mathcal{H}$ (i.e., there exists $H_i \in \mathcal{H}$ with $H_i \subseteq R$) is at least $1 - \epsilon$.*

Proof. The probability of failure is at most $(1 - \mu^s)^m \leq e^{-\mu^s m}$. The result follows. □

Putting these two facts together, note that either a set in $\mathcal{H}$ is encompassed by a random set with probability $1 - \epsilon$, or $\mathcal{H}$ has a nonempty s'-pith, as long as $s' \geq (1/\mu)^s \cdot \ln(1/\epsilon) \cdot s$. Finally, we have the third fact.

FACT 4.10. *Suppose $\mathcal{H}$ consists of sets of size at most s, and a set S intersects every set in the s'-pith of $\mathcal{H}$. Then either S encompasses a set in $\mathcal{H}$ or $\mathcal{H}' = \{H \backslash S : H \in \mathcal{H}\}$ has a disjoint subcollection of size at least s'/s.*

Proof. Say S does not encompass a set in $\mathcal{H}$. Then every set in $\mathcal{H}'$ is nonempty. If $\mathcal{H}'$ does not have a disjoint subcollection of size s'/s, then by Fact 4.8 $\mathcal{H}'$ has a nonempty s'-pith. But if a set T is in the s'-pith of $\mathcal{H}'$, then $T \backslash S$ is in the s'-pith of $\mathcal{H}$, contradicting the definition of S. □

This strange last fact is actually an important key to the whole proof.

Labeling the nodes. We now can describe how we will inductively label a node z on level k of the game tree, assuming it is Alice's turn at that node. First, we define the constants s_k to obey $s_k \geq (1/\mu)^{s_{k-1}} \cdot \ln(1/\epsilon) \cdot s_{k-1}$. Next, we assign labels as follows:

- If all children of z are labeled B-EASY-WIN, then certainly we can give z the label B-EASY-WIN.
- If there exists a child of z that is labeled A-EASY-WIN, then we can give z the label A-EASY-WIN (Alice would just choose that child on her turn).

If neither of these cases occur, then we know that all of the children of z are either labeled DIFFICULT-WIN-WIN or B-EASY-WIN (with at least one labeled DIFFICULT-WIN-WIN). Let $\mathcal{X}$ be the union of all the collections $A\text{-}\mathcal{H}_{z_i}$, over all children z_i labeled DIFFICULT-WIN-WIN. $\mathcal{X}$ contains the helper sets that we know Alice can use to win from any such child. There are two cases:

- Suppose $\mathcal{X}$ has a large disjoint subcollection (specifically, of size at least s_k/s_{k-1}). Then label z as A-EASY-WIN.

- Else, label z as DIFFICULT-WIN-WIN, letting $A\text{-}\mathcal{H}_z$ equal $\mathcal{X}$ and letting $B\text{-}\mathcal{H}_z$ equal the s_k-pith of $\mathcal{X}$.

The correctness of the first bullet follows quickly from Fact 4.9—with high probability a random cheating set R chosen by Alice will encompass one of the sets $X \in \mathcal{X}$, and Alice can then choose the child z_i associated with X (i.e., $X \in A\text{-}\mathcal{H}_{z_i}$) and force the output into $X \cup R = R$ with high probability.

As for the second bullet, Alice certainly has a difficult win (that is, she can win with any helper set $X \in \mathcal{X}$) because she can choose the child associated with X.

The crux of the proof is showing that Bob has a difficult win from this point, using any helper set S from the pith of $\mathcal{X}$. Note first that by Fact 4.8 and the fact that we did not fall into the first bullet, we know this pith is nonempty.

It suffices to show that no matter what child Alice chooses, Bob can win using S—that is, force the output into $R \cup S$ with high probability, where R is a random set. Certainly, if she chooses a B-EASY-WIN node, Bob can win, even without S. So suppose she chooses a child node z_i labeled DIFFICULT-WIN-WIN. Inductively, we know Bob could win from this node z_i using any of the helper sets in $B\text{-}\mathcal{H}_{z_i}$ and that the pith of $B\text{-}\mathcal{H}_{z_i}$ is $A\text{-}\mathcal{H}_{z_i} \subseteq \mathcal{X}$. Since S is in the pith of $\mathcal{X}$, we know in particular that it intersects every set in $A\text{-}\mathcal{H}_{z_i}$.

Applying Fact 4.10, we then have two cases, both of which ensure Bob has some $T \in B\text{-}\mathcal{H}_{z_i}$ encompassed by his cheating set $R \cup S$, allowing him to win:

- S encompasses a set T in $B\text{-}\mathcal{H}_{z_i}$.
- $T' = \{T \backslash S : T \in B\text{-}\mathcal{H}_{z_i}\}$ has a large disjoint subcollection (i.e., of size s_{k-1}/s_{k-2}), in which case the random set will encompass one of these sets $T \backslash S$ with high probability (by Fact 4.9), and thus $T \subseteq R \cup S$.

This completes the induction and the proof sketch. The bulk of the ideas in the main proof were demonstrated above. What remains to flesh out is proper bookkeeping of parameters, namely the randomly chosen set and the sizes of the helper sets $A\text{-}\mathcal{H}_z$ and $B\text{-}\mathcal{H}_z$, to derive the precise round complexity bound. Jumping ahead, notice that the induction will stop at $\log^* N$ rounds because the sizes of these helper sets grow as tower in the number of rounds—each s_{k+1} is exponential in s_k. Thus, if there are more than $\log^* N$ rounds, the helper sets could contain all of the elements of the universe, rendering them useless for violating the Statistical Criterion.

4.2.2. Proof of Theorem 4.6. We proceed by backwards induction on the game tree of the protocol.

DEFINITION 4.11. *Given a protocol* Π *with* r *rounds and constants* ϵ *and* μ*, let* $h(r, \epsilon, \mu) = g(r, r, \epsilon, \mu)$*, where*

$$g(0, r, \epsilon, \mu) = 1,$$
$$g(k, r, \epsilon, \mu) = \ln(r/\epsilon) \cdot (r/\mu)^{g(k-1, r, \epsilon, \mu)} \cdot g(k-1, r, \epsilon, \mu).$$

For readability, we write s_k for $g(k, r, \epsilon, \mu)$, as r, ϵ, and μ will remain fixed throughout the proof. So

$$s_0 = 1,$$
$$s_k = \ln(r/\epsilon) \cdot (r/\mu)^{s_{k-1}} \cdot s_{k-1}.$$

Now, fix a protocol Π with r rounds, and consider the game tree T it induces (see Definition 2.2).

We inductively label the nodes of the tree as either A-EASY-WIN, B-EASY-WIN, or DIFFICULT-WIN-WIN. For each of the DIFFICULT-WIN-WIN nodes, we will also associate

two collections of sets (which are subsets of $\mathcal{U}$), $A\text{-}\mathcal{H}_z$ and $B\text{-}\mathcal{H}_z$, as defined below. The sets in these collections will correspond to the sets S and T of case 3 of Theorem 4.6. As we will see, the labels have been chosen to indicate the power of one or both of the players to manipulate the output effectively from that point in the tree.

DEFINITION 4.12. *Fix a protocol Π. Let z be a node on its game tree at level k (where leaves are at level 0). Assume it is Alice's turn at this node. (If it is Bob's turn, swap "A"/"A" with "B"/"B" everywhere in the description below.)*

If $k = 0$ (i.e., z is a leaf of the tree), then label z as DIFFICULT-WIN-WIN. *Moreover, let $B\text{-}\mathcal{H}_z = A\text{-}\mathcal{H}_z = \{\{x\}\}$, where x is the output of the protocol ending at node z.*

If $k > 0$, consider the children $z_1, \ldots, z_\ell$ of z. Use the following rules to label the nodes:

1. *If there exists $1 \le i \le \ell$ such that z_i is in case* A-EASY-WIN, *then label z as* A-EASY-WIN.
2. *If, for all $1 \le i \le \ell$, z_i is in case* B-EASY-WIN, *then label z as* B-EASY-WIN.
3. *Otherwise, denote $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i} = \{S : z_i$ is* DIFFICULT-WIN-WIN *and $S \in A\text{-}\mathcal{H}_{z_i}\}$. That is, $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$ is the union of the collections of sets associated with all children of z that are labeled* DIFFICULT-WIN-WIN. *Now, let $\mathcal{P}$ denote the largest disjoint subcollection of $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$ (break ties arbitrarily), and let s_k, s_{k-1} be defined as in Definition* 4.11.
   There are two cases:
   (a) $|\mathcal{P}| \ge s_k/s_{k-1} \Rightarrow$ *label z as* A-EASY-WIN.
   (b) $|\mathcal{P}| < s_k/s_{k-1} \Rightarrow$ *label z as* DIFFICULT-WIN-WIN, *and define $A\text{-}\mathcal{H}_z$ to be $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$, and $B\text{-}\mathcal{H}_z$ to be the s_k-pith of $A\text{-}\mathcal{H}_z$ (i.e., all sets of size at most s_k intersecting all sets in $A\text{-}\mathcal{H}_z$).*

Intuitively, this structure defines the power of the players at various stages of the protocol. The A-EASY-WIN, B-EASY-WIN, and DIFFICULT-WIN-WIN nodes refer to cases 1, 2, and 3 of Theorem 4.6, respectively. Moreover, the sets in the collections $B\text{-}\mathcal{H}_z$ and $A\text{-}\mathcal{H}_z$ will correspond to S and T in case 3 of Theorem 4.6.

We will codify this power in Lemma 4.14. Before stating it, it will help to define the following.

DEFINITION 4.13. *Let $\Pi = (A, B, f)$ be a protocol, and let T be its equivalent game tree (see Definition* 2.2*). For any node $z = (m_1, \ldots, m_{r-k})$ on level k of the tree T, let $\Pi_z = (A_z, B_z, f_z)$ be a protocol of k rounds, where $f_z((m'_1, \ldots, m'_k)) = f((m_1, \ldots, m_{r-k}, m'_1, \ldots, m'_k))$, and where A_z and B_z denote the strategies of A and B conditioned on history z (i.e., we choose their coin tosses r_A and r_B uniformly from those consistent with the history).*

Intuitively, Π_z is the protocol induced by starting the protocol at node z (i.e., assuming all messages leading to z are fixed in advance).

LEMMA 4.14. *Fix ϵ and μ, and suppose the protocol has r turns. Let z be some node on the tree at level k, at which it is Alice's turn to play. Throughout, let R be a random subset of $\mathcal{U}$ of expected density $k\mu/r$.*

1. *If z is in case* A-EASY-WIN, *then*

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}\left[\Pi_z(A^*, B) \in R\right]\right\}\right] \ge 1 - k\epsilon/r,$$

where Π_z is the protocol induced by beginning at node z, as defined in Definition 4.13. *(We say Alice can "win" from node z.)*

2. *If z is in case* B-EASY-WIN*, then, similarly,*

$$\mathop{\mathrm{E}}_{R}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_z(A,B^*)\in R\right]\right\}\right]\geq 1-k\epsilon/r.$$

(We say Bob can "win" from node z.)

3. *If z is in case* DIFFICULT-WIN-WIN*, then*
   (a) *B-$\mathcal{H}_z$ and A-$\mathcal{H}_z$ are nonempty;*
   (b) *for any $T\in A$-$\mathcal{H}_z$,*

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}\left[\Pi_z(A^*,B)\in R\cup T\right]\right\}\right]\geq 1-k\epsilon/r;$$

   (c) *for any $S\in B$-$\mathcal{H}_z$,*

$$\mathop{\mathrm{E}}_{R}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_z(A,B^*)\in R\cup S\right]\right\}\right]\geq 1-k\epsilon/r.$$

(We say both Alice and Bob "win" from node z, with "helper sets" T and S, respectively.)

*Moreover, the same (with "Alice"/"A"/"*A*" exchanged for "Bob"/"B"/"*B*," respectively) holds for all nodes for which it is Bob's turn.*

Lemma 4.14 more precisely asserts Theorem 4.6 at each level of the game tree. To use this lemma to prove Theorem 4.6, we simply need to apply it with $k=r$ and z being the root of the game tree. Certainly, if z_r is in case 1 or 2 of Lemma 4.14, it is in case 1 or 2 of Theorem 4.6, respectively. If z_r is in case 3 of Lemma 4.14, then subcases 3(b) and 3(c) directly prove subcases 3(a) and 3(b), respectively, where the sets in A-$\mathcal{H}_z$ and B-$\mathcal{H}_z$ of 3(a) and 3(b) correspond precisely to the sets T and S we need in those subcases of the theorem. The existence of such sets is guaranteed by subcase 3(a) of the lemma.

Thus, after proving Lemma 4.14, all that will remain will be to bound the function $h(r,\epsilon,\mu)$ to prove Theorem 4.6, which we will do in Lemma 4.20.

We prove Lemma 4.14 by induction on the levels of the tree.

Base case: $k=0$. So z is a leaf node, and the output of Π_z is just deterministically fixed at, say, x. According to Definition 4.12, A-$\mathcal{H}_z = B$-$\mathcal{H}_z = \{\{x\}\}$, and we are in case DIFFICULT-WIN-WIN. Since the density of R is chosen to be zero (it is $k\mu/r$), $R=\emptyset$, and so we need to show that

$$\max_{B^*}\left\{\Pr_{A}\left[\Pi_z(A,B^*)\in\{x\}\right]\right\}=1,$$

and similarly with A and B swapped. This, of course, holds because the output is fixed at x.

Inductive step. Suppose Lemma 4.14 holds for nodes on all levels up to level $k-1$. We will show that it holds for an arbitrary node z on level k. Assume it is Alice's turn at z. There are several possibilities.

CLAIM 4.15. *If z is in case* B-EASY-WIN*, then*

$$\mathop{\mathrm{E}}_{R}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_z(A,B^*)\in R\right]\right\}\right]\geq 1-k\epsilon/r,$$

where R is a random subset of expected density $k\mu/r$.

Proof. We will use Definition 4.12 and the inductive hypothesis to show that every child node z_i is "good" for Bob—that is, on average over R, B^* can make the

outcome land in R with probability at least $1-(k-1)\epsilon/r$. Then certainly the same holds for node z, since Alice cannot help but move to such a node.

Formally, we first notice that it suffices to show

$$\mathop{\mathrm{E}}_{R'}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_z(A,B^*)\in R'\right]\right\}\right]\geq 1-(k-1)\epsilon/r,$$

where R' is a random subset of expected density $\mu'=(k-1)\mu/r$, since a random set R of expected density $k\mu/r$ "contains" such an R'. (Formally, R can be obtained by first picking R' and then adding each element $x\notin R'$ to R with probability $(\mu/r)/(1-(k-1)\mu/r)$.)

Now, for z to be labeled B-EASY-WIN, we must have used rule 2 of Definition 4.12. Thus, all of the children of z are in case B-EASY-WIN. By the inductive hypothesis,

$$\mathop{\mathrm{E}}_{R'}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_{z_i}(A,B^*)\in R'\right]\right\}\right]\geq 1-(k-1)\epsilon/r \tag{4.1}$$

for each child z_i of z. Since at node z it is Alice's turn, we have

$$\begin{aligned}\mathop{\mathrm{E}}_{R'}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_z(A,B^*)\in R'\right]\right\}\right]&=\mathop{\mathrm{E}}_{R',\,z_i\leftarrow\mathcal{Z}}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_{z_i}(A,B^*)\in R'\right]\right\}\right]\\&\geq 1-(k-1)\epsilon/r,\end{aligned}$$

where $\mathcal{Z}$ is the distribution according to which Alice chooses child z_i of z when playing honestly, and the last inequality is by (4.1).

CLAIM 4.16. *If z is in case* A-EASY-WIN*, then*

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}\left[\Pi_z\left(A^*,B\right)\in R\right]\right\}\right]\geq 1-k\epsilon/r,$$

where R is a random subset of expected density $k\mu/r$.

Proof. By Definition 4.12, z could have been labeled A-EASY-WIN by either rule 1 or rule 3(a).

In rule 1, z has a child z_j that is in case A-EASY-WIN. Since it is Alice's turn at node z, if she can choose a node z_j "good" for her, then node z will be "good" for her too. Formally, by the inductive hypothesis applied to z_j, and again noting that a random set of expected density $k\mu/r$ "contains" a random set of expected density $(k-1)\mu/r$, we have that

$$\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_{B}\left[\Pi_{z_j}(A^*,B)\in R\right]\right\}\right]\geq 1-(k-1)\epsilon/r.$$

But $\max_{A^*}\{\Pr_B[\Pi_z(A^*,B)\in R]\}$ is at least $\max_{A^*}\{\Pr_B[\Pi_{z_j}(A^*,B)\in R]\}$, since node z is Alice's turn and she can always choose z_j. Taking expectations of both sides, the claim is proven for this case.

The alternative possibility is that z is in A-EASY-WIN because of rule 3(a). So among the sets $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$ (for all children z_i in DIFFICULT-WIN-WIN), we can find a disjoint subcollection $\mathcal{P}$, where $|\mathcal{P}|\geq s_k/s_{k-1}$.

Intuitively, since no A-EASY-WIN nodes are available among the children of z, Alice cannot simply choose such a branch as above. However, we know that from the DIFFICULT-WIN-WIN nodes, Alice can ensure the output lands in $S\cup R$ for any $S\in A\text{-}\mathcal{H}_{z_i}$, with high probability over the choice of a random set R. But this is true for many possible sets S—not only at a given child but also across all the potential children that are in case DIFFICULT-WIN-WIN (i.e., any $S\in\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$). Thus, we can

expect that with sufficiently many disjoint sets in $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$, the random set R will encompass some $S \in \bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$ with high probability. The inductive hypothesis will then give the desired result.

Formally, since $\mathcal{P} \subseteq \bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$ consists of at least $s_k/s_{k-1} = \ln(r/\epsilon)(r/\mu)^{s_{k-1}}$ (disjoint) sets of size at most s_{k-1}, Fact 4.9 tells us that

$$\mathop{\mathrm{E}}_{R_1}\left[\exists S \in \bigcup_{z_i} A\text{-}H_{z_i},\ S \subseteq R_1\right] \geq 1 - \epsilon/r, \tag{4.2}$$

where R_1 is a random subset of expected density μ/r.

For any $S \in \bigcup_{z_i} A\text{-}\mathcal{H}_{z_i}$, we can then assert

$$\mathop{\mathrm{E}}_{R_2}\left[\max_{A^*}\left\{\Pr_B[\Pi_z(A^*, B) \in R_2 \cup S]\right\}\right] \geq 1 - (k-1)\epsilon/r, \tag{4.3}$$

where R_2 is a random subset of expected density $(k-1)\mu/r$. This comes from applying the inductive hypothesis to the child z_j such that $S \in A\text{-}\mathcal{H}_{z_j}$ and noting that $\max_{A^*}\{\Pr_B[\Pi_z(A^*, B) \in R_2 \cup S]\}$ is at least $\max_{A^*}\{\Pr_B[\Pi_{z_j}(A^*, B) \in R_2 \cup S]\}$ (because at node z it is Alice's turn).

Now, since a random subset R of expected density $k\mu/r$ "contains" $R_1 \cup R_2$, where R_1 and R_2 are independent random subsets of expected densities μ/r and $(k-1)\mu/r$, respectively,[3] we can combine (4.2) and (4.3) to derive

$$\begin{aligned}\mathop{\mathrm{E}}_{R}\left[\max_{A^*}\left\{\Pr_B[\Pi_z(A^*, B) \in R]\right\}\right] &\geq (1 - \epsilon/r) \cdot (1 - (k-1)\epsilon/r) \\ &\geq 1 - k\epsilon/r.\end{aligned}$$

The claim follows. ∎

The final possibility is that z is in case DIFFICULT-WIN-WIN. Since z is not a leaf, this can come about only by rule 3(b) from Definition 4.12. That is, no children of z are in case A-EASY-WIN, and at least some are in DIFFICULT-WIN-WIN. Moreover, among $\bigcup_{z_i} A\text{-}\mathcal{H}_{z_i} = A\text{-}\mathcal{H}_z$ the largest (maximal) disjoint subcollection $\mathcal{P}$ has fewer than s_k/s_{k-1} elements.

We must prove the following: $B\text{-}\mathcal{H}_z$ is nonempty, $A\text{-}\mathcal{H}_z$ is nonempty, Alice can win from this node with a helper set from $A\text{-}\mathcal{H}_z$, and Bob can win from this node with a helper set from $B\text{-}\mathcal{H}_z$ (see Lemma 4.14).

CLAIM 4.17. *$B\text{-}\mathcal{H}_z \neq \emptyset$ and $A\text{-}\mathcal{H}_z \neq \emptyset$.*

Proof. We have already established that z has children in case DIFFICULT-WIN-WIN (this follows from Definition 4.12 and from our assumption that $z \in$ DIFFICULT-WIN-WIN). By the inductive hypothesis on such a child z_i, $A\text{-}\mathcal{H}_{z_i}$, and thus $A\text{-}\mathcal{H}_z$ is nonempty. Since the largest disjoint subcollection $\mathcal{P}$ of $A\text{-}\mathcal{H}_z$ has size less than s_k/s_{k-1} and since all $S \in \mathcal{P}$ have size at most s_{k-1}, Fact 4.8 tells us that the s_k-pith of $A\text{-}\mathcal{H}_z$—namely, $B\text{-}\mathcal{H}_z$—is nonempty. ∎

CLAIM 4.18. *For any $S \in A\text{-}\mathcal{H}_z$, $\mathrm{E}_R[\max_{A^*}\{\Pr_B[\Pi_z(A^*, B) \in S \cup R]\}] \geq 1 - k\epsilon/r$, where R is a random subset of expected density $k\mu/r$.*

The proof of this claim is identical to the proof of (4.3) in the proof of Claim 4.16, noting also that a random set R of expected density $k\mu/r$ contains a random set R_2 of expected density $(k-1)\mu/r$.

[3] Formally, R can be obtained by first picking R_1 and R_2 and adding each element $x \notin R_1 \cup R_2$ to $R_1 \cup R_2$ with probability $(\mu/r) \cdot ((k-1)\mu/r)/[(1-\mu/r) \cdot (1-(k-1)\mu)/r]$.

The final claim required to prove Lemma 4.14 is the following.

CLAIM 4.19. *For any $S \in B\text{-}\mathcal{H}_z$, $\mathrm{E}_R[\max_{B^*}\{\Pr_A[\Pi_z(A, B^*) \in S \cup R]\}] \geq 1 - k\epsilon/r$, where R is a random subset of expected density $k\mu/r$.*

This claim is the heart of the entire proof. All we know now is that there is at least one DIFFICULT-WIN-WIN node that is a child of the current node z and that among the corresponding sets in $A\text{-}\mathcal{H}_z$, the largest disjoint subcollection $\mathcal{P} \subseteq A\text{-}\mathcal{H}_z$ contains fewer than s_k/s_{k-1} sets. That $\mathcal{P}$ is so small is a limitation on the power of Alice, who would like there to be enough such disjoint sets in $\mathcal{P}$ that she could choose randomly and encompass a set in $\mathcal{P}$ with high probability. The key to this proof is converting this *limitation* on Alice into an *ability* for Bob to cheat.

Proof. Fix a set $S \in B\text{-}\mathcal{H}_z$, which recall is the s_k-pith of $A\text{-}\mathcal{H}_z$. Since an honest Alice will choose a child z_i at random, it suffices to prove the following for each child z_i:

$$\mathop{\mathrm{E}}_{R}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_{z_i}(A, B^*) \in S \cup R\right]\right\}\right] \geq 1 - k\epsilon/r, \tag{4.4}$$

where R is a random subset of expected density $k\mu/r$. So fix an arbitrary child z_i. Looking to Definition 4.12, the only way we could have defined z to be in case DIFFICULT-WIN-WIN is if all children z_i are in either case B-EASY-WIN or case DIFFICULT-WIN-WIN. So z_i is in one of these two cases.

If z_i is in case B-EASY-WIN, then we are done by the inductive hypothesis. So suppose z_i is in case DIFFICULT-WIN-WIN. Applying the inductive hypothesis to z_i, we know that $B\text{-}\mathcal{H}_{z_i}$ is nonempty. Moreover, for any $T \in B\text{-}\mathcal{H}_{z_i}$,

$$\mathop{\mathrm{E}}_{R_1}\left[\max_{B^*}\left\{\Pr_{A}\left[\Pi_{z_i}(A, B^*) \in T \cup R_1\right]\right\}\right] \geq 1 - (k-1)\epsilon/r, \tag{4.5}$$

where R_1 is a random subset of expected density $(k-1)\mu/r$.

We have that S is in the s_k-pith of $A\text{-}\mathcal{H}_z$, which means in particular that it intersects every set in $A\text{-}\mathcal{H}_{z_i}$, which in turn is the s_{k-1}-pith of $B\text{-}\mathcal{H}_{z_i}$ (whose sets are of size s_{k-2}, when $k > 1$). By Fact 4.10, either there exists a set T in $B\text{-}\mathcal{H}_{z_i}$ such that $T \subseteq S$ (in which case (4.4) follows immediately from (4.5)), or else $\mathcal{T} = \{T \backslash S : T \in B\text{-}\mathcal{H}_{z_i}\}$ has a disjoint subcollection of size s_{k-1}/s_{k-2}. (When $k = 1$, $B\text{-}\mathcal{H}_{z_i}$ contains only the set $T = \{x\}$, where x is the output of the protocol at leaf z_i, and we also have $x \in S$ because S intersects every set in $A\text{-}\mathcal{H}_{z_i} = \{\{x\}\}$. So we have $T \subseteq S$, and (4.4) follows immediately from (4.5).)

Informally, there are not many disjoint sets in $B\text{-}\mathcal{H}_{z_i}$—if there were, we would have labeled z_i as a case B-EASY-WIN node for Bob. That said, by intersecting every (small) set that intersected every set in $B\text{-}\mathcal{H}_{z_i}$, S captures the lack of disjointness of $B\text{-}\mathcal{H}_{z_i}$ in the first place. Once the elements of S are removed from consideration, the result has a large number of disjoint sets.

Returning to the proof of Claim 4.19, by Fact 4.9 we may conclude the following:

$$\Pr_{R_2}[\exists T' \in \mathcal{T},\, T' \subseteq R_2] \geq 1 - \epsilon/r,$$

where R_2 is a random subset of expected density μ/r. By the definition of $\mathcal{T}$, this in turn implies

$$\Pr_{R_2}[\exists T \in B\text{-}H_{z_i},\, T \subseteq S \cup R_2] \geq 1 - \epsilon/r.$$

Using (4.5) and choosing R through independent choices of R_1 and R_2 as in the proof of Claim 4.16, we are done. ∎

Taking together Claims 4.15, 4.16, 4.17, 4.18, and 4.19, the proof of Lemma 4.14 is complete.

To conclude Theorem 4.6, it remains to prove that the function h defining the set sizes s_k does not grow too fast in the number of rounds. Intuitively, the reason the lower bound holds only for protocols with fewer than $\log^* N - \log^* \log^* N - O(1)$ rounds is that these "helper sets" must have much fewer than N elements to be useful, but this function h grows as a tower—where the height (and base) of the tower grow with the number of rounds. Our challenge is to lower bound the number of rounds that keep this tower of size $o(N)$.

LEMMA 4.20. *Recall the definition $h(r, \epsilon, \mu) = g(r, r, \epsilon, \mu)$, where we define*

$$g(0, r, \epsilon, \mu) = 1,$$
$$g(k, r, \epsilon, \mu) = \ln(r/\epsilon) \cdot (r/\mu)^{g(k-1,r,\epsilon,\mu)} \cdot g(k-1, r, \epsilon, \mu).$$

There exists a constant C such that when $r < \log^* N - \log^*(\max\{\log^* N, 1/\epsilon, 1/\mu\}) - C$, we have $h(r, \epsilon, \mu) \le \mu N$.

Proof. First, bound r by $\log^* N$. Again, for shorthand we will write s_k for $g(k, r, \epsilon, \mu)$. Thus, we have that

$$s_k = \ln(r/\epsilon) \cdot (r/\mu)^{s_{k-1}} \cdot s_{k-1}.$$

Notice that this is no more than $(r \ln(r/\epsilon)/\mu)^{s_{k-1}}$. ($xy \le x^y$ if $x \ge 2$ and $y \ge 1$.) Letting $d = (r \ln(r/\epsilon)/\mu)$, we can then bound s_k by t_k, where t_k is defined by $t_0 = 1$ and $t_k = d^{t_{k-1}}$.

This means that we can set $k = \log_d^* N - 1$ (recall that by our definition, $\log_b^* N$ is always an integer, for any b or N) and still have $s_k \le t_k \le \log N \le \mu N$, where we may assume that the last inequality holds, because otherwise $\log^* N - \log^*(1/\mu) - C < 0$ and the lemma is vacuously true. It remains only to relate this to a base 2 logarithm.

CLAIM 4.21. *If $d \ge 4$, then $\log_d^* N \ge \log^* N - \log^*(2 \log d)$.*

Proof. Recall that $\log^{(k)} N$ is k iterated logarithms of N. We claim the following.

CLAIM 4.22. *For $k \le \log_d^* N$, $d \ge 4$, $\log^{(k)} N \le (2 \log d) \log_d^{(k)} N$.*

Proof. The base case $k = 0$ is clear. Assume, then, that

$$\log^{(k-1)} N \le (2 \log d) \log_d^{(k-1)} N.$$

Applying log to both sides, we have that

$$\begin{aligned} \log^{(k)} N &\le \log(2 \log d) + \log(\log_d^{(k-1)} N) \\ &\le \log(2 \log d) + (\log_d^{(k)} N)(\log d) \\ &\le (2 \log d)(\log_d^{(k)} N), \end{aligned}$$

where the last line follows because for $d \ge 4$, $d \ge 2 \log d$ and for $k \le \log_d^* N$, $\log_d^{(k)} N \ge 1$. ∎

Plugging in $k = \log_d^* N$, then we have that $\log^{(\log_d^* N)} N \le 2 \log d$. Applying $\log^*(2 \log d)$ logarithms to both sides, we have $\log^{(\log_d^* N + \log^*(2 \log d))} N \le 1$. Since $\log^* N$ is defined to be the least k such that $\log^{(k)} N \le 1$, it follows that $\log^* N \le \log_d^* N + \log^*(2 \log d)$. ∎

Thus we have that we can set k to be at least $\log_d^* N - 1$ and s_k will be no more than $\log N$. Moreover,

$$\begin{aligned}\log_d^* N - 1 &\geq \log^* N - \log^*(2\log d) - 1 \\ &= \log^* N - \log^*(2\log((r\ln(r/\epsilon))/\mu)) - 1 \\ &\geq \log^* N - \log^*(\max\{\log^* N, 1/\epsilon, 1/\mu\}) - O(1). \qquad \square\end{aligned}$$

By applying Lemma 4.14 to the root of the tree and using Lemma 4.20, we prove Theorem 4.6 and thus Theorem 4.3.

5. Multiplicative lower bounds. In this section, we concentrate on lower bounds regarding multiplicative guarantees—and indeed show that no protocol exists that provides constant multiplicative guarantees to both players. This is a very strong limitation on the ability of protocols to limit a cheating player's power in this regard.

An initial lower bound. Proposition 4.1 can be adapted to provide a quick lower bound for multiplicative guarantees.

PROPOSITION 5.1. *In any random selection protocol,* $(\rho_A - 1)/\rho_A + (\rho_B - 1)/\rho_B \geq 1 - 1/N$. *Moreover,* $\epsilon_A + (\rho_B - 1)/\rho_B \geq 1 - 1/N$ *(or equivalently,* $\epsilon_A \geq 1/\rho_B - 1/N$*).*

Proof. The results follow immediately from Proposition 4.1 and from the second part of Lemma 2.11. $\square$

This lower bound for multiplicative guarantees is not very strong—ρ is a number from 1 to N, but this lower bound is satisfied (for instance) as long as both ρ_A and ρ_B are at least 2. In Theorem 5.3, we will prove that $\rho_A \rho_B \geq N$, which is a substantially stronger result.

On the other hand, when looking at one player getting a statistical guarantee and the other player getting a multiplicative guarantee, Proposition 5.1 does provide some useful information. Specifically, it tells us that (minus a small $1/N$ term) we can always expect the statistical guarantee for one player to be worse than the reciprocal of the multiplicative guarantee to the other player. This explains inverse relationships in existing protocols of [DGW94] (where $\varepsilon = 1/\text{poly}(n)$ and $\rho = \text{poly}(n)$) and [GSV98] (where $\varepsilon = \text{poly}(n) \cdot 2^{-k}$ and $\rho = 2^k$ for any k).[4] Notice that these earlier works focus on the case of nonconstant guarantees ($\varepsilon \to 0$ and $\rho \to \infty$). Earlier, we showed that the Iterated Random Shift Protocol achieves simultaneous *constant* statistical and multiplicative guarantees. From Theorem 4.3 and Lemma 2.11, it follows that our protocol has optimal round complexity up to a factor of $2 + o(1)$ among those achieving simultaneous constant statistical and multiplicative guarantees.

COROLLARY 5.2. *For every two constants* $\epsilon_A < 1$ *and* ρ_B*, there exists a constant* C *such that any protocol* Π *selecting from a universe of size* N *and achieving statistical guarantee* ϵ_A *and multiplicative guarantee* ρ_B *will have at least* $\log^* N - \log^* \log^* N - C$ *rounds.*

A tight lower bound. Despite the inability of the above to give a strong lower bound for simultaneous multiplicative guarantees, in this section we present a tight lower bound in this setting, which follows from the work of Goldreich, Goldwasser, and Linial [GGL98].

THEOREM 5.3 (Theorem 2.10, restated; see [GGL98]). *For any protocol* Π*,* $\rho_A \cdot \rho_B \geq N$.

[4] Actually, the protocol of [GSV98] does not provide a multiplicative guarantee of 2^k but rather ensures that the probability that the output lands in any set T of density μ is at most $2^k \cdot \mu + o(1)$. Our lower bound also applies to this more general type of guarantee.

COROLLARY 5.4. *In any protocol* Π, $\max\{\rho_A, \rho_B\} \geq \sqrt{N}$.

Recalling that the multiplicative guarantee ρ_A is the greatest factor by which Bob can improve the probability that a single element is chosen over uniform, we conclude the following.

In any random selection protocol, at least one of the players can improve the probability that a single element is chosen by a factor *exponential* in the length of the output (which equals $\log N$).

Goldreich, Goldwasser, and Linial [GGL98] showed a more general result than Theorem 5.3 (for multiparty protocols) using different language and a moderately involved proof. We include below the simplification of their proof for the two-party case.

Proof of Theorem 5.3. Fix some element v of the universe. Now, consider the game tree T of the protocol (see Definition 2.2). At each node z of the tree, denote the protocol induced by beginning at node z to be Π_z. Then define

$$\begin{aligned}
\phi_A^z &= \max_{A^*} \Pr_B[\Pi_z((A^*, B)) = v], \\
\phi_B^z &= \max_{B^*} \Pr_A[\Pi_z((A, B^*)) = v], \\
p_z &= \Pr_{A,B}[\Pi_z((A, B)) = v].
\end{aligned}$$

That is, ϕ_A^z (resp., ϕ_B^z) is the highest probability Alice (resp., Bob) can make the output to be v, given that the protocol is now at node z and that Bob (resp., Alice) is playing honestly. p_z is the probability that v is chosen starting from z and assuming both players play honestly.

To prove the theorem, we will show that for every node z on T, $\phi_A^z \cdot \phi_B^z \geq p_z$. Applying this fact to the root r of the tree r and noting that we can choose v so that $p_r \geq 1/N$, the theorem follows easily.

We will prove $\phi_A^z \cdot \phi_B^z \geq p_z$ by backwards induction on the levels of the tree.

When z is a leaf, the protocol is complete. If v is the output of the protocol at leaf z, then $\phi_A^z = \phi_B^z = p_z = 1$. Otherwise, $\phi_A^z = \phi_B^z = p_z = 0$.

Now, suppose that the lemma holds for all children of z—denote them $z_1, \ldots, z_m$. Thus, we know $\phi_A^{z_i} \phi_B^{z_i} \geq p_{z_i}$ for all children z_i. Suppose also, without loss of generality, that at node z it is Alice's turn.

Suppose an honest Alice chooses child node z_i with probability λ_i. Then $p_z = \sum \lambda_i p_{z_i}$, and $\phi_B^z = \sum \lambda_i \phi_B^{z_i}$. When considering ϕ_A^z, however, Alice will cheat and choose the best child available. Thus, $\phi_A^z = \max_{z_i} \phi_A^{z_i}$, and so in particular for all i, $\phi_A^z \geq \phi_A^{z_i}$.

Now, just compute

$$\phi_A^z \phi_B^z = \phi_A^z \sum \lambda_i \phi_B^{z_i} \geq \sum \lambda_i \phi_A^{z_i} \phi_B^{z_i} \geq \sum \lambda_i p_{z_i} = p_z. \qquad \square$$

To understand this result intuitively, suppose that there were only one path down the tree that led to v being chosen as the output. At each node along that path, starting from the root, there is a certain probability that an honest player will choose the (unique) next node in the path. So the probability that v is chosen is the product of these probabilities when both players play honestly. If Alice (resp., Bob) is cheating, then the probability that v will be chosen is the product of the probabilities at nodes where it is Bob's (resp., Alice's) turn. In this case, $\phi_A^z \cdot \phi_B^z = p_z$. More paths yielding v merely provide more options to the cheating player, and so $\phi_A^z \cdot \phi_B^z \geq p_z$.

Note that, unlike Proposition 4.1, this result relies centrally on the assumption that, when one player is cheating, the other player is playing *honestly*.

Theorem 5.3 is, in fact, tight. In Proposition 3.10, we noted that the Random Set Protocol—in which Alice chooses a uniformly random subset of fixed size K and Bob chooses a random element of this set—achieves multiplicative guarantees of K and N/K for the two players.

Note that one negative aspect of the Random Set Protocol is that it is not efficient—sending a description of the random subset requires communication linear in N (rather than polylog(N)). This is certainly not necessary to achieve $\rho_A \rho_B = N$, however: other very simple and efficient protocols achieve this tradeoff. Specifically, instead of using all sets of size K, we can use any subcollection such that every element of $[N]$ is contained in the same number of sets. For example, if $N = K \cdot L$ for an integer L, then we can view the universe as $[K] \times [L]$ and use only the sets of the form $S_a = [K] \times \{a\}$ for each $a \in [L]$, and so the communication becomes $\log L + \log K = \log N$. The optimality of such a trivial protocol suggests that, ultimately, multiplicative guarantees are not by themselves likely to be sufficient metrics of study for two-party random selection protocols.

Acknowledgments. We thank Oded Goldreich, Grant Schoenebeck, and Alexander Healy for several helpful discussions. We are also grateful to anonymous reviewers for suggestions that greatly improved the presentation.

REFERENCES

[AN93] N. Alon and M. Naor, *Coin-flipping games immune against linear-sized coalitions*, SIAM J. Comput., 22 (1993), pp. 403–417.

[Amb04] A. Ambainis, *A new protocol and lower bounds for quantum coin flipping*, J. Comput. System Sci., 68 (2004), pp. 398–416.

[BL89] M. Ben-Or and N. Linial, *Collective coin-flipping*, in Randomness and Computation, S. Micali, ed., Academic Press, New York, 1989.

[Blu82] M. Blum, *Coin flipping by telephone*, in Proceedings of the 24th IEEE COMPCOM, 1982, pp. 133–137.

[BN00] R. B. Boppana and B. O. Narayanan, *Perfect-information leader election with optimal resilience*, SIAM J. Comput., 29 (2000), pp. 1304–1320.

[CCM98] C. Cachin, C. Crépeau, and J. Marcil, *Oblivious transfer with a memory-bounded receiver*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 493–502.

[Cle86] R. Cleve, *Limits on the security of coin flips when half the processors are faulty*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 364–369.

[CI93] R. Cleve and R. Impagliazzo, *Martingales, Collective Coin Flipping, and Discrete Control Processes*, manuscript, 1993.

[Dam94] I. Damgård, *Interactive hashing can simplify zero-knowledge protocol design without computational assumptions*, in Advances in Cryptology—CRYPTO '93, Lecture Notes in Comput. Sci. 403, Springer, Berlin, 1994, pp. 100–109.

[DGW94] I. Damgård, O. Goldreich, and A. Wigderson, *Hashing Functions Can Simplify Zero-Knowledge Protocol Design (Too)*, Technical report RS-94-39, BRICS, University of Aarhus, Aarhus, Denmark, 1994.

[DHRS04] Y. Ding, D. Harnik, A. Rosen, and R. Shaltiel, *Constant-round oblivious transfer in the bounded storage model*, in Proceedings of the 1st Theory of Cryptography Conference, Lecture Notes in Comput. Sci. 2951, Springer, Berlin, 2004, pp. 446–472.

[Fei99] U. Feige, *Noncryptographic selection protocols*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 142–153.

[GGL98] O. Goldreich, S. Goldwasser, and N. Linial, *Fault-tolerant computation in the full information model*, SIAM J. Comput., 27 (1998), pp. 506–544.

[GSV98] O. Goldreich, A. Sahai, and S. Vadhan, *Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 399–408.

[KO04] J. Katz and R. Ostrovsky, *Round-optimal secure two-party computation*, in Advances in Cryptology—CRYPTO '04, Lecture Notes in Comput. Sci. 3152, Springer, Berlin, 2004, pp. 335–354.

[Lau83] C. Lautemann, **BPP** *and the polynomial hierarchy*, Inform. Process. Lett., 17 (1983), pp. 215–217.

[Lin01] Y. Lindell, *Parallel coin-tossing and constant-round secure two-party computation*, J. Cryptology, 16 (2003), pp. 143–184.

[NOVY98] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, *Perfect zero-knowledge arguments for* **NP** *using any one-way permutation*, J. Cryptology, 11 (1998), pp. 87–108.

[ORV94] R. Ostrovsky, S. Rajagopalan, and U. Vazirani, *Simple and efficient leader election in the full information model*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 234–242.

[RSZ02] A. Russell, M. Saks, and D. Zuckerman, *Lower bounds for leader election and collective coin-flipping in the perfect information model*, SIAM J. Comput., 31 (2002), pp. 1645–1662.

[RZ01] A. Russell and D. Zuckerman, *Perfect information leader election in* $\log^* n + O(1)$ *rounds*, J. Comput. System Sci., 63 (2001), pp. 612–626.

[Sak89] M. Saks, *A robust noncryptographic protocol for collective coin flipping*, SIAM J. Discrete Math., 2 (1989), pp. 240–244.

[San04] S. Sanghvi, *A Study of Two-Party Random Selection Protocols*, undergraduate thesis, Harvard University, Cambridge, MA, 2004.

[San05] S. Sanghvi, *The round complexity of two-party random selection*, slides of presentation given at STOC 2005. Available online from http://eecs.harvard.edu/~salil/papers/randsel-abs.html.

[SV05] S. Sanghvi and S. Vadhan, *The round complexity of two-party random selection*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 338–347.

[Yao86] A. Yao, *How to generate and exchange secrets*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 162–167.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 551–607

SHORT PCPS WITH POLYLOG QUERY COMPLEXITY*

ELI BEN-SASSON† AND MADHU SUDAN‡

Abstract. We give constructions of probabilistically checkable proofs (PCPs) of length $n \cdot \text{polylog}\, n$ proving satisfiability of circuits of size n that can be verified by querying $\text{polylog}\, n$ bits of the proof. We also give analogous constructions of locally testable codes (LTCs) mapping n information bits to $n \cdot \text{polylog}\, n$ bit long codewords that are testable with $\text{polylog}\, n$ queries. Our constructions rely on new techniques revolving around properties of codes based on relatively *high*-degree polynomials in *one* variable, i.e., Reed–Solomon codes. In contrast, previous constructions of short PCPs, beginning with [L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 21–31] and until the recent [E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Robust PCPs of proximity, shorter PCPs, and applications to coding*, in Proceedings of the 36th ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 13–15], relied extensively on properties of *low*-degree polynomials in *many* variables. We show how to convert the problem of verifying the satisfaction of a circuit by a given assignment to the task of verifying that a given function is close to being a Reed–Solomon codeword, i.e., a univariate polynomial of specified degree. This reduction also gives an alternative to using the "sumcheck protocol" [C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *J. ACM*, 39 (1992), pp. 859–868]. We then give a new PCP for the special task of proving that a function is close to being a Reed–Solomon codeword. The resulting PCPs are not only shorter than previous ones but also arguably simpler. In fact, our constructions are also more natural in that they yield locally testable codes first, which are then converted to PCPs. In contrast, most recent constructions go in the opposite direction of getting locally testable codes from PCPs.

Key words. probabilistically checkable proofs (PCPs), PCPs of proximity, locally testable codes, Reed–Solomon codes

AMS subject classification. 68Q17

DOI. 10.1137/050646445

1. Introduction.

Probabilistically checkable proof (PCP) systems as formulated in [20, 3, 2] are proof systems that allow efficient probabilistic verification based on querying a few bits of a proof. Formally, a PCP system is given by a PCP-verifier that probabilistically queries a few bits of a purported proof of a claimed theorem and accepts valid proofs of true theorems with probability 1, while accepting any claimed proof of false assertions with low probability, say, at most 1/2. The celebrated PCP theorem [3, 2] asserts that for any language in NP there exists a PCP-verifier that reads just a constant number of bits from a proof of polynomial length. Subsequently, it was shown in [28, 26] that the number of queries can be made as small as *three* bits, while rejecting proofs of false assertions with probability arbitrarily close to (but

*Received by the editors November 30, 2005; accepted for publication (in revised form) December 5, 2006; published electronically May 23, 2008.

http://www.siam.org/journals/sicomp/38-2/64644.html

†Computer Science Department, Technion—Israel Institute of Technology, Haifa, 32000, Israel (eli@cs.technion.ac.il). This author is a Landau Fellow who was supported by the Taub and Shalom Foundations. This author's work was also supported by an Alon Fellowship of the Israeli Council for Higher Education, an International Reintegration Grant from the European Community, and grants from the Israeli Science Foundation and the US-Israel Binational Science Foundation. This work was done while the author was at the Radcliffe Institute for Advanced Study, Cambridge, MA.

‡Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (madhu@mit.edu). The work of this author was supported in part by NSF Award CCR-0312575. This work was done while the author was at the Radcliffe Institute for Advanced Study, Cambridge, MA.

larger than) 1/2. Such query-efficient proofs translate to strong inapproximability results for many combinatorial optimization problems; see [7, 8, 26, 28, 38].

Somewhat surprisingly, PCPs are rarely appreciated for their positive properties, i.e., as methods of transforming proofs into extremely efficiently verifiable formats. Instead their negative implications for combinatorial optimization dominate their study. In principle, PCPs could form the semantic analogue of error-correcting codes: Error-correcting codes are used to preserve data for long periods of time; PCPs may be used to preserve data, with a promise of integrity with respect to any fixed Boolean property, for long periods of time. However such uses seemed infeasible using current PCP constructions, which are *too long* and *too complex*. This forms the motivation of our work, which tries to find *shorter and simpler PCPs*.

A number of works [5, 37, 27, 24, 11, 9] have been focused on optimizing the length of the PCP. In addition to the inherent motivation mentioned above, the length of PCPs also plays an important role in their use in cryptography (e.g., in CS proofs [30, 35] and their applications [6, 13]) and is closely related to the construction of locally testable codes [24, 11, 9]. Simplifying PCP constructions has long been a goal within the study of PCPs, though little progress had been achieved in this direction until Dinur's recent surprising proof of the PCP theorem by gap amplification [18] continuing the combinatorial approach taken in [19]. Although we also construct simpler PCPs, our approach by contrast relies on adding algebraic structure instead of combinatorics.

PCPs. Our main result, Theorem 2.2, is a PCP construction that blows up the proof length by only a polylogarithmic factor resulting in a PCP of *quasilinear* length. (Throughout this paper, a function $f : \mathbb{N}^+ \to \mathbb{N}^+$ is said to be *quasilinear* if $f(n) = n \cdot \text{polylog}\, n$.) These short proofs can be verified by querying a polylogarithmic number of bits of the proof. By way of comparison, the recent results of Ben-Sasson et al. [9] give proofs of length $n \cdot \exp(\text{poly}\log\log n)$ with a query complexity of $\text{poly}\log\log n$. Thus, while the query complexity of our PCPs is higher than that of most recent results, the proof size is smaller.

PCPs of proximity. The results of [9] are actually for a stronger notion of PCPs, called *PCPs of proximity* (PCPPs). This notion was simultaneously introduced (under the name *assignment testers*) in [19] and a similar notion also appeared earlier in [40]. Informally, a PCPP-verifier's input includes two oracles, a "claimed theorem" and the "proof," and the verifier confirms that the claimed theorem is *close* in, say, Hamming distance, to a true theorem. It does so by making few oracle queries into the theorem and the proof. In contrast, recall that a PCP-verifier had unlimited access to the "claimed theorem" but verified that it was true *exactly* as stated. Theorem 2.10 gives a construction of PCPPs for all languages in NP with shorter proofs of proximity, though with larger query complexity than that of [9].

Locally testable codes. PCPs typically go hand-in-hand with *locally testable codes* (LTCs); for a detailed discussion of LTCs, see [24, 23] and references therein. Briefly, LTCs are error-correcting codes with relatively large rate and distance. Additionally, the amount of noise in a received word can be bounded from above by querying only a sublinear number of positions of the received word. Specifically, these codes have an associated tester that reads very few symbols of a received word and accepts codewords with probability 1, while rejecting words that are far from all codewords with constant probability (say, 1/2). Theorem 2.13 constructs LTCs with parameters similar to those of our PCPs. Namely, the codes have linear distance while the codeword to message ratio and the query complexity of the tester are polylogarithmic.

We highlight the fact that our work first constructs LTCs with polylogarithmic rate and query complexity, after which PCPs with the same parameters are derived as a consequence. While the early work of Babai et al. [5] also had this feature, constructions of smaller LTCs (in particular, those in [24, 11, 9]) reverse this direction, getting PCPs first and then deriving LTCs as a consequence. Our work thus achieves one of the goals associated with LTCs, namely, offering benefits and insights into PCPs via direct construction of LTCs.

Our techniques. Although our construction is algebraic as in prior PCP constructions, our techniques are significantly different and thus interesting in their own right. All previous algebraic PCPs (i.e., those excluding the combinatorial construction of [19]) start with a PCP based on the properties of multivariate polynomials over some finite field. Some key ingredients in such constructions are the following: (1) a *low-degree test*, i.e., a method to test if a function given by an oracle is close to being a low-degree multivariate polynomial, (2) a *self-corrector*, i.e., a procedure to compute the value of a multivariate polynomial at a given point, given oracle access to a polynomial that is close to this polynomial, (3) a *zero-tester*, i.e., an efficient procedure to verify if a function given by an oracle is close to a multivariate polynomial that is zero on every point in a prespecified subset of its domain, and (4) a reduction from verifying satisfiability to zero-testing. Typical solutions to the above problems yield a query complexity that is polynomial in the number of variables and the degree of the multivariate polynomial. This query complexity can then be reduced using a set of techniques referred to as *proof composition*.

Our solution follows a similar outline (though we do not need a self-corrector) except that, for the most part, we work only with univariate polynomials. This forms the essence of our technical advantage, giving PCPs with smaller proof length. The length of PCPs is well known to grow with the number of variables in the polynomials used to construct them, and reducing this number was an obvious way to try to reduce PCP length. However, reducing the number of variables increases the degree of the associated polynomials, and since solutions to steps (1)–(3) above had query complexity polynomial in the degree, previous solutions needed to use a large number of variables to significantly reduce the number of queries. In our case, we propose analogous questions for *univariate* polynomials and give query-efficient solutions for them, leading to short PCPs. We describe our solutions to the steps (1)–(4) in reverse order.

We start with the reduction from satisfiability to testing zero polynomials, which is step (4) above. The usual reduction is a transformation from a Boolean formula ϕ to a constraint C on pairs of polynomials along with subsets S_1 and S_2 of the multivariate domains with the following property: ϕ is satisfiable iff there exist polynomials P_1, P_2 that are zero on S_1, S_2, respectively, and furthermore $C(P_1, P_2)$ holds. (To enable "easy verification," $C(P_1, P_2)$ needs to be of a special form, but we will not get into this now.) In general, these reductions are simple, and our version of the reduction is as well. However in our case, the reductions appear particularly natural since we deal with a very small number of variables. In section 5 we describe our natural way of reducing NP-complete problems to problems about testing zeros of polynomials. In the end we use a somewhat more complex solution due only to our goal of extreme length efficiency; even in this case the full proof is only a few pages long.

Next we move to the zero-testing problem, which is step (3) above. We reduce this to two univariate low-degree testing questions, along with a natural consistency test between the two polynomials. The query complexity is a *constant* independent of

the degrees of the polynomials we are working with. Furthermore, it directly reduces zero-testing to low-degree testing while most previous solutions relied on some form or other of the self-correcting question. Put together, our solutions for steps (3) and (4) give a short and simple reduction from verifying NP statements to testing the degree of a univariate function. Furthermore, these reductions add only a *constant* number of queries to the query complexity of the low-degree testing protocol. This highlights the importance of the low-degree testing problem for univariate polynomials, which we describe below.

Reed–Solomon codes and proofs of proximity. The problem at the heart of our PCPs is the following: Given a finite field $\mathbb{F}$, a degree bound d, and oracle access to a function $f : \mathbb{F} \to \mathbb{F}$, test if f is close to a polynomial of degree at most d. Specifically, if f is a degree-d polynomial, then the test must always accept. On the other hand, if f is δ-far from every degree-d polynomial, i.e., the value of f needs to be changed on at least δ-fraction of the points in $\mathbb{F}$ to get a degree-d polynomial, then the test must reject with high probability. The objective is to do this while querying the oracle for f as few times as possible. The functions derived by evaluating polynomials of a specified degree over a field are known as *Reed–Solomon codes*, which we sometimes refer to by the name *RS-codes*. Our goal is thus to provide an efficient test for membership in these codes.

It is easy to see that, as such, the problem above allows no very efficient solutions: A tester that accepts all degree-d polynomials with probability 1 must probe the value of f in at least $d+2$ places before it can reject any function. This is too many queries for our purpose. This is where the notion of PCPPs comes to the rescue. Whereas it is hard to test if function f described by the oracle represents a degree-d polynomial with fewer than $\Omega(d)$ queries, it is conceivable (and indeed implied by previous works, for example, by [9]) that one can use an auxiliary proof oracle π to "prove" that f is close to the evaluations of a degree-d polynomial. More formally, our new task is thus to design a PCPP-verifier that makes a few queries to a pair of oracles (f, π), where we allow π to return elements of $\mathbb{F}$ as answers, and the following holds: If f is a degree-d polynomial, there exists a valid proof π so that (f, π) is always accepted by the tester. If f is δ-far from every degree-d polynomial, then for every π, the pair (f, π) must be rejected with high probability.

Since the property of being a degree-d polynomial over $\mathbb{F}$ can be efficiently verified (in time $|\mathbb{F}| \cdot \operatorname{polylog} |\mathbb{F}|$), we can apply the final theorem of Ben-Sasson et al. [9], which gives length-efficient proofs for any property relative to the time it takes to verify the property deterministically, to get moderately efficient solutions to this problem. Unfortunately, such a solution would involve proof oracles of length $|\mathbb{F}| \cdot \exp(\operatorname{poly} \log \log |\mathbb{F}|)$, which is longer than we can allow. Their solution would also not satisfy our (subjective) simplicity requirement. However it does confirm that our goal of making $o(d)$ queries is attainable.

Our main technical result is a PCPP for Reed–Solomon codes. This proof of proximity has length $O(n \cdot \operatorname{polylog} n)$ and query complexity $\operatorname{polylog} n$ for RS-codes over a field $\mathbb{F}$ of cardinality n and characteristic 2. We also describe some variations, such as PCPPs for RS-codes over certain prime fields, but these are not needed for our final PCP results. Our proof of proximity consists of an encoding of an efficient FFT-like evaluation of the low-degree polynomial. Our analysis makes crucial (black-box) use of Polishchuk and Spielman's [37] analysis of a natural low-degree test for bivariate polynomials.

We remark that almost all ingredients in the construction of our PCPs, including

the PCPPs for RS-codes, are simple. The simplicity of the PCP also means that the "hidden constants" in the construction are relatively small and the building blocks we use can be implemented with relative ease. In fact, our main building blocks, namely, the PCP of proximity for Reed–Solomon codes and its verifier described in Theorem 3.2, have been recently implemented successfully in code [12], resulting in PCPPs of length $\approx \frac{1}{4}n \cdot \log^4 n$ for RS-codes over binary fields of size n.

Recent developments. One of the main problems left open by this work was obtaining quasilinear PCPs and PCPPs with *constant* query complexity. This was recently solved by Dinur in [18] by applying her novel proof of the PCP theorem by gap amplification to our Theorem 2.2. Dinur provides a general transformation that takes any PCP of length $\ell(n)$ where the verifier makes $q(n)$ queries and converts it into a PCP of length $\ell(n) \cdot q(n)^{O(1)}$ where the verifier makes $O(1)$ queries. Applying this to the trivial PCP that makes $O(n)$ queries yields a simple proof of the PCP theorem, though with long proofs. On the other hand, applying this transformation to our PCP yields a quasilinear PCP with constant query complexity.

With the exception of [5], the running time of all previously known PCP- and PCPP-verifiers, including ours, is polynomial in the size of the input. Recently, it was shown in [10] that the running time of our PCPP-verifier can be reduced to be polylogarithmic, maintaining the query complexity and proof length of our PCPP construction in Theorem 2.10.

Organization of this paper. In section 2 we present formal definitions of the notions of PCPs, LTCs, and PCPPs, and we present the formal statements of our main theorems about these concepts. In section 3 we introduce the main technical notions used in this paper, namely, Reed–Solomon codes, some computationally important subclasses of Reed–Solomon codes, and algebraic satisfiability problems. We state our technical results about these problems and then show how our main theorems (i.e., the ones stated in section 2) follow from these technical results. In sections 4–7, we prove our technical results. A more detailed breakdown of these results is given at the end of section 3.

2. Definitions and main results.

Preliminaries. Unless specified otherwise, our alphabet of choice is $\Sigma = \{0, 1\}$ and all logarithms are taken to base 2. For a function $t : \mathbb{N}^+ \to \mathbb{N}^+$, recall that NTIME($t(n)$) is the class of languages $L \subseteq \Sigma^*$ decidable in nondeterministic time $t(n)$ on inputs of length n.

2.1. PCPs.

The following is a variant of the standard definition of PCPs [3], where the running time of the verifier is allowed to grow exponentially with the randomness. This is done following [10] to allow a statement of results about languages whose nondeterministic decision time is superpolynomial. Recall that an oracle machine is said to be *nonadaptive* if its queries do not depend on previous oracle answers. We stress that all oracle machines considered in this paper, and, in particular, the following PCP-verifier and the PCPP-verifier of Definition 2.4, are nonadaptive.

DEFINITION 2.1 (PCP). *For functions $r, q : \mathbb{N}^+ \to \mathbb{N}^+$ an $(r(n), q(n))$-*PCP-verifier *is a probabilistic machine V with oracle access to a* probabilistically checkable proof*, or simply, a* proof*, denoted π. On input x of length n, V runs in time $2^{O(r(n))}$, tosses $r(n)$ coins, makes $q(n)$ nonadaptive queries to the proof, and outputs either* `accept` *or* `reject`*. We denote by $V^{\pi}[x; R]$ the output of V on input x, proof π, and random coins R.*

For constant $s \in [0, 1]$, a language $L \subseteq \Sigma^*$ is said to belong to the class of

languages

$$\mathbf{PCP}_s[\mathsf{randomness}\ r(n), \mathsf{query}\ q(n)]$$

if there exists an $(r(n), q(n))$-PCP-verifier V_L such that the following hold:

- Perfect completeness: *If $x \in L$ then $\exists \pi$ such that* $\Pr_R[V_L^{\pi}[x; R] = \texttt{accept}] = 1$.
- Soundness: *If $x \notin L$ then $\forall \pi$ we have* $\Pr_R[V_L^{\pi}[x; R] = \texttt{accept}] \leq s$.

Our first main result is the following. Recall the definition of a *proper complexity function* from [36, Definition 7.1], where a function $f(n)$ is proper if it can be computed in time $\text{polylog}\, n$.

THEOREM 2.2 (quasilinear PCPs). *For any proper complexity function $t : \mathbb{N}^+ \to \mathbb{N}^+$,*

$$\text{NTIME}(t(n)) \subseteq \mathbf{PCP}_{\frac{1}{2}}\mathsf{randomness}\ \log(t(n) \cdot \text{polylog}\, t(n)), \mathsf{query}\ \text{polylog}\, t(n)].$$

Remark 2.3. The parameters of Theorem 2.2 have been recently improved. In particular, [18] reduced the query complexity to $O(1)$ and [10] reduced the verifier's running time to $\text{poly}\, n + \text{polylog}\, t(n)$ (as opposed to $t(n) \cdot \text{polylog}\, t(n)$). In both cases all other parameters remain unchanged.

Since without loss of generality the proof is of size at most $2^{\mathsf{randomness}} \times \mathsf{query}$ the previous theorem implies that the probabilistically checkable proof for $x \in L$ is quasilinear in the running time of the nondeterministic machine deciding L.

In contrast, the recent results of [9] give proofs of length $n \cdot \exp(\text{poly}\log\log n)$ with a query complexity of $\text{poly}\log\log n$ and slightly longer proofs with constant query complexity. Thus, while the query complexity of our PCPs is higher than that of the previous state of the art, their length is shorter.

2.2. Proximity and proofs of proximity. We now formalize the notion of verifying proofs of theorems where even the theorem is not known but rather is provided as an oracle to the verifier. The verifier, in such a case, can hope only to certify that the theorem is "close" to one that is true. To define this notion we first need to formalize the notion of "closeness," or proximity.

We will work with a variety of distance measures $\Delta : \Sigma^N \times \Sigma^N \to [0, 1]$, where a distance measure satisfies the properties (1) $\Delta(x, x) = 0$, (2) $\Delta(x, y) = \Delta(y, x)$, and (3) $\Delta(x, z) \leq \Delta(x, y) + \Delta(y, z)$. The most common/natural one, and the target of most of our theorems, will be *relativized Hamming distance* over the alphabet Σ, denoted $\mathsf{Hamming}_{\Sigma}(\cdot, \cdot)$. Formally, for $y = (y_1, \ldots, y_N), y' = (y'_1, \ldots, y'_N) \in \Sigma^N$,

$$\mathsf{Hamming}_{\Sigma}(y, y') = |\{i : y_i \neq y'_i\}|/N.$$

For our proofs we use other distance measures on strings which may weigh different coordinates differently. For example, given a set $I \subseteq [N]$ we may consider the distance $\mathsf{Hamming}_{\Sigma,I}(y, y') = |\{i \in I : y_i \neq y'_i\}|/|I|$. Note that a convex combination of distance measures is also a distance measure, and this describes many other distance measures we use later.

Given a distance measure $\Delta : \Sigma^N \times \Sigma^N \to [0, 1]$ and a set $S \subseteq \Sigma^N$ we define the distance of an element $y \in \Sigma^N$ from S to be

$$\Delta(y, S) = \begin{cases} \min_{s \in S} \Delta(y, s), & S \neq \emptyset, \\ 1, & S = \emptyset. \end{cases}$$

We are now ready to describe PCPs of proximity (PCPPs)/assignment testers [9, 19]. We follow the general formulation as appearing in [9]. In this formulation, the input comes in two parts (x, y), where $x \in \Sigma^*$ is given explicitly to the verifier and $y \in \Sigma^*$ is given as oracle. In addition, the verifier is given oracle access to a proof. The verifier is allowed to read x in its entirety, but its queries to y are counted as part of its query complexity, i.e., together with the queries to the proof. Throughout this paper we assume without loss of generality the explicit input of a pair instance includes a specification of the length of the implicit input. If unspecified we set the length to be $t(|x|)$. Formally, we assume the explicit input is of the form $x = (x', N)$, where $N = |y|$. The size of the explicit input is the size of x'.

DEFINITION 2.4 (PCPP-verifier). *For functions $r, q : \mathbb{N}^+ \to \mathbb{N}^+$ an $(r(n), q(n))$-*PCPP-verifier *is a probabilistic machine V with oracle access to an* implicit input y *and a* proof of proximity*, or simply, a* proof*, denoted π. On explicit input $x = (x', N)$ with $|x'| = n$, and N an integer, verifier V runs in time $2^{O(r(n))}$, tosses $r(n)$ coins, makes at most $q(n)$ nonadaptive queries in total to the two oracles, y of size N and π, and outputs either* `accept` *or* `reject`*. We denote by $V^{(y,\pi)}[x; R]$ the output of the PCPP-verifier on input x and random coins R.*

PCPPs refer to languages consisting of pairs of strings where the elements in these pairs refer to the two parts of the input in Definition 2.4. Thus, we define a *pair language* to be subset of $\Sigma^* \times \Sigma^*$. It is useful for us to measure the complexity of a pair language as a function of its first input. So PAIR-TIME$(t(n))$ is the set of languages L such that there exists a machine M that takes time $t(|x|)$ on input (x, y) such that $L = \{(x, y) : M(x, y) = \texttt{accept}\}$. One notable pair language in PAIR-TIME$(n \cdot \operatorname{polylog} n)$ is CKTVAL, the language of pairs (C, w), where C is a Boolean circuit with N inputs and w is an assignment satisfying C. PAIR-NTIME$(t(n))$ is defined similarly, this time allowing M to be a nondeterministic machine.

For a pair language L and $x \in \Sigma^*$, $x = (x', N)$, let

$$L_x \triangleq \{y \in \Sigma^N : (x, y) \in L\}.$$

PCPP-verifiers are intended to accept implicit inputs in L_x and reject implicit inputs that are far from being in L_x. This gives rise to classes of pair languages defined in terms of PCPPs.

DEFINITION 2.5 (PCPP). *For functions $r, q : \mathbb{N}^+ \to \mathbb{N}^+$,* soundness parameter *$s \in [0, 1]$, family of* distance measures *$\Delta = \{\Delta_N : \Sigma^N \times \Sigma^N \to [0, 1]\}_{N \in \mathbb{N}^+}$, and* proximity parameter *$\delta \in [0, 1]$ we say the pair language L belongs to the class of languages*

$$\mathbf{PCPP}_{s,\delta}\left[\begin{array}{l}\mathsf{randomness}\ r(n),\\ \mathsf{query}\ q(n),\\ \mathsf{distance}\ \Delta\end{array}\right]$$

if there exists an $(r(n), q(n))$-PCPP-verifier V_L such that the following hold for all $(x, y), |y| = N$:

- Perfect completeness: *If $(x, y) \in L$ then $\exists \pi$ such that* $\Pr_R[V_L^{(y,\pi)}[x; R] = \texttt{accept}] = 1$.
- Soundness: *If $\Delta_N(y, L_x) \geq \delta$, then $\forall \pi$* $\Pr_R[V_L^{(y,\pi)}[x; R] = \texttt{accept}] \leq 1 - s$.

Remark 2.6. As mentioned earlier, our main results (for example, Theorem 2.10) target the relative Hamming distance. However, to prove these we shall need to use PCPPs with different distance measures (see subsection 3.4).

Our constructions of PCPPs come naturally with a somewhat different soundness condition than the one required in Definition 2.5. On the one hand, they do not achieve a soundness error of an absolute constant. On the other hand, they satisfy the additional property that a PCPP-verifier for L_x rejects *every* string y with probability proportional to the distance of y from L_x. We formalize this "strong" soundness condition below and then state a general transformation from PCPPs with strong soundness to the weaker version above. (The term "strong" is derived from the analogous definition of *strong locally testable codes* [24, Definition 2.1].)

DEFINITION 2.7 (strong PCPP). *For r, q, Δ as in Definition 2.5 and* soundness function $s : (0,1] \times \mathbb{N}^+ \to (0,1]$*, we say language L belongs to the class*

$$\mathbf{Strong\text{-}PCPP}_{s(\delta,n)} \left[\begin{array}{l} \mathsf{randomness}\ r(n), \\ \mathsf{query}\ q(n), \\ \mathsf{distance}\ \Delta \end{array} \right]$$

if there exists an $(r(n), q(n))$-PCPP-verifier V_L with perfect completeness as in Definition 2.5 and for all (x, y), $|y| = N$, the following holds:

- Strong soundness: $\forall \pi \quad \Pr_R[V_L^{(y,\pi)}[x; R] = \mathtt{accept}] \leq 1 - s(\Delta_N(y, L_x), n)$.

Remark 2.8. Naturally, one expects the soundness function to be nondecreasing. Formally, we say $s : (0,1] \times \mathbb{N}^+ \to [0,1]$ is *nondecreasing* if for all $n \in \mathbb{N}^+$ the function $s(\cdot, n) : (0,1] \to (0,1]$ is nondecreasing. This implies that the farther y is from L_x, the higher the rejection probability or soundness. Indeed, all soundness functions considered in this paper are nondecreasing.

Notice that a "weak" PCPP, with soundness parameter s_0 and distance parameter s_0, is also a "strong" PCPP with a threshold soundness function $s(\delta, n)$ that evaluates to 0 on $\delta' < \delta_0$ and to s_0 on $\delta' \geq \delta_0$. A converse of this is also true. To see this one needs only to amplify the soundness error from $s(\delta, n)$ to some fixed desired constant s'. The now standard application of randomness efficient sampling allows such amplification with little additional cost in randomness. Indeed, using the expander-neighborhood sampler of [25] (see also [22, section C.4]) we get the following proposition, given here without proof. (For a proof see [9, Lemma 2.11].)

PROPOSITION 2.9 (strong PCPPs imply "weak" ones). *Let $s : (0,1] \times \mathbb{N}^+ \to (0,1]$ be a nondecreasing soundness function as defined in Remark 2.8. If a pair language L belongs to*

$$\mathbf{Strong\text{-}PCPP}_{s(\delta,n)} \left[\begin{array}{l} \mathsf{randomness}\ r(n), \\ \mathsf{query}\ q(n), \\ \mathsf{distance}\ \Delta \end{array} \right],$$

then, for every $s', \delta \in (0,1)$, the language L belongs to

$$\mathbf{PCPP}_{s',\delta} \left[\begin{array}{l} \mathsf{randomness}\ r(n) + O(\frac{1}{s(\delta,n)} \cdot \log \frac{1}{s'}), \\ \mathsf{query}\ O\left(\frac{q(n) \cdot \log 1/s'}{s(\delta,n)}\right), \\ \mathsf{distance}\ \Delta \end{array} \right].$$

Furthermore, the proof queried by the "weak" PCPP-verifier is of the same length as that queried by the "strong" one.

We are now ready to state our main result for PCPPs.

THEOREM 2.10 (quasilinear PCPPs). *For any proper complexity function t :*

$\mathbb{N}^+ \to \mathbb{N}^+$,

$$\text{PAIR-NTIME}(t(n)) \subseteq \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} t(n)} \left[\begin{array}{l} \mathsf{randomness}\ \log(t(n) \cdot \operatorname{polylog} t(n)), \\ \mathsf{query}\ \operatorname{polylog} t(n), \\ \mathsf{distance}\ \mathsf{Hamming}_\Sigma \end{array} \right].$$

Consequently, as implied by Proposition 2.9*, for any* $s, \delta \in (0,1)$*,*

$$\text{PAIR-NTIME}(t(n)) \subseteq \mathbf{PCPP}_{s,\delta} \left[\begin{array}{l} \mathsf{randomness}\ \log(t(n) \cdot \operatorname{polylog} t(n)), \\ \mathsf{query}\ \operatorname{polylog} t(n), \\ \mathsf{distance}\ \mathsf{Hamming}_\Sigma \end{array} \right].$$

Furthermore, the length of the proof queried by the PCPP-verifier (in both the strong and weak cases) is $t(n) \cdot \operatorname{polylog} t(n)$*.*

Remark 2.11. As in the case of Theorem 2.2, the parameters of Theorem 2.10 have been recently improved. In particular, [18] reduced the query complexity to $O(1)$ and [10] reduced the verifier running time to $\operatorname{poly} n + \operatorname{polylog} t(n)$. In both cases all other parameters remain unchanged.

The previous state of the art with respect to PCPPs [9] gave proofs of length $n \cdot \exp(\operatorname{poly}\log\log n)$ with a query complexity of $\operatorname{poly}\log\log n$ (and slightly longer proofs with constant query complexity). Once again, our query complexity is somewhat higher but our proofs are somewhat shorter.

2.3. Locally testable codes. We now move to the third notion addressed by this paper—that of LTCs.

For field $\mathbb{F}$ and integers n, k, d, a linear $[n,k,d]_{\mathbb{F}}$-code is an injective linear map $C : \mathbb{F}^k \to \mathbb{F}^n$ such that for every pair $x \neq y \in \mathbb{F}^k$, $\mathsf{Hamming}_{\mathbb{F}}(C(x), C(y)) \geq d/n$. We point out that all codes considered in this paper are linear. The *alphabet* of C is $\mathbb{F}$, the *blocklength* is n, the *dimension* is k, the *rate* is k/n, and the *distance* is d. The *image* of C is the linear space $\text{Image}(C) = \{C(a) : a \in \mathbb{F}^k\}$. Often a code is identified with its image.

Loosely speaking, a linear $[n,k,d]_{\mathbb{F}}$-code is said to be *locally testable* if a *tester*, i.e., a randomized machine with oracle access to the supposed codeword, can distinguish with high probability between words in the code and words that are far from it, while making only $o(k)$ random queries into a purported codeword. The following is essentially Definition 2.1 from [24].

DEFINITION 2.12 (locally testable codes). *A randomized polynomial time oracle machine* T *is called a* (δ, q, γ)-tester *for the linear* $[n,k,d]_{\mathbb{F}}$ *code* C *if it satisfies the following two conditions:*

- *For any* $w \in \text{Image}(C)$*,*

$$\Pr[T^w[R] = \texttt{accept}] = 1,$$

  where $T^w[R]$ *denotes the output of the tester on oracle* w *and random coins* R*.*
- *For any* $w \in \mathbb{F}^n$ *such that* $\mathsf{Hamming}_{\mathbb{F}}(w, \text{Image}(C)) \geq \delta$*,*

$$\Pr[T^w[R] = \texttt{reject}] \geq \gamma.$$

A code is said to be (δ, q, γ)-locally testable *if it has a* (δ, q, γ)*-tester.*

THEOREM 2.13 (locally testable codes with polylogarithmic rate). *Let $\delta, \gamma \in (0,1)$, $\Sigma = \mathbb{F}_2$ be the field of two elements and let n be any power of 2. Then, there exists a linear $[N = n \cdot \text{polylog}\, n, K = n/8, D = N/8]_\Sigma$-code that is $(\delta, \text{polylog}\, n, \gamma)$-locally testable. Furthermore, encoding, decoding, and testing can be performed in time polynomial in n.*

Remark 2.14. As with Theorems 2.2 and 2.10, the query complexity of Theorem 2.13 has been recently improved in [18] to $O(1)$, leaving all other parameters unchanged.

We remark that we also give LTCs over a variety of other fields and other choices of n (see Theorems 3.2 and 3.4 for details). Also if we relax the requirement that the code be linear, then the theorem above follows immediately from Theorem 2.10 (and [9, section 4.1]) without any restrictions on the choice of n.

3. Technical ingredients of our constructions. In this section we introduce the main technical ingredients of our paper and prove the three main theorems (Theorems 2.2, 2.10, and 2.13) of our paper, assuming these ingredients. Recall that these theorems promise short PCPs, PCPPs, and LTCs. We stress that while the construction of PCPs and LTCs follows easily from the PCPP construction, this is not the approach in our paper.

We start by constructing an LTC, based on one of the most popular codes, namely, the Reed–Solomon code. We give a PCPP for a language whose elements are essentially Reed–Solomon codewords. Recalling the fact that Reed–Solomon codes are evaluations of univariate polynomials of bounded degree, this result shows how it is possible to prove that a function given as an oracle is close to some polynomial of bounded degree. Subsection 3.1 below describes the actual language based on Reed–Solomon codes and states the PCPP construction that we obtain for this language. This immediately leads to a proof of Theorem 2.13.

We then move to the constructions of PCPs and PCPPs for general NTIME languages. These constructions are obtained by first reducing the NTIME language under consideration to an algebraic version of SAT that we call an algebraic constraint satisfaction problem, and then giving PCPs (and PCPPs) for algebraic constraint satisfaction problems. We define algebraic constraint satisfaction problems and state their completeness for NTIME in subsection 3.2.

The advantage of algebraic constraint satisfaction problems is that the natural "classical" proofs of satisfiability for these problems come in the form of two univariate polynomials of bounded degree, say, f, g, that satisfy some simple constraints. For example, in the PCP construction, the verifier knows some set $H \subseteq \mathbb{F}$ and would like to verify that $g(x) = 0$ for every $x \in H$. The PCPPs for Reed–Solomon codes already show how to prove/verify that the functions f and g are close to some polynomials of bounded degree. In subsection 3.3 we augment this PCPP so as to test that it vanishes on the set H, and this leads us to a proof of Theorem 2.2. Finally, in subsection 3.4 we describe the additional ingredients needed to get a PCPP for NTIME and prove Theorem 2.10 modulo these ingredients.

3.1. PCPPs for Reed–Solomon codes. We start by defining the Reed–Solomon codes and a pair language based on these codes. We then describe two cases of Reed–Solomon codes where we can obtain PCPPs for membership in the language. This yields our main theorem (Theorem 2.13) on LTCs.

DEFINITION 3.1 (Reed–Solomon codes and pair language). *The* evaluation *of a polynomial $P(z) = \sum_{i=0}^{d} a_i z^i$ over $S \subseteq \mathbb{F}, |S| = n$ is the function $p : S \to \mathbb{F}$ defined by*

$p(s) = P(s)$ for all $s \in S$. The formal sum $P(z)$ is called the polynomial corresponding *to (the function) p. The* Reed–Solomon code *of degree at most d over $\mathbb{F}$, evaluated at S, is*

$$\mathrm{RS}(\mathbb{F}, S, d) \triangleq \{p : S \to \mathbb{F} \mid p \text{ is an evaluation of a polynomial of degree } \leq d \text{ over } S\}.$$

The pair language PAIR-RS *is defined as follows. The explicit input is a triple $(\mathbb{F}, S, d)$, where $\mathbb{F}$ is a description of a finite field,*[1] *$S \subseteq \mathbb{F}$, and d is an integer. The size of the explicit input is assumed to be $|S| + O(1)$ field elements because in all our applications both d and $\mathbb{F}$ can be described using $\log |\mathbb{F}|$ bits. The implicit input is a function $p : S \to \mathbb{F}$. The size of the implicit input is $|S|$ field elements. A pair $((\mathbb{F}, S, d), p)$ is in* PAIR-RS *iff $p \in \mathrm{RS}(\mathbb{F}, S, d)$ and the explicit input is in the format described above.*

Notice that $\mathrm{RS}(\mathbb{F}, S, d)$ is the image of a linear $[n, d+1, n-d]_{\mathbb{F}}$-code. To see this, set $S = \{\xi_1, \ldots, \xi_n\}$ and consider the linear map sending $(a_0, \ldots, a_d) \in \mathbb{F}^{d+1}$ to the codeword $(P(\xi_1), \ldots, P(\xi_n))$ for $P(z) = \sum_{i=0}^{d} a_i z^i$.

Next we state our main technical results, namely, quasilinear length proofs of proximity for Reed–Solomon codes. Our results hold for certain "well-behaved" fields and evaluation sets, including fields of characteristic 2 (Theorem 3.2) and multiplicative subgroups that are sufficiently smooth (Theorem 3.4). As is customary when discussing Reed–Solomon codes, our distance measure is the relative Hamming distance over alphabet $\mathbb{F}$, denoted $\mathsf{Hamming}_{\mathbb{F}}$, and our alphabet is the underlying field. In particular, queries are answered by field elements.

3.1.1. Fields of characteristic two.

THEOREM 3.2 (PCPPs for RS-codes over fields of characteristic 2). *Let* PAIR-ADDITIVE-RS *be the restriction of* PAIR-RS *to pairs $((\mathrm{GF}(2^\ell), S, d), p)$, where $\mathrm{GF}(2^\ell)$ is the Galois field of size $n = 2^\ell$ and characteristic 2 and $S \subseteq \mathbb{F}$ is $\mathrm{GF}(2)$-linear. (Recall that S is $\mathrm{GF}(2)$-linear iff for all $\alpha, \beta \in S$ we have $\alpha + \beta \in S$.) Then,*

$$\text{PAIR-ADDITIVE-RS} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ O(1), \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathrm{GF}(2^\ell)} \end{array} \right].$$

Consequently (using Proposition 2.9), for any $s, \delta \in (0, 1)$,

$$\text{PAIR-ADDITIVE-RS} \in \mathbf{PCPP}_{s,\delta} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ \operatorname{polylog} n, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathrm{GF}(2^\ell)} \end{array} \right].$$

Furthermore, the proof queried by the "weak" PCPP-verifier is of the same length as that queried by the "strong" one.

Remark 3.3. The proof of Theorem 3.2 can be modified to obtain (strong) PCPPs with parameters as above for some other fields also. In particular, we can get PCPPs for $\mathbb{F}$ of characteristic $\leq \operatorname{polylog} n$ as long as the evaluation set S is linear over a subfield of $\mathbb{F}$ of size $\operatorname{polylog} n$. For simplicity, and since this suffices for our applications, we prove the result only for characteristic 2.

We prove Theorem 3.2 in section 6. Here we note that Theorem 3.2 immediately leads to a construction of LTCs. In particular, we use it to prove Theorem 2.13 later in this section. But before doing so, we describe a different collection of fields $\mathbb{F}$ and sets S where we can derive PCPPs for Reed–Solomon codes.

[1] An explicit description for such a field could be via a prime a and an irreducible polynomial $g(x)$ over $\mathrm{GF}(a)$.

3.1.2. RS-codes over smooth fields. For this part, and throughout the rest of this paper, let $\mathbb{F}^*$ denote the multiplicative group of a finite field $\mathbb{F}$. The *order* of $\omega \in \mathbb{F}^*$, denoted $\mathrm{ord}(\omega)$, is the smallest positive integer n such that $\omega^n = 1$. The multiplicative group generated by ω is $\langle\omega\rangle \triangleq \{\omega^0, \omega^1, \ldots, \omega^{n-1}\}$.

THEOREM 3.4 (PCPPs for smooth RS-codes). *Let* PAIR-SMOOTH-RS *be the restriction of* PAIR-RS *to pairs* $((\mathbb{F}, \langle\omega\rangle, d), p)$*, where* $\mathrm{ord}(\omega) = n$ *is a power of* 2*. Then,*

$$\text{PAIR-SMOOTH-RS} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ O(1), \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Consequently (using Proposition 2.9*), for any* $s, \delta \in (0, 1)$*,*

$$\text{PAIR-SMOOTH-RS} \in \mathbf{PCPP}_{s,\delta} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ \operatorname{polylog} n, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Furthermore, the proof queried by the "weak" PCPP-verifier is of the same length as that queried by the "strong" one.

Remark 3.5. Examination of the proof of Theorem 3.4 shows that it can be extended to $\langle\omega\rangle$ of size n that is $\operatorname{polylog} n$-smooth; i.e., all prime factors of n are at most $\operatorname{polylog} n$. For simplicity, we state and prove our theorem only for the multiplicative case of a 2-smooth n.

While not immediately evident, prime fields satisfying the requirements of the previous theorem abound. In section 7 we discuss this and provide an alternative proof of the quasilinear PCP Theorem 2.2 that relies on such prime fields. Notice the intersection of PAIR-ADDITIVE-RS and PAIR-SMOOTH-RS is empty. Indeed, a field with a multiplicative subgroup of size 2^k must be of size $c \cdot 2^k + 1$ for integer c, whereas the size of a field of characteristic 2 is a power of 2. Next we show how to construct LTCs using the PCPPs for RS-codes over fields of characteristic two.

3.1.3. Proof of quasilinear LTC—Theorem 2.13.

Proof of Theorem 2.13. Given an integer $n = 2^t$ we use Theorem 3.2 above applied to the field $\mathbb{F}$ of size n, with $S = \mathbb{F}$ and $d = n/8$. The resulting Reed–Solomon code has rate $\Omega(1)$ and relative distance at least 7/8. We then convert the PCPP for this code into an LTC over $\mathbb{F}$ using a standard conversion. Here we simply sketch this step. For a formal proof, see [9, Proposition 4.1].

The codewords of the LTC are in one-to-one correspondence with the codewords of the Reed–Solomon code. The codeword of the LTC corresponding to a polynomial p consists of two parts. The first part is simply the Reed–Solomon encoding of p repeated sufficiently often so that the first part takes at least, say, half of the coordinates of the LTC. The second part consists of the PCPP that p is a member of the language. The LTC-verifier simply simulates the PCPP-verifier using the first half as the oracle for the implicit input and the second half as the proof oracle, along with some spot-checks to verify that the first part repeats the same codeword several times. It is straightforward to see that the rate of this LTC is asymptotically bounded by the length of the Reed–Solomon codewords divided by the length of their PCPP, and the query complexity is similar to that of the PCPP-verifier. It is easy to see that the LTC so obtained has a relative distance of at least 7/16 (i.e., half the relative distance of the Reed–Solomon code).

It remains to convert this code into a binary code. This is also straightforward using the idea of concatenation of codes. We pick a small error-correcting code with $|\mathbb{F}|$ codewords of length $\ell = O(\log |\mathbb{F}|)$ and distance, say, at least $.4\ell$ and represent elements of $\mathbb{F}$ as codewords of this code. This converts the LTC obtained in the previous paragraph into a binary code with relative distance at least .4 times the distance of that code, which yields a relative distance of at least $7/40 > 1/8$. The verifier of the LTC above can now be simulated on this binary code with a multiplicative increase in the query complexity by a factor of $O(\log |\mathbb{F}|)$. □

Thus the PCPP for Reed–Solomon codes immediately leads to short LTCs. Additionally as we discuss in the upcoming sections, it also forms the central ingredient in our PCP and PCPP constructions.

3.2. Algebraic constraint satisfaction problems. To obtain length-efficient PCPs and PCPPs we reduce $L \in \text{NTIME}(t(n))$ to an *algebraic* constraint satisfaction problem. We describe this problem by comparing it to a combinatorial analogue, namely, 3SAT. A 3-CNF formula ψ with n variables and m clauses can be viewed as a mapping $\psi : \{0,1\}^n \to \{0,1\}^m$, sending an assignment to the characteristic vector of the set of clauses satisfied by it. The "natural" proof of satisfiability of a 3-CNF formula is a vector a of n bits. The proof proves the satisfiability of ψ if $\psi(a) = \vec{1}$. The typical advantage of this proof is that verification is a sequence of local steps, i.e., to verify that the jth coordinate of $\psi(a)_j = 1$, we need only examine three coordinates of a.

An instance of an algebraic constraint satisfaction ϕ similarly can be viewed as a mapping $\phi : \mathbb{F}[x] \to \mathbb{F}[x]$ from polynomials to polynomials. A candidate proof for the algebraic problem is a low-degree (univariate) polynomial $A \in \mathbb{F}[x]$ over finite field $\mathbb{F}$, called the *proof polynomial*. The map ϕ would map A to a polynomial P of slightly larger degree. ϕ would be considered satisfiable if $P = \phi(A)$ vanishes on a prespecified subset H of $\mathbb{F}$. Finally, for "local verifiability," we will expect that computing $P(x_0)$ requires knowledge of A at very few places, denoted k. But here we place some very strong restrictions on the local neighborhoods. Whereas in 3SAT, there was no simple relationship between a clause index j and the variables participating in the clause, in algebraic constraint satisfaction problems, we expect $P(x_0)$ to depend on A on some set of points of the form $\{\text{AFF}_1(x_0), \ldots, \text{AFF}_k(x_0)\}$, where $\text{AFF}_i(x) = a_i x + b_i$ is an affine map. Moreover, we insist that the computation of $P(x_0)$ from x_0 and $A(\text{AFF}_1(x_0)), \ldots, A(\text{AFF}_1(x_0))$ itself be algebraically simple. Combining all these ingredients leads to the following definition.

DEFINITION 3.6 (univariate algebraic constraint satisfaction problem (CSP)). *Instances of the language* ALGEBRAIC-CSP *are tuples of the form* $\phi = (\mathbb{F}, \{\text{AFF}_1, \ldots, \text{AFF}_{k'}\}, H, C)$, *where* $\mathbb{F}$ *is a field,* $\text{AFF}_i(x) \triangleq a_i x + b_i$ *is an affine map over* $\mathbb{F}$ *specified by* $a_i, b_i \in \mathbb{F}$, $H \subseteq \mathbb{F}$, *and* $C : \mathbb{F}^{k'+1} \to \mathbb{F}$ *is a polynomial of degree at most* $|H|$ *in its first variable. The* size *of* ϕ *is* $|\mathbb{F}|$.

A polynomial $A \in \mathbb{F}[x]$ *is said to* satisfy *the instance* $\phi \in$ ALGEBRAIC-CSP *iff* $\deg(A) \le |H| - 1$ *and for all* $x \in H$,

$$C(x, A(\text{AFF}_1(x)), \ldots, A(\text{AFF}_{k'}(x))) = 0. \tag{3.1}$$

The instance ϕ *is in* ALGEBRAIC-CSP *iff there exists a polynomial satisfying it.*

For integers k, d, *let* ALGEBRAIC-CSP$_{k,d}$ *be the restriction of* ALGEBRAIC-CSP, *to instances as above where* $k' \le k$ *and the degree of* C *in all but the first variable is at most* d.

Our main theorem on PCPs is obtained by reducing NTIME languages to ALGEBRAIC-CSP, while preserving the length of instances to within polylogarithmic factors.

THEOREM 3.7 (ALGEBRAIC-CSP is NTIME-complete). *There exist integers k, d such that for any proper complexity function $t : \mathbb{N}^+ \to \mathbb{N}^+$ and $L \in \mathrm{NTIME}(t(n))$ the following hold.*

1. *L is reducible to* ALGEBRAIC-CSP$_{k,d}$ *in time* $\mathrm{poly}\, t(n)$.
2. *The reduction maps an instance of L of size n to an instance of* ALGEBRAIC-CSP$_{k,d}$ *over field* $\mathrm{GF}(2^\ell)$ *of size* $2^\ell \le t(n)\,\mathrm{polylog}\, t(n)$ *and characteristic* 2, *where* $100(kd+1)(|H|-1) < 2^\ell \le 200(kd+1)(|H|-1)$.

The proof of this theorem is given in section 5.

Remark 3.8. Inspection of the proof of Theorem 3.7 gives $k = 10$ and $d = 8$. More careful optimization can give $k = 9$ and $d = 1$; i.e., C is multilinear in all but the first variable. Favoring simplicity over constant optimization, we omit this proof. Additionally, one can obtain the theorem for any $\mathbb{F}$ as long as $|\mathbb{F}| > |H|$. However, to derive Theorems 2.2 and 2.10 we need $|\mathbb{F}| \gg |H|$.

Very similar algebraic reductions are prevalent in many previous PCPs [5, 3, 2, 37, 39, 11, 9], starting with [5], and our reduction follows that of Polishchuk and Spielman [37]. However, all previous reductions used multivariate polynomials to perform degree reduction. Namely, a message (or assignment) of length n is encoded by an m-variate polynomial of degree $\approx m \cdot n^{1/m}$ (allowing proximity testing with $n^{1/m}$ queries). In contrast, our reduction does not reduce the degree at all; in fact it slightly increases it. The PCPPs for the RS-code described earlier allow us to tolerate this and verify proximity to high-degree polynomials with very small query complexity—logarithmic in the degree.

For our PCPP construction we need to modify the reduction above so that it works appropriately for pair languages. Suppose we wish for a reduction R from a pair language L to a pair language L'. Note that such a reduction can only work with the explicit input of pair languages. Furthermore, the reduction should say something about (the proximity of) the implicit input to an accepting pair. The following definition of a "systematic reduction" (borrowing a phrase from coding theory) specifies our needs.

DEFINITION 3.9 (systematic reduction). *A* systematic reduction *from a pair language L to L' is given by a pair of functions (R, m), $R : \Sigma^* \to \Sigma^*$, and $m : \Sigma^* \times \mathbb{Z}^+ \to \mathbb{Z}^+$ satisfying the following properties:*

- *For every x, the function $m'(i) = m(x, i)$ restricted to the domain $\{1, \ldots, N\}$ is injective and maps to the range $\{1, \ldots, N'\}$, where N denotes the length of the implicit input associated with x, and N' denotes the length of the implicit input associated with $R(x)$.*
- *If $y \in L_x$, then there exists a $y' \in \Sigma^*$ such that $y' \in L'_{R(x)}$ and $y_i = y'_{m(i)}$ for every $i \in \{1, \ldots, N\}$.*
- *If $y' \in L'_{R(x)}$, then $y \in L_x$, where y is the string given by $y_i = y'_{m(i)}$ for $i \in \{1, \ldots, N\}$.*

The running time of the reduction is the maximum of the computation times of $R(x)$ and $m(x, i)$, measured as a function $|x|$.

We next state a variant of Theorem 3.7 giving systematic reductions from pair languages to a language related to ALGEBRAIC-CSP. To this end, we define PAIR-ALGEBRAIC-CSP to be the pair language whose explicit inputs are instances ϕ as in Definition 3.6 and whose implicit inputs are mappings $y : \mathbb{F} \to \mathbb{F}$. A pair

(ϕ, y) is in PAIR-ALGEBRAIC-CSP iff the polynomial corresponding to y satisfies ϕ. We now state our theorem about the completeness of PAIR-ALGEBRAIC-CSP for PAIR-NTIME.

THEOREM 3.10. *There exist integers k, d such that for any proper complexity function $t : \mathbb{N}^+ \to \mathbb{N}^+$ and $L \in$ PAIR-NTIME$(t(n))$ the following hold.*

1. *L is reducible to* PAIR-ALGEBRAIC-CSP$_{k,d}$ *by a systematic reduction, given by the pair (R, m), in time* poly $t(n)$.
2. *For $x \in \{0,1\}^n$, the instance $R(x)$ is an instance of* ALGEBRAIC-CSP$_{k,d}$ *over field* GF(2^ℓ) *of size $2^\ell \le t(n)\,\mathrm{polylog}\, t(n)$ and characteristic 2, where $100(kd+1)(|H|-1) < 2^\ell \le 200(kd+1)(|H|-1)$.*

Since Theorem 3.10 is proved by a minor modification of the proof of Theorem 3.7, we prove them together in section 5.

3.3. Vanishing RS-codes and the PCP construction. The completeness of ALGEBRAIC-CSP for NP, combined with the PCPPs for Reed–Solomon codes, suggests a natural approach to building PCPs. In order to prove that some input instance x belongs to some NP language L, the verifier transforms x into an instance ϕ of ALGEBRAIC-CSP. To prove that ϕ is satisfiable, the prover can write a table of the assignment function $A : \mathbb{F} \to \mathbb{F}$. Furthermore, the prover can also write a table of the transformed polynomial $P = \phi(A)$. In order to verify that this is a valid proof of the satisfiability of ϕ, the verifier need only verify the following three properties: (1) The degrees of A and P are as specified; (2) A and P satisfy the relationship $P = \phi(A)$; and (3) P vanishes on the set H. The PCPP for Reed–Solomon codes solves the problem in (1) above. The locality in the definition of ALGEBRAIC-CSP turns out to lead to a simple solution to step (2) above as well. This leaves us to solve the problem in step (3). In this section we abstract this problem, calling it the vanishing RS-code problem, and state our result showing how to verify this. We then formalize the argument above to get a formal proof of Theorem 2.2.

We remark that the problem in step (3) is a special case of a common problem in all previous algebraic PCPs [4, 5, 20, 3, 2, 37, 39, 11, 9], where the goal is to test whether an m-variate function f, given as an oracle, is close to some polynomial p that *vanishes* on a set H^m for some prespecified subset $H \subset \mathbb{F}$. Our setting specializes this to the case where the functions are univariate (i.e., $m = 1$) as opposed to multivariate in the above mentioned results. This motivates the following pair language.

DEFINITION 3.11 (vanishing RS-codes). *A polynomial $P(z)$ over field $\mathbb{F}$ is said to* vanish *over $H \subseteq \mathbb{F}$ iff for all $h \in H, P(h) = 0$. For field $\mathbb{F}$, subsets S, H of $\mathbb{F}$, and integer d, the H-vanishing RS-code is*

$$\mathrm{VRS}(\mathbb{F}, S, H, d) \triangleq \{p \in \mathrm{RS}(\mathbb{F}, S, d) : \text{ The polynomial correspoding to } p \text{ vanishes on } H\}.$$

The pair language PAIR-VRS *is defined as follows. The explicit input is a quadruple $(\mathbb{F}, S, H, d)$, where $\mathbb{F}$ is a description of a finite field, $S, H \subseteq \mathbb{F}$, and d is an integer. The implicit input is a function $p : S \to \mathbb{F}$. The size of both the implicit and explicit inputs is $O(|S|)$. A pair $((\mathbb{F}, S, H, d), p)$ is in* PAIR-VRS *iff $p \in \mathrm{VRS}(\mathbb{F}, S, H, d)$.*

Note that in the above definition we do not require H to be a subset of S. The following lemma reduces testing proximity to the vanishing RS-code to testing proximity to the standard RS-code.

LEMMA 3.12 (PCPPs for PAIR-VRS). *Suppose a field $\mathbb{F}$, $S \subseteq \mathbb{F}$, and integer d*

are such that

$$\mathrm{RS}(\mathbb{F}, S, d) \in \mathbf{Strong\text{-}PCPP}_{s(\delta)} \left[\begin{array}{l} \mathsf{randomness}\ r, \\ \mathsf{query}\ q, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Then, for any $H \subset \mathbb{F}$ *and* $s'(\delta) = \min\{\delta/2, s(\delta/2)\}$,

$$\mathrm{VRS}(\mathbb{F}, S, H, d) \in \mathbf{Strong\text{-}PCPP}_{s'(\delta)} \left[\begin{array}{l} \mathsf{randomness}\ \max\{r, \log |S|\}, \\ \mathsf{query}\ q + 2, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Applying the previous lemma to Theorems 3.2 and 3.4 we immediately get the following corollary.

COROLLARY 3.13 (quasilinear PCPPs for vanishing RS-codes). *Let* PAIR-ADDITIVE-VRS *be the restriction of* PAIR-VRS *to pairs* $((\mathbb{F}, S, H, d), p))$, *where* $\mathbb{F}, S$ *are as defined in Theorem* 3.2. *Similarly, let* PAIR-SMOOTH-VRS *be the restriction of* PAIR-VRS *to pairs where* $\mathbb{F}, S$ *are as defined in Theorem* 3.4. *Let* $|S| = n$. *Then,*

$$\begin{array}{l}\text{PAIR-ADDITIVE-VRS,} \\ \text{PAIR-SMOOTH-VRS}\end{array} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ O(1), \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Consequently (using Proposition 2.9*), for any* $s, \delta \in (0, 1)$,

$$\begin{array}{l}\text{PAIR-ADDITIVE-VRS,} \\ \text{PAIR-SMOOTH-VRS}\end{array} \in \mathbf{PCPP}_{s,\delta} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ \operatorname{polylog} n, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Furthermore, the proof queried by the "weak" PCPP-verifier is of the same length as that queried by the "strong" one.

Lemma 3.12 generalizes to the case of multivariate polynomials and can replace the sumcheck-based protocols in previous PCP constructions [5, 3, 2, 37, 27, 24, 11, 9]. We describe this problem and our solution to it in subsection 4.4. We remark that our solution is both simpler and somewhat more efficient than the previous solutions (alas we do not need it for any of our own constructions). For our PCP construction the univariate version above suffices, as we show next.

3.3.1. Proof of quasilinear PCP—Theorem 2.2. We now show how to use the PCPP for (vanishing) RS-codes to prove Theorem 2.2, which gives short PCPs for all NTIME languages.

Proof of 2.2. *Overview.* We need to show that $L \in \mathrm{NTIME}(t(n))$ has short PCPs. We start by reducing an instance $\psi, |\psi| = n$ of L to an instance ϕ of ALGEBRAIC-CSP$_{k,d}$ of quasilinear size in n. As our proof, we request an evaluation p_0 of the polynomial A satisfying ϕ. Additionally, we request an evaluation p_1 of the polynomial from (3.1) appearing in Definition 3.6. To verify that A satisfies ϕ, we need only test that (i) p_0 is of sufficiently low degree, (ii) p_0, p_1 are consistent, i.e., p_1 is the evaluation of the polynomial from (3.1), and (iii) the polynomial corresponding to p_1 vanishes on H. We test (i) using Theorem 3.2, (iii) using Corollary 3.13, and (ii) using an additional consistency test with constant query complexity. Details follow.

Let $\phi = \{\mathrm{GF}(2^\ell), \{\textsc{Aff}_1, \ldots, \textsc{Aff}_k\}, H, C\}$ be the instance of ALGEBRAIC-CSP$_{k,d}$ that ψ is reduced to via Theorem 3.7. Let $\mathbb{F} = \mathrm{GF}(2^\ell)$ and notice that $|\mathbb{F}| = \Theta(t(n) \operatorname{polylog} t(n))$.

Probabilistically checkable proof. The verifier expects oracle access to a proof comprised of

- function $p_0 : \mathbb{F} \to \mathbb{F}$ and proof of proximity π_0 to $\mathrm{RS}(\mathbb{F}, \mathbb{F}, |H| - 1)$ as per Theorem 3.2 and
- function $p_1 : \mathbb{F} \to \mathbb{F}$ and proof of proximity π_1 to $\mathrm{VRS}(\mathbb{F}, \mathbb{F}, H, (kd+1)(|H|-1))$ as per Corollary 3.13.

Notice that Theorem 3.2 and Corollary 3.13 imply that the size of the proof is quasilinear in $|\mathbb{F}|$, which is quasilinear in $t(n)$.

Verifier operation. The verifier tosses $\log(t(n) \cdot \mathrm{polylog}\, t(n))$ coins and runs the following subtests using the same random coins in all and accepting iff all subtests accept.

- Invoke the PCPP-verifier for the Reed–Solomon code from the second part of Theorem 3.2 on explicit input $(\mathbb{F}, \mathbb{F}, |H| - 1)$, implicit input p_0, and proof π_0, using proximity parameter $\delta = 1/10k$ and soundness half.
- Invoke the PCPP-verifier for vanishing Reed–Solomon code from the second part of Corollary 3.13 on explicit input $(\mathbb{F}, \mathbb{F}, H, |H| - 1)$, implicit input p_1, and proof π_1, using proximity parameter $1/100$ and soundness half.
- Select random $x \in \mathbb{F}$, query $p_0(x), p_0(\textsc{Aff}_1(x)), \ldots, p_0(\textsc{Aff}_k(x))$ and $p_1(x)$; accept iff

$$p_1(x) = C(x, p_0(x), p_0(\textsc{Aff}_1(x)), \ldots, p_0(\textsc{Aff}_k(x))).$$

Basic parameters. Randomness and query complexity are as claimed, by Theorem 3.2 and Corollary 3.13.

Completeness. Suppose $\psi \in L$. Then $\phi \in \text{ALGEBRAIC-CSP}_{k,d}$ by Theorem 3.7. Suppose A satisfies ϕ as per Definition 3.6. Let p_0 be the evaluation of A on $\mathbb{F}$. Let

$$B(x) \triangleq C(x, A(\textsc{Aff}_1(x)), \ldots, A(\textsc{Aff}_k(x))). \tag{3.2}$$

Notice that $\deg(B) \le \deg_{x_0}(C) + \sum_{i=1}^{k} \deg_{x_i}(C) \cdot \deg(A) \le (kd+1)(|H|-1)$. Furthermore, B vanishes on H because A satisfies ϕ. Let p_1 be the evaluation of B on $\mathbb{F}$. We conclude that $p_0 \in \mathrm{RS}(\mathbb{F}, \mathbb{F}, |H| - 1)$ and $p_1 \in \mathrm{VRS}(\mathbb{F}, \mathbb{F}, H, (kd+1)(|H|-1))$, so by the completeness property of Theorem 3.2 and Corollary 3.13 there exist proofs π_0, π_1 causing the first two subtests of the verifier to accept. Finally, for all $x \in \mathbb{F}$ we have by construction

$$\begin{aligned} p_1(x) = B(x) &= C(x, A(\textsc{Aff}_1(x)), \ldots, A(\textsc{Aff}_k(x))) \\ &= C(x, p_0(\textsc{Aff}_1(x)), \ldots, p_0(\textsc{Aff}_1(x))). \end{aligned}$$

We conclude that the last subtest also accepts with probability 1, completing the proof of the completeness statement.

Soundness. Suppose $\psi \notin L$. Then $\phi \notin$ ALGEBRAIC-CSP by Theorem 3.7. There are several cases to consider:

- If p_0 is $(1/10k)$-far from $\mathrm{RS}(\mathbb{F}, \mathbb{F}, |H| - 1)$ then Theorem 3.2 implies the first subtest rejects with probability $1/2$. Similarly, if p_1 is not $(1/100)$-close to $\mathrm{VRS}(\mathbb{F}, \mathbb{F}, H, (kd+1)(|H|-1))$ then Corollary 3.13 implies the second subtest rejects with probability $1/2$.
- Otherwise, let A be the unique polynomial of degree $\le |H| - 1$ that is closest to p_0 and let $B(x)$ be as defined in (3.2). Let $p_2 : \mathbb{F} \to \mathbb{F}$ be defined by

$$p_2(x) = C(x, p_0(\textsc{Aff}_1(x)), \ldots, p_0(\textsc{Aff}_k(x))).$$

A union bound implies p_2 is $(1/10)$-close to the valuation of B on $\mathbb{F}$ because p_0 is $(1/10k)$-close to the valuation of A and $\text{AFF}_i(x)$ is uniformly distributed on $\mathbb{F}$ when x is uniformly distributed on $\mathbb{F}$.
Let B' be the (unique) polynomial closest to p_1. Notice that $B \neq B'$ because if $B = B'$, then B vanishes on H, implying A satisfies ϕ. Summing up, we have p_1 is $(1/100)$-close to B' and p_2 is $(1/10)$-close to B, where $B \neq B'$ are polynomials of degree $\leq |\mathbb{F}|/100$, so they agree on at most $1/100$ fraction of their entries. Thus, the third subtest accepts with probability at most $(1/10) + (1/100) + (1/100) < 1/2$. The soundness analysis is complete and with it we have proved Theorem 2.2. □

3.4. Systematic RS-codes and quasilinear PCPPs. We now turn to the task of building PCPPs for PAIR-NTIME languages. It is relatively straightforward to convert the PCP-verifier for ALGEBRAIC-CSP constructed in the previous section into a PCPP-verifier for PAIR-ALGEBRAIC-CSP. Unfortunately, this is not sufficient to imply a PCPP-verifier for all PAIR-NTIME languages, despite the systematic reduction given by Theorem 3.10.[2]

To get to the underlying issue, consider pair language L in PAIR-NTIME, and consider the task of proving/verifying if $(x, y) \in L$, where x is explicit and y is given as an oracle. Using the systematic reduction of Theorem 3.10, a PCPP-verifier could convert x to an instance $\phi = R(x)$ of ALGEBRAIC-CSP. It now demands proof oracles for a polynomial A satisfying ϕ, along with other ingredients as in the proof of Theorem 2.2 that prove that A satisfies ϕ. The PCPP-verifier can now verify that A satisfies ϕ with few queries. It still needs to verify that A is consistent with the implicit input y. Using the "systematic" nature of the given reduction, it also knows that y ought to be "contained" in A. More specifically, it knows that there is some subset H of A such that the evaluation of the polynomial A on the set H should be equal to the string y. In what follows we abstract this problem as that of building a PCPP-verifier for "systematic" Reed–Solomon codes. Such a verifier is given two oracles, one for a function $f : H \to \mathbb{F}$ (representing the implicit input y above), and the other for a (supposedly polynomial) function $p : S \to \mathbb{F}$, and attempts to verify that p is a polynomial of the appropriate degree that agrees with the function f. We formalize the task below and state our main result for this task.

DEFINITION 3.14 (systematic RS-codes). *For field $\mathbb{F}$, subsets $S, H \subseteq \mathbb{F}, |H| \leq |S|/2$, and integer $d \leq |S|/2$ let $\text{RS}_{\text{sys}}(\mathbb{F}, S, H, d)$ be the set of pairs of functions $(f : H \to \mathbb{F}, p : S \to \mathbb{F})$, such that $p \in \text{RS}(\mathbb{F}, S, d)$ and the polynomial corresponding to p agrees with f on H. The pair language* PAIR-RS_{sys} *is the set of pairs with explicit input $(\mathbb{F}, S, H, d)$ as above and implicit input $(f, p) \in \text{RS}_{\text{sys}}(\mathbb{F}, S, H, d)$. Similarly, the pair language* PAIR-ADDITIVE-RS_{sys} *(*PAIR-SMOOTH-RS_{sys}*, respectively) is the restriction of* PAIR-RS_{sys} *to pairs with $\mathbb{F}, S$ as in Theorem* 3.2 *(Theorem* 3.4*, respectively).*

Notice in Definition 3.14 we do not require H to be a subset of S, nor do we assume $H \cap S = \emptyset$. Furthermore, we allow d to be greater than $|H| - 1$, in which case there are several polynomials of degree d that all agree with f on H.

Recall that when building PCPP-verifiers we need to specify our distance measure. Since typically $|H| \ll |S|$, the standard Hamming distance is not a good measure because under this measure (f, p) is close to RS_{sys} whenever p is low degree, regardless

[2]Indeed, we do not know of a generic reduction that, when given a pair language L with a systematic reduction to L' and a PCPP-verifier for L', can construct an efficient PCPP-verifier for L.

of the amount of agreement between p and f. To amend this we use the following weighted Hamming distance:

$$\mathsf{Hamming}^{\frac{1}{2}}_{\mathbb{F}}((f,p),(f',p')) = \frac{1}{2}(\mathsf{Hamming}_{\mathbb{F}}(f,f') + \mathsf{Hamming}_{\mathbb{F}}(p,p')).$$

The main theorem of this section gives an efficient PCPP for the language of systematic Reed–Solomon codes. Its proof is deferred to subsection 4.3.

THEOREM 3.15 (PCPPs for systematic RS-codes).

$$\begin{array}{l}\text{PAIR-ADDITIVE-RS}_{\text{sys}},\\ \text{PAIR-SMOOTH-RS}_{\text{sys}}\end{array} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n}\left[\begin{array}{l}\mathsf{randomness}\ \log(n\cdot\operatorname{polylog} n),\\ \mathsf{query}\ \operatorname{polylog} n,\\ \mathsf{distance}\ \mathsf{Hamming}^{\frac{1}{2}}_{\mathbb{F}}\end{array}\right].$$

3.4.1. Proof of quasilinear PCPP Theorem 2.10. We now show how to use Theorems 3.10 and 3.15 to construct efficient PCPP-verifiers for all PAIR-NTIME languages.

Proof of Theorem 2.10. Let (x,y) be an instance of PAIR-$L \in \text{NTIME}(t(n))$ with $|y| = N$. The PCPP-verifier starts by reducing x to an instance $\phi = (\mathbb{F}, \{\text{AFF}_1, \ldots, \text{AFF}_k\}, H, C\}$ of ALGEBRAIC-CSP as in Theorem 3.10. Let $m : \{1,\ldots,N\} \to \mathbb{F}$ be the efficiently computable injective mapping as per Definition 3.9. Let $m([N]) = \{m(1),\ldots,m(N)\} \subseteq \mathbb{F}$. From here on we view y as a function from $m([N])$ to $\mathbb{F}$ by associating $\{0,1\}$ with the same elements in $\mathbb{F}$. Verifier expects oracle access to a proof of proximity comprised of the following:

- A PCP π for x as described in the proof of Theorem 2.2. In particular, the PCP is comprised of functions $p_0, p_1 : \mathbb{F} \to \mathbb{F}$ and subproofs π_0, π_1.
- A proof of proximity for PAIR-ADDITIVE-$\text{RS}_{\text{sys}}(\mathbb{F},\mathbb{F},m([N]),|H|-1)$, denoted π_2.

Verifier operation. The verifier invokes the PCP-verifier described in the proof of Theorem 2.2 on explicit input x and proof π. Reusing randomness, the verifier invokes the PCPP-verifier for the pair language PAIR-ADDITIVE-RS_{sys} described in Theorem 3.15 on explicit input $(\mathbb{F},\mathbb{F},m([N]),|H|-1)$, implicit input pair (y,p_0), and proof π_2. The verifier accepts iff both subverifiers accept. Notice that Theorems 2.2 and 3.15 imply the randomness and query complexity are as claimed.

Completeness. Suppose $(x,y) \in$ PAIR-L and let ϕ be the instance of ALGEBRAIC-CSP that x is reduced to as per Theorem 3.7. By Theorem 3.10, y agrees with p_0 on $m([N])$ and p_0 is an evaluation of a polynomial satisfying ϕ. Completeness now follows from Theorems 2.2 and 3.15.

Soundness. Suppose y is δ-far from L_x in distance measure $\mathsf{Hamming}^{\frac{1}{2}}_{\mathbb{F}}$. There are several cases to consider. First, if $x \notin L$ in which case L_x is empty and $\delta = 1$, then Theorem 2.2 implies the first subtest of our verifier rejects with probability $\delta/\operatorname{polylog} n$. From here on we assume $x \in L$. If p_0 is not $(1/100)$-close to an evaluation of a polynomial of degree $\le |H|-1$ that satisfies ϕ, then the soundness part of the proof of Theorem 2.2 implies the first subtest of our verifier rejects with probability $\ge 1/(100\cdot\operatorname{polylog} n) \ge \delta/\operatorname{polylog} n$. Finally, suppose p_0 is $(1/100)$-close to evaluation of a polynomial A satisfying ϕ and let $y' : m([N]) \to \mathbb{F}$ be the evaluation of A on $m([N])$. Theorem 3.10 implies y' satisfies x, so by assumption y is δ-far from y'. Thus, the pair of functions (y,p_0) is $(\delta/2)$-far from $\text{RS}_{\text{sys}}(\mathbb{F},\mathbb{F},m([N]),|H|-1)$, so

Theorem 3.15 implies that the second subtest rejects (y, p_0) with probability at least $\delta/\operatorname{polylog} n$, completing the proof of Theorem 2.10. □

3.5. Organization of the rest of the paper. We have thus far proved our main theorems (Theorems 2.2, 2.10, and 2.13) assuming the NTIME-completeness result for ALGEBRAIC-CSP (Theorems 3.7 and 3.10) and PCPPs for RS-codes (Theorem 3.2), for vanishing Reed–Solomon codes (Lemma 3.12), and for systematic Reed–Solomon codes (Theorem 3.15). In the following sections we give proofs for these claims. We remark that the sections may be read in any order. The order in which they are sequenced here merely reflects our opinion of the complexity of the proofs.

In section 4 we assume a PCPP for Reed–Solomon codes and give PCPPs for vanishing and systematic Reed–Solomon codes. We also show how to verify vanishing properties of multivariate polynomials in this section (see Lemma 4.7 in subsection 4.4), a result that may be of independent interest. In section 5 we prove the NTIME completeness of ALGEBRAIC-CSP and PAIR-ALGEBRAIC-CSP. In section 6 we give the PCPP for RS-codes over fields of characteristic two, which is our central technical result. Finally, in section 7, we give an analogous PCPP for RS-codes over smooth fields.

4. PCPPs for vanishing and systematic Reed–Solomon codes. In this section, we show how one can test various properties of polynomials given by an oracle, once we have the ability to test that an oracle is close to a polynomial.

The first such property is to verify that a univariate function f is close to some polynomial P that *vanishes* on a prespecified set H. This property was used crucially in building a PCP for NP languages in subsection 3.3.

The PCPP for vanishing RS-codes immediately leads to a PCPP to verify if two given oracles f_1 and f_2 are close to polynomials that agree on the prespecified set of inputs. (This task reduces to verifying whether $f_1 - f_2$ represents a vanishing RS-code.) We refer to this property as "agreeing" Reed–Solomon codes.

We then use the PCPP for agreeing Reed–Solomon codes to get a PCPP for systematic RS-codes. Recall that here, our goal was to take two oracles for functions $f : H \to \mathbb{F}$ and $p : S \to \mathbb{F}$ and verify that p is close to a polynomial P that agrees with f on H. This PCPP was crucial to our PCPP Theorem 2.10.

Finally, we show how to extend our PCPP for vanishing RS-codes to a PCPP for vanishing multivariate polynomial codes even though we do not use this result anywhere in the paper. (However, it was extensively used in previous PCP constructions.)

We note that all the constructions are quite simple and differ from previous such "tests" in a crucial way. Whereas previous tests of properties as above attempt to use the fact that the oracle being tested is close to a polynomial, they do not seem to explicitly use the fact that a low-degree test is available and can be used to test other functions that may be provided by the prover. Our constructions exploit this additional feature to simplify many known tests.

4.1. PCPPs for vanishing Reed–Solomon—Proof of Lemma 3.12. Recall the notion of a vanishing RS-code from Definition 3.11.

LEMMA 4.1 (Lemma 3.12, restated). *Suppose a field $\mathbb{F}$, $S \subseteq \mathbb{F}$, and an integer d are such that*

$$\mathrm{RS}(\mathbb{F}, S, d) \in \mathbf{Strong\text{-}PCPP}_{s(\delta)} \left[\begin{array}{l} \mathsf{randomness}\ r, \\ \mathsf{query}\ q, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Then, for any $H \subset \mathbb{F}$ and $s'(\delta) = \min\{\delta/2, s(\delta/2)\}$,

$$\mathrm{VRS}(\mathbb{F}, S, H, d) \in \mathbf{Strong\text{-}PCPP}_{s'(\delta)} \left[\begin{array}{l} \mathsf{randomness}\ \max\{r, \log|S|\}, \\ \mathsf{query}\ q+2, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

The key observation in our proof is that a degree-d polynomial $P(z)$ vanishes on H iff the polynomial $g_H(z) \triangleq \prod_{h \in H}(z-h)$ divides $P(z)$, i.e., iff there exists a polynomial $\tilde{P}, \deg(\tilde{P}) \leq d - |H|$ such that $P(z) = g_H(z) \cdot \tilde{P}(z)$.

Proof. The PCPP-verifier for $\mathrm{VRS}(\mathbb{F}, S, H, d)$ has oracle access to implicit input $p : S \to \mathbb{F}$ and a proof combined of two parts: (i) a function $\tilde{p} : S \to \mathbb{F}$ (a supposed evaluation of $\tilde{P}$ on S) and (ii) a proof of proximity $\tilde{\pi}$ to $\mathrm{RS}(\mathbb{F}, S, d - |H|)$. Notice that the proof length is $|S| + |\tilde{\pi}|$. The verifier operates as follows.

- Toss $\max\{r, \log|S|\}$ random coins. Let R denote the random outcome.
- Invoke an assumed RS-verifier using randomness R on explicit input $(\mathbb{F}, S, d - |H|)$, implicit input $\tilde{p}$, and proof $\tilde{\pi}$. Reject if the verifier rejects. Otherwise,
- pick random $\alpha \in S$ (using randomness R); read $p(\alpha)$ and $\tilde{p}(\alpha)$; accept iff $p(\alpha) = g_H(\alpha) \cdot \tilde{p}(\alpha)$.

Notice that $g_H(\alpha)$ can be computed in time $\mathrm{poly}(|H| \cdot \log|\mathbb{F}|)$ by the verifier because H is given as an explicit input. Thus, the running time is as claimed, and so are the randomness and query complexity, by construction. Completeness follows by taking $\tilde{p}$ to be the evaluation of $\tilde{P}(z) \triangleq P(z)/g_H(z)$ and taking $\tilde{\pi}$ to be $\tilde{p}$'s proof of proximity to $\mathrm{RS}(\mathbb{F}, S, d - |H|)$.

As to the soundness, assume p is δ-far from $\mathrm{VRS}(\mathbb{F}, S, H, d)$. If $\tilde{p}$ is $\delta/2$-far from $\mathrm{RS}(\mathbb{F}, S, d - |H|)$, then by assumption the RS-verifier rejects $\tilde{p}$ with probability at least $s(\delta/2)$ and we are done. Otherwise, $\tilde{p}$ is $\delta/2$-close to some polynomial Q of degree $\leq d - |H|$. Let $q : S \to \mathbb{F}$ be the evaluation of Q on S. Notice that the function $\tilde{p} \cdot g_H$ is $\delta/2$-close to $q \cdot g_H$ and the latter function is, by construction, a codeword of $\mathrm{VRS}(\mathbb{F}, S, H, d)$. By assumption, p is $\delta/2$-far from $\tilde{p} \cdot g_H$; hence the last subtest of the verifier rejects with probability at least $\delta/2$. This completes our proof. ∎

4.2. Agreeing Reed–Solomon codes. We now show how to extend the PCPP of the previous section to test if two polynomials agree on a given set. Two polynomials $P_1(z), P_2(z)$ are said to *agree* on $H \subseteq \mathbb{F}$ if $P_1(z) = P_2(z)$ for all $z \in H$. Below we formalize the problem of testing agreement.

DEFINITION 4.2 (agreeing RS-codes). *For field $\mathbb{F}$, subsets $S, H \subseteq \mathbb{F}$, and integers $|S|/2 \geq d_1 \geq d_2$ let $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H, d_1, d_2)$ be the set of pairs of functions $p_1, p_2 : S \to \mathbb{F}$, such that $p_1 \in \mathrm{RS}(\mathbb{F}, S, d_1), p_2 \in \mathrm{RS}(\mathbb{F}, S, d_2)$, and the polynomial corresponding to p_1 agrees with the polynomial corresponding to p_2 on H. The pair language* PAIR-$\mathrm{RS}_{\mathrm{agr}}$ *is the set of pairs with explicit input $(\mathbb{F}, S, H, d_1, d_2)$ as above and implicit input $(p_1, p_2) \in \mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H, d_1, d_2)$. Similarly, the pair language* PAIR-ADDITIVE-$\mathrm{RS}_{\mathrm{agr}}$ *(*PAIR-SMOOTH-$\mathrm{RS}_{\mathrm{agr}}$*, respectively) is the restriction of* PAIR-$\mathrm{RS}_{\mathrm{agr}}$ *to pairs with $\mathbb{F}, S$ as in Theorem* 3.2 *(Theorem* 3.4*, respectively).*

LEMMA 4.3. *Assume a field $\mathbb{F}$, $S \subseteq \mathbb{F}$, and integers $d_2 \le d_1 \le |S|/2$ satisfy*

$$\begin{matrix}\mathrm{RS}(\mathbb{F}, S, d_1),\\ \mathrm{RS}(\mathbb{F}, S, d_2),\\ \mathrm{VRS}(\mathbb{F}, S, d_1)\end{matrix} \in \mathbf{Strong\text{-}PCPP}_{s(\delta,|S|)} \left[\begin{array}{l}\text{randomness } r,\\ \text{query } q,\\ \text{distance Hamming}_{\mathbb{F}}\end{array}\right],$$

where s is monotone nondecreasing in δ. Then for any $H \subset \mathbb{F}$,

$$\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H, d_1, d_2) \in \mathbf{Strong\text{-}PCPP}_{s(\delta/8,|S|)} \left[\begin{array}{l}\text{randomness } r,\\ \text{query } q+2,\\ \text{distance Hamming}_{\mathbb{F}}\end{array}\right].$$

Proof. The main idea is that $P_1(z)$ agrees with $P_2(z)$ on H iff $P_1(z) - P_2(z)$ vanishes on H, so we apply the PCPP from Lemma 3.12 to this difference. Details follow.

The verifier for $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H, d_1, d_2)$ has oracle access to implicit inputs $p_1, p_2 : S \to \mathbb{F}$ and proof of proximity comprised of

- proof of proximity π_1 to $\mathrm{RS}(\mathbb{F}, S, d_1)$,
- proof of proximity π_2 to $\mathrm{RS}(\mathbb{F}, S, d_2)$, and
- proof of proximity π_3 to $\mathrm{VRS}(\mathbb{F}, S, d_1)$.

The verifier's operation is to invoke the following three subtests using the same randomness across all tests and accepting iff all of them accept:

- Invoke verifier for $\mathrm{RS}(\mathbb{F}, S, d_1)$ on implicit input p_1 and proof π_1.
- Invoke verifier for $\mathrm{RS}(\mathbb{F}, S, d_2)$ on implicit input p_2 and proof π_2.
- Invoke verifier for $\mathrm{RS}_H(\mathbb{F}, S, d_1)$ on implicit input $p_1 - p_2$ and proof π_3. Querying $p_1 - p_2$ on $\alpha \in S$ is performed by querying $p_1(\alpha), p_2(\alpha)$ and taking their difference.

All properties can be checked as in the proof of Lemma 3.12. For an illustration, consider the soundness. Suppose the pair of functions (p_1, p_2) is δ-far from $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H, d_1, d_2)$. If either one of p_1, p_2 is $\delta/8$-far from $\mathrm{RS}(\mathbb{F}, S, d_1), \mathrm{RS}(\mathbb{F}, S, d_2)$, respectively, then (the first/second subtest of) the verifier rejects with probability $s(\delta/8, |S|)$. Otherwise, $q = (p_1 - p_2)$ is $\delta/4$-close to a polynomial of degree d_1 that by assumption does not vanish on H. Since $d_1 \le |S|/2$ and $\delta \le 1$ we conclude q is $1/4$-far from $\mathrm{VRS}(\mathbb{F}, S, H, d_1)$ in which case the third subtest of the verifier rejects with probability $\ge s(1/4, |S|) \ge s(\delta/8, |S|)$. The last inequality follows from monotonicity of s. ∎

The previous lemma combined with Lemma 3.12 and Theorems 3.2 and 3.4 immediately implies the following.

COROLLARY 4.4 (PCPPs for agreeing RS-codes).

$$\begin{matrix}\text{PAIR-ADDITIVE-RS}_{\mathrm{agr}},\\ \text{PAIR-SMOOTH-RS}_{\mathrm{agr}}\end{matrix} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n} \left[\begin{array}{l}\text{randomness } \log(n \cdot \operatorname{polylog} n),\\ \text{query } O(1),\\ \text{distance Hamming}_{\mathbb{F}}\end{array}\right].$$

4.3. PCPP for systematic Reed–Solomon codes: Proof of Theorem 3.15. Recall the definition of the systematic RS-code (Definition 3.14) and the related notation presented in subsection 3.4. We now prove Theorem 3.15, restated below.

THEOREM 4.5 (Theorem 3.15, restated).

$$\begin{array}{l}\text{PAIR-ADDITIVE-RS}_{\text{sys}},\\ \text{PAIR-SMOOTH-RS}_{\text{sys}}\end{array} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n}\left[\begin{array}{l}\text{randomness } \log(n\cdot\operatorname{polylog} n),\\ \text{query } \operatorname{polylog} n,\\ \text{distance Hamming}_{\mathbb{F}}^{\frac{1}{2}}\end{array}\right].$$

To prove the theorem we apply induction to the following lemma, the proof of which appears below. Roughly, the lemma says that if we have PCPPs for systematic and agreeing RS-codes of size $n/2$, then we can construct systematic codes of size n. Intuitively, the proof is as follows. To verify that a message is indeed encoded by a codeword, we split the message into two parts of equal length. We ask for the encoding of each of these submessages and verify agreement of the subencodings with the encoding of the large message. This part uses PCPPs for agreeing codes described in the previous section. Then we pick one of the submessages at random and verify that it is consistent with its supposed subencoding and for this part we use induction. Details follow.

LEMMA 4.6. *If $H, S \subseteq \mathbb{F}$, and $d \le |S|/2$ satisfy the following conditions:*

1. *there exist $S_0, S_1 \subseteq S, |S_0|, |S_1| = |S|/2$, and a partition $H_0 \cup H_1 = H, |H_0|, |H_1| = |H|/2$ and these sets are computable in time t_0,*
2. *we have*

$$\operatorname{RS}(\mathbb{F}, S, d) \in \mathbf{Strong\text{-}PCPP}_{s_1(\delta,|S|)}\left[\begin{array}{l}\text{randomness } r_1,\\ \text{query } q_1,\\ \text{distance Hamming}_{\mathbb{F}}\end{array}\right],$$

3. *for $i = 0, 1$ we have*

$$\operatorname{RS}_{\text{agr}}(\mathbb{F}, S, H_i, d, |H_i|-1) \in \mathbf{Strong\text{-}PCPP}_{s_2(\delta,|S|)}\left[\begin{array}{l}\text{randomness } r_2,\\ \text{query } q_2,\\ \text{distance Hamming}_{\mathbb{F}}\end{array}\right],$$

4. *for $i = 0, 1$ we have*

$$\operatorname{RS}_{\text{sys}}(\mathbb{F}, S_i, H_i, |H_i|-1) \in \mathbf{Strong\text{-}PCPP}_{s_3(\delta,|S_i|)}\left[\begin{array}{l}\text{randomness } r_3,\\ \text{query } q_3,\\ \text{distance Hamming}_{\mathbb{F}}^{\frac{1}{2}}\end{array}\right],$$

and s_3 is subadditive, i.e., $s_3(\delta_0, |S_0|) + s_3(\delta_1, |S_1|) \ge s_3(\delta_0 + \delta_1, |S_i|)$,

then for any $0 < \alpha < 1/16$,

$$\operatorname{RS}_{\text{sys}}(\mathbb{F}, S, H, d) \in \mathbf{Strong\text{-}PCPP}_{s(\delta,|S|)}\left[\begin{array}{l}\text{randomness } r,\\ \text{query } q,\\ \text{distance Hamming}_{\mathbb{F}}^{\frac{1}{2}}\end{array}\right],$$

where

$$\begin{aligned} s(\delta, |S|) &= \min\left\{s_1(\alpha\delta, |S|), s_2(\alpha\delta, |S|)/2, \frac{1}{2}s_3((2-\alpha)\delta, |S|/2)\right\},\\ r &= \max\{r_1, r_2, r_3\} + 1,\\ q &= q_1 + q_2 + q_3. \end{aligned}$$

Proof of Theorem 3.15. Consider first the case of PAIR-ADDITIVE-$\mathrm{RS}_{\mathrm{sys}}$. We need to prove there exists a constant $c \geq 1$ that will be specified later such that for any $\mathbb{F}$ of characteristic 2, linear space $S \subseteq \mathbb{F}$, $|S| = n$, set $H \subset \mathbb{F}, |H| = 2^\ell \leq |S|/2$, and $d \leq |S|/2$ we have

$$\mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S, H, d) \in \mathbf{Strong\text{-}PCPP}_{\delta/(\log n)^c} \left[\begin{array}{l} \mathsf{randomness}\ \log(n \cdot \mathrm{polylog}\, n), \\ \mathsf{query}\ \mathrm{polylog}\, n, \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}} \end{array} \right].$$

Our proof is by induction on n, using Lemma 4.6. The base case of constant n follows because a verifier can query all entries in the implicit input and reject any pair of functions (f, p) that is not in $\mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S, H, d)$.

As to the inductive step, partition H into two equal sets H_0, H_1 arbitrarily. Let $S_0 = S_1$ be a $(k-1)$-dimensional space. Part 1 of Lemma 4.6 holds by construction. Parts 2 and 3 follow from Theorem 3.2 and Corollary 4.4, respectively, with

$$s_1(\delta, n), s_2(\delta, n) \geq \delta/(\log n)^{c'}, \quad q_1, q_2 = \mathrm{polylog}\, n, \quad r_1, r_2 = \log(n \cdot \mathrm{polylog}\, n),$$

where c' is a constant. Part 4 holds by induction with

$$s_3(\delta, n/2) = \delta/((\log n) - 1)^c, \quad q_3 = \mathrm{poly}((\log n) - 1), \quad r_3 \leq \log(n \cdot \mathrm{polylog}\, n).$$

Notice that s_3 is subadditive. Apply Lemma 4.6 with $\alpha = 1/\log n$. Randomness and query complexities follow immediately and the verifier's running time is polynomial. As to soundness, notice that $\frac{1}{2} s_3((2-\alpha)\delta, n/2) \geq (1 - 1/\log n)\delta/(\log n - 1)^c \geq \delta/\log^c n$. Thus, by selecting $c > c' + 1$ we conclude that

$$s(\delta, n) \geq \min\{\delta/\log^{c'+1} n, \delta/\log^c n\} \geq \delta/\log^c n.$$

This completes the proof.

Regarding PAIR-SMOOTH-$\mathrm{RS}_{\mathrm{sys}}$, change $S = \langle \omega \rangle$ and $S_0 = S_1 = \langle \omega^2 \rangle$, and use Theorem 3.4. The rest of the proof is identical. □

Proof of Lemma 4.6. To prove that p is an evaluation of a polynomial P that agrees with f on H, we request that the prover provide evaluations of the polynomials that agree with P on the two partitions of H. We test agreement of p with these two polynomials, denoted p_0, p_1, and then split f to two corresponding parts and recurse. Details follow. We describe the proof of proximity, followed by the verifier's operation, and conclude with completeness and soundness analysis.

Proof of proximity. The proof for the implicit input pair $f : H \to \mathbb{F}, p : S \to \mathbb{F}$ is defined recursively. In the base case ($|S| = O(1)$) the proof is empty. Otherwise, it is comprised of

- one proof of proximity π to $\mathrm{RS}(\mathbb{F}, S, d)$,
- two functions $p_0, p_1 : S \to \mathbb{F}$,
- two proofs of proximity π_0, π_1 to $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H_0, d, |H_0| - 1)$ and $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H_1, d, |H_1| - 1)$, respectively, and
- two proofs of proximity π'_0, π'_1 to $\mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S_0, H_0, |H_0| - 1)$ and to $\mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S_1, H_1, |H_1| - 1)$, respectively, defined recursively.

Verifier operation. Let $f_i : H_i \to \mathbb{F}$ be the restriction of the function f to domain H_i and let p'_i be the restriction of the function p_i to domain S_i for $i = 0, 1$. The verifier tosses $r = \max\{r_1, r_2, r_3\} + 1$ coins, sets $i \in \{0, 1\}$ according to the first coin, and performs the following subtests reusing the remaining $r - 1$ coins across different tests:

- Invoke verifier for $\mathrm{RS}(\mathbb{F}, S, d)$ on input p and proof π.
- Invoke verifier for $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H_i, d, |H_i| - 1)$ on input pair (p, p_i) and proof π_i.
- Invoke verifier for $\mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S_i, H_i, |H_i| - 1)$ on input pair (f_i, p'_i) and proof π'_i.
- Accept iff all aforementioned tests accept.

Basic properties. The randomness is r, by construction. The query complexity is the sum of queries made by the various subtests, as claimed.

Completeness. Assume $(f, p) \in \mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S, H, d)$. Since p is of degree $\leq d$ there exists a proof π accepted by the first subtest of the verifier with probability one. Let p_i be the polynomial of degree $|H_i| - 1$ that agrees with f_i (on H_i). By construction p agrees with p_i on H_i. Thus, there exist proofs π_i accepted by the second subtest of the verifier with probability 1. Finally, notice that $(f_i, p'_i) \in \mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S_i, H_i, |H_i| - 1)$, so there exist subproofs π'_i causing the third test of the verifier to accept with probability one.

Soundness. Assume the distance of (f, p) from $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H, d)$ is exactly δ. There are several cases to consider. (i) If p is $\alpha\delta$-far from $\mathrm{RS}(\mathbb{F}, S, d)$, then the first test of the verifier rejects with probability $s_1(\alpha\delta, |S|)$. (ii) If for some $i \in \{0, 1\}$ the distance of (p, p_i) from $\mathrm{RS}_{\mathrm{agr}}(\mathbb{F}, S, H_i, d, |H_i| - 1)$ is greater than $\alpha\delta$, then the second test of the verifier rejects with probability $s_2(\alpha\delta, |S|)/2$. The factor half decrease in rejection probability is due to the random selection of i. (iii) Otherwise, because (i) does not hold and $d \leq |S|/2$ and $\alpha < 1/16$, we conclude that p is $\alpha\delta$-close to a unique polynomial P, so f is $((2 - \alpha)\delta)$-far from the evaluation of P on H. Similarly, because (ii) does not hold, we conclude that each of p_0, p_1 is 1/8-close to the unique polynomial agreeing with P on H_i.

For $i = 0, 1$, let δ_i be the distance of (f_i, p'_i) from $\mathrm{RS}_{\mathrm{sys}}(\mathbb{F}, S_i, H_i, |H_i| - 1)$ using measure $\mathsf{Hamming}^{\frac{1}{2}}_{\mathbb{F}}$. Notice that $\delta_0 + \delta_1 \geq (2 - \alpha)\delta$ because p_i is 1/8-close to the evaluation of P on H, so p'_i is 1/4-close to the evaluation of P on H_i, while f is $((2 - \alpha)\delta)$-far from it. By induction, the rejection probability of the third subtest in this case is at least

$$\frac{1}{2}(s_3(\delta_0, |S|/2) + s_3(\delta_1, |S|/2)) \geq \frac{1}{2} s_3(\delta_0 + \delta_1, |S|/2) \geq \frac{1}{2} s_3((2 - \alpha)\delta, |S|/2).$$

Summing up, our rejection probability is at least as claimed and this completes our proof. □

4.4. PCPPs for multivariate polynomials and vanishing Reed–Muller codes. Finally, we give a generalization of Lemma 3.12 to the case of multivariate polynomials. This generalization would suffice to replace the sumcheck-based protocols in previous PCP constructions [5, 3, 2, 37, 27, 24, 11, 9].

In the multivariate problem we are given sets $S, H \subset \mathbb{F}$ and oracle access to a multivariate function $f : S^m \to \mathbb{F}$. We are asked to verify that f is close to a polynomial of degree $\leq d$ in each variable that evaluates to zero on H^m. Once again, we do not need to assume $H \subset S$. Recall that evaluations of low-degree multivariate polynomials form the well-known *Reed–Muller* code. We denote by $\mathrm{RM}(\mathbb{F}, S, d, m)$ the set of functions $p : S^m \to \mathbb{F}^m$ that are evaluations of m-variate polynomials of maximal individual degree d. We denote by $\mathrm{VRM}(\mathbb{F}, S, H, d, m)$ its subcode consisting of all evaluations of polynomials that vanish on H^m. Our main lemma of this section is the following.

LEMMA 4.7 (multivariate zero testing). *Suppose field $\mathbb{F}$, set $S \subseteq \mathbb{F}$, and integers d, m satisfy*

$$\mathrm{RM}(\mathbb{F}, S, d, m) \in \mathbf{Strong\text{-}PCPP}_{s(\delta)} \left[\begin{array}{l} \text{randomness } r, \\ \text{query } q, \\ \text{distance Hamming}_{\mathbb{F}} \end{array} \right].$$

Then, for any $H \subset \mathbb{F}$ and $s'(\delta) = \min\{s(\delta), 1 - ((m+1)\delta + (\frac{d}{|S|})^m)\}$,

$$\mathrm{VRM}(\mathbb{F}, S, H, d, m) \in \mathbf{Strong\text{-}PCPP}_{s'(\delta)} \left[\begin{array}{l} \text{randomness } r + m \log |S|, \\ \text{query } (m+1)(q+1), \\ \text{distance Hamming}_{\mathbb{F}} \end{array} \right].$$

Notice that the query complexity of previous solutions to this problem depended polynomially on the size of H. Our solution has query complexity that is independent of H and is based on a straightforward characterization of VRM that resembles Alon's combinatorial Nullstellensatz [1]). Before proving the lemma we first recall some (relatively well-known) results on testing proximity to Reed–Muller codes.

Testing proximity to multivariate polynomials. It is easy to extend the PCPP for the RS-code into one for the Reed–Muller code (based on multivariate polynomials), given the extensive literature on testing multivariate polynomials using axis parallel lines [4, 5, 20, 3, 37, 21].

For a set $S \subseteq \mathbb{F}$ and an m-variate function $f : S^m \to \mathbb{F}$, let $\delta_m^d(f)$ be the fractional distance of f from $\mathrm{RM}(\mathbb{F}, S, d, m)$. Let $\delta_{m,i}^d(f)$ denote the fractional distance of f from a polynomial of degree d in the ith variable, and an unbounded degree in all other variables. Finally, let $\mathbb{E}[\delta_{m,i}^d(f)]$ be the expectation of $\delta_{m,i}^d$ over random $i \in [m]$. The following lemma is a rephrasing of [3, Lemma 5.2.1]. Notice that Lemma 6.13 is a special case of it with tighter parameters.

LEMMA 4.8 (see [3]). *There exists a universal constant c such that for every $S \subset \mathbb{F}$ such that $|S| \geq \mathrm{poly}(m, d)$,*

$$\delta_m^d(f) \leq c \cdot m \cdot \mathbb{E}[\delta_{m,i}^d(f)].$$

This lemma and Theorem 3.2 imply short PCPPs for Reed–Muller codes.

LEMMA 4.9 (RM PCP of proximity). *Let $S \subset \mathbb{F}$ and d, m be integers such that $|S| \geq \mathrm{poly}(m, d)$ for the polynomial of Lemma 4.8 and suppose*

$$\mathrm{RS}(\mathbb{F}, S, d) \in \mathbf{Strong\text{-}PCPP}_{s(\delta)} \left[\begin{array}{l} \text{randomness } r, \\ \text{query } q, \\ \text{distance Hamming}_{\mathbb{F}} \end{array} \right].$$

Then

$$\mathrm{RM}(\mathbb{F}, S, d, m) \in \mathbf{Strong\text{-}PCPP}_{s(\delta)/m} \left[\begin{array}{l} \text{randomness } r + \log(m \cdot |S|^{m-1}), \\ \text{query } q, \\ \text{distance Hamming}_{\mathbb{F}} \end{array} \right].$$

Proof. The proof for a purported RM-codeword is the collection of proofs of proximity for each axis parallel line (to the RS-code). A line parallel to the ith axis is $\{(b_1, \ldots, b_{i-1}, x_i, b_{i+1}, \ldots, b_m) : x_i \in S\}$, where $b_1, \ldots, b_m \in S$. The verifier selects a random axis parallel line and invokes the RS-verifier of Definition 6.7 on the line and its proof. The proof follows from Lemma 4.8. □

Remark 4.10. A more query-efficient test can be constructed when $S = \mathbb{F}$. Instead of axis parallel lines, we use an ϵ-biased set of directions as in [11]. This results in proofs of similar length and query complexity and slightly larger randomness, but the soundness is as large as $\Omega(s(\delta))$ and independent of m.

Testing proximity to vanishing multivariate polynomials. We now move to the proof of Lemma 4.7. The catch in immediately extending the univariate verifier of Lemma 3.12 to even the bivariate case is that the "factoring" concept does not extend immediately. Specifically, if we are given that a bivariate polynomial $Q(x, y)$ has a zero at (α, β) this does not imply that $Q(x, y)$ has some nice factors. However, one can abstract a nice property about Q from this zero. Specifically, we can say that there exist polynomials $A(x, y), B(x, y)$ (of the right degree) such that $Q(x, y) = A(x, y) \cdot (x - \alpha) + B(x, y) \cdot (y - \beta)$. Thus to prove that $Q(\alpha, \beta) = 0$, we may ask the prover to give an evaluation of $Q(x, y)$, $A(x, y)$, and $B(x, y)$. We can then test that Q, A, and B are of low degree and that they satisfy the identity above. Extending this idea to m-variate polynomials that are zero on an entire generalized rectangle is straightforward. The technical lemma giving the identity is included below. The lemma is also a key ingredient in Alon's combinatorial Nullstellensatz [1]. We include a proof for completeness.

LEMMA 4.11. *Let $Q(x_1, \ldots, x_m)$ be a polynomial over $\mathbb{F}_Q$ of degree d in each of m variables. Let $H \subseteq \mathbb{F}_Q$ and let $g_H(z) \stackrel{\text{def}}{=} \prod_{\beta \in H}(z - \beta)$. Then Q evaluates to 0 on H^m iff there exist m-variate polynomials $A_1, \ldots, A_m$ of individual degree at most d such that $Q(\vec{x}) = \sum_{i=1}^{m} A_i(\vec{x}) \cdot g_H(x_i)$.*

Remark 4.12. The lemma above is intentionally sloppy with degree bounds. While tighter degree bounds on A_i's can be obtained, this will not be needed for our PCPs.

Proof. One direction is immediate. If $Q(\vec{x}) = \sum_{i=1}^{m} A_i(\vec{x}) \cdot g_H(x_i)$ then $Q(\vec{\alpha}) = 0$ for every $\vec{\alpha} \in H^m$. The other direction is proved in three steps. First, we show that for any polynomial $P(x_1, \ldots, x_m)$ of degree d_j in x_j, and any $i \in \{1, \ldots, m\}$, there exist polynomials $B(x_1, \ldots, x_m)$ and $C(x_1, \ldots, x_m)$ of degree at most d_j in x_j, with the degree of C in x_i being at most $\min\{d_j, |H|-1\}$, such that $P(\vec{x}) = B(\vec{x}) \cdot g_H(x_i) + C(\vec{x})$. Second, we show that there exist polynomials $A_1, \ldots, A_m$ and R with the A_i's having degree at most d in each variable and R having degree at most $|H|-1$ in each variable such that $Q(\vec{x}) = \sum_{i=1}^{m} A_i(\vec{x}) \cdot g_H(x_i) + R(\vec{x})$, where Q is the polynomial from the lemma statement. In the final step, we show that $R(\vec{x}) = 0$, concluding the proof.

Step 1. Recall that any polynomial $f(x_i)$ can be written as $q(x_i) \cdot g_H(x_i) + r(x_i)$, where r has degree less than $|H|$. Applying this fact to the monomials x_i^D for nonnegative D we find that there exist polynomials $q_D(x_i)$ and $r_D(x_i)$, with degree of q_D being at most D and degree of r_D being less than $|H|$, such that $x_i^D = q_D(x_i) \cdot g_H(x_i) + r_D(x_i)$. Now consider any polynomial $P(x_1, \ldots, x_m)$ of degree d_i in x_i. Suppose $P(\vec{x}) = \sum_{D=0}^{d_i} P_i(\vec{x}') \cdot x^D$, where $\vec{x}' = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m)$. Writing the monomials x_i^D in terms of the q_D's and r_D's, we get

$$P(\vec{x}) = \left(\sum_{D=0}^{d_i} P_i(\vec{x}') q_D(x_i) \right) \cdot g_H(x_i) + \left(\sum_{D=0}^{d_i} P_i(\vec{x}') r_D(x_i) \right).$$

Letting $B(\vec{x}) = \sum_{D=0}^{d_i} P_i(\vec{x}') q_D(x_i)$ and $C(\vec{x}) = (\sum_{D=0}^{d_i} P_i(\vec{x}') r_D(x_i))$ yields the polynomials as claimed. In particular, the degrees of B and C in any variable are no more than that of P, and the degree of C in x_i is smaller than $|H|$.

Step 2. We now claim that there exist polynomials $A_1, \ldots, A_m$ and $R_0, \ldots, R_m$ such that for every $j \in \{0, \ldots, m\}$, $Q(\vec{x}) = \sum_{i=0}^{j} A_i(\vec{x}) \cdot g_H(x_i) + R_j(\vec{x})$, with A_i's being of degree at most d in each variable and R_j being of degree less than $|H|$ in $x_1, \ldots, x_j$ and of degree at most d in the remaining variables. The proof is straightforward by induction on j, with the induction step using Step 1 on the polynomial $P() = R_j()$ and the variable x_{j+1}. The final polynomials $A_1, \ldots, A_m$ and $R = R_m$ are the polynomials as required to yield the subclaim of this step.

Step 3. Finally, we note that for every $\vec{\alpha} \in H^m$, we have $R(\vec{\alpha}) = Q(\vec{\alpha}) - \sum_{i=1}^{m} A_i(\vec{\alpha}) \cdot g_H(\alpha_i) = 0 - \sum_{i=1}^{m} 0 = 0$. But R is a polynomial of degree less than $|H|$ in each variable and is zero on the entire box H^m. This can happen only if $R \equiv 0$. Thus we get that $Q(\vec{x}) = \sum_{i=1}^{m} A_i(\vec{x}) \cdot g_H(x_i)$, with the A_i's being of degree at most d in each variable, as required in the completeness condition. ∎

Proof of Lemma 4.7. As a proof of the proximity of $q \in \mathbb{F}^{S^m}$ to the code $\mathrm{VRM}(\mathbb{F}, S, d, m)$ our verifier expects (i) the evaluations of $A_1, \ldots, A_m$ from Lemma 4.11 on S^m, denoted $a_1, \ldots, a_m$, and (ii) for each of $q, a_1, \ldots, a_m$, a proof of proximity of A_i to $\mathrm{RM}(\mathbb{F}, S, d, m)$. Proof length is as claimed. The verifier operates as follows. First, it tests proximity of each of $q, a_1, \ldots, a_m$ to $\mathrm{RM}(\mathbb{F}, S, d, m)$. Then, a random $\langle \alpha_1, \ldots, \alpha_m \rangle \in S^m$ is selected and the verifier accepts iff $q(\vec{\alpha}) = \sum_{i=1}^{m} g_H(\alpha_i) \cdot a_i(\vec{\alpha})$. The query complexity is as claimed. Completeness follows from Lemma 4.11. As to the soundness, if any of $q, a_1, \ldots, a_m$ is δ-far from $\mathrm{RM}(\mathbb{F}, S, d, m)$, the verifier rejects with probability $s(\delta)$. Otherwise, q is δ close to a polynomial Q that does not vanish on H^m. If $A_1, \ldots, A_m$ are the polynomials closest to $a_1, \ldots, a_m$, respectively, then by Lemma 4.11 we get $Q(\vec{x}) \neq \sum_i A_i(\vec{x}) \cdot g_H(x_i)$ and Q has degree at most d in each variable. Thus, the two polynomials agree on $\leq d^m$ points, so the acceptance probability of the verifier is $\leq (m+1)\delta + (\frac{d}{|S|})^m$ as claimed. ∎

5. Quasilinear reductions of NTIME(n) to ALGEBRAIC-CSP.

In this section we show the completeness of ALGEBRAIC-CSP for NTIME classes, thereby proving Theorem 3.7 (restated below). We also show how to modify this proof to get a proof of Theorem 3.10, which shows the completeness of PAIR-ALGEBRAIC-CSP for PAIR-NTIME classes under systematic reductions.

THEOREM 5.1 (Theorem 3.7, restated). *There exist integers k, d such that for any proper complexity function $t : \mathbb{N}^+ \to \mathbb{N}^+$ and $L \in \mathrm{NTIME}(t(n))$ the following hold:*

1. *L is reducible to $\mathrm{ALGEBRAIC\text{-}CSP}_{k,d}$ in time $\mathrm{poly}\, t(n)$.*
2. *An instance of L of size n is reduced to an instance of $\mathrm{ALGEBRAIC\text{-}CSP}_{k,d}$ over field $\mathrm{GF}(2^\ell)$ of size $2^\ell \leq t(n)\,\mathrm{polylog}\, t(n)$ and characteristic 2, where $100(kd+1)(|H|-1) < 2^\ell \leq 200(kd+1)(|H|-1)$.*

5.1. Warmup—quadratic size reduction.

To illustrate the ideas used in the proof of Theorem 3.7, we start with a simpler proof of a weaker version of it, where the size blowup is quadratic rather than quasilinear. Our starting point is the following NP-complete language essentially from Cook's theorem [14]. (See also [36, proof of Theorem 8.2]).

DEFINITION 5.2 (domino tiling). *A domino tiling instance over alphabet Σ is a tuple of constraints $\psi = \{\hat{C}_{ij} : i, j \in \{0, \ldots, n-2\}\}$, where each constraint is a mapping $\hat{C}_{ij} : \Sigma^3 \to \{\mathtt{accept}, \mathtt{reject}\}$. An instance is satisfiable iff there exists a mapping $\hat{A} : \{0, n^2 - 1\} \to \Sigma$ such that for all $i, j \in \{0, \ldots, n-2\}$*

$$\hat{C}_{i,j}(\hat{A}(in+j), \hat{A}(in+j+1), \hat{A}((i+1)n+j) = \mathtt{accept}.$$

The language DOMINO-TILING$_\Sigma$ *is the set of all satisfiable instances over alphabet* Σ.

THEOREM 5.3 (DOMINO-TILING is NTIME-complete [14]). *There exists a finite size alphabet* Σ *such that if* $L \in \text{NTIME}(t(n))$ *for a proper complexity function* $t : \mathbb{N}^+ \to \mathbb{N}^+$, *then* L *is reducible to* DOMINO-TILING$_\Sigma$ *under quadratic size reductions.*

Our warmup version of Theorem 3.7 is the following.

THEOREM 5.4. *For every finite alphabet* Σ, *the language* DOMINO-TILING$_\Sigma$ *is reducible under linear sized reductions to* ALGEBRAIC-CSP$_{4,|\Sigma|}$.

Notice that although the reduction from DOMINO-TILING to ALGEBRAIC-CSP is linear, the reduction from a language $L \in \text{NTIME}(t(n))$ to DOMINO-TILING incurs a quadratic size blowup.

Proof. We reduce an instance ψ of DOMINO-TILING$_\Sigma$ to an instance $\phi = (\mathbb{F}, \{\text{AFF}_1, \ldots, \text{AFF}_4\}, H, C)$ as in Definition 3.6. We will make crucial use of the fact that the constraint $\hat{C}_{ij}$ depends on assignment entries whose indices are *linear functions* of i and j.

Fix $\mathbb{F}$ to be any finite field satisfying $100n^2 < |\mathbb{F}| \leq 200n^2$. Let ω be a generator of $\mathbb{F}^*$. Associate Σ with arbitrary elements of $\mathbb{F}$. View an assignment to ψ as a mapping $\hat{A} : \{w^{in+j} : i, j \in \{0, \ldots, n-1\}\} \to \Sigma$ where the domain and range of this mapping are subsets of $\mathbb{F}$. The arithmetized instance ϕ will be satisfied only by polynomials A that are a low-degree extension of an assignment $\hat{A}$ that satisfies ψ. Thus, the constraint polynomial C will ensure that (i) A takes only values in Σ on $I = \{w^{in+j} : i, j \in \{0, \ldots, n-1\}\}$ and (ii) the evaluation of A on I produces an assignment $\hat{A}$ that satisfies ψ. Details follow.

Define

$$\text{AFF}_1(x) = x;\ \text{AFF}_2(x) = x \cdot \omega^n;\ \text{AFF}_3(x) = x \cdot \omega, \text{AFF}_4(x) = x \cdot \omega^{-n^2},$$

$$H = I \cup \omega^{n^2} \cdot I = \{\{w^{in+j} : i \in \{0, \ldots, 2n-1\}, j \in \{0, \ldots, n-1\}\}.$$

Notice that $|I| = n^2$ and $|H| = 2n^2$. We now define the constraint polynomial C.

Notice that $\hat{C}_{i,j}$ can be interpreted as a function from $\Sigma^3 \subset \mathbb{F}^3$ to $\{0,1\} \subset \mathbb{F}$ and we associate 0 with `accept` and 1 with `reject`. Arithmetize this constraint by a trivariate polynomial $C_{i,j} : \mathbb{F}^3 \to \mathbb{F}$ of degree at most $|\Sigma| - 1$ in each variable, satisfying

$$C_{i,j}(\sigma_1, \sigma_2, \sigma_3) = \hat{C}_{i,j}(\sigma_1, \sigma_2, \sigma_3) \quad \forall \sigma_1, \sigma_2, \sigma_3 \in \Sigma. \tag{5.1}$$

For $\omega^{in+j} \in H$, let $P_{i,j}(x)$ be the unique polynomial of degree $|H| - 1$ that evaluates to 1 on ω^{in+j} and to 0 on every other element in H. Finally, let $P_\Sigma(x) = \prod_{\sigma \in \Sigma}(x - \sigma)$ be the unique monic nonzero polynomial of degree $|\Sigma|$ whose set of roots is precisely Σ. The constraint polynomial is

$$C(x, y_1, \ldots, y_4) = \sum_{i,j=0}^{n-2} P_{i,j}(x) \cdot C_{i,j}(y_1, y_2, y_3) + \sum_{i=n}^{2n-1} \sum_{j=0}^{n-1} P_{i,j}(x) \cdot P_\Sigma(y_4). \tag{5.2}$$

The polynomial $P_{i,j}$ is often used to "bundle" together many constraints and verify that all of them are satisfied, forming the algebraic analogue of an AND gate. The second summand on the right-hand side of (5.2) corresponds to the set of constraints (i) mentioned above, and the first summand corresponds to (ii).

Notice that C has degree $|H|-1$ in its first variable and degree $|\Sigma|$ in the remaining variables. We conclude that ϕ is a legal instance of ALGEBRAIC-CSP$_{4,|\Sigma|}$.

Completeness. Suppose $\psi \in$ DOMINO-TILING$_\Sigma$ and let $\hat{A}$ be a proof for ψ. Let A be the low-degree extension of $\hat{A}$; i.e., A is a polynomial of degree $\leq n^2 - 1$ satisfying $A(\omega^{in+j}) = \hat{A}(in + j)$ for all $i, j \in \{0, \ldots, n-1\}$. We now prove for all $x \in H$

$$C(x, A(x), A(\omega^n x), A(\omega x), A(\omega^{-n^2} x)) = 0.$$

If $x = \omega^{in+j} \in H$ then by definition of $P_{i,j}$ at most one summand of (5.2) can be nonzero. There are two cases to consider:

- $i \leq n-2$: The summand to consider is $P_{i,j}(\omega^{in+j}) \cdot \hat{C}_{i,j}(\hat{A}(\omega^{in+j}), \hat{A}(\omega^{in+j+1}), \hat{A}(\omega^{(i+1)n+j}))$. This summand vanishes because $\hat{A}$ satisfies $\hat{C}_{i,j}$.
- $i \geq n$: The summand to consider is $P_{i,j}(\omega^{in+j}) \cdot P_\Sigma(\hat{A}(\omega^{in+j}))$, which vanishes because $\hat{A}$ evaluates to Σ on I.

We conclude that $\phi \in$ ALGEBRAIC-CSP$_{4,|\Sigma|}$.

Soundness. Suppose $\phi \in$ ALGEBRAIC-CSP$_{4,|\Sigma|}$ and let A witness this. Let $\hat{A} : \{0, \ldots, n^2 - 1\}$ be defined by $\hat{A}(in + j) = A(\omega^{in+j})$. We claim $\hat{A}$ satisfies ψ. First, notice that the range of A on inputs from I is Σ. If this is not the case and $A(\omega^{in+j}) \notin \Sigma$, then $P_\Sigma(A(\omega^{in+j})) \neq 0$, so (5.2) does not vanish on $x = \omega^{n^2+in+j} \in H$.

Since A evaluates to Σ on I and for $\sigma_1, \sigma_2, \sigma_3 \in \Sigma$ (5.1) implies $C_{i,j}(\sigma_1, \sigma_2, \sigma_3) = 0$ iff $\hat{C}(\sigma_1, \sigma_2, \sigma_3) =$ `accept`, we conclude that $\hat{A}$ satisfies ψ so $\psi \in$ DOMINO-TILING$_\Sigma$. This completes our proof. □

5.2. Quasilinear size reduction. In this section we prove Theorem 3.7 and show that ALGEBRAIC-CSP is NTIME($t(n)$)-complete under quasilinear size reductions. Our proof is similar to that of Polishchuk and Spielman [37]; however, our ending point is a problem over univariate polynomials.

Overview. The reason we chose DOMINO-TILING as our starting point in the previous section was because this language was NP-complete and additionally had "nice" structure, in the sense that each constraint ($\hat{C}_{i,j}$) depended on assignment entries whose indices are *linear functions* of the constraint index. The problem with DOMINO-TILING is that the reduction from an arbitrary language in NTIME($t(n)$) to it results in instances of size $t^2(n)$. Thus, we are looking for an NP-complete language that has a similar "nice" structure, yet whose blowup factor, when reducing from a language in NTIME($t(n)$), is only quasilinear.

One such language is DE BRUIJN COLORING, first presented by Polishchuk and Spielman [37], based on a construction of [5]. First we will describe this language and state its completeness. Then we will arithmetize it and reduce it to ALGEBRAIC-CSP. The crucial observation in the arithmetization, given in Proposition 5.11, is that the de Bruijn graph can be embedded in an "affine" graph over a finite field (see Definition 5.9).

DE BRUIJN COLORING. Let $\sigma : \{0,1\}^k \to \{0,1\}^k$ be the cyclic permutation operator; i.e., for $w \in \{0,1\}^k, w = (w_1, \ldots, w_k)$ let $\sigma(w) = (w_k, w_1, \ldots, w_{k-1})$. Let $u \oplus v$ denote the bitwise xor of $u, v \in \{0,1\}^k$ and let $e_i \in \{0,1\}^k$ be the sequence that is zero on all but the ith coordinate, where it is one.

DEFINITION 5.5 (wrapped de Bruijn graph [39]). *The k-dimensional* wrapped de Bruijn graph *is the following directed graph $B_k = (V, E)$. Let m be the smallest power of 2 satisfying $m > 5k$. The vertex set is*

$$V = \{(w, i) : w \in \{0,1\}^k, i \in \{0, \ldots, m-1\}\}.$$

Each vertex $v = (w, i) : w \in \{0,1\}^k, i \in \{0, \ldots, m-1\}$, has two neighbors:

$$N_0(v) = (\sigma(w), (i + 1 \bmod m)), \quad N_1(v) = ((\sigma(w)) \oplus e_1, (i + 1 \bmod m)).$$

Remark 5.6. The definition in [39, section 4.3.2] is slightly different from the above; namely, it fixes $m = 5k + 1$. However, Theorem 5.8 holds for any $m > 5n$, as inspection of [39, section 4.3] reveals.

DEFINITION 5.7 (DE BRUIJN COLORING). *Let* $\hat{\Sigma} = \{0,1\}^4$. *The language* DE BRUIJN COLORING *has as its space of instances tuples of the form* $\psi = \{B_k, \hat{C}\}$, *where* $B_k = (V, E)$ *is a* k*-dimensional wrapped de Bruijn graph, and* $\hat{C} = \{\hat{C}_v : v \in V\}$ *is a set of constraints, where* $\hat{C}_v : \hat{\Sigma}^3 \to \{\texttt{accept}, \texttt{reject}\}$.

An instance is in the language DE BRUIJN COLORING *iff there exists an assignment* $\hat{A} : V \to \hat{\Sigma}$ *such that for all* $v \in V$ *we have* $\hat{C}_v(\hat{A}(v), \hat{A}(N_0(v)), \hat{A}(N_1(v))) = \texttt{accept}$.

THEOREM 5.8. DE BRUIJN COLORING *is NP-complete. Moreover, for any proper complexity function* $t : \mathbb{N}^+ \to \mathbb{N}^+$, *a language* $L \in \mathrm{NTIME}(t(n))$ *is reducible in time* $\mathrm{poly}\, t(n)$ *to an instance* $\psi = \{B_k, \hat{C}\}$ *of* DE BRUIJN COLORING, *where* $k = \lceil \log(t(n) \cdot O(\log^2 t(n))) \rceil$.

Proof. $L \in \mathrm{NTIME}(t(n))$ is reducible in time $\mathrm{poly}\, t(n)$ to an instance of CKTSAT of size $O(t(n) \log t(n))$ [29, 15]. This instance is reducible in time $\mathrm{poly}\, t(n)$ to an instance of DE BRUIJN COLORING of size $t(n)\,\mathrm{polylog}\, t(n)$ [37]. (See [39, section 4.3] for details.) ∎

To arithmetize an instance of DE BRUIJN COLORING we embed B_k in an *affine graph* as defined below. Recall an injective graph homomorphism of G to H is an injective mapping $f : V(G) \to V(H)$ such that if $(u, v) \in E(G)$ then $(f(u), f(v)) \in E(H)$. Further recall an affine map $\mathrm{AFF} : \mathbb{F} \to \mathbb{F}$ is of the form $\mathrm{AFF}(z) = az + b$ for $a, b \in \mathbb{F}$.

DEFINITION 5.9 (affine graph). *Let* $\mathcal{A}$ *be a set of affine maps over a field* $\mathbb{F}$. *The* affine graph $G(\mathbb{F}, \mathcal{A})$ *over* $\mathbb{F}$, *generated by* $\mathcal{A}$, *is the directed graph over vertex set* $\mathbb{F}$, *where each vertex* $v \in \mathbb{F}$ *is connected to* $\mathrm{AFF}(v)$ *for all* $\mathrm{AFF} \in \mathcal{A}$. *Notice that the outdegree of this graph is at most* $|\mathcal{A}|$.

We will use the following elementary properties of primitive polynomials (see [32, section 3.1]).

PROPOSITION 5.10. *Let* $S(x)$ *be a primitive polynomial of degree* s *over* GF(2). *Then, denoting* $\xi_i = x^i (\mathrm{mod}\ S(x))$, *we have that* $\xi_1, \ldots, \xi_{2^s} = \xi_0$ *are distinct polynomials over* GF(2) *of degree less than* s.

We now define a graph homomorphism injecting B_k to an affine graph of outdegree eight over $\mathrm{GF}(2^\ell)$ for any $\ell > k + \log 5k + 2$. Briefly, a vertex $(w, i) \in B_k$ will be mapped to a polynomial $p_{(w,i)} \in \mathrm{GF}(2^\ell)$. We will show that the polynomials corresponding to $(w, i+1)$ and $(w \oplus e_i, i+1)$ can be obtained by applying two out of eight possible affine shifts to $p_{(w,i)}$. In what follows, addition and multiplication are in $\mathrm{GF}(2^\ell)$ and we identify $\{0,1\}$ with GF(2).

PROPOSITION 5.11. *Let* m *be the smallest power of* 2 *satisfying* $m > 5k$. *Let* $\mathrm{GF}(2^\ell) = \mathrm{GF}(2)[x]/q(x)$, *where* $q(x)$ *is an irreducible polynomial of degree* ℓ. *Let* $S(x)$ *be a primitive polynomial of degree* $s = \log m$ *(note that* s *is an integer), and let* ξ_i *be as defined in Proposition* 5.10. *For* $((w_1, \ldots, w_k), i), w_j \in \{0,1\}, i \in [m]$, *let*

$$g(w) = x^s \cdot \sum_{j=1}^{k} w_j x^j \ ; \quad h(i) = \xi_i \ ; \quad f(w,i) = g(w) + h(i). \tag{5.3}$$

Then, the mapping $f : V(B_k) \to \mathrm{GF}(2^\ell)$ *is an injective homomorphism of* B_k

into the affine graph $G(\mathrm{GF}(2^\ell), \mathcal{A})$*, where*

(5.4)
$$\mathcal{A} = \left\{ \textsc{Aff}_b(\alpha) \triangleq x \cdot \alpha + b_1 S(x) + b_2 x^{s+1} + b_3 x^{s+k+1} : b = (b_1, b_2, b_3) \in \{0,1\}^3 \right\}.$$

Proof. Our mapping is injective. Note that $\deg(h(i)) < s$ for all $i \in [m]$, whereas the minimal degree of a nonzero term of $g(w)$ is $s+1$. Thus, $f(w,i) = f(w',i')$ iff $g(w) = g(w')$ and $h(i) = h(i')$. The former happens by definition iff $w = w'$ and the latter happens iff $i = i'$ because Proposition 5.10 implies $\xi_i \neq \xi_{i'}$ for all $i \neq i' \in [m]$.

To prove f is a homomorphism, we need to show that if $((w,i),(w',(i+1 \bmod m)))$ is an edge of $B_{k,m}$ then $f(w',(i+1 \bmod m)) = \textsc{Aff}_b(f(w,i))$ for some $b \in \{0,1\}^3$. There are eight cases to consider. Recall w' is either $\sigma(w)$ or $(\sigma(w)) \oplus e_1$. Note that $\ell > k+s+1$ so for all $w \in \{0,1\}^k$ we have that $x \cdot g(w) \bmod (q(x))$ is equal to $x \cdot g(w)$ as polynomials over $\mathrm{GF}(2)$. By definition of g we get

$$g(\sigma(w)) = \begin{cases} x \cdot g(w), & w_k = 0, \\ x \cdot g(w) + x^{s+k+1} + x^{s+1}, & w_k = 1. \end{cases}$$

Similarly,

$$g((\sigma(w)) \oplus e_1) = \begin{cases} x \cdot g(w) + x^{s+1}, & w_k = 0, \\ x \cdot g(w) + x^{s+k+1}, & w_k = 1. \end{cases}$$

Finally, by definition of h we get

$$h(i+1 \bmod m) = \begin{cases} x \cdot h(i), & \deg(h(i)) < s-1, \\ x \cdot h(i) + S(x), & \deg(h(i)) = s-1. \end{cases}$$

Our claim follows from the definition of $\textsc{Aff}_b$ and the previous equations. ∎

Proof of Theorem 3.7. We prove Theorem 3.7 for $k = 10$ and $d = |\hat{\Sigma}| = 16$, where $\hat{\Sigma}$ is from Definition 5.7. By Theorem 5.8 it suffices to show a polynomial time reduction sending an instance $\psi = \{B_k, \hat{C}\}$ of de Bruijn Coloring to an instance of ALGEBRAIC-CSP over a field of size $2^k \cdot \operatorname{poly} k$. We reduce in time $\operatorname{poly} 2^\ell$ to an instance over $\mathrm{GF}(2^\ell)$ for any $\ell > k + (\log 5k) + 2$ and the reduced instance is of the form

$$\phi = \{\mathrm{GF}(2^\ell), \{\textsc{Aff}', \textsc{Aff}''\} \cup \mathcal{A}, H, C(x, y_0, y_1, z_{000}, \ldots, z_{111})\},$$

where $\textsc{Aff}'(x) = x$, $\textsc{Aff}''(x) = \zeta - x$ (for ζ to be defined later), and $\mathcal{A}$ is as in (5.4).

Embed $\hat{\Sigma}$ in $\mathrm{GF}(2^\ell)$ arbitrarily and associate `accept` with 0 and `reject` with 1. As in the proof of Theorem 5.4, we view the constraint $\hat{C}_v$ as a mapping from $\hat{\Sigma} \subset \mathrm{GF}(2^\ell)$ to $\{0,1\}$. Recall that Proposition 5.11 showed $V(B_k)$ can be embedded in $G(\mathrm{GF}(2^\ell), \mathcal{A})$ via the embedding f from (5.3). Notice that the outdegree of $G(\mathrm{GF}(2^\ell), \mathcal{A})$ is greater than the outdegree of B_k; thus when arithmetizing $\hat{C}_v$ we must take into account which of the eight neighbors of $f(v)$ in $G(\mathrm{GF}(2^\ell), \mathcal{A})$ are maps of the neighbors of v in B_k. Let $b_0(v), b_1(v) \in \{0,1\}^3$ denote the two relevant neighbors of $f(v)$ in $G(\mathrm{GF}(2^\ell), \mathcal{A})$ satisfying $f(N_0(v)) = \textsc{Aff}_{b_0(v)}(f(v))$ and $f(N_1(v)) = \textsc{Aff}_{b_1(v)}(f(v))$. Let $C_v(y, z_{b_0}, z_{b_1})$ be the trivariate polynomial of degree at most $|\hat{\Sigma}| - 1$ in each variable, agreeing with $\hat{C}_v$ on inputs in $\hat{\Sigma}^3$. Let $I = \{f(v) : v \in V\}$. Let $\zeta \in \mathrm{GF}(2^\ell)$ satisfy $(\zeta + I) \cap I = \emptyset$, where $\zeta + I = \{\zeta + \xi : \xi \in I\}$ and set $H = I \cup (\zeta + I)$. Such ζ exists because viewing elements of $\mathrm{GF}(2^\ell)$ as polynomials over $\mathrm{GF}(2)$ modulo an irreducible polynomial of degree ℓ, all elements in I

have degree at most $k+s$. Now, let $P_h(x)$ be the polynomial of degree $|H|-1$, that is, 1 when $x=h$ and 0 for all $x=h' \in H, h' \neq h$. Finally, let $P_{\hat{\Sigma}}(y)$ be the nonzero polynomial of degree $|\hat{\Sigma}|$ whose roots are precisely the elements of $\hat{\Sigma}$. We are ready to define the constraint polynomial of ϕ:

$$C(x, y_0, y_1, z_{000}, \ldots, z_{111}) = \sum_{v \in I} P_v(x) \cdot C_v(y_0, z_{b_0(v)}, z_{b_1(v)}) + \sum_{h \in H \setminus I} P_h(x) \cdot P_{\hat{\Sigma}}(y_1). \tag{5.5}$$

The second summand on the right-hand side checks that all vertices receive colors in $\hat{\Sigma}$ and the first summand checks that all coloring constraints are satisfied. The polynomials P_v, P_h are used to "bundle" all constraints into one polynomial.

Note that $\deg_x(C) \leq |H|-1$ and the degree in the remaining variables is at most $|\hat{\Sigma}|$. Thus, ϕ is a legal instance of ALGEBRAIC-CSP$_{k,d}$.

Completeness. Suppose $\psi \in$ de Bruijn Coloring and let $\hat{A} : V \to \hat{\Sigma}$ witness this. Let A be the polynomial of degree $\leq |V|-1$ satisfying $A(f(v)) = \hat{A}(v)$ for all $v \in V$. We claim A satisfies ϕ. We need to show for all $x \in H$

$$C\left(x, A(x), A(\zeta - x), A(\textsc{Aff}_{000}(x)), \ldots, A(\textsc{Aff}_{111}(x))\right) = 0.$$

As in the proof of Theorem 5.4, when $x \in H$, at most one summand of (5.5) may be nonzero, by definition of P_v. We split the proof into cases.

- $x \in I$: Let $v = x$. The summand to consider is $P_v(v) \cdot C_v(A(v), A(\textsc{Aff}_{b_0(v)}(v)), A(\textsc{Aff}_{b_1(v)}(v)))$, which vanishes because $\hat{C}(\hat{A}(v), \hat{A}(N_0(v)), \hat{A}(N_1(v)) = \texttt{accept}, f(N_0(v)) = \textsc{Aff}_{b_0(v)}(f(v))$, and $f(N_1(v)) = \textsc{Aff}_{b_1(v)}(f(v))$.
- $x \in H \setminus I$: The summand to consider is $P_v(x) \cdot P_{\hat{\Sigma}}(A(\zeta - x))$. By construction of H and selection of ζ we have $\zeta - x \in I$. By construction A takes on values in $\hat{\Sigma}$ on $\zeta - x$, so the summand vanishes.

Soundness. Suppose $\phi \in$ ALGEBRAIC-CSP$_{k,d}$ and let A witness this. Let $\hat{A} : I \to \mathrm{GF}(2^\ell)$ be the evaluation of A on I. First we claim that the range of $\hat{A}$ is $\hat{\Sigma}$. Indeed, assume $A(x) \notin \hat{\Sigma}$ for $x \in I$. Let $x' = \zeta + x$ and notice that $x' \in H \setminus I$. Then the second summand of (5.5) does not vanish on x'. Since all other summands vanish by construction of P_v, we reach a contradiction. We conclude that $\hat{A}$ is a legal assignment to ψ.

Next, we claim that $\hat{A}$ satisfies ψ. Consider the constraint $\hat{C}_v$. Equation (5.5) holds for v and $P_{v'}(v) = 0$ for all $v' \neq v, v' \in H$, implying $C_v(A(v), A(\textsc{Aff}_{b_0(v)}(v)), A(\textsc{Aff}_{b_1(v)}(v))) = 0$. Recall from the previous paragraph that $A(v), A(\textsc{Aff}_{b_0(v)}(v)), A(\textsc{Aff}_{b_1(v)}(v)) \in \hat{\Sigma}$. By construction of C_v we conclude that $\hat{C}_v(\hat{A}(v), \hat{A}(N_0(v)), \hat{A}(N_1(v))) = \texttt{accept}$. This completes our proof. ∎

5.3. Systematic reduction to PAIR-ALGEBRAIC-CSP. We now show how to modify the reduction of the previous section to apply it to pair languages and get a systematic reduction, thus proving Theorem 3.10.

Proof. Consider the sequence of reductions applied to an instance x of L and resulting in an instance ϕ of ALGEBRAIC-CSP. First, we reduce x to an instance C of CktSAT along the lines of [29, 15]. Inspection reveals that this reduction is systematic. Indeed, the implicit input y is embedded into the inputs of C, and C accepts only inputs $y \in L_x$.

In the next step we reduce C to an instance ψ of DE BRUIJN COLORING. Once again, inspection of this reduction shows it is systematic [39, section 4.3]. In particular, this latter reduction embeds all nodes of C including its inputs in the first layer of the wrapped de Bruijn graph and each input node is mapped to a unique vertex. By construction, a coloring of the resulting de Bruijn graph is legal only if the colors of the vertices corresponding to inputs form an assignment satisfying C. Similarly, any assignment satisfying C can be extended to a coloring satisfying the constraints of ψ.

Finally, consider the reduction from DE BRUIJN COLORING to ALGEBRAIC-CSP. Notice that each vertex of the de Bruijn graph is mapped to a distinct element of $\mathbb{F}$ (Proposition 5.11). Additionally, by construction we map the colors of the de Bruijn coloring problem to distinct elements of $\mathbb{F}$. By construction, ϕ is satisfied by A iff A is the low-degree extension of a coloring that satisfies ψ. We have seen that all steps of our reduction are systematic; hence so is their concatenation. This completes our proof. □

6. PCPPs for Reed–Solomon codes over fields of characteristic 2. In this section we give a PCPP-verifier for RS-codes when the field is of characteristic 2 and the set of evaluation points is a linear subspace of the field over GF(2), thereby proving Theorem 3.2 (restated below).

An overview of the proof appears in subsection 6.1. This is followed by a formal description of the proof of proximity and verifier in subsection 6.2 and the analysis of its basic properties in subsection 6.3. The analysis of the soundness follows in subsection 6.4. We conclude with a formal proof of Theorem 3.2 in subsection 6.5.

THEOREM 6.1 (Theorem 3.2, restated). *Let* PAIR-ADDITIVE-RS *be the restriction of the language* PAIR-RS *to pairs* $((\mathrm{GF}(2^\ell), L, d), p)$*, where* $\mathrm{GF}(2^\ell)$ *is the Galois field of size* $n = 2^\ell$ *(and characteristic* 2*) and* $L \subseteq \mathbb{F}$ *is* GF(2)*-linear. Then,*

$$\text{PAIR-ADDITIVE-RS} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n} \begin{bmatrix} \mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n), \\ \mathsf{query}\ O(1), \\ \mathsf{distance}\ \mathsf{Hamming}_{\mathrm{GF}(2^\ell)} \end{bmatrix}.$$

Remark 6.2. For simplicity, we first prove the theorem for the special case of degree $d = |L|/8 - 1$. Then we show in Proposition 6.14 that this implies that the theorem holds for all degrees.

6.1. Sketch of proof of Theorem 3.2. At a high level, we attempt a reduction from the task of testing a univariate polynomial to the task of testing a bivariate polynomial of significantly smaller degree. We then invoke an analysis of a "bivariate low-degree test" by Polishchuk and Spielman [37], which reduces the task of testing bivariate polynomials back to the task of testing univariate polynomials, of much smaller degree than the original. Recursing on this idea leads to the full test. We note that crucial to our obtaining short PCPPs is the evaluation of the bivariate polynomial on a carefully selected, algebraically structured, subset of points. This set is very different from the sets typically used in previous PCP constructions, e.g., in [5, 2, 17], which are product sets usually consisting of the whole field.

We start by considering the polynomial $P(z)$ of degree $< n/8$ evaluated on the linear space $L \subset \mathrm{GF}(2^\ell)$ of cardinality n and address the task of "testing" it. Our starting point is that for any polynomial $q(z)$ of degree $\approx \sqrt{n}$, we can define a bivariate polynomial $Q(x, y)$ of degree $\approx \sqrt{n}$ in each variable that "captures" all the information of P. Specifically, we can reconstruct P from Q using the identity $P(z) = Q(z, q(z))$.

PROPOSITION 6.3. *Given any pair of polynomials $P(z), q(z)$, there exists a unique bivariate polynomial $Q(x,y)$ with $\deg_x(Q) < \deg(q)$ and $\deg_y(Q) = \lfloor \deg(P)/\deg(q) \rfloor$, such that $P(z) = Q(z, q(z))$.*

Proof. We use division over the ring of bivariate polynomials $\mathbb{F}[z,y]$ (see [16] for more details). Fix the lexicographic ordering on terms where $z > y$; i.e., terms are ordered first by their degree in z and then by their degree in y. Divide $P(z)$ by $(y - q(z))$, obtaining

$$P(z) = Q'(z,y) \cdot (y - q(z)) + Q(z,y). \tag{6.1}$$

By the basic properties of division in this ring Q is uniquely defined, and $\deg_y(Q) = \lfloor \deg(P)/\deg(q) \rfloor$ and $\deg_z(Q) < \deg(q)$. To complete the proof set $y = q(z)$ and notice that the first summand on the right-hand side of (6.1) vanishes. □

The presentation of P of degree $\approx n$ as a bivariate polynomial Q of individual degree $\approx \sqrt{n}$ is useful, because testing of bivariate polynomials reduces to testing of univariate polynomials of roughly the same degree using well-known "low-degree tests" and their analysis. This leads us to the hope that Q might provide a good "proof" that P is of low degree. More to the point, to prove that a table of evaluations of P corresponds to the evaluations of a polynomial of low degree, the prover can provide a table of evaluations of a bivariate polynomial Q, prove that Q has degree $\sqrt{n}$ in each variable, and then prove that Q is consistent with the table of evaluations of P.

To completely describe the above approach, all we need to do is describe which set of points we will specify Q on so as to achieve both tasks: (i) verifying that Q has low degree, and (ii) that it is consistent with P. However, this leads to conflicting goals. To test that Q has low degree, using a bivariate verifier, we need to know its values on some subset $X \times Y$, where $X, Y \subseteq \mathrm{GF}(2^\ell)$. To make this efficient, we need to make $|X|, |Y| \approx \sqrt{n}$. On the other hand, to test its consistency with P, the natural approach is to ask for its values on the set

$$T = \{(z, q(z)) | z \in L\}.$$

Unfortunately the set T, which depends on the selection of $q(z)$, is far from being of the form $X \times Y$. For starters, the projection of T onto its first coordinate has cardinality n while we would like this projection to be of cardinality $O(\sqrt{n})$.

Our solution is to ask the prover to provide the evaluation on *both* sets of points. This leads to a problem of checking consistency between the two sets and to do so we pick $q(z)$ in a way that will ensure T is compatible with $X \times Y$. In particular, we choose $q(z)$ to be a special *linearized* polynomial as defined in [32, Chapter 3, section 4]. A polynomial $q(z)$ over $\mathrm{GF}(2^\ell)$ is said to be linearized if $q(x+y) = q(x) + q(y)$ for every $x, y \in \mathrm{GF}(2^\ell)$. A linearized polynomial defines a linear map over $\mathrm{GF}(2^\ell)$ and we abuse notation and use q to denote this map. For $S \subset \mathrm{GF}(2^\ell)$, let $q(S) = \{q(s) : s \in S\}$. The linearized polynomial we use and its useful properties are listed below.

PROPOSITION 6.4. *For L a linear subspace of $\mathrm{GF}(2^\ell)$ that is a direct sum of the linear spaces L_0, L_1, let*

$$q(z) = q_{L_0}(z) \triangleq \prod_{\alpha \in L_0} (z - \alpha).$$

- *The polynomial $q(z)$ is linearized.*
- *The kernel of (the linear map defined by) q is L_0.*
- *$q(L) = q(L_1)$ and $q(L_1)$ is a linear space of dimension $\dim(L_1)$.*

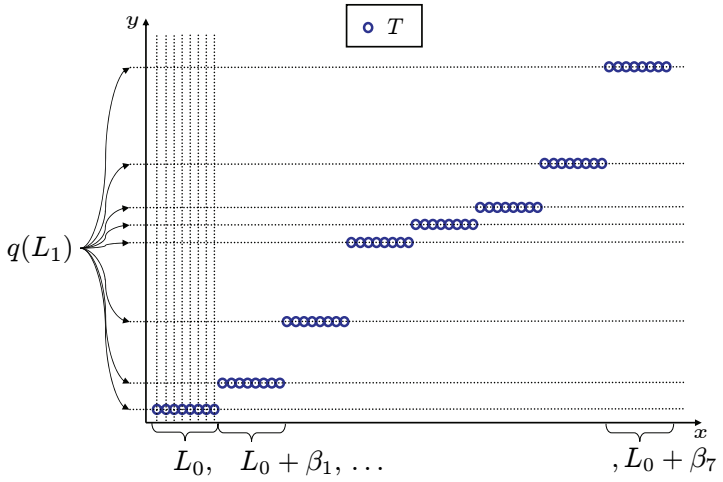


FIG. 1. *Here* $\mathbb{F} = \mathrm{GF}(2^6)$ *is the field with* 64 *elements and* q *is a linearized polynomial of degree* 8. *We plot the set of points* $T \subset \mathbb{F} \times \mathbb{F}$ *defined by* $T = \{(z, q(z)) : z \in \mathbb{F}\}$. *Notice* T *can be partitioned into eight product sets, each set being a product of an affine shift of* L_0 *and some* $\beta \in L_1$.

- q *is a one-to-one map from* L_1 *to* $q(L_1)$*; i.e., for* $\beta \neq \beta'$ *we get* $q(\beta) \neq q(\beta')$.
- q *is an* $|L_0|$ *to one map on* L*, where, for* $\beta \in L_1$*, the affine space* $L_0 + \beta \triangleq \{\alpha + \beta : \alpha \in L_0\}$ *is mapped to* $q(\beta)$.

Proof. The first part is proved by induction on the dimension of L. The base case (dimension zero) is easy, as $q_{L_0}(z) = z$ is clearly linearized. For the inductive step, let $L_0 = \mathrm{span}(\hat{L}, \alpha)$, where $\dim(\hat{L}) = k - 1$ and $\alpha \in \mathrm{GF}(2^\ell)$. Let $\hat{q}(z) = q_{\hat{L}}(z)$ be the linearized polynomial whose set of roots is $\hat{L}$. Clearly, $q_{L_0}(z) = \hat{q}(z) \cdot \hat{q}(\alpha + z)$ because addition and subtraction are the same in fields of characteristic 2. So

$$\begin{aligned} q_{L_0}(x+y) &= \hat{q}(x+y) \cdot \hat{q}(\alpha + x + y) = \hat{q}^2(x) + \hat{q}^2(y) + \hat{q}(\alpha)(\hat{q}(x) + \hat{q}(y)) \\ &= \hat{q}(x) \cdot \hat{q}(\alpha + x) + \hat{q}(y) \cdot \hat{q}(\alpha + y) = q_{L_0}(x) + q_{L_0}(y). \end{aligned}$$

We conclude that q_{L_0} is a linearized polynomial. The second part follows because $\deg(q) = |L_0|$ and the elements of L_0 are all roots of q.

The last three parts follow via basic linear algebra from our previous assertions that q defines a linear map with kernel L_0. ∎

With Proposition 6.4 in hand, we return to the task of providing a proof of proximity for the evaluation of a polynomial on the set of points L. Write L as the direct sum of L_0, L_1, with $\dim(L_0) = \lfloor \dim(L)/2 \rfloor$ and $\dim(L_1) = \lceil \dim(L)/2 \rceil$ (so $|L_0|, |L_1| \approx \sqrt{|L|}$), and take $q(z) = q_{L_0}(z)$ as described above. The last part of Proposition 6.4 implies that q partitions T into the disjoint union of $|L_1|$ lines, where each line is a product of a set of size $\approx \sqrt{|L|}$ with a singleton set (see Figure 1):

$$T = \bigcup_{\beta \in L_1} \{\{L_0 + \beta\} \times \{q(\beta)\}\}.$$

This suggests requesting the evaluation of Q on the set of points $(L_0 \times q(L_1)) \cup T$, the cardinality of which is $\leq 2n$. With such an evaluation in hand we can use the

subset $L_0 \times q(L_1)$ to perform a bivariate low-degree test, by testing proximity to the RS-code of degree $\approx \sqrt{n}$ of a random row/column of this product set. The consistency of Q's evaluation on the product set $L_0 \times q(L_1)$ and on the set T can also be addressed, by reading $Q(x, q(\beta))$ for all points $x \in L_0 \cup (L_0 + \beta)$ for $\beta \in L_1$. This consistency is precisely what is needed to connect the evaluation of P on the set L, that is isomorphic to T, to the evaluation of the bivariate Q on the product set $L_0 \times q(L_1)$. We have reduced our original problem of size n to $O(\sqrt{n})$ identical problems of size $O(\sqrt{n})$.

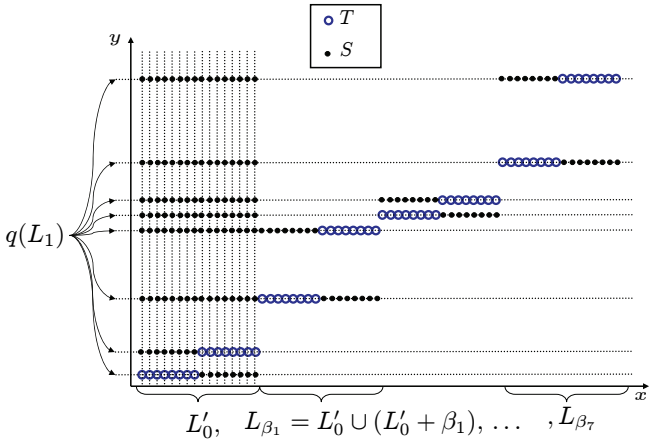


FIG. 2. *The proof of proximity for P is the evaluation of Q on the set of points denoted S. Notice it has a large subset that is the product set $L'_0 \times q(L_1)$, allowing for bivariate low degree testing. Additionally, $S \cup T$ can be partitioned into eight rows and each row is a linear space.*

Our description so far leads to a proof of proximity of size $O(n)$ that can be tested by making $O(\sqrt{n})$ queries. However, the *robustness* of our tests can be used to decrease the query complexity further, at the price of increasing the proof length. Informally, robustness means the following. If a function $f : (L_0 \times q(L_1)) \cup T \to \mathrm{GF}(2^\ell)$ is δ-far from being a low-degree bivariate polynomial, then the expected distance of a random row/column of f from a low-degree univariate polynomial is $\Omega(\delta)$. To apply recursion, notice that all of our tests verify proximity to Reed–Solomon codewords evaluated on *linear subspaces* of $\mathrm{GF}(2^\ell)$. To see this notice that L_0 and $q(L_1)$ are linear spaces, and so is $L_0 \cup (L_0 + \beta) = \mathrm{span}(L_0, \beta)$. Using recursion we conclude that to test proximity to the RS-code of size n it suffices to test proximity to RS-codes of size $\approx \sqrt{n}$, which can be done by testing proximity to the RS-code of size $\approx n^{1/4}$, etc. Applying this recursion a $\log \log n$ number of times reduces the degree to a constant and gives us our proofs of length $n \cdot \mathrm{polylog}\, n$.

From intuition to proof. Our rigorous analysis follows the intuition above, with one technical difference regarding the degree of the bivariate polynomial Q. To use the bivariate low-degree test on Q, we need its evaluation on a product set of points $X \times Y$, where $|X| \gg \deg_x(Q)$ and $|Y| \gg \deg_y(Q)$. In our case Proposition 6.3 gives us only $|X| > \deg_x(Q)$. As to y, we get $|Y| > 8 \deg_y(Q)$, which is sufficient. So we need to enlarge X. This is done by taking a linear space $L'_0 \supset L_0$ of dimension $\dim(L_0) + 2$ and asking for the evaluation of Q on $(L'_0 \times q(L_1)) \cup T$. This causes a new problem, because $L'_0 \cup (L_0 + \beta)$ is not a linear space, as $\dim(L'_0) > \dim(L_0)$. This

problem is fixed by asking for the evaluation of Q on the linear space $L'_0 \cup (L'_0 + \beta)$. The resulting set of points is described in Figure 2.

6.2. The RS proof of proximity and its associated verifier. First we define the structure of the proof of proximity for $\mathrm{RS}(\mathrm{GF}(2^\ell), L, d)$ and then describe the verifier's operation. As explained in the previous section, the proof for a purported low-degree polynomial $p : L \to \mathrm{GF}(2^\ell)$ is an evaluation of a low-degree bivariate polynomial related to p on a carefully chosen subset of $\mathrm{GF}(2^\ell) \times \mathrm{GF}(2^\ell)$, concatenated with a sequence of subproofs for RS-codes of smaller size. To formally define the proof of proximity we use the following notation throughout this section.

Given basis $(b_1, \ldots, b_k)$ for L, let

$$L_0 \triangleq \mathrm{span}(b_1, \ldots, b_{\lfloor k/2 \rfloor}); \quad L'_0 \triangleq \mathrm{span}(b_1, \ldots, b_{\lfloor k/2 \rfloor + 2}); \quad L_1 \triangleq \mathrm{span}(b_{\lfloor k/2 \rfloor + 1}, \ldots, b_k). \tag{6.2}$$

Fix $q(x) \triangleq q_{L_0}(x)$. Notice that $L'_0 \cap L_1 = \mathrm{span}(b_{\lfloor k/2 \rfloor + 1}, b_{\lfloor k/2 \rfloor + 2})$, and, in particular, this intersection is nonempty. For $\beta \in L_1$ let

$$L_\beta \triangleq \begin{cases} \mathrm{span}(L'_0, b_{\lfloor k/2 \rfloor + 3}), & \beta \in L'_0, \\ \mathrm{span}(L'_0, \beta) & \text{otherwise.} \end{cases} \tag{6.3}$$

A *partial* bivariate function f over $\mathrm{GF}(2^\ell)$ is a function with a partial domain $f : S \to \mathrm{GF}(2^\ell)$, where $S \subset \mathrm{GF}(2^\ell) \times \mathrm{GF}(2^\ell)$. The β-row of S is the set $R_\beta = \{\alpha : (\alpha, \beta) \in S\}$ (this set might be empty). The *restriction* of f to the β-row is the univariate function $f|^{\leftrightarrow}_\beta : R_\beta \to \mathrm{GF}(2^\ell)$ that agrees with f on its inputs, i.e., $f|^{\leftrightarrow}_\beta(\alpha) = f(\alpha, \beta)$. Similarly, the α-column of S is $C_\alpha = \{\beta : (\alpha, \beta) \in S\}$, and the restriction of f to it is $f|^{\updownarrow}_\alpha : C_\alpha \to \mathrm{GF}(2^\ell)$ defined by $f|^{\updownarrow}_\alpha(\beta) = f(\alpha, \beta)$.

Definition 6.5 (Reed–Solomon proof of proximity). *The proof of proximity for a purported codeword of the Reed–Solomon code* $\mathrm{RS}(\mathrm{GF}(2^\ell), L, n/8 - 1)$ *is defined by induction on* $k = \dim(L)$. *If* $k \le 6$ *then it is empty. Otherwise, the proof is a pair* $\pi = \{f, \Pi\}$, *where* f *is a partial bivariate function over partial domain* $S \subset \mathrm{GF}(2^\ell) \times \mathrm{GF}(2^\ell)$ *defined next and* Π *is a sequence of proofs of proximity for RS-codes over (smaller) linear spaces.*

Partial domain. *Let*

$$T \triangleq \bigcup_{\beta \in L_1} \{\{L_0 + \beta\} \times \{q(\beta)\}\}; \qquad S \triangleq \left(\bigcup_{\beta \in L_1} \{L_\beta \times \{q(\beta)\}\} \right) \setminus T. \tag{6.4}$$

Auxiliary proofs. *For each* $\beta \in L_1$, *the sequence of proofs* Π *has a unique subproof for an RS-codeword over* L_β *of degree* $|L_\beta|/8 - 1$, *denoted* $\pi^{\leftrightarrow}_\beta$. *For each* $\alpha \in L'_0$, *the sequence* Π *includes a unique subproof for an RS-codeword over* $q(L_1)$ *of degree* $|q(L_1)|/8 - 1$, *denoted* $\pi^{\updownarrow}_\alpha$. *Formally,*

$$\Pi \triangleq \{\pi^{\leftrightarrow}_\beta : \beta \in L_1\} \cup \{\pi^{\updownarrow}_\alpha : \alpha \in L'_0\}.$$

The next proposition shows that $S \cup T$ can be decomposed into rows and columns that are linear spaces (of size $\approx \sqrt{|L|}$). This gives some explanation of our peculiar choice of the set S as described in the previous section and shown in Figure 2.

Proposition 6.6. *The set* $S \cup T$ *is the disjoint union of* $q(\beta)$*-rows for* $\beta \in L_1$. *The* $q(\beta)$*-row of* $S \cup T$ *is the linear space* L_β. *Similarly, for every* $\alpha \in L'_0$, *the* α*-column of* $S \cup T$ *is the linear space* $q(L_1)$.

Proof. By construction of S, to prove the claim about the rows of $S \cup T$ it suffices to show that the $q(\beta)$-row of T is a subset of L_β. By the last part of Proposition 6.4 this row is

$$\{\gamma \in L : q(\gamma) = q(\beta)\} = q^{(-1)}(q(\beta)) \cap L = L_0 + \beta \subset L_\beta.$$

The inclusion above follows by definition from (6.3). This completes the proof of the claim about the rows.

Now consider the α-column of $S \cup T$ for $\alpha \in L_0'$. By (6.3) we have $L_0' \subset L_\beta$ for every $\beta \in L_1$ and $L_\beta \times \{q(\beta)\} \subset S$, so $(\alpha, q(\beta)) \in S$ implying $q(L_1)$ is a subset of the α-column. However, by the first part of our proposition, the only nonempty rows of $S \cup T$ are the $q(\beta)$-rows. So we conclude that the α-column of $S \cup T$ is precisely $q(L_1)$. ∎

DEFINITION 6.7 (RS-verifier). *The verifier for proximity to* $\mathrm{RS}(\mathrm{GF}(2^\ell), L, d = |L|/8 - 1)$ *is denoted* $\mathrm{V_{RS}}^{(p,\pi)}(\mathrm{GF}(2^\ell), L, d)$. *It receives as explicit inputs the description of the field* $\mathrm{GF}(2^\ell)$*, a basis* $(b_1, \ldots, b_k)$ *for* L*, and the degree parameter* $d = |L|/8 - 1$. *The implicit input of the verifier is the purported codeword* $p : L \to \mathrm{GF}(2^\ell)$ *and its purported proof is* $\pi = \{f, \Pi\}$ *as described in Definition* 6.5. *The verifier operates as follows.*

Base case ($|L| \le 64$). *The verifier reads* p *in entirety and accepts iff* $p \in \mathrm{RS}(\mathrm{GF}(2^\ell), L, |L|/8 - 1)$.

Recursion ($|L| > 64$). *Let* $\hat{p} : T \to \mathrm{GF}(2^\ell)$ *be the partial bivariate function corresponding to* p*,*

$$\hat{p}(\gamma, q(\gamma)) = p(\gamma) \text{ for } \gamma \in L. \tag{6.5}$$

Notice that $\hat{p}$ *is well defined because the mapping* $\gamma \mapsto (\gamma, q(\gamma))$ *is a bijection from* L *to* T. *Let*

$$\hat{f} : S \cup T \to \mathrm{GF}(2^\ell) \tag{6.6}$$

be the function that agrees with f *on* S *and with* $\hat{p}$ *on* T. *Notice that* $\hat{f}$ *is well defined because* $S \cap T = \emptyset$. *The verifier sets* $L_0 = \mathrm{span}(b_1, \ldots, b_{\lfloor k/2 \rfloor})$*, computes the coefficients of the polynomial* $q(x) = q_{L_0}(x)$*, and performs one of the following two tests with probability half each.*

Row test. *Pick* $\beta \in L_1$ *at random. Let* L_β *be as in* (6.3). *Invoke*

$$\mathrm{V_{RS}}^{(\hat{f}|^{\rightarrow}_{q(\beta)}, \pi^{\rightarrow}_{\beta})}(\mathrm{GF}(2^\ell), L_\beta, |L_\beta|/8 - 1).$$

Column test. *Pick* $\alpha \in L_0'$ *at random. Let* L_1 *be as in* (6.2). *Compute a basis for* $q(L_1)$ *and invoke*

$$\mathrm{V_{RS}}^{(\hat{f}|^{\updownarrow}_{\alpha}, \pi^{\updownarrow}_{\alpha})}(\mathrm{GF}(2^\ell), q(L_1), |q(L_1)|/8 - 1).$$

Remark 6.8. The "inner" verifiers, i.e., the row and column tests, restrict their attention to special subsets of p and π. To simplify our analysis, we assume these special subsets are copied to an "inner oracle" before invocation of an inner test. This assumption can be made without loss of generality because the verifier is nonadaptive; i.e., its operation does not depend on the implicit input given to it. Furthermore, the indices of the queries needed at the bottom of the recursion can be computed efficiently given the random coins used through the recursion as can be verified by inspection of the proof of Proposition 6.9.

The following subsections analyze the performance of V_{RS}. Specifically, the next subsection analyzes the simple properties including the running time, query complexity, randomness/size complexity, and the completeness. The soundness analysis is addressed in subsection 6.4.

6.3. Basic properties.

PROPOSITION 6.9. $\mathrm{V}_{\mathrm{RS}}{}^{(p,\pi)}(\mathrm{GF}(2^\ell), L, |L|/8-1)$ *makes at most* 64 *queries into* p *and* π. *It tosses at most* $k + O(\log k)$ *random coins (recall that* $k = \dim(L)$*) and runs in time* $\operatorname{poly} \ell$. *The size of the proof* π *accessed by* $\mathrm{V}_{\mathrm{RS}}{}^{(p,\pi)}(\mathrm{GF}(2^\ell), L, |L|/8-1)$ *is* $2^k \cdot \operatorname{poly} k = |L| \cdot \operatorname{polylog} |L|$.

Proof. The query complexity is easy to verify. In the base case, the verifier reads 64 field elements. In the inductive case the verifier invokes V_{RS} which by induction makes 64 queries.

Regarding randomness complexity, in the base case the verifier tosses zero coins. In the inductive case, the verifier tosses one coin to determine which test to perform—row or column. It then tosses $k/2 + O(1)$ coins to determine the inner call and then $(k/2 + O(1)) + O(\log(k/2 + O(1)))$ coins in the recursive call. Adding up, we get a total of $k + O(\log k)$ coins. The size of the proof can be similarly analyzed or bounded by $2^{\text{randomness}}$ to get the same bound.

We now analyze the running time, which is the sum of two processes.

The preprocessing time. This is the time required by the *outer* verifier $\mathrm{V}_{\mathrm{RS}}{}^{(p,\pi)}[\mathrm{GF}(2^\ell), L, |L|/8-1]$ to prepare the explicit input for invoking an *inner* verifier on a row/column. Notice that $q(x)$ can be computed and evaluated in polynomial time in $|L_0|$ and ℓ and so can the bases for L_0, L_0', L_β, L_1, and $q(L_1)$. Thus, the preprocessing time is polynomial.

The index translation time. Suppose the outer verifier conducts an inner row test of the form

$$\mathrm{V}_{\mathrm{RS}}{}^{(\hat{f}|^{\leftrightarrow}_{q(\beta)}, \pi^{\leftrightarrow}_{\beta})}[\mathrm{GF}(2^\ell), q(L_\beta), |L_\beta|/8-1].$$

The case of a column test is analogous. A query to $\hat{f}|^{\leftrightarrow}_{q(\beta)}$ by the inner verifier is indexed by an element $\alpha \in L_\beta$. However, this query needs to be *translated* to a query to $\hat{f}$, which is a pair $(\alpha, \beta) \in S \cup T$. This translation is easily seen to be efficient given α and β. Furthermore, translating a query to $\hat{f}$ into a query to $f : S \to \mathbb{F}$ or $p : T \to \mathbb{F}$ is also easy. If $\beta = q(\alpha)$ we query $p(\alpha)$ because $(\alpha, \beta) \in T$, and otherwise we query $f(\alpha, \beta)$. This translation involves evaluating $q(\alpha)$, which can be done efficiently as argued above.

We conclude that for each level of the recursion, the running time of the preprocessing and index translation is at most polynomial in $|L|$ and ℓ. Since there are $O(\log \ell)$ levels of recursion, we conclude that the running time is as stated, completing our proof. □

Next we move to the completeness part of the proof.

PROPOSITION 6.10 (perfect completeness). *If* p *is the evaluation of a polynomial* P *of degree* $< |L|/8$, *then there exists a proof that causes the RS-verifier to accept with probability one.*

This part is straightforward given the intuition developed in the proof sketch of Theorem 3.2. If p is indeed low-degree, then there exists a proper low-degree bivariate polynomial Q that is consistent with it on all rows. Looking at Figure 2 we argue that S is a union of linear spaces and the restriction of Q to each row is low-degree and consistent with p. The formal proof follows.

Proof. To prove the proposition inductively, it suffices to construct $f : S \to \mathrm{GF}(2^\ell)$ so that the function $\hat{f} : S \cup T \to \mathrm{GF}(2^\ell)$ is such that for every $\beta \in L_1$, the $q(\beta)$-row of $\hat{f}$ is a codeword of $\mathrm{RS}(\mathrm{GF}(2^\ell), L_\beta, |L_\beta|/8 - 1)$, and for every $\alpha \in L_0'$, the α-column of $\hat{f}$ is a codeword of $\mathrm{RS}(\mathrm{GF}(2^\ell), L_1, |L_1|/8 - 1)$.

Using Proposition 6.3 we get $P(x) = Q(x, q(x))$ for $q(x) = q_{L_0}(x)$, where

$$\deg_x(Q) < |L_0| \quad \text{and} \quad \deg_y(Q) = \lfloor \deg(P)/\deg(q) \rfloor < (|L|/8)/|L_0| = |L_1|/8. \tag{6.7}$$

Set $f(\alpha, \beta') = Q(\alpha, \beta')$ for every $(\alpha, \beta') \in S$. If $(\alpha, \beta') \in T$ we have $\beta' = q(\alpha)$, so

$$\hat{p}(\alpha, \beta') = \hat{p}(\alpha, q(\alpha)) = p(\alpha) = P(\alpha) = Q(\alpha, q(\alpha)) = Q(\alpha, \beta').$$

Thus, $\hat{f}$ is the evaluation of Q on $S \cup T$. Consider the $q(\beta)$-row of $\hat{f}$ for $\beta \in L_1$. By Proposition 6.6 the $q(\beta)$-row of $S \cup T$ is L_β. By (6.3) we have $|L_\beta| = 8 \cdot |L_0|$ because $\dim(L_\beta) = \dim(L_0) + 3$. By (6.7) we have $\deg(Q(x, q(\beta)) \le \deg_x(Q) < |L_0|$. We conclude that the $q(\beta)$-row of $\hat{f}$ is indeed a member of $\mathrm{RS}(\mathrm{GF}(2^\ell), L_\beta, |L_\beta|/8 - 1)$.

Similarly, by Proposition 6.6 the α-column of $\hat{f}$ is $q(L_1)$. By Proposition 6.4 $\dim(q(L_1)) = \dim(L_1)$, so $|q(L_1)| = |L_1|$. By construction the α-column of $\hat{f}$ is the evaluation of $Q(\alpha, y)$ on $q(L_1)$. Equation (6.7) completes our proof, because $\deg(Q(\alpha, y)) \le \deg_y(Q) < |L_1|/8$. ∎

6.4. Soundness. Our analysis of the soundness is by induction. Assume $\mathrm{V_{RS}}$ accepts implicit input p and proof $\pi = \{f, \Pi\}$ with high probability. Let $\hat{p}, \hat{f}$ be the partial bivariate functions as defined in (6.5) and (6.6), respectively. We argue by induction that for most $\alpha \in L_0'$ and $\beta \in L_1$, the α-column and $q(\beta)$-row of $\hat{f}$ are close to polynomials of degree roughly $\sqrt{|L|}$. The analysis of Polishchuk and Spielman implies that $\hat{f}$ restricted to the product set $L_0' \times q(L_1)$ is very close to some low-degree bivariate polynomial. Then we claim that $\hat{p}$ is close to an evaluation of the same polynomial on the set of points T. This implies p is close to a degree-$|L|/8$ univariate polynomial, completing the analysis. Formally, we have the following.

LEMMA 6.11 (soundness). *There exists constant $c \ge 1$ such that for every integer k and ϵ, if*

$$\Pr[\mathrm{V_{RS}}^{(p,\pi)}(\mathrm{GF}(2^\ell), \mathrm{span}(b_1, \ldots, b_k), 2^k/8 - 1) = \mathsf{reject}] \le \epsilon,$$

then p is $\left(c^{\log k} \cdot \epsilon\right)$-close to $\mathrm{RS}(\mathrm{GF}(2^\ell), \mathrm{span}(b_1, \ldots, b_k), 2^k/8 - 1)$.

To prove the lemma, we need a version of the analysis of Polishchuk and Spielman of the bivariate test. The following lemma is directly implied by the main theorem in [37]. We defer its proof to section 6.6 below.

DEFINITION 6.12. *For set $S \subseteq \mathbb{F} \times \mathbb{F}$, partial bivariate function $f : S \to \mathbb{F}$, and nonnegative integers d_1, d_2, define $\delta^{(d_1,d_2)}(f)$ to be the fractional distance of f from a polynomial of degree d_1 in its first variable and d_2 in its second variable. Formally,*

$$\delta^{(d_1,d_2)}(f) \triangleq \min_{\{Q : S \to \mathbb{F} \mid \deg_x(Q) \le d_1, \deg_y(Q) \le d_2\}} \{\delta(f, Q)\}.$$

Let $\delta^{(d,*)}(f)$ and $\delta^{(*,d)}(f)$ denote the fractional distances when the degree in one of the variables is unrestricted.

LEMMA 6.13 (bivariate test on product set [37]). *There exists a universal constant $c_0 \ge 1$ such that the following holds. For every $A, B \subseteq \mathbb{F}$ and integers $d_1 \le |A|/4, d_2 \le |B|/8$ and function $f : A \times B \to \mathbb{F}$, it is the case that*

$$\delta^{(d_1,d_2)}(f) \le c_0 \cdot \left(\delta^{(d_1,*)}(f) + \delta^{(*,d_2)}(f)\right).$$

Proof of Lemma 6.11. The proof is by induction on k. Let $L = \mathrm{span}(b_1, \ldots, b_k)$ and let L_0, L_0', L_1, q be as defined in the beginning of subsection 6.2. We use the following constants, where $\hat{c}$ is a parameter to be minimized and c_0 is the universal constant from Lemma 6.13:

$$c_1 \triangleq \hat{c}^{\log(7/6)}/2; \quad c_2 \triangleq \frac{c_1}{3c_0}; \quad c_3 \triangleq \frac{3c_1}{16(3c_0+2)}.$$

We fix c to be the minimal $\hat{c}$ such that $c_3 \geq 2$ and $\frac{1}{c_3} + \frac{16}{c_1} \leq 1$. Notice that c_1, c_2, c_3 are strictly increasing functions of $\hat{c}$, so c is well defined (we do not attempt to minimize it).

The base case $k \leq 6$ is immediate. For the inductive case we assume that the lemma is true by induction for smaller dimension k' and, in particular, for the recursive calls of the RS-verifier, and we now prove it for dimension $k \geq 7$.

Let $\pi = (f, \Pi)$ be as in Definition 6.5 and assume that (p, π) is rejected by the verifier with probability at most ϵ. We assume without loss of generality that $\epsilon \leq c^{-\log k}$, for otherwise there is nothing to prove. We show below that p is within distance $c^{\log k} \cdot \epsilon$ of some RS-codeword. In what follows let $\hat{p}, \hat{f}$ be the partial bivariate functions defined in (6.5) and (6.6), respectively.

Step 1. *Restricting the bivariate function $\hat{f}$ to a product set $L_0' \times q(L_1)$.* Denote by $\epsilon(\alpha)$ the probability that the inner verifier rejects $\hat{f}|_\alpha^\updownarrow$ (and its proof), and similarly let $\epsilon(\beta)$ be the probability verifier rejects $\hat{f}|_{q(\beta)}^\leftrightarrow$. Let ϵ_{col} be the expectation of $\epsilon(\alpha)$ over random $\alpha \in L_0'$ and let ϵ_{row} be the similar expectation of $\epsilon(\beta)$ over random $\beta \in L_1$. By definition of the verifier, we have $\epsilon = \frac{1}{2}(\epsilon_{\mathrm{row}} + \epsilon_{\mathrm{col}})$. Since these quantities are nonnegative we get $\epsilon_{\mathrm{row}}, \epsilon_{\mathrm{col}} \leq 2\epsilon$.

Let $d_1 = |L_0| - 1$ and recall that $|L_\beta| = 8|L_0|$ for every $\beta \in L_1$. This follows from (6.2) and (6.3). First we bound $\delta^{(d_1,*)}(\hat{f})$. This quantity is the expectation over random $\beta \in L_1$ of the fractional distance of $\hat{f}|_{q(\beta)}^\leftrightarrow$ from a degree-d_1 univariate polynomial. Let $\delta^{(d_1)}(\hat{f}|_{q(\beta)}^\leftrightarrow)$ denote this distance. We get

$$\begin{aligned} \delta^{(d_1,*)}(\hat{f}) &= \mathbb{E}_{\beta \in L_1}\left[\delta^{(d_1)}(\hat{f}|_{q(\beta)}^\leftrightarrow)\right] \leq \mathbb{E}_{\beta \in L_1}\left[\epsilon(\beta) \cdot c^{\log(\dim(L_\beta))}\right] \\ &\leq \epsilon_{\mathrm{row}} \cdot c^{\log(\lfloor k/2 \rfloor + 3)} \leq 2\epsilon \cdot c^{\log \frac{6k}{7}} = \frac{\epsilon}{c_1} \cdot c^{\log k}. \end{aligned} \tag{6.8}$$

The first inequality follows by induction, the second follows because $\dim(L_\beta) = \lfloor k/2 \rfloor + 3$ for every $\beta \in L_1$, the third holds for $k \geq 7$, and the last equality is true for our setting of c_1.

Let f' be the restriction of $\hat{f}$ to $L_0' \times q(L_1)$; i.e., $f' : L_0' \times q(L_1) \to \mathrm{GF}(2^\ell)$ is the function that agrees with $\hat{f}$ on its domain. Since $|L_0'| = |L_\beta|/2$ we get from (6.8)

$$\delta^{(d_1,*)}(f') \leq \frac{2\epsilon}{c_1} \cdot c^{\log k}. \tag{6.9}$$

Let $d_2 = |L_1|/8 - 1$. By analogy to (6.8), we get by induction

$$\delta^{(*,d_2)}(f') \leq \epsilon_{\mathrm{col}} \cdot c^{\log(\lfloor k/2 \rfloor + 1)} \leq \frac{\epsilon}{c_1} \cdot c^{\log k}. \tag{6.10}$$

The conditions of Lemma 6.13 hold with respect to f' and $A = L_0', B = q(L_1)$, because $d_1 \leq |L_0'|/4$ and $d_2 \leq |q(L_1)|/8$. Thus, from (6.9), (6.10), Lemma 6.13, and

our setting of c_2, we conclude that f' is "close" to an evaluation of a low degree bivariate polynomial:

$$\delta^{(d_1,d_2)}(f') \leq \frac{\epsilon}{c_2} \cdot c^{\log k}. \tag{6.11}$$

Step 2. *Extending the analysis to the bivariate function* $\hat{p}$. Let Q be the degree-(d_1, d_2) polynomial closest to f'. We wish to bound the probability over random $(\alpha, \tilde{\beta}) \in T$ that $\hat{p}(\alpha, \tilde{\beta}) \neq Q(\alpha, \tilde{\beta})$. Let $\tilde{\beta} = q(\beta)$ and notice that $\tilde{\beta} \in q(L_1)$. This follows from Proposition 6.4. Call $\tilde{\beta}$ *good* if the polynomial closest to $\hat{f}|^{\leftrightarrow}_{\tilde{\beta}}$ is $Q(x, \tilde{\beta})$, and otherwise it is *bad*. We bound the probability as follows:

$$\begin{aligned}\Pr_{(\alpha,\tilde{\beta})\in T}\left[\hat{p}(\alpha,\tilde{\beta}) \neq Q(\alpha,\tilde{\beta})\right] \leq \Pr_{\tilde{\beta}\in q(L_1)}\left[\tilde{\beta} \text{ is bad }\right] \\ + \Pr_{(\alpha,\tilde{\beta})\in T}\left[\hat{p}(\alpha,\tilde{\beta}) \neq Q(\alpha,\tilde{\beta}) | \tilde{\beta} \text{ is good}\right].\end{aligned} \tag{6.12}$$

- *First summand of* (6.12). We start by bounding the probability of bad $\tilde{\beta}$. Let $\hat{f}|^{\leftrightarrow}_{\tilde{\beta}}{}'$ be the restriction of $\hat{f}|^{\leftrightarrow}_{\tilde{\beta}}$ to domain L_0' and let $Q_{\tilde{\beta}}(x)$ be the degree-d_1 polynomial closest to $\hat{f}|^{\leftrightarrow}_{\tilde{\beta}}$. If $\tilde{\beta}$ is bad, i.e., $Q_{\tilde{\beta}}(x) \neq Q(x, \tilde{\beta})$, then $\hat{f}|^{\leftrightarrow}_{\tilde{\beta}}{}'$ is either (3/8)-far from $Q(x, \tilde{\beta})$ or (3/8)-far from $Q_{\tilde{\beta}}(x)$. This is because $Q_{\tilde{\beta}}(x)$ and $Q(x, \tilde{\beta})$ can agree on at most $|L_0'|/4$ locations in L_0'. Thus, by (6.9) and (6.11), we get

  $$\Pr_{\tilde{\beta}\in q(L_1)}\left[\tilde{\beta} \text{ is bad }\right] \leq 2 \cdot \frac{8}{3} \cdot \max\left\{\frac{1}{c_2}, \frac{2}{c_1}\right\} \cdot \epsilon \cdot c^{\log k} \leq \frac{\epsilon}{c_3} \cdot c^{\log k}. \tag{6.13}$$

  The last inequality follows by bounding the maximum of two nonnegative numbers by their sum and holds for our setting of c_1, c_2, c_3.
- *Second summand of* (6.12). Let $T_{\text{good}} = \{(\alpha, \tilde{\beta}) \in T : \tilde{\beta} \text{ is good}\}$. Since $\hat{p}$ is a function on a subdomain of $\hat{f}$ we can bound the second summand in (6.12) as follows:

  $$\begin{aligned}\Pr_{(\alpha,\tilde{\beta})\in T}\left[\hat{p}(\alpha,\tilde{\beta}) \neq Q(\alpha,\tilde{\beta}) | \tilde{\beta} \text{ is good}\right] \leq \Pr_{(\alpha,\tilde{\beta})\in S}\left[\hat{f}(\alpha,\tilde{\beta}) \neq Q(\alpha,\tilde{\beta})\right] \\ \cdot \frac{|S|}{|T_{\text{good}}|}.\end{aligned} \tag{6.14}$$

  We already showed in (6.8) that $\Pr_S[\hat{f} \neq Q]$ is relatively small, so we need only to argue that $|T_{\text{good}}|$ is large relative to $|S|$. By Proposition 6.4 the $\tilde{\beta}$-row of T is an affine shift of L_0 by $\tilde{\beta}$. From the proof of the first part of Proposition 6.6 we conclude that the $\tilde{\beta}$-row of T is 1/8 fraction subset of the $\tilde{\beta}$-row of S, so (6.13) implies

  $$|T_{\text{good}}|/|S| = \frac{1}{8} \cdot (1 - \Pr[\tilde{\beta} \text{ is bad}]) \geq 1/16. \tag{6.15}$$

  The last inequality follows from (6.13) by our assumption that $\epsilon \cdot c^{\log k} \leq 1$ and because we set $c_3 \geq 2$.

Summing up from (6.13), (6.14), and (6.15) we get

$$\Pr_{(\alpha,\tilde{\beta})\in T}\left[\hat{p}(\alpha,\tilde{\beta}) \neq Q(\alpha,\tilde{\beta})\right] \leq \left(\frac{1}{c_3} + \frac{16}{c_1}\right) \epsilon \cdot c^{\log k} \leq \epsilon \cdot c^{\log k}. \tag{6.16}$$

The last inequality holds because we set c_1 and c_3 such that $\frac{1}{c_3} + \frac{16}{c_1} \leq 1$.

Step 3. *From bivariate $\hat{p}$ to univariate p.* Let $P(x) = Q(x, q(x))$. Notice that $\deg(P) \leq |L|/8 - 1$. This follows from the degree of Q and $\deg(q) = |L_0| = |L'_0|/4$. Using (6.16) and the definition $T = \{(\gamma, q(\gamma)) : \gamma \in L\}$ we conclude that for all but a $(\epsilon \cdot c^{\log k})$-fraction of L we have

$$p(\gamma) = \hat{p}(\gamma, q(\gamma)) = Q(\gamma, q(\gamma)) = P(\gamma).$$

The fractional distance of p from a degree-$|L|/8 - 1$ polynomial is as claimed, completing our proof. □

6.5. Proof of Theorem 3.2. In this subsection we complete the formal proof of Theorem 3.2. First consider the case of degree precisely $|L|/8 - 1$, dealt with in the preceding sections. In particular, the proof of proximity and its associated verifier are described in subsection 6.2. The query complexity, randomness, and proof length are argued in Proposition 6.9. Perfect completeness is asserted by Proposition 6.10. Soundness is analyzed in Lemma 6.11. This completes the proof of the special case. The following proposition, Proposition 6.14, generalizes the degree and completes the full proof of Theorem 3.2.

In what follows we say a soundness function $s : [0,1] \times \mathbb{N}^+ \to [0,1]$ is *monotone* if it increases with δ; i.e., for all n we have $\delta \geq \delta' \Rightarrow s(\delta, n) \geq s(\delta', n)$.

PROPOSITION 6.14. *Let L be either of the pair-languages* PAIR-ADDITIVE-RS *and* PAIR-SMOOTH-RS*, and let $L_{\frac{1}{8}}$ be the restriction of L to explicit pairs of the form $(\mathbb{F}, S, |S|/8 - 1)$. Suppose*

$$L_{\frac{1}{8}} \in \mathbf{Strong\text{-}PCPP}_{s(\delta,n)} \left[\begin{array}{l} \mathsf{randomness}\ r(n), \\ \mathsf{query}\ q(n), \\ \mathsf{distance\ Hamming}_{\mathbb{F}} \end{array} \right],$$

where $s(\delta, n)$ is monotone and $r(n) \geq \log n$. Then for $s'(\delta, n) = \min\{\delta/2, s(\delta/64, n)\}$,

$$L \in \mathbf{Strong\text{-}PCPP}_{s'(\delta,n)} \left[\begin{array}{l} \mathsf{randomness}\ r(n), \\ \mathsf{query}\ O(q(n)), \\ \mathsf{distance\ Hamming}_{\mathbb{F}} \end{array} \right].$$

Proof. Let (x, p) be an instance to L with explicit input $x = (\mathbb{F}, S, d')$. Denote $d = \frac{|S|}{8} - 1$ and let $V_{\frac{1}{8}}$ denote the verifier for $L_{\frac{1}{8}}$. We start with the case of $d' < d$. On explicit input $(\mathbb{F}, S, d' < d)$ the verifier expects a (concatenation of) two subproofs for $\mathrm{RS}(\mathbb{F}, S, d)$, denoted π_1, π_2. The verifier operates as follows:

- Toss $r(n)$ coins. Let $\mathcal{R}$ denote the random string.
- Invoke $V_{\frac{1}{8}}$ using randomness $\mathcal{R}$ on explicit input $(\mathbb{F}, S, d)$, implicit input p, and proof π_1.
- Fix $Q(z) \triangleq z^{d-d'}$ and set $p'(z) = p(z) \cdot Q(z)$. Invoke $V_{\frac{1}{8}}$ using randomness $\mathcal{R}$ on explicit input $(\mathbb{F}, S, d)$, implicit input p', and proof π_2.

Notice that querying $p'(\alpha)$ can be simulated by querying $p(\alpha)$ and multiplying the answer by $Q(\alpha)$. Additionally evaluating $Q(\alpha)$ can be done in time polylog $|\mathbb{F}|$. So the running time, query complexity, and randomness are essentially inherited from $V_{\frac{1}{8}}$. Completeness follows by observing that $\deg(p) \leq d'$ implies $\deg(p') \leq d$. As to soundness, there are two cases to consider. If p is $\delta/4$-far from $\mathrm{RS}(\mathbb{F}, S, d)$ then by assumption, the first subtest rejects with probability at least $s(\delta/4, n) \geq s(\delta/64, n)$ (the previous inequality follows from monotonicity). Otherwise, p is within relative distance $\delta/4 \leq 1/4$ of an evaluation of a polynomial P with $d' < \deg(P) \leq d$. In this

case, p' is 1/4-close to the evaluation of $P'(z) = Q(z) \cdot P(z)$, where $d < \deg(P') < |S|/4$. Thus, P' is 3/4-far from $\mathrm{RS}(\mathbb{F}, S, d)$, so the distance of p' from the same code is at least $1/2 > \delta/4$. We conclude that the second subtest rejects with probability $s(\delta/4, n) \geq s(\delta/64, n)$.

Next assume $d' > d$ and notice without loss of generality that $d' \leq 8(d+1)$ because otherwise every implicit input is a codeword. The key observation is that a polynomial $P(z)$ is of degree d' iff it can be written as a sum $P(z) = \sum_{i=0}^{7} z^{i(d+1)} \cdot P_i(z)$, where $\deg(P_i) = d_i \leq d$ can be uniquely and efficiently computed given d and d'. Let $L_{(\leq d)}$ be the restriction of L to instances of degree $\leq d$ and let $V_{(\leq d)}$ denote the verifier for $L_{(\leq d)}$. The proof for explicit input $(\mathbb{F}, S, d')$ consists of eight functions $p_0, \ldots p_7 : S \to \mathbb{F}$ and eight proofs of proximity to $L_{(\leq d)}$ denoted $\pi_0, \ldots, \pi_7$. On explicit input $(\mathbb{F}, S, d')$ and implicit input p, the verifier operates as follows:

- Toss $r(n)$ coins. Let $\mathcal{R}$ denote the random string.
- For $i = 0, \ldots, 7$, invoke $V_{(\leq d)}$ using randomness $\mathcal{R}$ on explicit input $(\mathbb{F}, S, d_i)$, implicit input p_i, and proof π_i.
- Using $\mathcal{R}$, select uniformly at random $\gamma \in S$. Accept iff $p(\gamma) = \sum_{i=0}^{7} \gamma^{i(d+1)} p_i(\gamma)$.

Proof length, randomness, completeness, running time, and query complexity follow from construction. As to soundness, assume that p is δ-far from $\mathrm{RS}(\mathbb{F}, S, d')$. There are two cases to consider. If $p(z)$ disagrees with $\sum_{i=0}^{7} z^{i(d+1)} p_i(z)$ on a $\delta/2$-fraction of $z \in S$, then the second subtest rejects with probability $\geq \delta/2$. Otherwise, $p(z)$ is $\delta/2$-close to $\sum_{i=0}^{7} z^{i(d+1)} p_i(z)$. In this case at least one p_i must be $\delta/16$-far from $\mathrm{RS}(\mathbb{F}, S, d_i)$. So the first part of this proof (for the case $d' < d$) implies the rejection probability is at least $s(\frac{\delta}{4 \cdot 16}, n)$. This completes our proof. ∎

6.6. Proof of Lemma 6.13. The lemma is an immediate corollary of the bivariate testing theorem of Polishchuk and Spielman [37, Theorem 9]. We use here the general version of it appearing in Spielman's thesis.

THEOREM 6.15 (see [39, Theorem 4.2.19]). *Let $\mathbb{F}$ be a field, $S, T \subseteq \mathbb{F}$. Let $R(x, y)$ be a polynomial over $\mathbb{F}$ of degree $(d, |T| - 1)$ and let $C(x, y)$ be a polynomial over $\mathbb{F}$ of degree $(|S| - 1, e)$. If*

$$\Pr_{(x,y) \in S \times T} [R(x, y) \neq C(x, y)] < \gamma^2 \quad \text{and} \quad 2\left(\frac{d}{|S|} + \frac{e}{|T|} + \gamma\right) < 1,$$

then there exists a polynomial $Q(x, y)$ of degree (d, e) such that

$$\Pr_{(x,y) \in S \times T} [R(x, y) \neq Q(x, y) \text{ or } C(x, y) \neq Q(x, y)] < 2\gamma^2.$$

To prove Lemma 6.13 we show the contrapositive form for $c_0 = 128$, making no attempt to optimize constants. We may assume without loss of generality that $\delta^{(d,*)}, \delta^{(*,e)} < 1/c_0$; otherwise the claim is trivial. Correct each row of f to its closest RS-codeword (breaking ties arbitrarily), obtaining a bivariate polynomial $R(x, y)$ of degree $(d, |T| - 1)$. By definition, $\Delta(R(x, y), f) = \delta^{(d,*)}(f)$. Similarly, correct the columns of f to obtain the polynomial $C(x, y)$ of degree $(|S| - 1, e)$ that is within fractional distance $\delta^{(*,e)}(f)$ of f. We get

$$\Pr_{(x,y) \in S \times T} [R \neq C] \leq \delta^{(d,*)}(f) + \delta^{(*,e)}(f) = \gamma^2 < 1/64.$$

Since $\gamma \leq 1/8, d \leq |S|/4$, and $e \leq |T|/8$, both conditions of Theorem 6.15 hold, allowing us to conclude that $R(x, y)$ is $(2\gamma^2)$-close to $\mathrm{RM}(\mathbb{F}, S \times T, (d, e))$. The triangle inequality completes the proof:

$$\delta^{(d,e)}(f) \leq \Delta(f, R) + \Delta(R, \mathrm{RM}(\mathbb{F}, S \times T, (d, e))) \leq 3\delta^{(d,*)}(f) + 2\delta^{(*,e)}(f).$$

7. PCPPs for Reed–Solomon codes over smooth fields. In this section we give a PCPP-verifier for Reed–Solomon codes over smooth fields, when the set S over which the polynomials are evaluated are multiplicative subfields of the field, thereby proving Theorem 3.4 (restated below). We also show it suffices for obtaining quasilinear PCPs (Theorem 2.2). Our presentation mirrors that of the additive case presented in section 6.

THEOREM 7.1 (Theorem 3.4, restated). *Let* PAIR-SMOOTH-RS *be the restriction of* PAIR-RS *to pairs* $((\mathbb{F}, \langle\omega\rangle, d), p)$*, where* $\mathrm{ord}(\omega) = n$ *is a power of* 2*. Then,*

$$\text{PAIR-SMOOTH-RS} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n}\left[\begin{array}{l}\mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n),\\ \mathsf{query}\ O(1),\\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}}\end{array}\right].$$

7.1. Proof overview. This section should be read as a continuation of subsection 6.1. The crucial property used in our constructions in section 6 was that the linearized polynomial $q(z)$ "nicely partitions" the linear space L. Specifically, $q(z)$ defines a linear map on L, its image is a linear space of size $\approx \sqrt{|L|}$, and for every value in its image, the set of preimages of that value form an affine space of size $\approx \sqrt{|L|}$.

In the smooth case, $\mathbb{F}$ contains a multiplicative subgroup $S = \langle\omega\rangle$ of size n. Assume that $\sqrt{n}$ is an integer and consider a polynomial $P(z)$ evaluated over S. Using Proposition 6.3 with the polynomial $q(z) \triangleq z^{\sqrt{n}}$ we get $P(z) = Q(z, z^{\sqrt{n}})$. Notice that $q(z)$ "nicely partitions" $\langle\omega\rangle$. Specifically, $q(\langle w\rangle) = \langle\omega^{\sqrt{n}}\rangle$ is of size $\sqrt{n}$ (recall $q(S) \triangleq \{q(s) : s \in S\}$), and for every value in the image of q, the set of its preimages is a multiplicative coset of $\langle\omega^{\sqrt{n}}\rangle$.

Thus, to prove proximity of $P(z)$ to $\mathrm{RS}(\mathbb{F}, S, d)$ we may ask for an evaluation of $Q(x, y)$ on the set of points $(X \times Y) \cup Z$, where $Z = \{(z, q(z)) : z \in \langle\omega\rangle\}$ and $X = Y = \langle\omega^{\sqrt{n}}\rangle$. In the additive case we used the fact, implied by Proposition 6.6, that the union of a linear space (L'_0) and an affine shift of it ($L'_0 + \beta$) form a linear space of slightly larger dimension. In the smooth case, it is not true in general that $\langle\omega^{\sqrt{n}}\rangle$ and a coset of it form a small multiplicative group. In fact, the smallest group containing both can be as large as $\langle\omega\rangle$. To overcome this problem, we define the *shifted Reed–Solomon code* (SRS-code), which is formed of evaluations of polynomials over a multiplicative group $\langle\omega\rangle$ and a coset of it of the form $\kappa\langle\omega\rangle \triangleq \{\kappa z : z \in \langle w\rangle\}$.

The crucial observation is that $q(z)$ "nicely partitions" each of $\langle\omega\rangle$ and $\kappa\langle\omega\rangle$ into $\sqrt{n}$ cosets of $\langle\omega^{\sqrt{n}}\rangle$ (see Figure 3). Indeed, the image of $q(\langle\omega\rangle) = \langle\omega^{\sqrt{n}}\rangle$ and $q(\kappa\langle\omega\rangle) = \kappa^{\sqrt{n}}\langle\omega^{\sqrt{n}}\rangle$. Similarly, for an element in $q(\langle\omega\rangle)$ of the form $\omega^{j\sqrt{n}}, j \in [\sqrt{n}]$, we get $q^{(-1)}(\omega^{j\sqrt{n}}) = \omega^j\langle\omega^{\sqrt{n}}\rangle$ and for an element in $q(\kappa\langle\omega\rangle)$ of the form $\kappa^{\sqrt{n}}\omega^{j\sqrt{n}}$ we get $q^{(-1)}(\kappa^{\sqrt{n}}\omega^{j\sqrt{n}}) = \kappa\omega^j\langle\omega^{\sqrt{n}}\rangle$. Thus, we will ask our prover to provide an evaluation of the bivariate polynomial Q on the points (see Figure 4)

$$\{(z, q(z)) : z \in \langle\omega\rangle \cup \kappa\langle\omega\rangle\} \bigcup \left(\langle\omega^{\sqrt{n}}\rangle \times (\langle\omega^{\sqrt{n}}\rangle \cup \kappa^{\sqrt{n}}\langle\omega^{\sqrt{n}}\rangle)\right).$$

By our previous discussion we notice that the restriction of Q to certain rows and columns forms a word of an SRS-code of length $\approx \sqrt{n}$.

This allows us to measure proximity to the SRS-code of length n by measuring proximity to SRS-codes of size $\approx \sqrt{n}$. As in the additive case of section 6 we use the bivariate testing lemma, Lemma 6.13, to apply recursion and obtain quasilinear

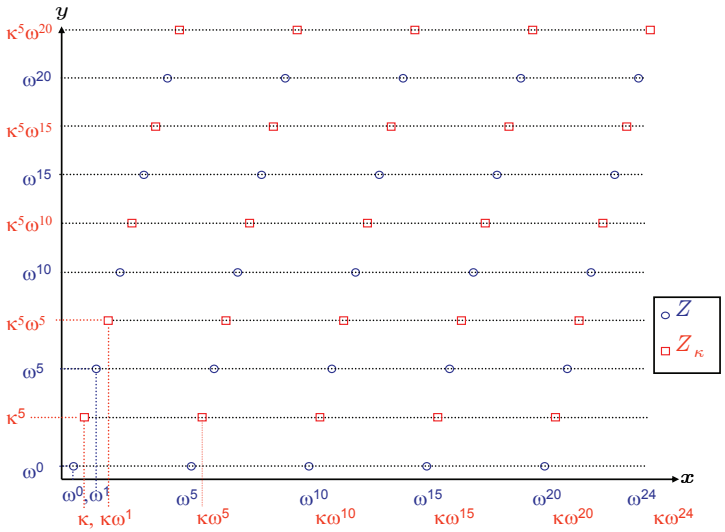


FIG. 3. *In this case,* $\mathbb{F} = \mathbb{Z}_{101}$. *Let* σ *generate* $\mathbb{F}^*$, *let* $\omega = \sigma^4$ *be an element of order* $n = 25$, *and let* $\kappa = \sigma^2$ *and* $q(z) = z^5$. *The elements on each axis are ordered by increasing powers of* σ *and the figure shows the subsets of points* $Z, Z_\kappa \subset \mathbb{F}^* \times \mathbb{F}^*$, *where* $Z = \{(z, q(z)) : z \in \langle\omega\rangle\}$ *and* $Z_\kappa = \{(\kappa z, q(\kappa z)) : z \in \langle\omega\rangle\}$.

sized proofs that can be tested with polylogarithmic query complexity. We need some technical modifications, arising from difficulties similar to the additive case. In particular, the degree of Q in its first variable is too large for applying Lemma 6.13, so we reduce this degree by breaking Q into a sum of several polynomials of sufficiently small degree. Additionally, we will not assume that $\sqrt{n}$ is an integer; rather we use the fact that $n = 2^k$ and work with the multiplicative subgroups generated by $n_0 = 2^{\lceil k/2 \rceil}, n_1 = 2^{\lfloor k/w \rfloor}$ that are of size $\approx \sqrt{n}$.

7.2. The shifted Reed–Solomon code. We prove Theorem 3.4 by proving a stronger statement about testing proximity to shifted RS-codes, defined next.

DEFINITION 7.2 (shifted Reed–Solomon code). *For* $\mathbb{F}$ *a finite field,* $\omega, \kappa \in \mathbb{F}^*, \mathrm{ord}(\omega) = n$, *and integer* d, *the degree-*d shifted Reed–Solomon (SRS)-code *over* $\langle\omega\rangle$ *with* shift κ *is*

$$\mathrm{SRS}(\mathbb{F}, d, \omega, \kappa) \triangleq \mathrm{RS}(\mathbb{F}, \langle\omega\rangle \cup \kappa\langle\omega\rangle, d).$$

Let PAIR-SMOOTH-SRS *be the pair language whose explicit inputs are triples* $(\mathbb{F}, S = \langle\omega\rangle \cup \kappa\langle\omega\rangle, d)$, *where* $\mathrm{ord}(\omega)$ *is a power of* 2 *and whose implicit inputs are functions* $p : S \to \mathbb{F}$. *The size of (explicit and implicit) inputs is* $\mathrm{ord}(\omega)$. *A pair* $((\mathbb{F}, S, d), p)$ *is in* PAIR-*SRS if* $p \in \mathrm{SRS}(\mathbb{F}, \omega, \kappa, d)$.

Notice that $\mathrm{SRS}(\mathbb{F}, \omega, 1, d) = \mathrm{RS}(\mathbb{F}, \langle\omega\rangle, d)$. Thus, Theorem 3.4 follows from the following theorem, the proof of which occupies the rest of the section.

THEOREM 7.3 (SRS PCP of proximity).

$$\text{PAIR-SMOOTH-SRS} \in \mathbf{Strong\text{-}PCPP}_{\delta/\operatorname{polylog} n}\left[\begin{array}{l}\mathsf{randomness}\ \log(n \cdot \operatorname{polylog} n),\\ \mathsf{query}\ O(\log|\mathbb{F}|),\\ \mathsf{distance}\ \mathsf{Hamming}_{\mathbb{F}}\end{array}\right].$$

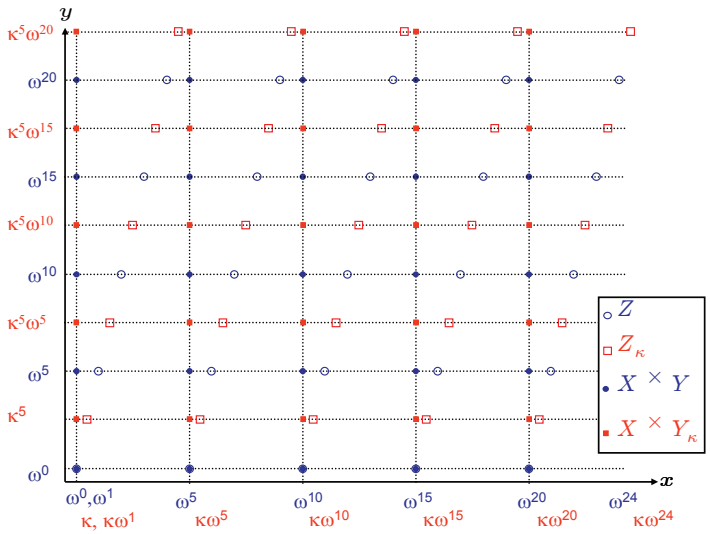


FIG. 4. *The proof of proximity for the smooth RS-code is the evaluation of Q on the set of points $Z \cup Z_\kappa \cup (X \times Y) \cup (X \times Y_\kappa)$. Notice that the restriction of this set to every row and column gives a pair—a multiplicative group of order $\sqrt{n}$ and a coset of it.*

As in the additive case, our proof will focus on the special case of degree $d = n/8 - 1$, and Proposition 6.14 generalizes this to an arbitrary degree.

7.3. The SRS proof of proximity and its associated verifier.

Notation. Recall $\mathrm{ord}(\omega) = n = 2^k$ for integer k. Let $n_0 = 2^{\lceil k/2 \rceil}$ and $n_1 = 2^{\lfloor k/2 \rfloor}$. Note that $n = n_0 \cdot n_1$ and $\sqrt{n/2} \le n_1 \le n_0 \le \sqrt{2n}$. For $r \le \mathrm{ord}(\alpha)$, let $\langle \alpha \rangle_r \triangleq \{\alpha^0, \alpha^1, \ldots, \alpha^{r-1}\}$. When dealing with a purported codeword of $\mathrm{SRS}(\mathbb{F}, \omega, \kappa, d)$ we treat it as a *pair* of functions, $p : \langle \omega \rangle \to \mathbb{F}$ and $p_\kappa : \kappa\langle \omega \rangle \to \mathbb{F}$.

DEFINITION 7.4 (SRS proof of proximity). *The proof of proximity for a purported codeword of the code $\mathrm{SRS}(\mathbb{F}, \omega, \kappa, n/8 - 1)$ is defined by induction on $n = \mathrm{ord}(\omega)$. If $n \le 16$ then it is empty. Otherwise, it is of the form*

$$\pi = (\{f^{(\ell)}, f_\kappa^{(\ell)}, g^{(\ell)}, g_\kappa^{(\ell)}, \{\pi^{(1,\beta,\ell)}, \pi^{(2,\beta,\ell)}\}_{\beta \in \langle \omega \rangle_{n_1}}, \{\pi^{(3,\tilde{\alpha},\ell)}\}_{\tilde{\alpha} \in \langle \omega^{n_1} \rangle}\}_{\ell \in \{0,\ldots,7\}}),$$

where

- $f^{(\ell)}, f_\kappa^{(\ell)} : \langle \omega^{n_1} \rangle \times \langle \omega \rangle_{n_1} \to \mathbb{F}$,
- $g^{(\ell)}, g_\kappa^{(\ell)} : \langle \omega^{n_1} \rangle \times \langle \omega^{n_0} \rangle \to \mathbb{F}$, *and*
- $\pi^{(1,\cdot,\ell)}, \pi^{(2,\cdot,\ell)}, \pi^{(3,\cdot,\ell)}$ *are proofs for* SRS*-codewords (over $\mathbb{F}$) of sizes n_0, n_0, n_1, respectively.*

We are now ready to describe the proximity tester.

DEFINITION 7.5 (SRS-verifier). *The verifier for proximity to* $\mathrm{SRS}(\mathbb{F}, \omega, \kappa, d = \mathrm{ord}(\omega)/8 - 1)$ *is denoted* $\mathrm{V}_{\mathrm{SRS}}^{\langle (p, p_\kappa), \pi \rangle}(\mathbb{F}, \omega, \kappa, d)$. *It receives as explicit input the parameters $\mathbb{F}, \omega, \kappa$ as defined in the statement of Theorem 7.3. The implicit input is a pair of functions $p : \langle \omega \rangle \to \mathbb{F}, p_\kappa : \kappa\langle \omega \rangle \to \mathbb{F}$. The proof π is as described in Definition 7.4. The verifier operates as follows.*

Base case ($n \le 16$). *The verifier reads p and p_κ in entirety and accepts iff $(p, p_\kappa) \in \mathrm{SRS}(\mathbb{F}, \omega, \kappa, 1)$.*

Recursion ($n \geq 32$). *The verifier computes* $n_0 = 2^{\lceil k/2 \rceil}, n_1 = 2^{\lfloor k/2 \rfloor}$ *and performs one of the following four tests with probability* $1/4$ *each.*

Outer. *Pick* $\tilde{\alpha} \in \langle \omega^{n_1} \rangle, \beta \in \langle \omega \rangle_{n_1}$ *uniformly at random; query* $p(\tilde{\alpha} \cdot \beta)$, $p_\kappa(\tilde{\alpha} \cdot \beta)$ *and* $f^{(\ell)}(\tilde{\alpha}, \beta)$, $f^{(\ell)}_\kappa(\tilde{\alpha}, \beta)$ *for every* $\ell \in \{0, \ldots, 7\}$*; accept iff* $p(\tilde{\alpha} \cdot \beta) = \sum_{\ell=0}^{7} (\tilde{\alpha} \cdot \beta)^{\ell n_0/8} \cdot f^{(\ell)}(\tilde{\alpha}, \beta)$ *and* $p_\kappa(\kappa\tilde{\alpha} \cdot \beta) = \sum_{\ell=0}^{7} (\kappa\tilde{\alpha} \cdot \beta)^{\ell n_0/8} \cdot f^{(\ell)}_\kappa(\tilde{\alpha}, \beta)$.

Inner. *Pick* $\ell \in \{0, \ldots, 7\}, \beta \in \langle \omega \rangle_{n_1}$ *at random and invoke*

$$\mathrm{V}_{\mathrm{SRS}}^{\langle (g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}|^{\leftrightarrow}_{\beta}), \pi^{(1,\beta,\ell)} \rangle}(\mathbb{F}, \omega^{n_1}, \beta, n_0/8 - 1).$$

Inner$_\kappa$. *Pick* $\ell \in \{0, \ldots, 7\}, \beta \in \langle \omega \rangle_{n_1}$ *at random and invoke*

$$\mathrm{V}_{\mathrm{SRS}}^{\langle (g^{(\ell)}_\kappa|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}_\kappa|^{\leftrightarrow}_{\beta}), \pi^{(2,\beta,\ell)} \rangle}(\mathbb{F}, \omega^{n_1}, \kappa\beta, n_0/8 - 1).$$

Inner$_c$. *Pick* $\ell \in \{0, \ldots, 7\}, \tilde{\alpha} \in \langle \omega^{n_1} \rangle$ *at random and invoke*

$$\mathrm{V}_{\mathrm{SRS}}^{\langle (g^{(\ell)}|^{\updownarrow}_{\tilde{\alpha}}, g^{(\ell)}_\kappa|^{\updownarrow}_{\tilde{\alpha}}), \pi^{(3,\alpha,\ell)} \rangle}(\mathbb{F}, \omega^{n_0}, \kappa^{n_0}, n_1/8 - 1).$$

The remaining subsections analyze the performance of this verifier, thus yielding Theorem 3.4. Specifically, the next subsection analyzes the simple properties including the query complexity, the randomness/size complexity, and the completeness. The hard part, the soundness analysis, is addressed in subsection 7.5.

7.4. Basic properties.

PROPOSITION 7.6. $\mathrm{V}_{\mathrm{SRS}}^{\langle (p, p_\kappa), \pi \rangle}(\mathbb{F}, \omega, \kappa, n/8 - 1)$ *makes at most* 32 *queries into* p, p_κ, π. *It tosses at most* $\log_2 n + O(\log \log n)$ *random coins and runs in time* $\mathrm{poly}\, n$. *The size of the proof* π *is* $O(n \cdot \mathrm{polylog}\, n)$.

Proof. The proof is straightforward from the definition. The query complexity is easy to verify. In the base case, the verifier reads 32 field elements. In the inductive case, if the verifier chooses to execute the **Outer** step, then it makes $18 < 32$ queries; else it makes a recursive query to $\mathrm{V}_{\mathrm{SRS}}^{\langle \rangle}$ which makes 32 queries by induction.

The randomness complexity is similar. In the base case the verifier tosses 0 coins. In the inductive case, the verifier tosses $O(1)$ coins to determine which step to perform. If it chooses the outer test, it picks $\tilde{\alpha}$ and β at random with $\log n + O(1)$ coins. If it chooses one of the inner tests, it tosses $\log \sqrt{n} + O(1)$ coins to determine the inner call, and then $\log \sqrt{n} + O(\log \log \sqrt{n})$ coins in the recursive call. Adding up, we get a total of $\log n + O(\log \log n)$ coins in all. Notice that all computations are simple and can be performed in time $\mathrm{poly}\, n$. Finally, the size of the proof is bounded by $2^{\mathrm{randomness}}$. ∎

Next we move to the completeness part of the proof. This part is straightforward given the intuition developed in subsection 7.1. We first generalize Proposition 6.3 and express a univariate polynomial as a sum of bivariate polynomials of low degree. We then use this to describe a proof π that is accepted with probability 1 when accompanying an SRS-codeword.

PROPOSITION 7.7. *Given positive integers* $d_1, d_2, L,$ *and* d *such that* $d_1 \cdot d_2 \cdot L \geq d$, *the following holds: For every univariate polynomial* $P(x)$ *of degree less than* d *there exists a sequence of* L *bivariate polynomial* $Q^{(0)}(y, z), \ldots, Q^{(L-1)}(y, z)$, *of degree less than* d_1 *in* y *and* d_2 *in* z, *such that*

$$P(x) = \sum_{\ell=0}^{L-1} x^{\ell \cdot d_1} Q^{(\ell)}(x, x^{L \cdot d_1}).$$

Furthermore, such a sequence is unique if $d_1 \cdot d_2 \cdot L = d$.

Proof. Let the a_i's be the coefficients of P; i.e., $P(x) = \sum_{i=0}^{d-1} a_i x^i$. Now let

$$Q^{(\ell)}(y,z) = \sum_{i=0}^{d_1-1} \sum_{j=0}^{d_2-1} a_{i+\ell\cdot d_1 + j\cdot d_1 \cdot L} y^i z^j,$$

where a_i is defined to be 0 if $i \geq d$. It can be verified by inspection that we have

$$P(x) = \sum_{\ell=0}^{L-1} x^{\ell \cdot d_1} Q^{(\ell)}(x, x^{L\cdot d_1}).$$

Uniqueness follows from a counting argument: the set of sequences of polynomials $Q^{(0)}, \ldots, Q^{(L-1)}$ forms a vector space of dimension $L \cdot d_1 \cdot d_2 = d$, the dimension of the space of polynomials of degree less than d. ∎

PROPOSITION 7.8 (completeness). *If* (p, p_κ) *equal the SRS encoding of some polynomial* P *of degree less than* $n/8$*, then there exists a proof that causes the SRS proximity tester to accept with probability one.*

Proof. The proof is by induction on n. Let $Q^{(0)}, \ldots, Q^{(7)}$ be the polynomials as given by Proposition 7.7 applied to P with integers $d_1 = n_0/8$, $d_2 = n_1/8$, $L = 8$, and $d = n/8$. Note that we have $d_1 \cdot d_2 \cdot L = d$, since $n_0 \cdot n_1 = n$. For every $\ell \in \{0, \ldots, 7\}$, we let $f^{(\ell)}(\tilde{\alpha}, \beta) = Q^{(\ell)}(\tilde{\alpha}\beta, \beta^{n_0})$, $f_\kappa^{(\ell)}(\tilde{\alpha}, \beta) = Q^{(\ell)}(\kappa\tilde{\alpha}\beta, \kappa^{n_0}\beta^{n_0})$, $g^{(\ell)}(\tilde{\alpha}, \tilde{\beta}) = Q^{(\ell)}(\tilde{\alpha}, \tilde{\beta})$, and $g_\kappa^{(\ell)}(\tilde{\alpha}, \tilde{\beta}) = Q^{(\ell)}(\tilde{\alpha}, \kappa^{n_0}\tilde{\beta})$ for every $\tilde{\alpha} \in \langle \omega^{n_1} \rangle$, $\beta \in \langle \omega \rangle_{n_1}$, and $\tilde{\beta} \in \langle \omega^{n_0} \rangle$.

Note that the above choice of table $f^{(\ell)}$, $f_\kappa^{(\ell)}$, $g^{(\ell)}$, $g_\kappa^{(\ell)}$ is such that the **Outer** test accepts with probability one. Specifically, we have

$$\begin{aligned} p(\tilde{\alpha}\cdot\beta) &= P(\tilde{\alpha}\cdot\beta) \\ &= \sum_{\ell \in \{0,\ldots,7\}} (\tilde{\alpha}\cdot\beta)^{\ell n_0/8} Q^{(\ell)}(\tilde{\alpha}\beta, \tilde{\alpha}^{n_0}\beta^{n_0}) \\ &= \sum_{\ell \in \{0,\ldots,7\}} (\tilde{\alpha}\cdot\beta)^{\ell n_0/8} Q^{(\ell)}(\tilde{\alpha}\beta, \beta^{n_0}) \\ &= \sum_{\ell \in \{0,\ldots,7\}} (\tilde{\alpha}\cdot\beta)^{\ell n_0/8} f^{(\ell)}(\tilde{\alpha}, \beta). \end{aligned}$$

Similarly we get $p_\kappa(\kappa\tilde{\alpha}\cdot\beta) = \sum_{\ell=0}^{7} (\kappa\tilde{\alpha}\cdot\beta)^{\ell n_0/8} \cdot f_\kappa^{(\ell)}(\tilde{\alpha}, \beta)$.

Now we describe how to set up the rest of the subproofs $\pi^{(\cdot,\cdot,\cdot)}$ such that the inner tests accept. For this part, note that the recursive calls to the SRS proximity verifiers access implicit input pairs that satisfy the completeness condition on smaller inputs. Consider, for example, the invocation

$$\mathrm{V}_{\mathrm{SRS}}^{\langle (g^{(\ell)}|_{\beta^{n_0}}^{\leftrightarrow}, f^{(\ell)}|_{\beta}^{\leftrightarrow}), \pi^{(1,\beta,\ell)} \rangle}(\mathbb{F}, \omega^{n_1}, \beta, n_0/8 - 1)$$

by **Inner** for some $\ell \in \{0, \ldots, 7\}$ and $\beta \in \langle \omega \rangle_{n_1}$. We may relate these implicit inputs to the polynomial $Q^{(\ell)}$ as follows: We have $g^{(\ell)}|_{\beta^{n_0}}^{\leftrightarrow}(\tilde{\alpha}) = g^{(\ell)}(\tilde{\alpha}, \beta^{n_0}) = Q^{(\ell)}(\tilde{\alpha}, \beta^{n_0})$, $f^{(\ell)}|_{\beta}^{\leftrightarrow}(\tilde{\alpha}) = f^{(\ell)}(\tilde{\alpha}, \beta) = Q(\tilde{\alpha}\cdot\beta, \beta^{n_0})$. Thus, if we let $P'(\tilde{\alpha}) = Q^{(\ell)}(\tilde{\alpha}, \beta^{n_0})$ and $\omega' = \omega^{n_1}$, then the pair $f^{(\ell)}|_{\beta}^{\leftrightarrow}, g^{(\ell)}|_{\beta^{n_0}}^{\leftrightarrow}$ is a codeword of the SRS-code $\mathrm{SRS}(\mathbb{F}, \omega', \beta, n_0/8-1)$ corresponding to the encoding of P', and thus (by induction) there exists a proof $\pi^{(1,\beta,\ell)}$ that causes the recursive verifier to accept with probability one. Similar reasoning shows that the verifier also accepts with probability one when invoking $\mathbf{Inner}_\kappa$ or $\mathbf{Inner}_c$. ∎

7.5. Soundness. We now argue the soundness of the SRS-verifier as follows. By induction, for most $\tilde{\alpha}$ and β, the functions $g^{(\ell)}|^{\updownarrow}_{\tilde{\alpha}}$, $g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}$, $g^{(\ell)}_{\kappa}|^{\updownarrow}_{\tilde{\alpha}}$, and $g^{(\ell)}_{\kappa}|^{\leftrightarrow}_{\beta^{n_0}}$ are close to polynomials of degree roughly $\sqrt{n}$. The bivariate testing lemma, Lemma 6.13, implies that $g^{(\ell)}$ and $g^{(\ell)}_{\kappa}$ are very close to some low-degree bivariate polynomials $Q^{(\ell)}$ and $Q^{(\ell)}_{\kappa}$. Furthermore, we will show $Q^{(\ell)} \equiv Q^{(\ell)}_{\kappa}$. Next, we claim the function $f^{(\ell)}(z, z^{n_0})$ is close to the function $Q^{(\ell)}(z, z^{n_0})$ and similarly $f^{(\ell)}_{\kappa}$ is close to $Q^{(\ell)}(\kappa z, (\kappa z)^{n_0})$. Finally, we claim that $p(z)$ is close to $\sum_{\ell=0}^{7} z^{\ell n_0/8} \cdot Q^{(\ell)}(z, z^{n_0})$, i.e., p is close to a low-degree univariate polynomial. Similarly p_κ is close to *the same* low-degree polynomial, where the consistency of p and p_κ follows from the equivalence of Q and Q_κ.

LEMMA 7.9 (soundness). *There exists a constant c such that for every ϵ the following holds. If*

$$\Pr\left[\mathrm{V}_{\mathrm{SRS}}^{\langle (p,p_\kappa),\pi\rangle}(\mathbb{F}, \omega, \kappa, \mathrm{ord}(\omega)/8 - 1) = \texttt{reject}\right] \leq \epsilon,$$

then (p, p_κ) is $(c^{\log\log \mathrm{ord}(\omega)} \cdot \epsilon)$-close to $\mathrm{SRS}(\mathbb{F}, \omega, \kappa, \mathrm{ord}(\omega)/8 - 1)$.

Proof. Let c_0 be as in Lemma 6.13. Let $c_1 = 128 \cdot c_0$, $c_2 = (320 + 2c_1)$, and $c_3 = 8c_2 + 4$. We prove the lemma for $c = c_3^2$, which is a (large) constant. Note that the conditions imply $c > 1$ and $c > (2 \cdot (256 + 4c_1))^2$ as will be used later.

We assume the lemma is true by induction for smaller n and in particular for the recursive calls to the various **Inner** tests, and we now prove it for n. Assume $c^{\log\log n} \cdot \epsilon \leq 1$ or else the claim is vacuously true. We use below the fact that $c^{\log\log n_0} \leq c^{\log\log \sqrt{2n}} \leq c^{\log\log n - \frac{1}{2}}$ for every $c \geq 1$ and $n \geq 16$.

Denote by $\epsilon_O(\tilde{\alpha}, \beta)$ the probability that the **Outer** verifier rejects (p, p_κ, π) on random choice $\tilde{\alpha}$ and β. Let ϵ_O denote the expectation of $\epsilon_O(\tilde{\alpha}, \beta)$ over the choice of $\tilde{\alpha}$ and β. Similarly let $\epsilon_I(\ell, \beta)$, $\epsilon_\kappa(\ell, \beta)$, and $\epsilon_c(\ell, \tilde{\alpha})$ denote the probability that **Inner**, **Inner**$_\kappa$, and **Inner**$_c$ reject on random choice ℓ, β, and $\tilde{\alpha}$. Let $\epsilon_I(\ell)$, $\epsilon_\kappa(\ell)$, and $\epsilon_c(\ell)$ denote the expectations of these quantities over β and $\tilde{\alpha}$, and let ϵ_I, ϵ_κ, and ϵ_c denote the expectations over β, $\tilde{\alpha}$, and ℓ. By definition of the tester, we have $\epsilon = \frac{1}{4} \cdot (\epsilon_O + \epsilon_I + \epsilon_\kappa + \epsilon_c)$. Since these quantities are nonnegative, we get $\epsilon_O, \epsilon_I, \epsilon_\kappa, \epsilon_c \leq 4\epsilon$. Similarly, we have $\epsilon_O(\ell), \epsilon_I(\ell), \epsilon_\kappa(\ell), \epsilon_c(\ell) \leq 32\epsilon$ for every $\ell \in \{0, \ldots, 7\}$.

For $\ell \in \{0, \ldots, 7\}$, denote by $Q^{(\ell)}(x, y)$ the polynomial of degree at most $n_0/8$ in x and $n_1/8$ in y that is closest to $g^{(\ell)}$ (on the domain $\langle \omega^{n_1} \rangle \times \langle \omega^{n_0} \rangle$), where ties are broken arbitrarily. Similarly let $Q^{(\ell)}_{\kappa}$ be the closest polynomial to $g^{(\ell)}_{\kappa}$. Let $P(z) = \sum_{\ell=0}^{7} z^{\ell n_0/8} \cdot Q^{(\ell)}(z, z^{n_0})$ and let $P_\kappa(z) = \sum_{\ell=0}^{7} z^{\ell n_0/8} \cdot Q^{(\ell)}_{\kappa}(\kappa z, z^{n_0})$. We show below that (p, p_κ) is close to the evaluation of P on $\langle \omega \rangle \cup \kappa \langle \omega \rangle$. (Among other facts, we also show that $P_\kappa(z) \equiv P(\kappa \cdot z)$.)

Step 1. *The functions $Q^{(\ell)}$ (and $Q^{(\ell)}_{\kappa}$).* By the inductive hypothesis applied to **Inner**(ℓ, β), we have that $(g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}|^{\leftrightarrow}_{\beta})$ is $(c^{\log\log n_0} \cdot \epsilon_I(\ell, \beta))$-close to the SRS encoding of some degree $n_0/8$ polynomial. Thus $g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}$ is at most $(2 \cdot c^{\log\log n_0} \cdot \epsilon_I(\ell, \beta))$-close to the RS encoding of some degree-$n_0/8$ polynomial. Averaging over β, we get that $g^{(\ell)}$ is $(2 \cdot c^{\log\log n_0} \cdot \epsilon_I(\ell))$-close to some bivariate polynomial of degree $n_0/8$ in x and arbitrary degree in y. A similar argument based on the **Inner**$_c$ tests yields that $g^{(\ell)}$ is $(2 \cdot c^{\log\log n_1} \cdot \epsilon_c(\ell))$-close to some bivariate polynomial of degree $n_1/8$ in y and arbitrary degree in x. Now applying Lemma 6.13, we get that $g^{(\ell)}$ is close to some polynomial of degree $n_0/8$ in x and $n_1/8$ in y. More specifically, we

have

$$\begin{aligned}\delta^{(n_0/8,n_1/8)}(g^{(\ell)}) &\le c_0 \cdot \left(\delta^{(n_0/8,*)}(g^{(\ell)}) + \delta^{(*,n_1/8)}(g^{(\ell)})\right)\\ &\le c_0 \cdot \left(2 \cdot c^{\log\log n_0} \cdot \epsilon_I(\ell) + 2 \cdot c^{\log\log n_1} \cdot \epsilon_c(\ell)\right)\\ &\le 64 \cdot c_0 \left(c^{\log\log n_0} + c^{\log\log n_1}\right) \cdot \epsilon\\ &\le 128 \cdot c_0 \cdot c^{\log\log n_0} \cdot \epsilon.\end{aligned}$$

Letting $c_1 \stackrel{\text{def}}{=} 128 \cdot c_0$, we have that $\delta(g^{(\ell)}, Q^{(\ell)}) \le c_1 \cdot c^{\log\log n_0} \cdot \epsilon$. A similar argument shows that $\delta(g_\kappa^{(\ell)}, Q_\kappa^{(\ell)}) \le c_1 \cdot c^{\log\log n_0} \cdot \epsilon$.

Step 2. *The functions* $f^{(\ell)}$ *and* $f_\kappa^{(\ell)}$. Next we move to the functions $f^{(\ell)}$ (for any $\ell \in \{0,\ldots,7\}$) and show that for most $\tilde{\alpha}, \beta$ $f^{(\ell)}(\tilde{\alpha},\beta) = Q^{(\ell)}(\tilde{\alpha}\cdot\beta, \beta^{n_0})$ (and similarly for most $\tilde{\alpha}, \beta$, $f_\kappa^{(\ell)}(\tilde{\alpha},\beta) = Q_\kappa^{(\ell)}(\kappa\cdot\tilde{\alpha}\cdot\beta, \beta^{n_0})$.

We first describe the argument informally. Consider a β such that $g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}$ and $f^{(\ell)}|^{\leftrightarrow}_{\beta}$ pass the **Inner** test with high probability *and* the SRS-codeword correspond to the encoding of $Q(\cdot,\beta^{n_0})$. For such β, we have $f^{(\ell)}|^{\leftrightarrow}_{\beta}(\tilde{\alpha},\beta) = Q(\tilde{\alpha}\cdot\beta,\beta^{n_0})$ for most $\tilde{\alpha}$. It remains to make this argument quantitative, and we do so below.

Define a β to be *good* if the fractional distance between $(g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}|^{\leftrightarrow}_{\beta})$ and the $\mathrm{SRS}(\mathbb{F}, n_0/8, \omega^{n_1}, \beta)$ encoding of $Q^{(\ell)}(\cdot,\beta^{n_0})$ is at most $1/8$. Let $\delta(\beta)$ denote the relative distance of $f^{(\ell)}|^{\leftrightarrow}_{\beta}$ to the projection of the SRS-codeword nearest to $(g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}|^{\leftrightarrow}_{\beta})$ onto the second half of the coordinates. Note that

$$\begin{aligned}&\Pr_{\tilde{\alpha},\beta}[f^{(\ell)}(\tilde{\alpha},\beta) \ne Q^{(\ell)}(\tilde{\alpha}\cdot\beta,\beta^{n_0})]\\ &\le \mathbb{E}_\beta[\delta(\beta)|\beta \text{ is good}] \cdot \Pr_\beta[\beta \text{ is good}] + \Pr_\beta[\beta \text{ is not good}]\\ &\le \mathbb{E}_\beta[\delta(\beta)] + \Pr_\beta[\beta \text{ is not good}].\end{aligned}$$

Note that the first term above is easily estimated as in Step 1. We get $\mathbb{E}_\beta[\delta(\beta)] \le (2 \cdot c^{\log\log n_0} \cdot \epsilon_I(\ell)) \le 64 \cdot c^{\log\log n_0} \cdot \epsilon$.

Next we describe two sets that cover the case where β is not good. Let S_1 be the set of all β such that the distance of $(g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}|^{\leftrightarrow}_{\beta})$ from every SRS-codeword is more than $\frac{1}{8}$. For every $\beta \in S_1$ note that the $\epsilon_I(\ell,\beta) \ge \frac{1}{8c^{\log\log n_0}}$. Thus, the probability that $\beta \in S_1$ is at most $8 \cdot c^{\log\log n_0} \cdot \epsilon_I(\ell) \le 256 \cdot c^{\log\log n_0} \cdot \epsilon$. Next, let S_2 be the set of β for which $(g^{(\ell)}|^{\leftrightarrow}_{\beta^{n_0}}, f^{(\ell)}|^{\leftrightarrow}_{\beta})$ is $\frac{1}{8}$-close to an SRS-codeword, but the SRS-codeword is not the encoding of $Q^{(\ell)}(\cdot,\beta^{n_0})$. For every $\beta \in S_2$, we have that $Q^{(\ell)}(\tilde{\alpha},\beta^{n_0})$ and $g^{(\ell)}(\tilde{\alpha},\beta^{n_0})$ disagree for at least $\frac{5}{8}$ fraction of the $\tilde{\alpha}$'s (since $Q^{(\ell)}(\cdot,\beta^{n_0})$ and the other SRS-codeword can agree on at most $n_0/8$ values of the $\tilde{\alpha}$'s). Since the distance between $g^{(\ell)}$ and $Q^{(\ell)}$ is at most $c_1 \cdot c^{\log\log n_0} \cdot \epsilon$, we get that the probability that $\beta \in S_2$ is at most $\frac{8}{5} \cdot c_1 \cdot c^{\log\log n_0} \cdot \epsilon \le 2c_1 \cdot c^{\log\log n_0} \cdot \epsilon$. Finally, we note that if β is not good, then $\beta \in S_1 \cup S_2$. Thus we get

$$\Pr_\beta[\beta \text{ is not good}] \le (256 + 2c_1) \cdot c^{\log\log n_0} \cdot \epsilon.$$

Putting the above together, and recalling $c_2 = (320+2c_1)$, we get $\Pr_{\tilde{\alpha},\beta}[f^{(\ell)}(\tilde{\alpha},\beta) \ne Q^{(\ell)}(\tilde{\alpha}\cdot\beta,\beta^{n_0})] \le c_2 \cdot c^{\log\log n_0} \cdot \epsilon$. Similarly we also get $\Pr_{\tilde{\alpha},\beta}[f_\kappa^{(\ell)}(\tilde{\alpha},\beta) \ne Q_\kappa^{(\ell)}(\kappa\cdot\tilde{\alpha}\cdot\beta,\beta^{n_0})] \le c_2 \cdot c^{\log\log n_0} \cdot \epsilon$.

Step 3. *The functions p and p_κ.* Next we move to the functions p and show that $p(z)$ usually equals $P(z) = \sum_{\ell=0}^{7} z^{\ell \cdot n_0/8} Q^{(\ell)}(z, z^{n_0})$ for $z \in \langle\omega\rangle$. Note that $\langle\omega\rangle$ is in one-to-one correspondence with $\{\tilde{\alpha} \cdot \beta\}$, where $\tilde{\alpha} \in \langle\omega^{n_1}\rangle$ and $\beta \in \langle\omega\rangle_{n_1}$, and so we are interested in estimating the probability that

$$p(\tilde{\alpha} \cdot \beta) \neq \sum_{\ell=0}^{7} (\tilde{\alpha}\beta)^{\ell \cdot n_0/8} Q^{(\ell)}(\tilde{\alpha}\beta, \beta^{n_0}).$$

We consider the following events: For $\ell \in \{0, \ldots, 7\}$, let E_ℓ be the event that $f^{(\ell)}(\tilde{\alpha}, \beta) \neq Q^{(\ell)}(\tilde{\alpha}\beta, \beta^{n_0})$. Further, let E' be the event that $p(\tilde{\alpha}\cdot\beta) \neq \sum_{\ell=0}^{7} (\tilde{\alpha}\beta)^{\ell \cdot n_0/8} \cdot f^{(\ell)}(\tilde{\alpha}\beta, \beta^{n_0})$. For any ℓ, we have that E_ℓ happens with probability at most $c_2 \cdot c^{\log\log n_0} \cdot \epsilon$. Further, E' happens with probability at most $\epsilon_O \leq 4\epsilon \leq 4 \cdot c^{\log\log n_0} \cdot \epsilon$, using $c \geq 1$. Furthermore, if none of the events E', $\{E_\ell\}_\ell$ occurs, then we do have $p(\tilde{\alpha} \cdot \beta) = \sum_{\ell=0}^{7} (\tilde{\alpha}\beta)^{\ell \cdot n_0/8} Q^{(\ell)}(\tilde{\alpha}\beta, \beta^{n_0})$. Thus, recalling $c_3 = 8c_2 + 4$, we get that $\delta(p, P) \leq c_3 \cdot c^{\log\log n_0} \cdot \epsilon$. Similarly, we get $\delta(p_\kappa, P_\kappa) \leq c_3 \cdot c^{\log\log n_0} \cdot \epsilon$. Combining, we get that $\delta((p, p_\kappa), (P, P_\kappa)) \leq c_3 \cdot c^{\log\log n_0} \cdot \epsilon$. By the definition of $c = c_3^2$ and the condition $c^{\log\log n_0} \leq c^{\log\log n - \frac{1}{2}}$, we get that the final proximity above is at most $c^{\log\log n} \cdot \epsilon$, as desired.

All that remains to be shown is that P and P_κ are consistent, i.e., that $P_\kappa(z) = P(\kappa \cdot z)$.

Step 4. *Consistency of the κ shifts.* We prove this part by showing that for every ℓ, Q and Q_κ are consistent, i.e., $Q_\kappa^{(\ell)}(x, y) = Q^{(\ell)}(x, \kappa^{n_0} y)$. This suffices, since we will then have

$$P_\kappa(z) = \sum_\ell z^{\ell n_0/8} Q_\kappa^{(\ell)}(\kappa z, z^{n_0}) = \sum_\ell z^{\ell n_0/8} Q^{(\ell)}(\kappa z, \kappa^{n_0} z^{n_0}) = P(\kappa z).$$

Fix $\ell \in \{0, \ldots, 7\}$. Define $\tilde{\alpha} \in \langle\omega^{n_1}\rangle$ to be *good* if $(g^{(\ell)}|_{\tilde{\alpha}}^{\uparrow}, g_\kappa^{(\ell)}|_{\tilde{\alpha}}^{\uparrow})$ is 1/8 close to some SRS-codeword *and* $g^{(\ell)}|_{\tilde{\alpha}}^{\uparrow}$ is 1/4 close to the evaluations of $Q^{(\ell)}(\tilde{\alpha}, \cdot)$, *and* $g_\kappa^{(\ell)}|_{\tilde{\alpha}}^{\uparrow}$ is 1/4 close to the evaluations of $Q_\kappa^{(\ell)}(\tilde{\alpha}, \cdot)$. It is straightforward to see that if $\tilde{\alpha}$ is good, then $Q_\kappa^{(\ell)}(\tilde{\alpha}, y) = Q^{(\ell)}(\tilde{\alpha}, \kappa^{n_0} y)$. Furthermore, if the fraction of good $\tilde{\alpha}$'s is more than 1/8, then we will have $Q_\kappa^{(\ell)}(x, y) = Q^{(\ell)}(x, \kappa^{n_0} y)$ as desired. So it suffices to bound the probability of $\tilde{\alpha}$ being not good (to be less than 7/8).

The three conditions above can be analyzed in a manner similar to the analysis of the probability of β not being *good* in Step 2. Specifically, we have the following: The probability that $(g^{(\ell)}|_{\tilde{\alpha}}^{\uparrow}, g_\kappa^{(\ell)}|_{\tilde{\alpha}}^{\uparrow})$ is not 1/8 close to some SRS-codeword is at most $8 \cdot c^{\log\log n_1} \cdot \epsilon_c(\ell) \leq 256 \cdot c^{\log\log n_0} \cdot \epsilon$. The probability that $g^{(\ell)}|_{\tilde{\alpha}}^{\uparrow}$ is 1/8 close to some SRS-codeword and not 1/4 close to the evaluations of $Q^{(\ell)}(\tilde{\alpha}, \cdot)$ is at most $2 \cdot c_1 \cdot c^{\log\log n_0} \cdot \epsilon$. Finally, the probability that $g_\kappa^{(\ell)}|_{\tilde{\alpha}}^{\uparrow}$ is 1/8 close to some SRS-codeword and not 1/4 close to the evaluations of $Q_\kappa^{(\ell)}(\tilde{\alpha}, \cdot)$ is at most $2 \cdot c_1 \cdot c^{\log\log n_0} \cdot \epsilon$. Combining the above we get that the probability that $\tilde{\alpha}$ is not good is at most $(256 + 4 \cdot c_1) \cdot c^{\log\log n_0} \cdot \epsilon$. In turn the final quantity is at most $(256 + 4 \cdot c_1) \cdot c^{\log\log n - \frac{1}{2}} \cdot \epsilon \leq \frac{1}{2} c^{\log\log n} \cdot \epsilon \leq \frac{1}{2} < \frac{7}{8}$ as desired. The first inequality follows from the fact that we have $c > (2 \cdot (256 + 4 \cdot c_1))^2$. This concludes the proof that Q and Q_κ and hence P and P_κ are consistent. Combined with Step 3, this concludes the soundness analysis. □

7.6. Proof of Theorem 3.4.

Proof of smooth SRS PCPP Theorem 7.3 (for special case of $d = n/8 - 1$). The verifier is formally defined in subsection 7.3. Its query complexity, randomness, and

proof length are given by Proposition 7.6. Its completeness is asserted by Proposition 7.8. Its soundness is analyzed in Lemma 7.9. □

Proof of Theorem 3.2. The statement for $d = n/8 - 1$ follows from Theorem 7.3 by setting $\kappa = 1$. The generalization to arbitrary degree d follows from Proposition 6.14. □

7.7. Proving Theorem 2.2 using smooth RS-codes. In this section we briefly outline the modifications needed to prove Theorem 2.2 using PCPPs for *smooth* RS-codes (Theorem 3.4). Our motivation is to present a proof of Theorem 2.2 in as general a setting as possible and in particular show that we do not require the underlying field to be of characteristic 2. Our exposition follows that of subsection 3.3.1.

Our first challenge is to show the existence and abundance of fields with a multiplicative subgroup of order that is a power of 2. A second problem is that we cannot embed de Bruijn graphs in an affine graph of constant degree (Proposition 5.11), because our fields are not of characteristic 2. To solve this problem we embed the de Bruijn graph in an affine graph of logarithmic degree. Thus, we end with a weaker version of Theorem 3.7 and we need to prove quasilinear PCPs for this weaker version, using Theorem 3.4. We now elaborate on each of these three issues.

Prime fields with 2*-smooth subgroups.* Theorem 3.4 holds only for Reed–Solomon codes $\mathrm{RS}(\mathbb{F}, \langle\omega\rangle, d)$, where $\mathrm{ord}(\omega)$ is a power of 2. The following (special case of a) theorem due to Linnik [33] shows that there is a polynomial time computable sequence $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ such that $n \le |\mathbb{F}_n| \le n^{O(1)}$ and $\mathbb{F}_n^*$ has an element ω the order of which is a power of 2.

THEOREM 7.10 (Linnik's theorem [33]). *There exists a constant* $1 < L < 6$ *such that for any sufficiently large* d*, there exists a prime of size* $\le d^L$ *such that* $d|(p-1)$.

Remark 7.11. The general statement of Linnik's theorem says that there exists a universal constant L such that for every pair of integers $0 < a < n$, there exists a prime $p < n^L$ such that $n|(p-a)$. The case stated above is derived from the general statement by setting $a = 1$.

Suppose we wish to find a field $\mathbb{F}_n$ of size $n^{O(1)}$ that has an element ω of order $\Theta(n)$ that is a power of 2. Let d be a power of 2 such that $n < d \le O(n)$. Let $\mathbb{F}_n$ be the prime field $\mathbb{Z}_p$ for p as in Linnik's theorem, Theorem 7.10. We have $|\mathbb{F}_n^*| = p-1 = k\cdot d$. Let σ be a generator of $\mathbb{F}^*$ and set $\omega = \sigma^k$. Then $\mathrm{ord}(\omega) = \Theta(n)$ is a power of 2. Notice that p and ω can be found in polynomial time (in n) by an exhaustive search. Finally, each element of $\mathbb{F}_n$ is represented by $O(\log n)$ bits.

Algebraic constraint satisfaction problems for PAIR-SMOOTH-RS. We now sketch a proof of a weaker version of Theorem 3.7. The weakness of this version refers to the fact that the number of affine functions is not constant but polylogarithmic. However, we will be able to prove this theorem without relying on fields of characteristic 2. Rather, we need our field only to be sufficiently large.

THEOREM 7.12 (ALGEBRAIC-CSP is NP-complete (weak version)). *There exists an integer* d *such that for any proper complexity function* $t : \mathbb{N}^+ \to \mathbb{N}^+$ *and* $L \in \mathrm{NTIME}(t(n))$*, the following hold.*

1. L *is reducible to* ALGEBRAIC-CSP *in time* $\mathrm{poly}\, t(n)$.
2. *Given any field* $\mathbb{F}$ *of size* $\Omega(t(n)\,\mathrm{polylog}\, t(n))$*, an instance of* L *of size* n *is reduced to an instance of* $\mathrm{ALGEBRAIC\text{-}CSP}_{\mathrm{polylog}\, n,d}$ *over* $\mathbb{F}$.

Proof. The reduction underlying Theorem 3.7 and described in subsection 5.2 relied on the existence of a homomorphism of the wrapped de Bruijn graph B_k (see Definition 5.5) into an affine graph (as per Definition 5.9) of constant degree over a

field of characteristic 2. This homomorphism, in turn, relies on the additive structure of the field (see the proof of Proposition 5.11).

As in the proof of Theorem 5.4, we will assume only that the underlying field is sufficiently large. We use the existence of an efficiently computable homomorphism of B_k into the hypercube of dimension $k + O(\log k)$ (for details see [31]). Next we notice the hypercube of dimension k' can be embedded into an affine graph over any finite field $\mathbb{F}, |\mathbb{F}| > 2^{k'}$. Indeed, fix $\omega \in \mathbb{F}^*$ with $\mathrm{ord}(\omega) \geq 2^{k'}$. Consider the affine graph G over vertex set $\langle\omega\rangle$ and edge set generated by $\{\omega^{(-1)^b \cdot 2^\ell}\}_{\ell \in [k'], b \in \{0,1\}}$. To see that the hypercube can be embedded into G, let $\bar{i} \in \{0, \ldots, 2^{k'} - 1\}$ denote the integer with binary representation $i \in \{0,1\}^{k'}$. Associate with i the element $\omega^{\bar{i}} \in \langle\omega\rangle$. We claim that the elements associated with i and $i + e_\ell$ (in the hypercube) are adjacent in G. Indeed, let b denote the ℓth bit of i and notice that i is associated with $\omega^{\bar{i}}$ whereas $i + e_\ell$ is associated with $\omega^{\bar{i}+(-1)^b 2^\ell} = \omega^{(-1)^b \cdot 2^\ell} \cdot \omega^{\bar{i}}$, so the corresponding vertices are adjacent in G.

From here on we follow the proof of Theorem 3.7, using the above defined affine graph G of degree $\mathrm{polylog}\, n$ instead of the constant degree graph used there. All other details are identical. Thus, our reduction results in an instance of ALGEBRAIC-CSP$_{\mathrm{polylog}\, n, O(1)}$. ∎

Quasilinear PCPs via PCPPs for smooth RS-codes. We now provide efficient PCPs for the instances of ALGEBRAIC-CSP given by Theorem 7.12 and thus provide an alternative proof of Theorem 2.2.

Proof of quasilinear PCP Theorem 2.2. Let ψ be an instance of $L \in \mathrm{NTIME}(t(n))$ of size n. Using Theorem 7.12 we reduce ψ to an instance $\phi = \{\mathbb{F}, \{\textsc{Aff}_1, \ldots, \textsc{Aff}_k\}, H, C\}$ of ALGEBRAIC-CSP$_{k,d}$ of size $n' = n \cdot \mathrm{polylog}\, n$, where $k = \mathrm{polylog}\, n, d = O(1)$ and $\mathbb{F}$ is the smallest prime field containing an element ω with $100kdn' < \mathrm{ord}(\omega) \leq 200kdn'$, where $\mathrm{ord}(\omega)$ is a power of 2. Linnik's theorem, Theorem 7.10, implies that $\mathbb{F}$ and ω exist and can be found in polynomial time (by an exhaustive search).

From here on our proof is essentially identical to the proof presented in subsection 3.3.1 and we use the notation given there. Notice that since $k = \mathrm{polylog}\, n$, the first subtest invokes an RS-verifier with proximity parameter $1/\mathrm{polylog}\, n$. However, Proposition 2.9 implies that the query complexity increases only by a factor of $\mathrm{polylog}\, n$. All other details are exactly as in subsection 3.3.1, and this completes the alternative proof of Theorem 2.2. ∎

Acknowledgments. We thank Oded Goldreich, Prahladh Harsha, Salil Vadhan, and Chris Umans for helpful discussions. We thank Don Coppersmith for pointing us to Linnik's theorem, which appears here as Theorem 7.10. We thank Venkatesan Guruswami, Subhash Khot, Jaikumar Radhakrishnan, and an anonymous referee for pointing out errors in previous versions of the paper. Finally, we thank the editor and anonymous referees for remarks that helped improve the clarity of the presentation.

REFERENCES

[1] N. Alon, *Combinatorial Nullstellensatz*, Combin. Probab. Comput., 8 (1999), pp. 7–29.
[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.
[3] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.
[4] L. Babai, L. Fortnow, and C. Lund, *Nondeterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[5] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 21–31.
[6] B. Barak, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 106–115.
[7] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.
[8] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistically checkable proofs and applications to approximation*, in Proceedings of the 25th ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 294–304.
[9] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Robust PCPs of proximity, shorter PCPs, and applications to coding*, in Proceedings of the 36th ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 13–15.
[10] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, *Short PCPs verifiable in polylogarithmic time*, in Proceedings of the 20th IEEE Conference on Computational Complexity, San Jose, CA, 2005, pp. 120–134.
[11] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson, *Randomness-efficient low degree tests and short PCPs via epsilon-biased sets*, in Proceedings of the 35th ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 612–621.
[12] A. Bhattacharyya, *Implementing Probabilistically Checkable Proofs of Proximity*, Technical report MIT-CSAIL-TR-2005-051 (MIT-LCS-TR-998), MIT, Cambridge, MA, 2005. Available online from http://publications.csail.mit.edu/2005trs.shtml.
[13] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, J. ACM, 51 (2004), pp. 557–594.
[14] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1971, pp. 151–158.
[15] S. Cook, *Short propositional formulas represent nondeterministic computations*, Inform. Process. Lett., 26 (1988), pp. 269–270.
[16] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, Springer-Verlag, Berlin, 1992.
[17] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra, *PCP characterizations of NP: Towards a polynomially-small error-probability*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 29–40.
[18] I. Dinur, *The PCP theorem by gap amplification*, in Proceedings of the 38th ACM Symposium on Theory of Computing, ACM, New York, 2006, pp. 241–250.
[19] I. Dinur and O. Reingold, *Assignment testers: Towards a combinatorial proof of the PCP theorem*, SIAM J. Comput., 36 (2006), pp. 975–1024.
[20] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.
[21] K. Friedl, Z. Hátsági, and A. Shen, *Low-degree tests*, in Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1994, pp. 57–64.
[22] O. Goldreich, *A Sample of Samplers—a Computational Perspective on Sampling*, Tech. rep. TR97-020, Electronic Colloquium on Computational Complexity, 1997.
[23] O. Goldreich, *Short Locally Testable Codes and Proofs (Survey)*, Tech. rep. TR05-014, Electronic Colloquium on Computational Complexity, 2005.
[24] O. Goldreich and M. Sudan, *Locally testable codes and PCPs of almost linear length (second revision)*, initial version appeared in Proceedings of the 43rd ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 13–22.
[25] O. Goldreich and A. Wigderson, *Tiny families of functions with random properties: A quality–size trade–off for hashing*, Random Structures Algorithms, 11 (1997), pp. 315–343.
[26] V. Guruswami, D. Lewin, M. Sudan, and L. Trevisan, *A tight characterization of NP with 3-query PCPs*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 18–27.
[27] P. Harsha and M. Sudan, *Small PCPs with low query complexity*, Comput. Complexity, 9 (2000), pp. 157–201.
[28] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
[29] F. C. Hennie and R. E. Stearns, *Two-tape simulation of multitape turing machines*, J. ACM, 13 (1966), pp. 533–546.

[30] J. Kilian, *A note on efficient zero-knowledge proofs and arguments (extended abstract)*, in Proceedings of the 24th ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 723–732.
[31] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.
[32] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd ed., Cambridge University Press, Cambridge, UK, 1997.
[33] Y. Linnik, *On the least prime in an arithmetic progression.* I. *The basic theorem*, Mat. Sb. N. S., 15 (57), 1944, pp. 139–178 (in Russian).
[34] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.
[35] S. Micali, *Computationally sound proofs*, SIAM J. Comput., 30 (2000), pp. 1253–1298.
[36] C. Papadimitriou, *Computational Complexity,* Addison–Wesley, Reading, MA, Longman, Harlow, UK, 1994.
[37] A. Polishchuk and D. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the 26th ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 194–203.
[38] A. Samorodnitsky and L. Trevisan, *A PCP characterization of NP with optimal amortized query complexity*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 191–199.
[39] D. Spielman, *Computationally Efficient Error-Correcting Codes and Holographic Proofs*, Ph.D. thesis, MIT, Cambridge, MA, 1995.
[40] M. Szegedy, *Many-valued logics and holographic proofs*, in Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, Prague, Czech Republic, 1999, pp. 676–686.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 608–628

LOWER-STRETCH SPANNING TREES*

MICHAEL ELKIN†, YUVAL EMEK‡, DANIEL A. SPIELMAN§, AND SHANG-HUA TENG¶

Abstract. We show that every weighted connected graph G contains as a subgraph a spanning tree into which the edges of G can be embedded with average stretch $O\left(\log^2 n \log\log n\right)$. Moreover, we show that this tree can be constructed in time $O\left(m \log n + n \log^2 n\right)$ in general, and in time $O(m \log n)$ if the input graph is unweighted. The main ingredient in our construction is a novel graph decomposition technique. Our new algorithm can be immediately used to improve the running time of the recent solver for symmetric diagonally dominant linear systems of Spielman and Teng from $m2^{\left(O(\sqrt{\log n \log\log n})\right)}$ to $m \log^{O(1)} n$, and to $O\left(n \log^2 n \log\log n\right)$ when the system is planar. Our result can also be used to improve several earlier approximation algorithms that use low-stretch spanning trees.

Key words. low-distortion embeddings, probabilistic tree metrics, low-stretch spanning trees

AMS subject classification. 68Q25

DOI. 10.1137/050641661

1. Introduction.

Let $G = (V, E, \ell)$ be a weighted connected graph, where $\ell : E \to \mathbb{R}_{>0}$ assigns a positive *length* to each edge. Given a spanning tree T of V, we define the *distance* in T between a pair of vertices $u, v \in V$, denoted $\mathrm{dist}_T(u, v)$, to be the sum of the lengths of the edges on the unique path in T between u and v. We can then define the *stretch*[1] of an edge $(u, v) \in E$ to be

$$\mathrm{stretch}_T(u, v) = \frac{\mathrm{dist}_T(u, v)}{\ell(u, v)},$$

and the average stretch over all edges of E to be

$$\text{ave-stretch}_T(E) = \frac{1}{|E|} \sum_{(u,v)\in E} \mathrm{stretch}_T(u, v).$$

Alon et al. [1] proved that every weighted connected graph $G = (V, E, \ell)$ of n vertices and m edges contains a spanning tree T such that

$$\text{ave-stretch}_T(E) = \exp\left(O(\sqrt{\log n \log\log n})\right)$$

*Received by the editors October 1, 2005; accepted for publication (in revised form) April 2, 2007; published electronically May 23, 2008.

http://www.siam.org/journals/sicomp/38-2/64166.html

†Department of Computer Science, Ben-Gurion University of the Negev, P.O.B. 653, Beer-Sheva 84105, Israel (elkinm@cs.bgu.ac.il). Part of this author's work was done at Yale University and was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under grant N00014-01-1-0795. This author's work was also partially supported by the Lynn and William Frankel Center for Computer Sciences.

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, P.O. Box 26, Rehovot 76100, Israel (yuval.emek@weizmann.ac.il).

§Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520-8285 (spielman@cs.yale.edu). This author's research was partially supported by NSF grant CCR-0324914.

¶Department of Computer Science, Boston University, 111 Cummington St., Boston, MA 02215 (steng@cs.bu.edu). This author's research was partially supported by NSF grants CCR-0311430 and ITR CCR-0325630.

[1]Our definition of the stretch differs slightly from that used in [1]: $\mathrm{dist}_T(u, v)/\mathrm{dist}_G(u, v)$, where $\mathrm{dist}_G(u, v)$ is the length of the shortest path between u and v. See section 1.1 for a discussion of the difference.

and that there exists a collection $\tau = \{T_1, \ldots, T_h\}$ of spanning trees of G and a probability distribution Π over τ such that for every edge $e \in E$,

$$\mathbf{E}_{T \leftarrow \Pi}\left[\text{stretch}_T(e)\right] = \exp\left(O(\sqrt{\log n \log\log n})\right).$$

The class of graphs considered in this context includes multigraphs that may contain self-loops and multiple edges between pairs of vertices. Considering multigraphs is essential for several applications (including some in [1]). Specifically, in some applications it is required to minimize various weighted averages of the stretches, where different edges may have different contributions to the average stretch, rather than a simple average as defined above. By allowing edge multiplicities, one can control the coefficients of these weighted averages, and thus our result is sufficiently general for such applications.

The result of [1] triggered the study of low-distortion embeddings into *probabilistic tree metrics*. Most notable in this context is the work of Bartal [3, 4] which shows that if the requirement that the trees T be *subgraphs* of G is abandoned, then the upper bound of [1] can be improved by finding a tree whose distances approximate those in the original graph with average distortion $O(\log n \cdot \log\log n)$. On the negative side, a lower bound of $\Omega(\log n)$ is known for both scenarios [1, 3]. The gap left by Bartal was recently closed by Fakcharoenphol, Rao, and Talwar [11], who have shown a tight upper bound of $O(\log n)$.

However, some applications of graph-metric approximation require trees that are subgraphs. Until now, no progress had been made on reducing the gap between the upper and lower bounds proved in [1] on the average stretch of subgraph spanning trees. The bound achieved in [1] for general weighted graphs had been the best bound known for *unweighted* graphs (where every edge admits a unit length), and even for *unweighted planar* graphs.

In this paper,[2] we significantly narrow this gap by improving the upper bound of [1] from $\exp(O(\sqrt{\log n \log\log n}))$ to $O(\log^2 n \log\log n)$. Specifically, we give an algorithm that for every weighted connected graph $G = (V, E, \ell)$ constructs a spanning tree $T \subseteq E$ that satisfies $\text{ave-stretch}_T(E) = O(\log^2 n \log\log n)$. The running time of our algorithm is $O(m \log n + n \log^2 n)$ for weighted graphs and $O(m \log n)$ for unweighted graphs. Note that the input graph need not be simple and its number of edges m can be much larger than $\binom{n}{2}$. However, as proved in [1], it is enough to consider graphs with at most $n(n+1)$ edges.

We begin by presenting a simpler algorithm that guarantees a weaker bound, $\text{ave-stretch}_T(E) = O(\log^3 n)$. As a consequence of the result in [1] that the existence of a spanning tree with average stretch $f(n)$ for every weighted graph implies the existence of a distribution of spanning trees in which every edge has expected stretch $f(n)$, our result implies that for every weighted connected graph $G = (V, E, \ell)$ there exists a probability distribution Π over a set $\tau = \{T_1, \ldots, T_h\}$ of spanning trees ($T \subseteq E$ for every $T \in \tau$) such that for every $e \in E$, $\mathbf{E}_{T \leftarrow \Pi}\left[\text{stretch}_T(e)\right] = O(\log^2 n \log\log n)$. Furthermore, our algorithm itself can be adapted to produce a probability distribution Π that guarantees a slightly weaker bound of $O(\log^3 n)$ in time $O(m \cdot \log^2 n)$.

In a subsequent paper, Emek and Peleg [9] proved that every series-parallel unweighted graph admits a spanning tree of average stretch $O(\log n)$. This bound is tight as it matches the lower bound established in [13].

[2] In a previous version of this paper, we proved the weaker bound on average stretch of $O((\log n \log\log n)^2)$. The improvement in this paper comes from rearranging the arithmetic in our analysis. Bartal [5] has obtained a similar improvement by other means.

1.1. Applications. For some of the applications listed below it is essential to define the stretch of an edge $(u,v) \in E$ as in [1], namely, $\text{stretch}_T(u,v) = \text{dist}_T(u,v)/\text{dist}_G(u,v)$. The algorithms presented in this paper can be adapted to handle this alternative definition of stretch, but this requires the computation of $\text{dist}_G(u,v)$ for every edge $(u,v) \in E$. The additional computation can be performed in a preprocessing stage independently of the algorithms themselves, but the time required for this preprocessing stage may dominate the running time of the algorithms.

1.1.1. Solving linear systems. Boman and Hendrickson [6] were the first to realize that low-stretch spanning trees could be used to solve symmetric diagonally dominant linear systems. They applied the spanning trees of [1] to design solvers that run in time

$$m^{3/2} 2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon),$$

where ϵ is the precision of the solution. Spielman and Teng [18] improved their results to

$$m 2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon).$$

Unfortunately, the trees produced by the algorithms of Bartal [3, 4] and Fakcharoenphol, Rao, and Talwar [11] cannot be used to improve these linear solvers, and it is currently not known whether it is possible to solve linear systems efficiently using trees that are not subgraphs. By applying the low-stretch spanning trees developed in this paper, we can reduce the time for solving these linear systems to

$$m \log^{O(1)} n \log(1/\epsilon),$$

and to $O(n \log^2 n \log\log n \log(1/\epsilon))$ when the systems are planar. Applying a reduction that was recently introduced by Boman, Hendrickson, and Vavasis [7], one obtains an $O(n \log^2 n \log\log n \log(1/\epsilon))$ time algorithm for solving the linear systems that arise when applying the finite element method to solve two-dimensional elliptic partial differential equations.

1.1.2. Alon–Karp–Peleg–West game. Alon et al. [1] constructed low-stretch spanning trees to upper-bound the value of a zero-sum two-player game that arose in their analysis of an algorithm for the k-server problem: at each turn, the *tree player* chooses a spanning tree T, and the *edge player* chooses an edge $e \in E$, simultaneously. The payoff to the edge player is 0 if $e \in T$ and $\text{stretch}_T(e)+1$ otherwise. They showed that if every n-vertex weighted connected graph G has a spanning tree T of average stretch $f(n)$, then the value of this game is at most $f(n)+1$. Our new result lowers the bound on the value of this game from $\exp\left(O(\sqrt{\log n \log\log n})\right)$ to $O\left(\log^2 n \log\log n\right)$.

1.1.3. MCT approximation. Our result can be used to dramatically improve the upper bound on the approximability of the *minimum communication cost spanning tree* (henceforth, *MCT*) problem. This problem was introduced in [14] and is listed as [ND7] in [12] and [8]. An instance of this problem is a weighted graph $G = (V, E, \ell)$ and a matrix $\{r(u,v) \mid u, v \in V\}$ of nonnegative requirements. The goal is to construct a spanning tree T that minimizes $\text{cost}(T) = \sum_{u,v \in V} r(u,v) \cdot \text{dist}_T(u,v)$.

Peleg and Reshef [16] developed a $2^{O(\sqrt{\log n \cdot \log\log n})}$ approximation algorithm for the MCT problem on metrics using the result of [1]. A similar approximation ratio can be achieved for arbitrary graphs. Therefore our result can be used to produce an efficient $O(\log^2 n \log\log n)$ approximation algorithm for the MCT problem on arbitrary graphs.

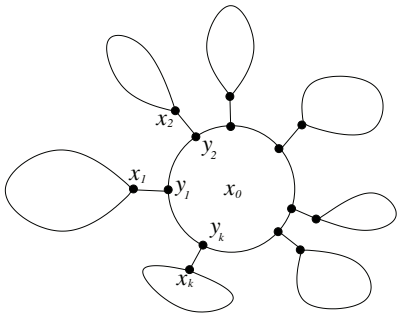


FIG. 1. *Star-decomposition.*

1.2. Our techniques. We build our low-stretch spanning trees by recursively applying a new graph decomposition that we call a *star-decomposition*. A star-decomposition of a graph is a partition of the vertices into sets that are connected into a star: a central set is connected to each other set by a single edge (see Figure 1). We show how to find star-decompositions that do not cut too many short edges and such that the radius of the graph induced by the star-decomposition is not much larger than the radius of the original graph.

Our algorithm for finding a low-cost star-decomposition applies a generalization of the ball-growing technique of Awerbuch [2] to grow *cones*, where the cone at a vertex x induced by a set of vertices S is the set of vertices whose shortest path to S goes through x.

1.3. The structure of the paper. In section 2, we define our notation. In section 3, we introduce the star-decomposition of a weighted connected graph. We then show how to use this decomposition to construct a subgraph spanning tree with average stretch $O(\log^3 n)$. In section 4, we present our star-decomposition algorithm. In section 5, we refine our construction and improve the average stretch to $O\left(\log^2 n \log\log n\right)$. Finally, we conclude the paper in section 6 and list some open questions.

2. Preliminaries. Throughout the paper, we assume that the input graph is a weighted connected multigraph $G = (V, E, \ell)$, where ℓ is a *length* function from E to the positive reals. Unless stated otherwise, we let n and m denote the number of vertices and the number of edges in the graph, respectively. The *cost* of an edge $e \in E$ is defined as the reciprocal of its length,[3] denoted by $\text{cost}(e) = 1/\ell(e)$. The *cost* of a set $F \subseteq E$ of edges, denoted $\text{cost}(F)$, is the sum of the costs of the edges in F.

Let u, v be two vertices in V. We define the *distance* between u and v, denoted $\text{dist}(u, v)$, to be the length of a shortest path between u and v in G. We write $\text{dist}_G(u, v)$ to emphasize that the distance is in the graph G. For a vertex subset $S \subseteq V$, we define the distance between u and S as $\text{dist}_G(u, S) = \min\{\text{dist}_G(u, x) \mid x \in S\}$.

For a set of vertices $S \subseteq V$, $G(S)$ is the subgraph induced on G by vertices in S. We write $\text{dist}_S(u, v)$ instead of $\text{dist}_{G(S)}(u, v)$ when G is understood. $E(S)$ is the set of edges with both endpoints in S. The *boundary* of S, denoted $\partial(S)$, is the set of edges with exactly one endpoint in S. If T is a set of vertices and $S \cap T = \emptyset$, then $E(S, T)$ is the set of edges with one endpoint in S and the other in T.

[3] In order to adapt our algorithms to the stretch definition of [1], we can simply define the cost of an edge as the reciprocal of the distance between its endpoints.

A *multiway partition* of V is a collection of pairwise-disjoint sets $\{V_1, \ldots, V_k\}$ such that $\bigcup_i V_i = V$. The *boundary* of a multiway partition, denoted $\partial(V_1, \ldots, V_k)$, is the set of edges with endpoints in different sets in the partition.

The *volume* of a set F of edges, denoted $\mathrm{vol}(F)$, is the size of the set $|F|$. The *volume* of a set S of vertices, denoted $\mathrm{vol}(S)$, is the number of edges with at least one endpoint in S.

Let v be a vertex in V. The *radius* of G with respect to v, denoted $\mathrm{rad}_G(v)$, is the smallest r such that every vertex of G is within distance (at most) r from v. Given a vertex subset $S \subseteq V$, we may write $\mathrm{rad}_S(v)$ instead of $\mathrm{rad}_{G(S)}(v)$ when G is understood. The *ball* of radius r around v, denoted $B(r, v)$, is the set of vertices at distance at most r from v. The *ball shell* of radius r around v, denoted $BS(r, v)$, is the set of vertices right outside $B(r, v)$; that is, $BS(r, v)$ consists of every vertex $u \in V - B(r, v)$ with a neighbor $w \in B(r, v)$ such that $\mathrm{dist}(v, u) = \mathrm{dist}(v, w) + \ell(u, w)$.

3. Spanning trees of $O(\log^3 n)$ stretch.

We present our first algorithm that generates a spanning tree with average stretch $O(\log^3 n)$. We first state the properties of the graph decomposition algorithm at the heart of our construction. We then present the construction and analysis of the low-stretch spanning trees. We defer the description of the graph decomposition algorithm and its analysis to section 4.

3.1. Low-cost star-decomposition.

Our graph decomposition algorithm produces *star-decompositions* of graphs, which we now define.

DEFINITION 3.1 (star-decomposition). *A multiway partition $\{V_0, \ldots, V_k\}$ is a* star-decomposition *of a weighted connected graph $G = (V, E, \ell)$ with center $x_0 \in V$ (see Figure 1) if $x_0 \in V_0$ and*

1. *for all $0 \le i \le k$, the subgraph induced on V_i is connected; and*
2. *for all $i \ge 1$, V_i contains an* anchor *vertex x_i that is connected to a vertex $y_i \in V_0$ by an edge $(x_i, y_i) \in E$. We call the edge (x_i, y_i) the* bridge *between V_0 and V_i.*

Let $r = \mathrm{rad}_G(x_0)$, and $r_i = \mathrm{rad}_{V_i}(x_i)$ for each $0 \le i \le k$. For $\delta, \epsilon \le 1/2$, a star-decomposition $\{V_0, \ldots, V_k\}$ is a (δ, ϵ)-star-decomposition if

a. *$r_0 \le (1 - \delta)r$;*
b. *$\mathrm{dist}(x_0, x_i) \ge \delta r$ for each $i \ge 1$; and*
c. *$\mathrm{dist}(x_0, x_i) + r_i \le (1 + \epsilon)r$ for each $i \ge 1$.*

The cost *of the star-decomposition is $\mathrm{cost}(\partial(V_0, \ldots, V_k))$.*

Note that if $\{V_0, \ldots, V_k\}$ is a (δ, ϵ)–star-decomposition of G, then the graph consisting of the union of the induced subgraphs on $V_0, \ldots, V_k$ and the bridge edges (y_i, x_i) has radius at most $(1 + \epsilon)$ times the radius of the original graph.

In section 4, we present an algorithm `StarDecomp` that satisfies the following cost guarantee. Let $\boldsymbol{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{y} = (y_1, \ldots, y_k)$.

LEMMA 3.2 (low-cost star-decomposition). *Let $G = (V, E, \ell)$ be a connected weighted graph and let x_0 be a vertex in V. Then for every positive $\epsilon \le 1/2$,*

$$(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{StarDecomp}(G, x_0, 1/3, \epsilon),$$

in time $O(m + n \log n)$, returns a $(1/3, \epsilon)$–star-decomposition of G with center x_0 satisfying

$$\mathrm{cost}(\partial(V_0, \ldots, V_k)) \le \frac{6\, m \log_2(m+1)}{\epsilon \cdot \mathrm{rad}_G(x_0)}.$$

On unweighted graphs, the running time is $O(m)$.

3.2. A divide-and-conquer algorithm. The basic idea of our algorithm is to use low-cost star-decompositions in a divide-and-conquer (recursive) algorithm to construct a spanning tree. Let $G = (V, E, \ell)$ be the graph input to the current recursive invocation of the algorithm. Recall that we write $n = |V|$ and $m = |E|$. Let $\widehat{n}$ and $\widehat{m}$ denote the number of vertices and number of edges, respectively, in the original graph, input to the first recursive invocation.

Before invoking our algorithm we apply a linear-time transformation from [1] that may decrease the number of edges in the given multigraph (recall that a multigraph of $\widehat{n}$ vertices may have an arbitrary number of edges). Given a weighted connected simple graph $G = (V, E, \ell)$ and a mapping $m : E \to \mathbb{Z}_{>0}$, let $G_m = (V, E_m, \ell_m)$ be the multigraph obtained from G by making $m(e)$ copies of each edge $e \in E$.

LEMMA 3.3 (established in [1]). *Let $G = (V, E, \ell)$ be a weighted connected simple graph. Then for every mapping $m : E \to \mathbb{Z}_{>0}$, there exists a mapping $m' : E \to \mathbb{Z}_{>0}$ such that $|E_{m'}| = \sum_{e \in E} m'(e) \le |V|\,(|V|+1)$ and* ave-stretch$_T(E_m) \le 2$ ave-stretch$_T(E_{m'})$ *for every spanning tree T of G. Moreover, the mapping m' can be computed in time linear in the size of G.*

Consequently, in what follows we assume that $\widehat{m} \le \widehat{n}(\widehat{n}+1)$.

We begin by showing how to construct low-stretch spanning trees for unweighted graphs, that is, when all edges have length 1. In particular, we use the fact that in this case the cost of a set of edges equals the number of edges in the set.

Fix $\alpha = (\log_{4/3}(\widehat{n} + 32))^{-1}$.
$T =$ `UnweightedLowStretchTree`$(G = (V, E), x_0)$.

1. If $|V| \le 2$, then return G. (If G contains multiple edges, return a single copy.)
2. $(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) =$ `StarDecomp`$(G, x_0, 1/3, \alpha)$.
3. For $0 \le i \le k$, set $T_i =$ `UnweightedLowStretchTree`$(G(V_i), x_i)$.
4. Set $T = \bigcup_i T_i \cup \bigcup_i (y_i, x_i)$.

THEOREM 3.4 (unweighted). *Let $G = (V, E)$ be an unweighted connected graph and let x_0 be a vertex in V. Then*

$$T = \texttt{UnweightedLowStretchTree}(G, x_0),$$

in time $O(\widehat{m} \log \widehat{n})$, returns a spanning tree of G satisfying

$$\mathrm{rad}_T(x_0) \le e \cdot \mathrm{rad}_G(x_0) \tag{1}$$

and

$$\text{ave-stretch}_T(E) \le O(\log^3 \widehat{m}). \tag{2}$$

Proof. For our analysis, we define a sequence of graphs that converges to T. For a graph G, we let

$$(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{StarDecomp}(G, x_0, 1/3, \alpha)$$

and recursively define

$$R_0(G) = G \quad \text{and} \qquad R_t(G) = \bigcup_i (y_i, x_i) \cup \bigcup_i R_{t-1}(G(V_i)).$$

The graph $R_t(G)$ is what one would obtain if we forced `UnweightedLowStretchTree` to return its input graph after t levels of recursion.

Let $\rho = \mathrm{rad}_G(x_0)$. Definition 3.1 implies $r_i \leq (1-\delta+\epsilon)r$ for every $1 \leq i \leq k$. By substituting $\delta = 1/3$, $\epsilon = \alpha$, and $r = \rho$, and because for all $\widehat{n} \geq 0$, $\alpha = (\log_{4/3}(\widehat{n}+32))^{-1} \leq 1/12$, we have $r_i \leq 3\rho/4$ for every $1 \leq i \leq k$. As $r_0 \leq 2\rho/3$, the depth of the recursion in `UnweightedLowStretchTree` is at most $\log_{4/3}\widehat{n}$, and we have $R_{\log_{4/3}\widehat{n}}(G) = T$. One can prove by induction that for every $t \geq 0$,

$$\mathrm{rad}_{R_t(G)}(x_0) \leq (1+\alpha)^t \mathrm{rad}_G(x_0).$$

Claim (1) now follows as $(1+\alpha)^{\log_{4/3}\widehat{n}} \leq e$. To prove claim (2), we note that

$$\begin{aligned}
(3)\qquad \sum_{(u,v)\in\partial(V_0,\ldots,V_k)} \mathrm{stretch}_T(u,v) &\leq \sum_{(u,v)\in\partial(V_0,\ldots,V_k)} (\mathrm{dist}_T(x_0,u) + \mathrm{dist}_T(x_0,v)) \\
&\leq \sum_{(u,v)\in\partial(V_0,\ldots,V_k)} 2e \cdot \mathrm{rad}_G(x_0), \text{ by (1)}, \\
&\leq 2e \cdot \mathrm{rad}_G(x_0) \left(\frac{6\,\widehat{m}\log_2(\widehat{m}+1)}{\alpha \cdot \mathrm{rad}_G(x_0)} \right), \text{ by Lemma 3.2}, \\
(4)\qquad &\leq 12e\,\widehat{m}\,(\log_2(\widehat{m}+1)) \left(\log_{4/3}(\widehat{n}+32)\right).
\end{aligned}$$

Applying this inequality to all graphs at all $\log_{4/3}\widehat{n}$ levels of the recursion, we obtain

$$\begin{aligned}
\sum_{(u,v)\in E} \mathrm{stretch}_T(u,v) &\leq 12e\,\widehat{m}\,(\log_2(\widehat{m}+1)) \left(\log_{4/3}\widehat{n}\right) \left(\log_{4/3}(\widehat{n}+32)\right) \\
&= O(\widehat{m}\log^3\widehat{m}).
\end{aligned}$$

(To justify the last inequality we remark that each edge ends up in $\partial(V_0,\ldots,V_k)$ on one of the levels of recursion, or it survives all the way down to a component of size two. In the latter case its contribution to the stretch is 1.)

The bound on the running time is obtained by Lemma 3.2 and because the depth of the recursion is $O(\log\widehat{n})$. □

We now extend our algorithm and proof to general weighted connected graphs. We begin by pointing out a subtle difference between general and unweighted graphs. In our analysis of `UnweightedLowStretchTree`, we used the facts that each edge length is 1 and $\mathrm{rad}_G(x_0) \leq n$ to show that the depth of recursion is at most $\log_{4/3} n$. In general weighted graphs, the ratio of $\mathrm{rad}_G(x_0)$ to the length of the shortest edge can be arbitrarily large. Thus, the recursion can be very deep. To compensate, we will contract all edges that are significantly shorter than the radius of their component. In this way, we will guarantee that each edge is active only in a logarithmic number of iterations.

Let $e = (u,v)$ be an edge in $G = (V,E,\ell)$. The contraction of e results in a new graph by identifying u and v as a new vertex whose neighbors are the union of the neighbors of u and v. All self-loops created by the contraction are discarded. We refer to u and v as the preimage of the new vertex.

We now state and analyze our algorithm for general weighted graphs.

Fix $\beta = (2\lceil \log_{4/3}(2\widehat{n} + 32)\rceil)^{-1}$.
$T = \texttt{LowStretchTree}(G = (V, E, \ell), x_0)$.

1. If $|V| \leq 2$, then return G. (If G contains multiple edges, return the shortest copy.)
2. Set $\rho = \mathrm{rad}_G(x_0)$.
3. Let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ be the graph obtained by contracting all edges in G of length less than $\beta\rho/\widehat{n}$.
4. $\left(\left\{\widetilde{V}_0, \ldots, \widetilde{V}_k\right\}, \widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}}\right) = \texttt{StarDecomp}(\widetilde{G}, x_0, 1/3, \beta)$.
5. For each i, let V_i be the preimage under the contraction of step 3 of vertices in $\widetilde{V}_i$, and let $(y_i, x_i) \in V_0 \times V_i$ be the edge of shortest length for which x_i is a preimage of $\widetilde{x}_i$ and y_i is a preimage of $\widetilde{y}_i$.
6. For $0 \leq i \leq k$, set $T_i = \texttt{LowStretchTree}(G(V_i), x_i)$.
7. Set $T = \bigcup_i T_i \cup \bigcup_i (y_i, x_i)$.

In what follows, we refer to the graph $\widetilde{G}$, obtained by contracting some of the edges of the graph G, as the *edge-contracted graph*.

THEOREM 3.5 (low-stretch spanning tree). *Let $G = (V, E, \ell)$ be a weighted connected graph and let x_0 be a vertex in V. Then*

$$T = \texttt{LowStretchTree}(G, x_0),$$

in time $O(\widehat{m}\log\widehat{n} + \widehat{n}\log^2\widehat{n})$, returns a spanning tree of G satisfying

$$\mathrm{rad}_T(x_0) \leq 2e \cdot \mathrm{rad}_G(x_0) \tag{5}$$

and

$$\text{ave-stretch}_T(E) = O(\log^3 \widehat{m}). \tag{6}$$

Proof. We first establish notation similar to that used in the proof of Theorem 3.4. Our first step is to define a procedure, SD, that captures the action of the algorithm in steps 2–4. We then define $R_0(G) = G$ and

$$R_t(G) = \bigcup_i (y_i, x_i) \cup \bigcup_i R_{t-1}(G(V_i)),$$

where

$$(\{V_0, \ldots, V_k\}, x_1, \ldots, x_k, y_1, \ldots, y_k) = \texttt{SD}(G, x_0, 1/3, \beta).$$

We now prove claim (5). Let $\rho = \mathrm{rad}_G(x_0)$. Let $s = \lceil \log_{4/3}(2\widehat{n})\rceil$ and let $\rho_s = \mathrm{rad}_{R_s(G)}(x_0)$. Each contracted edge is of length at most $\beta\rho/\widehat{n}$, and every path in the graph $G(V_i)$ contains at most n contracted edges; hence the insertion of the contracted edges into $G(V_i)$ increases its radius by an additive term of at most $\beta\rho$. Thus, we may prove by induction that

$$\mathrm{rad}_{R_s(G)}(x_0) \leq (1 + 2\beta)^s \cdot \mathrm{rad}_G(x_0).$$

As $(1 + 2\beta)^s \leq e$, we may conclude that ρ_s is at most $e \cdot \mathrm{rad}_G(x_0)$.

To determine how much larger the radius of T can be, we first note that Definition 3.1 implies $r_i \leq (1 - \delta + \epsilon)r$ for every $1 \leq i \leq k$. By substituting $\delta = 1/3$, $\epsilon = \beta$,

and $r = \rho$, we have $r_i \leq (2/3 + \beta)\rho$. When the contracted edges are inserted into $G(V_i)$, the radius r_i may increase by an additive term of at most $\beta\rho$; hence in each application of `StarDecomp` (actually, in each application of `SD`), $r_i \leq (2/3 + 2\beta)\rho$. Since $\beta = (2\lceil \log_{4/3}(2\widehat{n} + 32)\rceil)^{-1} \leq 1/24$ for all $\widehat{n} \geq 0$, it follows that $r_i \leq 3\rho/4$ for every $1 \leq i \leq k$. As $r_0 \leq 2\rho/3$, each component of G that remains after s levels of the recursion has radius at most $\rho(3/4)^s \leq \rho/2\widehat{n}$. We may also assume by induction that for the graph induced on each of these components, `LowStretchTree` outputs a tree of radius at most $2e(\rho/2\widehat{n}) = e\rho/\widehat{n}$. As there are at most $\widehat{n}$ of these components, we know that the tree returned by the algorithm has radius at most

$$e \cdot \mathrm{rad}_G(x_0) + \widehat{n} \times \frac{e}{\widehat{n}} \mathrm{rad}_G(x_0) = 2e \cdot \mathrm{rad}_G(x_0).$$

We now turn to the claim in (6), the bound on the stretch. In this part, we let $E_t \subseteq E$ denote the set of edges that are present at recursion depth t. That is, their endpoints appear in the same graph, and they are not identified by the contraction of short edges in step 2. We now observe that no edge can be present at more than $\lceil \log_{4/3}((2\widehat{n}/\beta) + 1)\rceil$ recursion depths. To see this, consider an edge (u, v) and let t be the first recursion level for which the edge is in E_t. Let ρ_t be the radius of the component in which the edge lies at that time. As u and v are not identified under contraction, they are at distance at least $\beta\rho_t/\widehat{n}$ from each other. (This argument can be easily verified, although the condition for edge contraction depends on the length of the edge rather than on the distance between its endpoints.) If u and v are still in the same graph on recursion level $t + \lceil \log_{4/3}((2\widehat{n}/\beta) + 1)\rceil$, then the radius of this graph is at most $\rho_t/((2\widehat{n}/\beta) + 1)$; thus its diameter is strictly less than $\beta\rho_t/\widehat{n}$, contradicting the lower bound on the distance between u and v.

Following the proof of the upper bound on $\sum_{(u,v)\in\partial(V_0,\ldots,V_k)} \mathrm{stretch}_T(u, v)$ in inequalities (3)–(4) in the proof of Theorem 3.4, we can show that the total contribution to the stretch at depth t is at most

$$O(\mathrm{vol}(E_t) \log^2 \widehat{m}).$$

Thus, the sum of the stretches over all recursion depths is

$$\sum_t O\left(\mathrm{vol}(E_t) \log^2 \widehat{m}\right) = O\left(\widehat{m} \log^3 \widehat{m}\right).$$

We now analyze the running time of the algorithm. On each recursion level, the dominant cost is that of performing the `StarDecomp` operations on each edge-contracted graph. Let V_t denote the set of vertices in all edge-contracted graphs on recursion level t. Then the total cost of the `StarDecomp` operations on recursion level t is at most $O(|E_t| + |V_t| \log |V_t|)$. We will prove in Lemma 3.6 that $\sum_t |V_t| = O(\widehat{n} \log \widehat{n})$, and as $\sum_t |E_t| = O(\widehat{m} \log \widehat{m})$, it follows that the total running time is $O(\widehat{m} \log \widehat{n} + \widehat{n} \log^2 \widehat{n})$. □

The following lemma shows that even though the number of recursion levels can be very large, the overall number of vertices in edge-contracted graphs appearing on different recursion levels is at most $O(\widehat{n} \log \widehat{n})$. This lemma is used only for the analysis of the running time of our algorithm; a reader interested only in the existential bound can skip it.

LEMMA 3.6. *Let V_t be the set of vertices appearing in edge-contracted graphs on recursion level t. Then $\sum_t |V_t| = O(\widehat{n} \log \widehat{n})$.*

Proof. Consider an edge-contracted graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ on recursion level t and let x be a vertex in $\widetilde{V}$. The vertex x was formed as a result of a number of edge contractions (maybe 0). Accordingly, x can be viewed as a *super-vertex*. Let $\chi(x)$ denote the set of original vertices that were identified together to form the super-vertex $x \in \widetilde{V}$.

We claim that for every super-vertex $x \in V_{t+1}$, there exists a super-vertex $y \in V_t$ such that $\chi(x) \subseteq \chi(y)$. To prove it, note that every graph on recursion level $t+1$ corresponds to a single component of a star-decomposition on recursion level t. Moreover, an edge that was contracted on recursion level $t+1$ must have been contracted on recursion level t as well. Therefore we can consider a directed forest $\mathcal{F}$ in which every node at depth t corresponds to some super-vertex in V_t and an edge leads from a node y at depth t to a node x at depth $t+1$, if $\chi(x) \subseteq \chi(y)$. Note that the roots of $\mathcal{F}$ correspond to super-vertices on recursion level 0 and the leaves of $\mathcal{F}$ correspond to the original vertices of the graph G.

In the proof of Theorem 3.5, we showed that every edge is present on $O(\log \widehat{n})$ recursion levels. Following a similar line of arguments, one can show that every super-vertex is present on $O(\log \widehat{n})$ (before it is decomposed into smaller super-vertices, each containing a subset of its vertices). Since there are $\widehat{n}$ vertices in the original graph G, there are $\widehat{n}$ leaves in $\mathcal{F}$. Therefore the overall number of nodes in the directed forest $\mathcal{F}$ is $O(\widehat{n} \log \widehat{n})$, and the bound on $\sum_t |V_t|$ holds. ∎

4. Star-decomposition. Our star-decomposition algorithm exploits two algorithms for growing sets. The first, `BallCut`, is the standard ball-growing technique introduced by Awerbuch [2], and was the basis of the algorithm of [1]. The second, `ConeCut`, is a generalization of ball-growing to cones. So that we can analyze this second algorithm, we abstract the analyses from the works of [15, 10]. Instead of nested balls, we consider *concentric systems*, which we now define.

DEFINITION 4.1 (concentric system). *A* concentric system *in a weighted graph $G = (V, E, \ell)$ is a family of vertex sets $\mathcal{L} = \{L_r \subseteq V : r \in \mathbb{R}_{\geq 0}\}$ such that*

1. *$L_0 \neq \emptyset$;*
2. *$L_r \subseteq L_{r'}$ for all $r < r'$; and*
3. *if a vertex $u \in L_r$ and (u, v) is an edge in E, then $v \in L_{r+\ell(u,v)}$.*

For example, for any vertex $x \in V$, the set of balls $\{B(r, x)\}$ is a concentric system. The *radius* of a concentric system $\mathcal{L}$ is $\operatorname{radius}(\mathcal{L}) = \inf\{r : L_r = V\}$. For each vertex $v \in V$, we define the *norm* of v with respect to $\mathcal{L}$ as $\|v\|_{\mathcal{L}} = \inf\{r : v \in L_r\}$.

LEMMA 4.2 (concentric system cutting). *Let $G = (V, E, \ell)$ be a connected weighted graph and let $\mathcal{L} = \{L_r\}$ be a concentric system. For every two reals $0 \leq \lambda < \lambda'$, there exists a real $r \in [\lambda, \lambda')$ such that*

$$\operatorname{cost}(\partial(L_r)) \leq \frac{\operatorname{vol}(L_r) + \tau}{\lambda' - \lambda} \max\left[1, \log_2\left(\frac{m + \tau}{\operatorname{vol}(E(L_\lambda)) + \tau}\right)\right],$$

where $m = |E|$ and

$$\tau = \begin{cases} 1 & \text{if } \operatorname{vol}(E(L_\lambda)) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Note that rescaling terms does not affect the statement of the lemma. For example, if all the lengths are doubled, then the costs are halved. Moreover, we may assume that $\lambda' \leq \operatorname{radius}(\mathcal{L})$, since otherwise, choosing $r = \operatorname{radius}(\mathcal{L})$ implies that $\partial(L_r) = \emptyset$, and the claim holds trivially.

Let $r_i = \|v_i\|_{\mathcal{L}}$, and assume that the vertices are ordered so that $r_1 \leq r_2 \leq \cdots \leq r_n$. We may now assume without loss of generality that each edge in the graph has minimal length. That is, an edge from vertex i to vertex j has length $|r_i - r_j|$. The reason we may make this assumption is that it only increases the costs of edges, making our lemma strictly more difficult to prove. (Recall that the cost of an edge is the reciprocal of its length.)

Let $B_i = L_{r_i}$. Our proof will make critical use of a quantity μ_i, which is defined to be

$$\mu_i = \tau + \operatorname{vol}(E(B_i)) + \sum_{(v_j,v_k)\in E: j\leq i<k} \frac{r_i - r_j}{r_k - r_j}.$$

That is, μ_i sums the edges inside B_i, proportionally counting edges that are split by the boundary of the ball. The two properties of μ_i that we exploit are

$$\mu_{i+1} = \mu_i + \operatorname{cost}(\partial(B_i))(r_{i+1} - r_i) \tag{7}$$

and

$$\tau + \operatorname{vol}(E(B_i)) \leq \mu_i \leq \tau + \operatorname{vol}(B_i). \tag{8}$$

Inequality (8) is trivial, and equality (7) follows from the definition by a straightforward calculation, as

$$\mu_i = \tau + \operatorname{vol}(E(B_{i+1})) - \operatorname{vol}(\{(v_j, v_{i+1}) \in E \mid j \leq i\}) + \sum_{(v_j,v_k)\in E: j\leq i<k} \frac{r_i - r_j}{r_k - r_j}$$

and

$$\operatorname{cost}(\partial(B_i))(r_{i+1} - r_i) = \sum_{(v_j,v_k)\in E: j\leq i<k} \frac{r_{i+1} - r_i}{r_k - r_j}.$$

Choose a and b so that $r_{a-1} \leq \lambda < r_a$ and $r_b < \lambda' \leq r_{b+1}$. Let $\nu = \lambda' - \lambda$. We first consider the trivial case in which $b < a$. (That is, $b = a-1$.) In that case, there is no vertex with norm between λ and λ'. Thus every edge crossing $L_{(\lambda+\lambda')/2}$ has length at least ν, and therefore cost at most $1/\nu$. Therefore, by setting $r = (\lambda + \lambda')/2$, we obtain

$$\operatorname{cost}(\partial(L_r)) \leq \operatorname{vol}(\partial(L_r))\frac{1}{\nu} \leq \operatorname{vol}(L_r)\frac{1}{\nu},$$

establishing the lemma in this case.

We now define

$$\eta = \log_2\left(\frac{m + \tau}{\operatorname{vol}(E(B_{a-1})) + \tau}\right).$$

Note that $B_{a-1} = L_\lambda$, by the choice of a. A similarly trivial case is when $[a, b]$ is nonempty and, in addition, there exists an $i \in [a-1, b]$ such that

$$r_{i+1} - r_i \geq \frac{\nu}{\eta}.$$

In this case, every edge in $\partial(B_i)$ has cost at most η/ν. By choosing r to be $\max\{r_i, \lambda\}$, we satisfy

$$\operatorname{cost}(\partial(L_r)) \le \operatorname{vol}(\partial(L_r))\frac{\eta}{\nu} \le \operatorname{vol}(L_r)\frac{\eta}{\nu};$$

hence, the lemma is established in this case.

In the remaining case that the set $[a, b]$ is nonempty and for all $i \in [a-1, b]$,

$$r_{i+1} - r_i < \frac{\nu}{\eta}, \tag{9}$$

we will prove that there exists an $i \in [a-1, b]$ such that

$$\operatorname{cost}(\partial(B_i)) \le \frac{\mu_i \eta}{\nu}.$$

Choosing $r = \max\{r_i, \lambda\}$ and applying (8), we will prove the lemma.

Assume by way of contradiction that

$$\operatorname{cost}(\partial(B_i)) > \mu_i \eta/\nu$$

for all $i \in [a-1, b]$. It follows, by (7), that

$$\mu_{i+1} > \mu_i + \mu_i(r_{i+1} - r_i)\eta/\nu$$

for all $i \in [a-1, b]$, which implies

$$\begin{aligned}
\mu_{b+1} &> \mu_{a-1} \prod_{i=a-1}^{b} (1 + (r_{i+1} - r_i)\eta/\nu) \\
&\ge \mu_{a-1} \prod_{i=a-1}^{b} 2^{((r_{i+1}-r_i)\eta/\nu)} \\
&= \mu_{a-1} \cdot 2^{(r_{b+1}-r_{a-1})\eta/\nu} \\
&\ge \mu_{a-1} \cdot ((m+\tau)/(\operatorname{vol}(E(B_{a-1})) + \tau)) \\
&\ge m + \tau,
\end{aligned}$$

where the second inequality holds by (9) since $1 + x \ge 2^x$ for every $0 \le x \le 1$ and the last inequality holds by (8). Thus we derive a contradiction. ∎

An analysis of the following standard ball-growing algorithm follows immediately by applying Lemma 4.2 to the concentric system $\{B(r, x)\}$.

$r = \texttt{BallCut}(G, x, \rho, \delta)$.

1. Set $r = \delta\rho$.
2. While $\operatorname{cost}(\partial(B(r,x))) > \frac{\operatorname{vol}(B(r,x))+1}{(1-2\,\delta)\rho}\log_2(m+1)$,
   (a) Find a vertex $v \notin B(r, x)$ that minimizes $\operatorname{dist}(x, v)$ and set $r = \operatorname{dist}(x, v)$.

COROLLARY 4.3 (weighted ball cutting). *Let $G = (V, E, \ell)$ be a connected weighted graph, and let $x_0 \in V$, $\rho = \operatorname{rad}_G(x_0)$, $r_0 = \texttt{BallCut}(G, x_0, \rho, 1/3)$, and $V_0 = B(r_0, x_0)$. Then $\rho/3 \le r_0 < 2\,\rho/3$ and*

$$\operatorname{cost}(\partial(V_0)) \le \frac{3(\operatorname{vol}(V_0) + 1)\log_2(|E| + 1)}{\rho}.$$

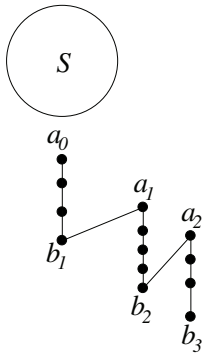


FIG. 2. *The vertex b_3 is in $C_S(l, a_0)$ if $\ell(b_1, a_1) + \ell(b_2, a_2) \leq l$.*

We now examine the concentric system that enables us to construct $V_1, \ldots, V_k$ in Lemma 3.2.

DEFINITION 4.4 (ideals and cones). *For any weighted graph $G = (V, E, \ell)$ and $S \subseteq V$, the set of* forward edges induced by S *is*

$$F(S) = \{(u \to v) : (u, v) \in E, \operatorname{dist}(u, S) + \ell(u, v) = \operatorname{dist}(v, S)\}.$$

For a vertex $v \in V$, the ideal of v induced by S, *denoted $I_S(v)$, is the set of vertices reachable from v by directed edges in $F(S)$, including v itself.*

For a vertex $v \in V$, the cone of width l around v induced by S, *denoted $C_S(l, v)$, is the set of vertices in V that can be reached from v by a path, the sum of the lengths of whose edges e that do not belong to $F(S)$ is at most l. Clearly, $C_S(0, v) = I_S(v)$ for all $v \in V$.*

That is, $I_S(v)$ is the set of vertices that have shortest paths to S that intersect v. Also, $u \in C_S(l, v)$ if there exist $a_0, \ldots, a_{k-1}$ and $b_1, \ldots, b_k$ such that $a_0 = v$, $b_k = u$, $b_{i+1} \in I_S(a_i)$, $(b_i, a_i) \in E$, and

$$\sum_{i=1}^{k-1} \ell(b_i, a_i) \leq l.$$

Refer to Figure 2 for an illustration of a cone.

We now establish that these cones form concentric systems.

PROPOSITION 4.5 (cones are concentric). *Let $G = (V, E, \ell)$ be a weighted graph and let $S \subseteq V$. Then for all $v \in V$, $\{C_S(l, v)\}_l$ is a concentric system in G.*

Proof. Clearly, $C_S(l, v) \subseteq C_S(l', v)$ if $l < l'$. Suppose that $u \in C_S(l, v)$ and $(u, w) \in E$. If $(u \to w) \in F(S)$, then $w \in C_S(l, v)$ as well. Otherwise, the path witnessing that $u \in C_S(l, v)$ followed by the edge (u, w) to w is a witness that $w \in C_S(l + \ell(u, w), v)$. ∎

$r = \texttt{ConeCut}(G, v, \lambda, \lambda', S)$.

1. Set $r = \lambda$.
2. If $\operatorname{vol}(E(C_S(\lambda, v))) = 0$, then set $\mu = (\operatorname{vol}(C_S(r, v)) + 1)\log_2(m + 1)$.
3. Otherwise, set $\mu = \operatorname{vol}(C_S(r, v))\log_2(m/\operatorname{vol}(E(C_S(\lambda, v))))$.
4. While $\operatorname{cost}(\partial(C_S(r, v))) > \mu/(\lambda' - \lambda)$,
   (a) Find a vertex $w \notin C_S(r, v)$ minimizing $\operatorname{dist}(w, C_S(r, v))$ and set $r = r + \operatorname{dist}(w, C_S(r, v))$.

COROLLARY 4.6 (cone cutting). *Let $G = (V, E, \ell)$ be a connected weighted graph, let v be a vertex in V, and let $S \subseteq V$. Then for any two reals $0 \leq \lambda < \lambda'$,* `ConeCut`$(G, v, \lambda, \lambda', S)$ *returns a real $r \in [\lambda, \lambda')$ such that*

$$\operatorname{cost}\left(\partial\left(C_S(r, v)\right)\right) \leq \frac{\operatorname{vol}\left(C_S(r, v)\right) + \tau}{\lambda' - \lambda} \max\left[1, \log_2 \frac{m + \tau}{\operatorname{vol}\left(E(C_S(\lambda, v))\right) + \tau}\right],$$

where $m = |E|$, and

$$\tau = \begin{cases} 1 & \text{if } \operatorname{vol}\left(E(C_S(\lambda, v))\right) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

We will use two other properties of the cones $C_S(l, v)$: that we can bound their radius (Proposition 4.7) and that their removal does not increase the radius of the resulting graph (Proposition 4.8). It is not too difficult to show that for every $S \subseteq V$ and every $x \in S$,

$$\operatorname{rad}_{C_S(l,x)}(x) \leq \max\{\operatorname{dist}(v, S) + 2l \mid v \in V\}.$$

To see this, observe that the $l = 0$ case is easy, and that for larger l, the extra distance traveled along forward edges is at most the distance traveled along nonforward edges.

However, we require a slightly more complicated inequality. Recall that for a vertex v and a real r, the ball shell $BS(r, v)$ consists of every vertex $u \in V - B(r, v)$ with a neighbor $w \in B(r, v)$ such that $\operatorname{dist}(v, u) = \operatorname{dist}(v, w) + \ell(u, w)$.

PROPOSITION 4.7 (radius of cones). *Let $G = (V, E, \ell)$ be a connected weighted graph, let $x_0 \in V$, and let $\rho = \operatorname{rad}_G(x_0)$. Let $r_0 < \rho$ and let $V_0 = B(r_0, x_0)$, $V' = V - V_0$, and $S = BS(r_0, x_0)$. Let x be any vertex in S and let $\psi = \rho - \operatorname{dist}_G(x_0, x)$. Then the cones $C_S(l, x)$ in the graph $G(V')$ satisfy*

$$\operatorname{rad}_{C_S(l,x)}(x) \leq \psi + 2l.$$

Proof. Let u be a vertex in $C_S(l, x)$, and let $a_0, \ldots, a_{k-1}$ and $b_1, \ldots, b_k$ be vertices in V' such that $a_0 = x$, $b_k = u$, $b_{i+1} \in I_S(a_i)$, $(b_i, a_i) \in E$, and

$$\sum_{i=1}^{k-1} \ell(b_i, a_i) \leq l.$$

These vertices provide a path connecting x to u inside $C_S(l, x)$ of length at most

$$\sum_{i=0}^{k-1} \operatorname{dist}_{G(V')}(a_i, b_{i+1}) + \sum_{i=1}^{k-1} \ell(b_i, a_i).$$

As the second term is at most l, we just need to bound the first term by $\psi + l$. To do this consider the distance of each of these vertices from x_0 in the graph G. Since $b_{i+1} \in I_S(a_i)$, and since S is a ball shell around x_0, it follows that

$$\operatorname{dist}_G(x_0, b_{i+1}) = \operatorname{dist}_G(x_0, a_i) + \operatorname{dist}_{G(V')}(a_i, b_{i+1}).$$

By the triangle inequality, we have

$$\operatorname{dist}_G(x_0, a_i) \geq \operatorname{dist}_G(x_0, b_i) - \ell(b_i, a_i).$$

Together, these imply

$$\rho \geq \mathrm{dist}_G(x_0, b_k) \geq \mathrm{dist}_G(x_0, a_0) - \sum_{i=1}^{k-1} \ell(b_i, a_i) + \sum_{i=0}^{k-1} \mathrm{dist}_{G(V')}(a_i, b_{i+1}).$$

The proposition follows since $\psi = \rho - \mathrm{dist}_G(x_0, a_0)$ and since $\sum_{i=1}^{k-1} \ell(b_i, a_i) \leq l$. ∎

PROPOSITION 4.8 (deleting cones). *Consider a connected weighted graph $G = (V, E, \ell)$, a vertex subset $S \subseteq V$, a vertex $x \in S$, and a real $l \geq 0$, and let $V' = V - C_S(l, x)$, $S' = S - C_S(l, x)$, and $\psi = \max_{v \in V} \mathrm{dist}(v, S)$. Then*

$$\max_{v \in V'} \mathrm{dist}_{V'}(v, S') \leq \psi.$$

Proof. Consider any $v \in V'$, and let P be a shortest path between S and v. Note that all the edges of P are forward edges, and thus if P intersects $C_S(l, x)$, then $v \in C_S(l, x)$. So, the shortest path from v to S in V must lie entirely in V'. ∎

The basic idea of `StarDecomp` is to first use `BallCut` to construct V_0 and then repeatedly apply `ConeCut` to construct $V_1, \ldots, V_k$.

$(\{V_0, \ldots, V_k, \boldsymbol{x}, \boldsymbol{y}\}) = \texttt{StarDecomp}(G = (V, E, \ell), x_0, \delta, \epsilon)$.
1. Set $\rho = \mathrm{rad}_G(x_0)$; set $r_0 = \texttt{BallCut}(G, x_0, \rho, \delta)$ and $V_0 = B(r_0, x_0)$.
2. Let $S = BS(r_0, x_0)$.
3. Set $G' = (V', E', \ell') = G(V - V_0)$, the weighted graph induced by $V - V_0$.
4. Set $(\{V_1, \ldots, V_k, \boldsymbol{x}\}) = \texttt{ConeDecomp}(G', S, \epsilon\rho/2)$.
5. For each $i \in [1 : k]$, set y_i to be a vertex in V_0 such that $(x_i, y_i) \in E$ and y_i is on a shortest path from x_0 to x_i. Set $\boldsymbol{y} = (y_1, \ldots, y_k)$.

$(\{V_1, \ldots, V_k, \boldsymbol{x}\}) = \texttt{ConeDecomp}(G, S, \Delta)$.
1. Set $G_0 = G$, $S_0 = S$, and $k = 0$.
2. While S_k is not empty,
   (a) Set $k = k + 1$.
   (b) Let x_k be an arbitrary vertex in S_{k-1} and set $r_k = \texttt{ConeCut}(G_{k-1}, x_k, 0, \Delta, S_{k-1})$.
   (c) Set $V_k = C_{S_{k-1}}(r_k, x_k)$. Set $G_k = G(V - \cup_{i=1}^{k} V_k)$ and $S_k = S_{k-1} - V_k$.
3. Set $\boldsymbol{x} = (x_1, \ldots, x_k)$.

Proof of Lemma 3.2. Let $\rho = \mathrm{rad}_G(x_0)$. By setting $\delta = 1/3$, Corollary 4.3 guarantees $\rho/3 \leq r_0 < (2/3)\rho$; thus $\mathrm{dist}_G(x_0, v) > \rho/3$ for every $v \in V - V_0$, and in particular, $\mathrm{dist}_G(x_0, x_i) > \rho/3$ for every $1 \leq i \leq k$. Applying $\Delta = \epsilon\rho/2$ and Propositions 4.7 and 4.8, we can bound for every i

$$\mathrm{dist}(x_0, x_i) + r_i \leq \rho + 2\Delta = \rho + \epsilon\rho.$$

Thus $\texttt{StarDecomp}(G, x_0, 1/3, \epsilon)$ returns a $(1/3, \epsilon)$–star-decomposition with center x_0.

To bound the cost of the star-decomposition that the algorithm produces, we use Corollaries 4.3 and 4.6 to show

$$\mathrm{cost}(\partial(V_0)) \leq \frac{3(1 + \mathrm{vol}(V_0))\log_2(m+1)}{\rho} \quad \text{and}$$

$$\mathrm{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \leq \frac{2(1 + \mathrm{vol}(V_j))\log_2(m+1)}{\epsilon\rho}$$

for every $1 \le j \le k$. Thus,

$$\begin{aligned}\operatorname{cost}(\partial(V_0,\dots,V_k)) &\le \sum_{j=0}^{k} \operatorname{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right)\\ &\le \frac{2\log_2(m+1)}{\epsilon\rho}\sum_{j=0}^{k}(\operatorname{vol}(V_j)+1)\\ &\le \frac{6\,m\log_2(m+1)}{\epsilon\rho}.\end{aligned}$$

To implement `StarDecomp` in $O(m + n\log n)$ time, we use a Fibonacci heap to implement steps (2) of `BallCut` and `ConeCut`. If the graph is unweighted, this can be replaced by a breadth-first search that requires $O(m)$ time. ∎

5. Improving the stretch. In this section, we improve the average stretch of the spanning tree to $O\left(\log^2 n \log\log n\right)$ by introducing a procedure `ImpConeDecomp` which refines `ConeDecomp`. This new cone decomposition trades off the volume of the cone against the cost of edges on its boundary (similar to Seymour [17]). Our refined star-decomposition algorithm `ImpStarDecomp` is identical to algorithm `StarDecomp`, except that it calls

$$(\{V_1,\dots,V_k,\boldsymbol{x}\}) = \texttt{ImpConeDecomp}(G', S, \epsilon\rho/2, t, \widehat{m})$$

at step 4, where t is a positive integer parameter whose role will be described soon.

$(\{V_1,\dots,V_k,\boldsymbol{x}\}) = \texttt{ImpConeDecomp}(G, S, \Delta, t, \widehat{m})$.

1. Set $G_0 = G$, $S_0 = S$, and $j = 0$.
2. While S_j is not empty,
   (a) Set $j = j + 1$ and $p = t - 1$.
   (b) Set x_j to be a vertex of S_{j-1}.
   (c) While $p > 0$,
      (i) $r_j = \texttt{ConeCut}(G_{j-1}, x_j, \frac{(t-p-1)\Delta}{t}, \frac{(t-p)\Delta}{t}, S_{j-1})$.
      (ii) If $\operatorname{vol}\left(E(C_{S_{j-1}}(r_j, x_j))\right) \le \frac{m}{2^{\log^{p/t}\widehat{m}}}$, then exit the loop; else $p = p - 1$.
   (d) Set $V_j = C_{S_{j-1}}(r_j, x_j)$, set $G_j = G(V - \cup_{i=1}^{j} V_i)$, and set $S_j = S_{j-1} - V_j$.
3. Set $\boldsymbol{x} = (x_1,\dots,x_k)$.

LEMMA 5.1 (improved low-cost star-decomposition). *Let G, x_0, and ϵ be as in Lemma 3.2, let t be a positive integer control parameter, and let $\rho = \operatorname{rad}_G(x_0)$. Then*

$$(\{V_0,\dots,V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{ImpStarDecomp}(G, x_0, 1/3, \epsilon, t, \widehat{m}),$$

in time $O(m + n\log n)$, returns a $(1/3,\epsilon)$–star-decomposition of G with center x_0 that satisfies

$$\operatorname{cost}(\partial(V_0)) \le \frac{6\operatorname{vol}(V_0)\log_2(\widehat{m}+1)}{\rho},$$

and for every index $j \in \{1,2,\dots,k\}$ there exists $p = p(j) \in \{0,1,\dots,t-1\}$ such that

$$\operatorname{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \le t \cdot \frac{4\operatorname{vol}(V_j)\log^{(p+1)/t}(\widehat{m}+1)}{\epsilon\rho}, \tag{10}$$

and unless $p = 0$,

$$\text{vol}\left(E(V_j)\right) \leq \frac{m}{2^{\log^{p/t}\widehat{m}}}. \tag{11}$$

Proof. In what follows, we call $p(j)$ the *index-mapping* of the vertex set V_j. We begin our proof by observing that $0 \leq r_j < \epsilon\rho/2$ for every $1 \leq j \leq k$. We can then show that $\{V_0, \ldots, V_k\}$ is a $(1/3, \epsilon)$–star-decomposition as we did in the proof of Lemma 3.2.

We now bound the cost of the decomposition. Clearly, the bound on $\text{cost}\left(\partial\left(V_0\right)\right)$ remains unchanged from that proved in Lemma 3.2 (because the algorithm that computes V_0 did not change), but here we upper-bound $\text{vol}\left(V_0\right) + 1$ by $2 \cdot \text{vol}\left(V_0\right)$. Fix an index $j \in \{1, 2, \ldots, k\}$, and let $p = p(j)$ be the final value of variable p in the loop above (that is, the value of p when the execution left the loop while constructing V_j). Observe that $p \in \{0, 1, \ldots, t-1\}$, and that unless the loop is aborted due to $p = 0$, we have $\text{vol}\left(E(V_j)\right) \leq \frac{m}{2^{\log^{p/t}\widehat{m}}}$ and inequality (11) holds.

For inequality (10), we split the discussion into two cases. First, consider the case $p = t - 1$. In this case, the inequality $\text{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \leq (\text{vol}\left(V_j\right) + 1)\log(\widehat{m}+1)(t/\Delta)$ follows directly from Corollary 4.6, and inequality (10) holds (recall that $\Delta = \epsilon\rho/2$).

Second, consider the case $p < t - 1$ and let r'_j be the value of the variable r_j set by the previous invocation of Algorithm `ConeCut` (which did not satisfy the test in line 2(c)(ii)). In this case, observe that at the beginning of the last iteration, $\text{vol}\left(E(C_{S_{j-1}}(r'_j, x_j))\right) > \frac{m}{2^{\log^{(p+1)/t}\widehat{m}}}$ (as otherwise the loop would have been aborted in the previous iteration). By Corollary 4.6,

$$\begin{aligned}
&\text{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \\
\leq &\frac{\text{vol}\left(V_j\right)}{\Delta/t} \times \max\left\{1, \log_2\left(\frac{m}{\text{vol}\left(E\left(C_{S_{j-1}}\left(\frac{(t-p-1)\Delta}{t}, x_j\right)\right)\right)}\right)\right\},
\end{aligned}$$

where $V_j = C_{S_{j-1}}(r_j, x_j)$. Since

$$\frac{(t-p-2)\Delta}{t} \leq r'_j < \frac{(t-p-1)\Delta}{t},$$

it follows that

$$\text{vol}\left(E\left(C_{S_{j-1}}\left(\frac{(t-p-1)\Delta}{t}, x_j\right)\right)\right) \geq \text{vol}\left(E(C_{S_{j-1}}(r'_j, x_j))\right) > \frac{m}{2^{\log^{(p+1)/t}\widehat{m}}}.$$

Therefore

$$\log^{(p+1)/t}\widehat{m} \geq \max\left\{1, \log_2\left(\frac{m}{\text{vol}\left(E\left(C_{S_{j-1}}\left(\frac{(t-p-1)\Delta}{t}, x_j\right)\right)\right)}\right)\right\},$$

and

$$\text{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \leq \frac{\text{vol}\left(V_j\right)\log^{(p+1)/t}\widehat{m}}{\Delta/t} = t \cdot \frac{2\,\text{vol}\left(V_j\right)\log^{(p+1)/t}\widehat{m}}{\epsilon\rho}.$$

The assertion follows. ∎

Our improved algorithm `ImpLowStretchTree`$(G, x_0, t, \widehat{m})$ is identical to the algorithm `LowStretchTree` except that in step 3 it calls `ImpStarDecomp`$(\widetilde{G}, x_0, 1/3, \beta, t, \widehat{m})$, and in step 5 it calls `ImpLowStretchTree`$(G(V_i), x_i, t, \widehat{m})$. We set $t = \log\log\widehat{m}$ throughout the execution of the algorithm.

THEOREM 5.2 (lower-stretch spanning tree). *Let $G = (V, E, \ell)$ be a connected weighted graph and let x_0 be a vertex in V. Then*

$$T = \texttt{ImpLowStretchTree}(G, x_0, t, \widehat{m}),$$

in time $O(\widehat{m}\log\widehat{n} + \widehat{n}\log^2\widehat{n})$, returns a spanning tree of G satisfying

$$\mathrm{rad}_T(x_0) \le 2e \cdot \mathrm{rad}_G(x)$$

and

$$\text{ave-stretch}_T(E) = O\left(\log^2\widehat{n}\log\log\widehat{n}\right).$$

Proof. The bound on the radius of the tree remains unchanged from that proved in Theorem 3.5.

We begin by defining a system of notation for the recursive process, assigning to each graph $G = (V, E)$ input to some recursive invocation of Algorithm `ImpLowStretchTree` a sequence $\sigma(G)$ of nonnegative integers. This is done as follows. If G is the original graph input to the first invocation of the recursive algorithm, then $\sigma(G)$ is empty. Assuming that the halt condition of the recursion is not satisfied for G, the algorithm continues, and some of the edges in E are contracted. Let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ be the resulting graph. (Recall that we refer to $\widetilde{G}$ as the edge-contracted graph.) Let $\{\widetilde{V}_0, \widetilde{V}_1, \ldots, \widetilde{V}_k\}$ be the star-decomposition of $\widetilde{G}$. Let $V_j \subseteq V$ be the preimage under edge contraction of $\widetilde{V}_j \subseteq \widetilde{V}$ for every $0 \le j \le k$. The graph $G(V_j)$ is assigned the sequence $\sigma(G(V_j)) = \sigma(G) \cdot j$. Note that $|\sigma(G)| = h$ implies that the graph G is input to the recursive algorithm on recursion level h. We warn the reader that the edge-contracted graph obtained from a graph assigned the sequence σ may have fewer edges than the edge-contracted graph obtained from the graph assigned the sequence $\sigma \cdot j$, because the latter may contain edges that were contracted out in the former.

We say that the edge e is *present* at recursion level h if there exists a graph G with $|\sigma(G)| = h$ such that e is not contracted out in $\widetilde{G}$. An edge e *appears* at the first level h at which it is present, and it *disappears* at the level at which it is present and its endpoints are separated by the star-decomposition or, if this never happens, then at the level of the leaf component of the recursion tree in which e appears. If an edge appears at recursion level h, then a path connecting its endpoints was contracted on every recursion level smaller than h, and no such path will be contracted on any recursion level greater than h. Moreover, an edge is never present at a level after it disappears. We define $h(e)$ and $h'(e)$ to be the recursion levels at which the edge e appears and disappears, respectively.

For every edge e and every recursion level i at which it is present, we let $U(e, i)$ denote the set of vertices $\widetilde{V}$ of the edge-contracted graph containing its endpoints. If $h(e) \le i < h'(e)$, then we let $W(e, i)$ denote the set of vertices $\widetilde{V}_j$ output by `ImpStarDecomposition` that contains the endpoints of e.

Recall that $p(j)$ denotes the index-mapping of the vertex set V_j in the star-decomposition. For each index $i \in \{0, 1, \ldots, t-1\}$, let $I_i = \{j \in \{1, 2, \ldots, k\} \mid p(j) = i\}$. For a vertex subset $U \subseteq V$, let $\mathrm{AS}(U)$ denote the average stretch that

the algorithm guarantees for the edges of $E(U)$. Let $\mathrm{TS}(U) = \mathrm{AS}(U) \cdot |E(U)|$. ($TS$ stands for the "total stretch.") Then by Lemma 5.1, the following recursive formula applies:

$$\begin{aligned}\mathrm{TS}(V) \leq & \left(\sum_{j=0}^{k} \mathrm{TS}(\widetilde{V}_j)\right) \\ & + 4e \times \left(6\,\log(\widehat{m}+1)\cdot \mathrm{vol}\left(\widetilde{V}_0\right) + 4\,\frac{t}{\beta}\sum_{p=0}^{t-1}\log^{(p+1)/t}(\widehat{m}+1)\sum_{j\in I_p}\mathrm{vol}\left(\widetilde{V}_j\right)\right) \\ & + \sum_{e\in E-\widetilde{E}} \mathrm{stretch}_T(e) \\ = & \ 4e \times \left(6\,\log(\widehat{m}+1)\cdot \mathrm{vol}\left(\widetilde{V}_0\right) + 4\,\frac{t}{\beta}\sum_{p=0}^{t-1}\log^{(p+1)/t}(\widehat{m}+1)\sum_{j\in I_p}\mathrm{vol}\left(\widetilde{V}_j\right)\right) \\ & + \sum_{j=0}^{k} \mathrm{TS}(V_j), \end{aligned} \tag{12}$$

where we recall $\beta = \epsilon = (2\lceil \log_{4/3}(2n+32)\rceil)^{-1}$.

For every edge e and for every $h(e) \leq i < h'(e)$, let $\pi_i(e)$ denote the index-mapping of the component $W(e,i)$ in the invocation of Algorithm `ImpConeDecomp` on recursion level i. For every index $p \in \{0,\ldots,t-1\}$, define the variable $l_p(e)$ as follows:

$$l_p(e) = \left|\{h(e) \leq i < h'(e) \mid \pi_i(e) = p\}\right|.$$

For a fixed edge e and an index $p \in \{0,1,\ldots,t-1\}$, every $h(e) \leq i < h'(e)$ such that $\pi_i(e) = p$ contributes $O(t/\beta)\cdot \log^{(p+1)/t}(\widehat{m}+1)$ to the right-hand term in (12) (that is, this is the contribution when an edge e is counted in a $\mathrm{vol}\left(\widetilde{V}_j\right)$ term). Summing p over $\{0,1,\ldots,t-1\}$, we obtain

$$\sum_{p=0}^{t-1} O(t/\beta) l_p(e) \log^{(p+1)/t}(\widehat{m}+1).$$

Soon, we will prove that

$$\sum_{e}\sum_{p=0}^{t-1} l_p(e)\log^{p/t}(\widehat{m}+1) \leq O(\widehat{m}\log_2\widehat{m}), \tag{13}$$

which implies that the sum of the contributions of all edges e in levels $h(e) \leq i < h'(e)$ to the right-hand term in (12) is

$$O\left(\frac{t}{\beta}\cdot \widehat{m}\log^{1+1/t}\widehat{m}\right) = O\left(t\cdot\widehat{m}\log^{2+1/t}\widehat{m}\right).$$

As $\mathrm{vol}\,(V_j)$ counts the internal edges of V_j as well as its boundary edges, we must also account for the contribution of each edge e at level $h'(e)$. At this level, it will be counted twice—once in each component containing one of its endpoints. Thus, at this stage, it contributes a factor of at most $O((t/\beta)\cdot\log\widehat{m})$ to the sum $\mathrm{TS}(V)$. Therefore all edges $e \in E$ at level $h'(e)$ contribute an additional factor of

$O(t \cdot \widehat{m} \log^2 \widehat{m})$. Summing over all the edges, we find that all the contributions to the right-hand term in (12) sum to at most

$$O\left(t \cdot \widehat{m} \log^{2+1/t} \widehat{m}\right).$$

Also, every $h(e) \le i < h'(e)$ such that the edge e belongs to the central component $\widetilde{V}_0$ of the star decomposition contributes $O(\log \widehat{m})$ to the left-hand term in (12). Since there are at most $O(\log \widehat{m})$ such i's, it follows that the contribution of the left-hand term in (12) to $\mathrm{TS}(V)$ sums up to an additive term of $O(\log^2 \widehat{m})$ for every single edge, and $(\widehat{m} \log^2 \widehat{m})$ for all edges.

It follows that $\mathrm{TS}(V) = O(t \cdot \widehat{m} \log^{2+1/t} \widehat{m})$. This is optimized by setting $t = \log \log \widehat{m}$, obtaining the desired upper bound of $O(\log^2 \widehat{n} \cdot \log \log \widehat{n})$ on the average stretch $\mathrm{AS}(V)$ guaranteed by Algorithm `ImpLowStretchTree`.

We now return to the proof of (13). We first note that $l_0(e)$ is at most $O(\log \widehat{m})$ for every edge e. We then observe that for each index $p > 0$ and each $h(e) \le i < h'(e)$ such that $\pi_i(e) = p$, $\mathrm{vol}\left(E(U(e,i))\right) / \mathrm{vol}\left(E(W(e,i))\right)$ is at least $2^{\log^{p/t} \widehat{m}}$ (by Lemma 5.1, (11)).

For $h(e) \le i < h'(e)$, let $g_i(e) = \mathrm{vol}\left(E(U(e,i+1))\right) / \mathrm{vol}\left(E(W(e,i))\right)$ (note that the vertex sets $W(e,i)$ and $U(e,i+1)$ correspond to the same component under different edge contraction. We then have

$$\prod_{1 \le p \le t-1} \left(2^{\log^{p/t} \widehat{m}}\right)^{l_p(e)} \le \widehat{m} \prod_{h(e) \le i < h'(e)} g_i(e);$$

hence $\sum_{p=1}^{t-1} l_p(e) \log^{p/t} \widehat{m} \le \log \widehat{m} + \sum_{h(e) \le i < h'(e)} \log g_i(e)$. We will next prove that

$$\sum_e \sum_{h(e) \le i < h'(e)} \log g_i(e) \le \widehat{m} \log \widehat{m}, \tag{14}$$

which implies (13).

Let E_i denote the set of edges present at recursion level i. For every edge $e \in E_i$ such that $i < h'(e)$, we have

$$\sum_{e' \in E(W(e,i))} g_i(e') = g_i(e) \mathrm{vol}\left(E(W(e,i))\right) = \mathrm{vol}\left(E(U(e,i+1))\right),$$

and so

$$\sum_{\substack{e \in E_i \\ h(e) \le i < h'(e)}} g_i(e) = \mathrm{vol}\left(E_{i+1}\right).$$

As each edge is present in at most $O(\log \widehat{m})$ recursion depths, $\sum_i \mathrm{vol}\left(E_i\right) \le \widehat{m} \log \widehat{m}$, which proves (14). ∎

6. Conclusion. At the beginning of the paper, we pointed out that the definition of stretch used in this paper differs slightly from that used by Alon et al. [1]. If one is willing to accept a longer running time, then this problem is easily remedied, as shown in section 1.1. If one is willing to accept a bound of $O(\log^3 n)$ on the stretch, then one can extend our analysis to show that the natural randomized variant of `LowStretchTree`, in which one chooses the radii of the balls and cones at random, works.

A natural open question is whether one can improve the stretch bound from $O\left(\log^2 n \log\log n\right)$ to $O(\log n)$. Algorithmically, it is also desirable to improve the running time of the algorithm to $O(m \log n)$. If we can successfully achieve both improvements, then we can use the Spielman–Teng solver to solve planar diagonally dominant linear systems in $O(n \log n \log(1/\epsilon))$ time.

As the average stretch[4] of any spanning tree in a weighted connected graph is at least 1, our low-stretch tree algorithm also provides an $O\left(\log^2 n \log\log n\right)$-approximation to the optimization problem of finding the spanning tree with the lowest average stretch. It remains open whether (a) our algorithm has a better approximation ratio and (b) one can in polynomial time find a spanning tree with better approximation ratio, e.g., $O(\log n)$ or even $O(1)$.

Acknowledgment. We are indebted to David Peleg, who was offered co-authorship on this paper.

REFERENCES

[1] N. Alon, R. M. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput., 24 (1995), pp. 78–100.
[2] B. Awerbuch, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.
[3] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 184–193.
[4] Y. Bartal, *On approximating arbitrary metrices by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 161–168.
[5] Y. Bartal, *private communication*, 2005.
[6] E. Boman and B. Hendrickson, *On Spanning Tree Preconditioners*, manuscript, Sandia National Labs, Livermore, CA, 2001.
[7] E. Boman, B. Hendrickson, and S. Vavasis, *Solving Elliptic Finite Element Systems in Near-Linear Time with Support Preconditioners*, manuscript, Sandia National Labs, Livermore, CA, and Cornell University, Ithaca, NY, available online from http://arXiv.org/abs/cs/0407022 (2004).
[8] P. Crescenzi and V. Kann, eds., *A Compendium of NP Optimization Problems*, http://www.nada.kth.se/~viggo/problemlist/.
[9] Y. Emek and D. Peleg, *A tight upper bound on the probabilistic embedding of series-parallel graphs*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 1045–1053.
[10] G. Even, J. Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, J. ACM, 47 (2000), pp. 585–616.
[11] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 448–455.
[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
[13] A. Gupta, I. Newman, Y. Rabinovich, and A. Sinclair, *Cuts, trees and ℓ_1-embeddings of graphs*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 399–408.
[14] T. C. Hu, *Optimum communication spanning trees*, SIAM J. Comput., 3 (1974), pp. 188–195.
[15] T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.
[16] D. Peleg and E. Reshef, *Deterministic polylogarithmic approximation for minimum communication spanning trees*, in Proceedings of the 25th International Colloquium on Automata, Languages and Programming, 1998, pp. 670–681.
[17] P. D. Seymour, *Packing directed circuits fractionally*, Combinatorica, 15 (1995), pp. 281–288.
[18] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 81–90.

[4] In the context of the optimization problem of finding a spanning tree with the lowest average stretch, the stretch is defined as in [1].

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 629–657

IMPROVED APPROXIMATION ALGORITHMS FOR MINIMUM WEIGHT VERTEX SEPARATORS*

URIEL FEIGE†, MOHAMMADTAGHI HAJIAGHAYI‡, AND JAMES R. LEE§

Abstract. We develop the algorithmic theory of vertex separators and its relation to the embeddings of certain metric spaces. Unlike in the edge case, we show that embeddings into L_1 (and even Euclidean embeddings) are insufficient but that the additional structure provided by many embedding theorems does suffice for our purposes. We obtain an $O(\sqrt{\log n})$ approximation for minimum ratio vertex cuts in general graphs, based on a new semidefinite relaxation of the problem, and a tight analysis of the integrality gap which is shown to be $\Theta(\sqrt{\log n})$. We also prove an optimal $O(\log k)$-approximate max-flow/min-vertex-cut theorem for arbitrary vertex-capacitated multicommodity flow instances on k terminals. For uniform instances on any excluded-minor family of graphs, we improve this to $O(1)$, and this yields a constant-factor approximation for minimum ratio vertex cuts in such graphs. Previously, this was known only for planar graphs, and for general excluded-minor families the best known ratio was $O(\log n)$. These results have a number of applications. We exhibit an $O(\sqrt{\log n})$ pseudoapproximation for finding balanced vertex separators in general graphs. In fact, we achieve an approximation ratio of $O(\sqrt{\log \mathsf{opt}})$, where opt is the size of an optimal separator, improving over the previous best bound of $O(\log \mathsf{opt})$. Likewise, we obtain improved approximation ratios for treewidth: in any graph of treewidth k, we show how to find a tree decomposition of width at most $O(k\sqrt{\log k})$, whereas previous algorithms yielded $O(k \log k)$. For graphs excluding a fixed graph as a minor (which includes, e.g., bounded genus graphs), we give a constant-factor approximation for the treewidth. This in turn can be used to obtain polynomial-time approximation schemes for several problems in such graphs.

Key words. graph separators, sparsest cut, embeddings, multicommodity flows

AMS subject classifications. 68Q25, 68W25

DOI. 10.1137/05064299X

1. Introduction.

Given a graph $G = (V, E)$, one is often interested in finding a small "separator" whose removal from the graph leaves two sets of vertices of roughly equal size (say, of size at most $2|V|/3$) with no edges connecting these two sets. The separator itself may be a set of edges, in which case it is called a *balanced edge separator*, or a set of vertices, in which case it is called a *balanced vertex separator*. In the present work, we focus on vertex separators.

Balanced separators of small size are important in several contexts. They are often the bottlenecks in communication networks (when the graph represents such a network) and can be used in order to provide lower bounds on communication tasks (see, e.g., [37, 35, 9]). Perhaps more importantly, finding balanced separators of small size is a major primitive for many graph algorithms and, in particular, for those that are based on the divide and conquer paradigm [39, 9, 36].

*Received by the editors October 18, 2005; accepted for publication (in revised form) July 12, 2007; published electronically May 23, 2008.
http://www.siam.org/journals/sicomp/38-2/64299.html

†Microsoft Research, Redmond, WA 98052, and Department of Computer Science and Applied Mathematics, Weizmann Institute, Rehovot 76100, Israel (urifeige@microsoft.com, uriel.feige@weizmann.ac.il).

‡Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (hajiagha@theory.csail.mit.edu). This work was done while this author was an intern in the Microsoft Research Theory Group.

§Computer Science and Engineering, University of Washington, Seattle, WA 98195 (jrl@cs.washington.edu). This work was done while this author was an intern in the Microsoft Research Theory Group.

Certain families of graphs always have small vertex separators. For example, trees always have a vertex separator containing a single vertex. The well-known planar separator theorem of Lipton and Tarjan [39] shows that every n-vertex planar graph has a balanced vertex separator of size $O(\sqrt{n})$ and, moreover, that such a separator can be found in polynomial time. This was later extended to show that more general families of graphs (any family of graphs that excludes some minor) have small separators [25, 2]. However, there are families of graphs (for example, expander graphs) in which the smallest separator is of size $\Omega(n)$.

Finding the smallest separator is an NP-hard problem (see, e.g., [15]). In the current paper, we study approximation algorithms that find vertex separators whose sizes are not much larger than the optimal separator of the input graph. These algorithms can be useful in detecting small separators in graphs that happen to have small separators, as well as in demonstrating that an input graph does not have any small vertex separator (and hence, for example, does not have serious bottlenecks for routing).

Much of the previous work on approximating vertex separators piggy-backed on work for approximating edge separators. For graphs of bounded degree, the sizes of the minimum edge and vertex separators are the same up to a constant multiplicative factor, leading to a corresponding similarity in terms of approximation ratios. However, for general graphs (with no bound on the degree), the situation is different. For example, every edge separator for the star graph has $\Omega(n)$ edges, whereas the minimum vertex separator has just one vertex. One can show that approximating vertex separators is at least as hard as approximating edge separators (see [15]). As to the reverse direction, it is known only that approximating vertex separators is at least as easy as approximating edge separators in *directed graphs* (a notion that will not be discussed in this paper).

The previous best approximation ratio for vertex separators is based on the work of Leighton and Rao [36]. They presented an algorithm based on linear programming that approximates the minimum edge separator within a ratio of $O(\log n)$. They observed that their algorithm can be extended to work on directed graphs and hence gives an approximation ratio of $O(\log n)$ also for vertex separators, using the algorithm for (directed) edge separators as a black box. More recently, Arora, Rao, and Vazirani [7] presented an algorithm based on semidefinite programming that approximates the minimum edge separator within a ratio of $O(\sqrt{\log n})$. Their remarkable techniques, which are a principal component in our algorithm for vertex separators, are discussed more in the following section.

In the present work, we formulate new linear and semidefinite program (SDP) relaxations for the vertex separator problem and then develop rounding algorithms for these programs. The rounding algorithms are based on techniques that were developed in the context of edge separators, but we exploit new properties of these techniques and adapt and enhance them to the case of vertex separators. Using this approach, we improve the best approximation ratio for vertex separators to $O(\sqrt{\log n})$. In fact, we obtain an $O(\sqrt{\log \mathsf{opt}})$ approximation, where opt is the size of an optimal separator. (An $O(\log \mathsf{opt})$ approximation can be derived from the techniques of [36].) In addition, we derive and extend some previously known results in a unified way, such as a constant factor approximation for vertex separators in planar graphs (a result originally proved in [5]), which we extend to any family of graphs excluding a fixed minor.

Before delving into more details, let us mention two aspects in which edge and vertex separators differ. One is the notion of a *minimum ratio cut*, which is an

important notion used in our analysis. For edge cuts, all "natural" definitions of such a notion are essentially equivalent. For vertex separators, this is not the case. One consequence of this is that our algorithms provide a good approximation for *vertex expansion* in bounded degree graphs but not in general graphs. This issue will be discussed in section 2. Another aspect where there is a distinction between edge and vertex separators is that of the role of embeddings into L_1 (a term that will be discussed later). For edge separators, if the linear program or SDP relaxations happen to provide such an embedding (i.e., if the solution is an L_1 metric), then they in fact yield an optimal edge separator. For vertex separators, embeddings into L_1 seem to be insufficient, and we give a number of examples that demonstrate this deficiency. Our rounding techniques for the vertex separator case are based on embeddings with small average distortion into a line, a concept that was first systematically studied by Rabinovich [41].

As mentioned above, finding small vertex separators is a basic primitive that is used in many graph algorithms. Consequently, our improved approximation algorithm for minimum vertex separators can be plugged into many of these algorithms, improving either the quality of the solution that they produce or their running time. Rather than attempting to provide in this paper a survey of all potential applications, we shall present one major application, that of improving the approximation ratio for *treewidth*, and discuss some of its consequences.

1.1. Some related work. An important concept that we shall use is the *ratio* of a vertex separator (A, B, S). Given a weight function $\pi : V \to \mathbb{R}_+$ on vertices and a set $S \subseteq V$ which separates G into two disconnected pieces A and B, we can define the *sparsity* of the separator by

$$\frac{\pi(S)}{\min\{\pi(A), \pi(B)\} + \pi(S)}.$$

Indeed, most of our effort will focus on finding separators (A, B, S) for which the sparsity is close to minimal among all vertex separators in G.

In the case of edge separators, there are intimate connections between approximation algorithms for minimum ratio cuts and the theory of metric embeddings. In particular, Linial, London, and Rabinovich [38] and Aumann and Rabani [8] show that one can use L_1 embeddings to round the solution to a linear relaxation of the problem. For the case of vertex cuts, we will show that L_1 embeddings (and even Euclidean embeddings) are insufficient but that the additional structure provided by many embedding theorems does suffice. This structure corresponds to the fact that many embeddings are of *Fréchet type*; i.e., their basic component takes a metric space X and a subset $A \subseteq X$ and maps every point $x \in X$ to its distance to A. This includes, for instance, the classical theorem of Bourgain [14].

The seminal work of Leighton and Rao [36] showed that, in both the edge and vertex cases, one can achieve an $O(\log n)$ approximation algorithm for minimum ratio cuts, based on a linear relaxation of the problem. Their solution also yields the first approximate max-flow/min-cut theorems in a model with uniform demands. The papers [38, 8] extend their techniques for the *edge case* to nonuniform demands. Their main tool is Bourgain's theorem [14], which states that every n-point metric space embeds into L_1 with $O(\log n)$ distortion.

Recently, Arora, Rao, and Vazirani [7] exhibited an $O(\sqrt{\log n})$ approximation for finding minimum ratio edge cuts, based on a known semidefinite relaxation of the problem, and a fundamentally new technique for exploiting the global structure of

the solution. This approach, combined with the embedding technique of Krauthgamer et al. [32], has been extended further to obtain approximation algorithms for minimum ratio edge cuts with nonuniform demands. In particular, using [7, 32] and the quantitative improvements of Lee [34], Chawla, Gupta, and Räcke [17] exhibit an $O(\log n)^{3/4}$ approximation. More recently, Arora, Lee, and Naor [6] have improved this bound almost to that of the uniform case, yielding an approximation ratio of $O(\sqrt{\log n}\log\log n)$.

On the other hand, progress on the vertex case has been significantly slower. In the sections that follow, we attempt to close this gap by providing new techniques for finding approximately optimal vertex separators.

Since the initial (conference) publication of this manuscript, we have learned of two other papers which contain some independently discovered, overlapping results. All three papers first appeared in STOC 2005. In particular, the work of Agarwal et. al. [1] gives an $O(\sqrt{\log n})$-approximation for a *directed* version of the Sparsest Cut problem which implies a similar result for vertex cuts by a well-known reduction (see, e.g., [36]). Their algorithm is also based on rounding an SDP (though they use a different relaxation). Second, the paper of Chekuri, Khanna, and Shepherd [18] shows that the max-multicommodity-flow/min-vertex-cut gap for product demands *in planar graphs* is bounded by a universal constant. As discussed later, we prove this theorem not only for planar graphs but also for any *excluded-minor family of graphs*.

1.2. Results and techniques. In section 2, we introduce a new semidefinite relaxation for the problem of finding minimum ratio vertex cuts in a general graph. In preparation for applying the techniques of [7], the relaxation includes so-called *triangle inequality* constraints on the variables. The program contains strictly more than one variable per vertex of the graph, but the SDP is constructed carefully to lead to a single metric of negative type[1] on the vertices that contains all the information necessary to perform the rounding.

In section 3, we exhibit a general technique for rounding the solution to optimization problems involving "fractional" vertex cuts. These are based on the ability to find line embeddings with small *average* distortion, as defined by Rabinovich [41] (though we extend his definition to allow for arbitrary weights in the average). In [41], it is proved that one can obtain line embeddings with constant average distortion for metrics supported on planar graphs. This is observed only as an interesting structural fact, without additional algorithmic consequences over the known average distortion embeddings into all of L_1 [42, 31]. For the vertex case, we will see that this additional structure is crucial.

Using the seminal results of [7], which can be viewed as a line embedding (see section A.2), we then show that the solution of the semidefinite relaxation can be rounded to a vertex separator whose ratio is within $O(\sqrt{\log n})$ of the optimal separator. For the SDP used in [7] for approximating minimum ratio edge cuts, only a constant lower bound is known for the integrality gap. Recent work of Khot and Vishnoi [30] shows that in the nonuniform demand case, the integrality gap must tend to infinity with the size of the instance. In contrast, we show that our analysis is tight by exhibiting an $\Omega(\sqrt{\log n})$ integrality gap for the SDP in section 5. Interestingly, this gap is achieved by an L_1 metric. This shows that L_1 metrics are not as intimately connected to vertex cuts as they are to edge cuts and that the use of the structural

[1] A metric space (X, d) is said to be of *negative type* if $d(x, y) = ||f(x) - f(y)||^2$ for some map $f : X \to L_2$.

theorems discussed in the previous paragraph is crucial to obtaining an improved bound.

We exhibit an $O(\log k)$-approximate max-flow/min-vertex-cut theorem for general instances with k commodities. The best previous bound of $O(\log^3 k)$ is due to [22] (they actually show this bound for directed instances with symmetric demands, but this implies the vertex case). The result is proved in section 4. A well-known reduction shows that this theorem implies the edge version of [38, 8] as a special case. Again, our rounding makes use of the general tools developed in section 3 based on average-distortion line embeddings. In sections 4.2 and 4.4, we show that any approach based on low-distortion L_1 embeddings and Euclidean embeddings, respectively, must fail since the integrality gap can be very large even for spaces admitting such embeddings. Using the improved line embeddings for metrics on graphs which exclude a fixed minor [41] (based on [31] and [42]), we also achieve a constant-factor approximation for finding minimum ratio vertex cuts in these families. This answers an open problem asked in [19].

By improving the approximation ratios for balanced vertex separators in general graphs and graphs excluding a fixed minor, we improve the approximation factors for a number of problems relating to graph-theoretic decompositions such as treewidth, branchwidth, and pathwidth. For instance, we show that in any graph of treewidth k, we can find a tree decomposition of width at most $O(k\sqrt{\log k})$. This improves upon the $O(\log n)$-approximation of Bodlaender et al. [11] and the $O(\log k)$-approximation algorithm of Amir [4]. A result of Seymour and Thomas [44] shows that a decomposition of width $1.5k$ can be found efficiently in planar graphs. We offer a significant generalization by giving an algorithm that finds a decomposition of width $O(k)$ whenever the input graph excludes a fixed minor. See section 6.3 and Theorem 6.4 and Corollary 6.5 for a discussion of these problems, along with salient definitions, and a list of the problems to which our techniques apply.

Improving the approximation factor for treewidth in general graphs and graphs excluding a fixed minor to $O(\sqrt{\log n})$ and $O(1)$, respectively, answers an open problem of [19] and leads to an improvement in the running time of approximation schemes and subexponential fixed parameter algorithms for several NP-hard problems on graphs excluding a fixed minor. For instance, we obtain the first polynomial-time approximation schemes (PTASs) for problems like *minimum feedback vertex set* and *connected dominating set* in such graphs (see Theorem 6.6 for more details). Finally, our techniques yield an $O(g)$-approximation algorithm for the vertex separator problem in graphs of genus at most g. It is known that such graphs have balanced separators of size $O(\sqrt{gn})$ [25] and that these separators can be found efficiently [28] (earlier, [3] gave a more general algorithm which, in particular, finds separators of size $O(\sqrt{g^{3/2}n})$). Our approximation algorithms thus find separators of size $O(\sqrt{g^3 n})$, but when the graph at hand has a smaller separator, our algorithms perform much better than the worst-case bounds of [25, 3, 28].

2. A vector program for minimum ratio vertex cuts. Let $G = (V, E)$ be a graph with nonnegative vertex weights $\pi : V \to [1, \infty)$. For a subset $U \subseteq V$, we write $\pi(U) = \sum_{u \in U} \pi(u)$. A vertex separator partitions the graph into three parts, S (the set of vertices in the separator), A, and B (the two parts that are separated). We use the convention that $\pi(A) \geq \pi(B)$. We are interested in finding separators that minimize the ratio of the "cost" of the separator to its "benefit." Here, the cost of a separator is simply $\pi(S)$. As to the benefit of a separator, it turns out that there is more than one natural way in which one can define it. The distinction between the

various definitions is relatively unimportant whenever $\pi(S) \le \pi(B)$, but it becomes significant when $\pi(S) > \pi(B)$. We elaborate on this below.

In analogy to the case of edge separators, one may wish to take the benefit to be $\pi(B)$. Then we would like to find a separator that minimizes the ratio $\pi(S)/\pi(B)$. However, there is evidence that no polynomial-time algorithm can achieve an approximation ratio of $O(|V|^{\delta})$ for this problem (for some $\delta > 0$). See section A.1 for details.

For the use of separators in divide and conquer algorithms, the benefit is in the reduction in size of the parts that remain. This reduction is $\pi(B) + \pi(S)$ rather than just $\pi(B)$, and the quality of a separator is defined to be

$$\frac{\pi(S)}{\pi(B) + \pi(S)}.$$

This definition is used in the introduction to our paper and in some other earlier work (e.g., [5]).

As a matter of convenience, we use a slightly different definition. We shall define the *sparsity* of a separator to be

$$\alpha_\pi(A, B, S) = \frac{\pi(S)}{\pi(A \cup S) \cdot \pi(B \cup S)}.$$

Under our convention that $\pi(A) \ge \pi(B)$, we have that $\pi(V)/2 \le \pi(A \cup S) \le \pi(V)$, and the two definitions differ by a factor of $\Theta(\pi(V))$.

We define $\alpha_\pi(G)$ to be the minimum over all vertex separators (A, B, S) of $\alpha_\pi(A, B, S)$. The problem of computing $\alpha_\pi(G)$ is NP-hard (see [15]). Our goal is to give algorithms for finding separators (A, B, S) for which $\alpha_\pi(A, B, S) \le O(\sqrt{\log k})\, \alpha_\pi(G)$, where $k = |\mathrm{supp}(\pi)|$ is the number of vertices with nonzero weight in G.

Let us pause for a moment to discuss an aspect of approximation algorithms for $\alpha_\pi(G)$ that is often overlooked. The optimal solution minimizing $\alpha_\pi(A, B, S)$ is indeed a nontrivial separator in the sense that both A and B are nonempty (unless the underlying graph G is a clique). However, when $\pi(S)$ is large relative to $\pi(B)$ in the optimal separator, sets S', B' that only approximately minimize $\alpha_\pi(A', B', S')$ might correspond to trivial separators in the sense that B' is empty. Hence approximation algorithms for $\alpha_\pi(G)$ might return trivial separators rather than nontrivial ones. Whenever this happens, we assume as a convention that the algorithm instead returns a minimum weight vertex cut in G. These cuts are nontrivial and can be found in polynomial time (see section 3 for example), and the corresponding value of $\alpha_\pi(A, B, S)$ is not larger than that for any trivial separator. (In fact, for trivial separators $\alpha_\pi(A, B, S) = 1/\pi(V)$, whereas for every nontrivial separator, whether optimal or not, one always has $\alpha_\pi(A, B, S) \le 1/\pi(V)$.)

Before we move on to the main algorithm, let us define

$$\tilde{\alpha}_\pi(A, B, S) = \pi(S)/[\pi(A) \cdot \pi(B \cup S)].$$

Note that $\alpha_\pi(A, B, S)$ and $\tilde{\alpha}_\pi(A, B, S)$ are equivalent up to a factor of 2 whenever $\pi(A) \ge \pi(S)$. Hence in this case it will suffice to find a separator (A, B, S) with $\alpha_\pi(A, B, S) \le O(\sqrt{\log k})\, \tilde{\alpha}_\pi(G)$. Allowing ourselves to compare $\alpha_\pi(A, B, S)$ to $\tilde{\alpha}_\pi(G)$ rather than $\alpha_\pi(G)$ eases the formulation of the semidefinite relaxations that follow. When $\pi(S) > \pi(A)$, $\tilde{\alpha}$ no longer provides a good approximation to α. However, in this case $\pi(S) > \pi(B)$, and returning a minimum weight vertex cut provides a constant-factor approximation to $\alpha_\pi(G)$.

2.1. The quadratic program. We present a quadratic program for the problem of finding minimum ratio vertex cuts. All constraints in this program involve only terms that are quadratic (products of two variables). Our goal is for the value of the quadratic program to be equal to $\tilde{\alpha}_\pi(G)$. Let $G = (V, E)$ be a vertex-weighted graph, and let (A^*, B^*, S^*) be an optimal separator according to $\tilde{\alpha}_\pi(\cdot)$, i.e., such that $\tilde{\alpha}_\pi(G) = \tilde{\alpha}_\pi(A^*, B^*, S^*)$.

With every vertex $i \in V$, we associate three indicator 0/1 variables, x_i, y_i, and s_i. It is our intention that for every vertex exactly one indicator variable will have the value 1 and that the other two will have value 0. Specifically, $x_i = 1$ if $i \in A^*$, $y_i = 1$ if $i \in B^*$, and $s_i = 1$ if $i \in S^*$. To enforce this, we formulate the following two sets of constraints.

Exclusion constraints. These force at least two of the indicator variables to be 0:

$$x_i \cdot y_i = 0,\ x_i \cdot s_i = 0,\ y_i \cdot s_i = 0 \text{ for every } i \in V.$$

Choice constraints. These force the nonzero indicator variable to have value 1:

$$x_i^2 + y_i^2 + s_i^2 = 1 \text{ for all } i \in V.$$

The combination of exclusion and choice constraints implies the following *integrality constraints*, which we formulate here for completeness, even though they are not explicitly included as part of the quadratic program: $x_i^2 \in \{0, 1\}$, $y_i^2 \in \{0, 1\}$, $s_i^2 \in \{0, 1\}$ for all $i \in V$.

Edge constraints. This set of $2|E|$ constraints expresses the fact that there are no edges connecting A and B:

$$x_i \cdot y_j = 0 \text{ and } x_j \cdot y_i = 0 \text{ for all } (i, j) \in E.$$

Now we wish to express the fact that we are minimizing $\tilde{\alpha}_\pi(A, B, S)$ over all vertex separators (A, B, S). To simplify our presentation, it will be convenient to assume that we know the value $K = \pi(A^*) \cdot \pi(B^* \cup S^*)$. We can make such an assumption because the value of K can be guessed (since eventually we will need only to know K within a factor of 2, say, there are only $O(\log \pi(V))$ different values to try). Alternatively, the assumption can be dropped at the expense of a more complicated relaxation.

Spreading constraint. The following constraint expresses our guess for the value of K:

$$\tfrac{1}{2} \sum_{i,j \in V} \pi(i)\pi(j)(x_i - x_j)^2 \geq K.$$

Notice that $(x_i - x_j)^2 = 1$ if and only if $\{x_i, x_j\} = \{0, 1\}$.

The objective function. Finally, we write the objective we are trying to minimize:

$$\text{minimize} \qquad \frac{1}{K} \sum_{i \in V} \pi(i) s_i^2.$$

The above quadratic program computes exactly the value of $\tilde{\alpha}_\pi(G)$ and hence is NP-hard to solve.

2.2. The vector relaxation. We relax the quadratic program of section 2.1 to a "vector" program that can be solved up to arbitrary precision in polynomial time. The relaxation involves two aspects.

Interpretation of variables. All variables are allowed to be arbitrary vectors in $\mathbb{R}^d$, rather than in $\mathbb{R}$. The dimension d is not constrained and might be as large as the number of variables (i.e., $3n$).

Interpretation of products. The original quadratic program involved products over pairs of variables. Every such product is interpreted as an inner product between the respective vector variables. The exclusion constraints merely force vectors to be orthogonal (rather than forcing one of them to be 0), and the integrality constraints are no longer implied by the exclusion and choice constraints. The choice constraints imply (among other things) that no vector has norm greater than 1, and the edge constraints imply that whenever $(i, j) \in E$, the corresponding vectors x_i and y_j are orthogonal.

2.3. Adding valid constraints. We now strengthen the vector program by adding more valid constraints. This should be done in a way that will not violate feasibility (in cases where the original quadratic program was feasible) and, moreover, that preserves polynomial-time solvability (up to arbitrary precision) of the resulting vector program. It is known that this last condition is satisfied if we add only constraints that are linear over inner products of pairs of vectors, and this is indeed what we shall do. The reader is encouraged to check that every constraint that we add is indeed satisfied by feasible 0/1 solutions to the original quadratic program.

The 1*-vector.* We add the additional variable v to the vector program. It is our intention that variables whose value is 1 in the quadratic program will have value equal to that of v in the vector program. Hence v is a unit vector, and we add the constraint $v^2 = 1$.

Sphere constraints. For every vector variable z we add the constraint $z^2 = v \cdot z$. Geometrically, this forces all vectors to lie on the surface of a sphere of radius $\frac{1}{2}$ centered at $\frac{v}{2}$ because the constraint is equivalent to $(z - \frac{v}{2})^2 = \frac{1}{4}$.

Triangle constraints. For every three variables z_1, z_2, z_3 we add the constraint

$$(z_1 - z_2)^2 + (z_2 - z_3)^2 \geq (z_1 - z_3)^2.$$

This implies that every three variables (which are points on the sphere $S(\frac{v}{2}, \frac{1}{2})$) form a triangle whose angles are all at most $\pi/2$. We remark that we shall eventually use only those triangle constraints in which all three variables are x variables.

Removing the s_i vectors. In the upcoming sections we shall describe and analyze a rounding procedure for our vector program. It turns out that our rounding procedure does not use the vectors s_i—only the values $s_i^2 = 1 - x_i^2 - y_i^2$. Hence we can modify the choice constraints to

$$x_i^2 + y_i^2 \leq 1$$

and remove all explicit mention of the s_i vectors, without affecting our analysis for the rounding procedure. The full vector program follows.

$$
\begin{array}{lll}
\text{minimize} & \frac{1}{K}\sum_{i\in V}\pi(i)(1-x_i^2-y_i^2) & \\
\\
\text{subject to} & x_i, y_i, v \in \mathbb{R}^{2n}, & i \in V, \\
& x_i^2 + y_i^2 \le 1, & i \in V, \\
& x_i \cdot y_i = 0, & i \in V, \\
& x_i \cdot y_j = x_j \cdot y_i = 0, & (i,j) \in E, \\
& v^2 = 1, & \\
& v \cdot x_i = x_i^2,\ v \cdot y_i = y_i^2, & i \in V, \\
& \frac{1}{2}\sum_{i,j\in V}\pi(i)\pi(j)(x_i - x_j)^2 \ge K, & \\
& (x_i - x_j)^2 \le (x_i - x_h)^2 + (x_h - x_j)^2, & h,i,j \in V.
\end{array}
$$

In the following section, we will show how to use this SDP to obtain a solution which is within an $O(\sqrt{\log k})$ factor of the best vertex separator. In section 5, we show that this analysis is tight, even for a family of stronger (i.e. more constrained) vector programs.

3. Algorithmic framework for rounding. In this section, we develop a general algorithmic framework for rounding solutions to optimization problems on vertex cuts.

3.1. Capacities and demands. In the vector program of section 2, vertices have weights π. These weights served two purposes. One was as a measure of cost for the separator (we are charged $\pi(S)$ in the numerator of α_π). The other was as a measure of benefit of the separator (we get credit of $\pi(A \cup S)\pi(B \cup S)$ in the denominator). Here, we shall not insist on having one weight function serving both purposes. Instead, we allow the cost to be measured with respect to one weight function (say, π_1), and the benefit to be measured with respect to another weight function (say, π_2). It is customary to call these functions *capacity* and *demand*. Let us provide more details.

Vertices are assumed to have nonnegative capacities $\{c_v\}_{v\in V} \subseteq \mathbb{N}$. For simplicity of presentation, we are assuming here that capacities are integer, but all results of this paper can also be extended to the case of arbitrary nonnegative capacities. For a subset $S \subseteq V$, we define $\mathrm{cap}(S) = \sum_{v\in S} c_v$.

In its most general form, we have a *demand function* $\omega : V \times V \to \mathbb{R}_+$ which is *symmetric*, i.e. $\omega(u,v) = \omega(v,u)$. In interesting special cases, this demand function is induced by weights $\pi_2 : V \to \mathbb{R}_+$ via the relation $\omega(u,v) = \pi_2(u)\pi_2(v)$ for all $u, v \in V$.

Given a capacity function and a demand function, we define the *sparsity of* (A,B,S) by

$$\alpha^{\mathrm{cap},\omega}(A,B,S) = \frac{\mathrm{cap}(S)}{\sum_{u\in A\cup S}\sum_{v\in B\cup S}\omega(u,v)}.$$

We define the *sparsity of* G by $\alpha^{\mathrm{cap},\omega}(G) = \min\{\alpha^{\mathrm{cap},\omega}(A,B,S)\}$ where the minimum is taken over all vertex separators. Note that $\alpha_\pi(A,B,S) = \alpha^{\mathrm{cap},\omega}(A,B,S)$ when $c_v = \pi(v)$ and $\omega(u,v) = \pi(u)\pi(v)$ for all $u, v \in V$.

3.2. Line embeddings and distortion. A key feature of the vector program is that its solution is a set of vectors in high dimensional Euclidean space $\mathbb{R}^{2n}$. Moreover, the triangle constraints imply that for the x_i vectors, the square of their Euclidean distance also forms a metric. Technically, such a metric is said to be of *negative type*. Our rounding framework is based on properties of metric spaces.

Let (X, d) be a metric space. A map $f : X \to \mathbb{R}$ is called 1-*Lipschitz* if, for all $x, y \in X$,

$$|f(x) - f(y)| \leq d(x, y).$$

Given a 1-Lipschitz map f and a demand function $\omega : X \times X \to \mathbb{R}_+$, we define its *average distortion under* ω by

$$avd_\omega(f) = \frac{\sum_{x,y \in X} \omega(x, y) \cdot d(x, y)}{\sum_{x,y \in X} \omega(x, y) \cdot |f(x) - f(y)|}.$$

We say that a weight function ω is a *product weight* if it can be written as $\omega(x, y) = \pi(x)\pi(y)$ for all $x, y \in X$, for some $\pi : X \to \mathbb{R}_+$. We now state three theorems which give line embeddings of small average distortion in various settings. The proofs of these theorems are sketched in section A.2.

THEOREM 3.1 (Bourgain [14]). *If (X, d) is an n-point metric space, then for every weight function $\omega : X \times X \to \mathbb{R}_+$, there exists an efficiently computable* 1*-Lipschitz map $f : X \to \mathbb{R}$ with $avd_\omega(f) = O(\log n)$.*

THEOREM 3.2 (Rabinovich [41]). *If (X, d) is any metric space supported on a graph which excludes a K_r-minor, then for every* product weight $\omega_0 : X \times X \to \mathbb{R}_+$*, there exists an efficiently computable* 1*-Lipschitz map $f : X \to \mathbb{R}$ with $avd_{\omega_0}(f) = O(r^2)$.*

THEOREM 3.3 (Arora, Rao, and Vazirani [7]). *If (X, d) is an n-point metric of negative type, then for every* product weight $\omega_0 : X \times X \to \mathbb{R}_+$*, there exists an efficiently computable* 1*-Lipschitz map $f : X \to \mathbb{R}$ with $avd_{\omega_0}(f) = O(\sqrt{\log n})$.*

We also recall the following classical result.

LEMMA 3.4. *Let (Y, d) be any metric space and $X \subseteq Y$. Given a* 1*-Lipschitz map $f : X \to \mathbb{R}$, there exists a* 1*-Lipschitz extension $\tilde{f} : Y \to \mathbb{R}$, i.e., such that $\tilde{f}(x) = f(x)$ for all $x \in X$.*

Proof. One defines

$$\tilde{f}(y) = \sup_{x \in X} [f(x) - d(x, y)]$$

for all $y \in Y$. ∎

3.3. Menger's theorem. The following classical theorem is an important ingredient in our rounding framework.

THEOREM 3.5 (Menger's theorem). *A graph $G = (V, E)$ contains at least k vertex-disjoint paths between two nonadjacent vertices $u, v \in V$ if and only if every vertex cut that separates u from v has size at least k.*

It is well known that a smallest vertex cut separating u from v can be found in polynomial time (in the size of G) by deriving Menger's theorem from the max-flow/min-cut theorem (see, e.g., [45]).

Suppose that, in addition to a graph $G = (V, E)$, we have a set of nonnegative vertex capacities $\{c_v\}_{v \in V} \subseteq \mathbb{N}$. (For simplicity, we are assuming here that capacities are integers.) For a subset $S \subseteq V$, we define $\text{cap}(S) = \sum_{v \in S} c_v$. We have the following immediate corollary.

COROLLARY 3.6. *Let $G = (V, E)$ be a graph with vertex capacities. Then for any two nonadjacent vertices $u, v \in V$, the following two statements are equivalent:*

1. *Every vertex cut $S \subseteq V$ that separates u from v has* $\text{cap}(S) \geq k$.

2. *There exist u-v paths $p_1, p_2, \ldots, p_k \subseteq V$ such that for every $w \in V$,*

$$\#\{1 \le i \le k : w \in p_i\} \le c_w.$$

Furthermore, a vertex cut S of minimal capacity can be found in polynomial time.

Proof. The proof is by a simple reduction. From $G = (V, E)$ and the capacities $\{c_v\}_{v \in V}$, we create a new uncapacitated instance $G' = (V', E')$ and then apply Menger's theorem to G'.

To arrive at G', we replace every vertex $v \in V$ with a collection of representatives $v_1, v_2, \ldots, v_{c_v}$ (if $c_v = 0$, then this corresponds to deleting v from the graph). Now for every edge $(u, v) \in E$, we add edges $\{(u_i, v_j) : 1 \le i \le c_u,\ 1 \le j \le c_v\}$. It is not hard to see that every minimal vertex cut takes either all representatives of a vertex or none, giving a one-to-one correspondence between minimal vertex cuts in G and G'. ∎

Furthermore, given such a capacitated instance $G = (V, E)$, $\{c_v\}_{v \in V}$, along with $u, v \in V$, it is possible to find, in polynomial time, a vertex cut $S \subseteq V$ of minimal capacity which separates u from v.

3.4. Line embeddings and vertex separators. Having presented the tools that we shall be using (line embeddings, Menger's theorem), we present here an algorithmic framework based on an arbitrary line embedding $f : V \to \mathbb{R}$ for finding a vertex cut. Different instantiations of this algorithm may use different line embeddings f. The analysis of this algorithm will use, among other things, Menger's theorem. It will also involve a certain cost function $\mathsf{cost} : V \to \mathbb{R}_+$ that is left unspecified in this section. However, in later sections (e.g., section 3.5) the cost of a vertex will be instantiated to be the contribution of the vertex to the objective function of a relaxation of the minimum vertex separator problem (e.g., $\pi(i)(1 - x_i^2 - y_i^2)$ in the vector program). The key technical property of the algorithm is summarized in Lemma 3.7, and it relates the cost (which is the value of the relaxation) to the sparsity of the cut found by the algorithm. Hence Lemma 3.7 can be used in order to analyze the approximation ratio of algorithms that use this algorithmic framework.

Let $G = (V, E)$ be a graph with vertex capacities $\{c_v\}_{v \in V}$ and a demand function $\omega : V \times V \to \mathbb{R}_+$. Furthermore, suppose that we have a map $f : V \to \mathbb{R}$. We give the following algorithm, which computes a vertex cut (A, B, S) in G.

Algorithm FindCut(G, f)

1. Sort the vertices $v \in V$ according to the value of $f(v)$: $\{y_1, y_2, \ldots, y_n\}$.
2. For each $1 \le i \le n$,
3. Create the augmented graph $G_i = (V \cup \{s, t\}, E_i)$ with $E_i = E \cup \{(s, y_j), (y_k, t) : 1 \le j \le i,\ i < k \le n\}$.
4. Find the minimum capacity s-t separator S_i in G_i.
5. Let $A_i \cup \{s\}$ be the component of $G[V \cup \{s, t\} \setminus S_i]$ containing s, and let $B_i = V \setminus (A_i \cup S_i)$.
6. Output the vertex separator (A_i, B_i, S_i) for which $\alpha^{\mathrm{cap},\omega}(A_i, B_i, S_i)$ is minimal.

The analysis. Suppose that we have a cost function $\mathsf{cost} : V \to \mathbb{R}_+$. We say that the map $f : V \to \mathbb{R}$ is *edge-compatible with the cost function* cost if, for any $(u, v) \in E$, we have

$$|f(u) - f(v)| \le \frac{\mathsf{cost}(u) + \mathsf{cost}(v)}{2}. \tag{1}$$

We now move on to the main lemma of this section.

LEMMA 3.7 (charging lemma). *Let $G = (V, E)$ be any capacitated graph with demand function $\omega : V \times V \to \mathbb{R}_+$. Suppose additionally that we have a cost function $\mathsf{cost} : V \to \mathbb{R}_+$ and an edge-compatible map $f : V \to \mathbb{R}$. If α_0 is the sparsity of the minimum ratio vertex cut output by* FINDCUT(G, f)*, then*

$$\sum_{v \in V} c_v \cdot \mathsf{cost}(v) \geq \alpha_0 \sum_{u,v \in V} \omega(u, v)|f(u) - f(v)|.$$

Proof. Recall that we have sorted the vertices v according to the value of $f(v)$: $\{y_1, y_2, \ldots, y_n\}$. Let $C_i = \{y_1, \ldots, y_i\}$ and $\varepsilon_i = f(y_{i+1}) - f(y_i)$. First we have the following lemma which relates the size of the separators found to the average distance under f, according to ω.

LEMMA 3.8.

$$\sum_{i=1}^{n-1} \varepsilon_i \operatorname{cap}(S_i) \geq \alpha_0 \sum_{u,v \in V} \omega(u, v)|f(u) - f(v)|.$$

Proof. Using the fact that α_0 is the minimum sparsity of all cuts found by FINDCUT(G, f),

$$\begin{aligned} \operatorname{cap}(S_i) &\geq \alpha_0 \sum_{u \in A_i \cup S_i} \sum_{v \in B_i \cup S_i} \omega(u, v) \\ &\geq \alpha_0 \sum_{u \in C_i} \sum_{v \in V \setminus C_i} \omega(u, v). \end{aligned}$$

Note that the second inequality follows from the fact in FINDCUT(G, f) that since C_i contains A_i and $V \setminus C_i$ contains B_i, $A_i \cup S_i$ contains C_i and $B_i \cup S_i$ contains $V \setminus C_i$.

Multiplying both sides of the previous inequality by ε_i and summing over $i \in \{1, 2, \ldots, n-1\}$ prove the lemma. ∎

Now we come to the heart of the charging argument which relates the cost function to the capacity of the cuts occurring in the algorithm.

LEMMA 3.9 (charging against balls).

$$\sum_{v \in V} c_v \cdot \mathsf{cost}(v) \geq \sum_{i=1}^{n-1} \varepsilon_i \operatorname{cap}(S_i).$$

Proof. We first present an interpretation of the quantity $\sum_{i=1}^{n-1} \varepsilon_i \operatorname{cap}(S_i)$. Consider a nonnegative function g defined on the line segment $[f(y_1), f(y_n)]$ whose value at point z is defined as $g(z) = \operatorname{cap}(S_i)$, where i is the unique value such that z is in the half open interval $[f(y_i), f(y_{i+1}))$. Then $\sum_{i=1}^{n-1} \varepsilon_i \operatorname{cap}(S_i)$ is precisely $\int_{\mathbb{R}} g$.

Now, for every v, we present an interpretation of $c_v \cdot \mathsf{cost}(v)$. Consider a nonnegative function g_v whose value is c_v on the interval $[f(v) - \frac{1}{2}\mathsf{cost}(v),\ f(v) + \frac{1}{2}\mathsf{cost}(v)]$ and 0 elsewhere. Then $c_v \cdot \mathsf{cost}(v)$ is precisely $\int_{\mathbb{R}} g_v$. We shall refer to the support of g_v as the *ball* of v (as it is a ball centered at $f(v)$ of radius $\frac{1}{2}\mathsf{cost}(v)$).

Lemma 3.9 can now be rephrased as

$$\int_{\mathbb{R}} g(z)\, dz \leq \sum_v \int_{\mathbb{R}} g_v(z)\, dz.$$

We shall prove this inequality pointwise.

Consider an arbitrary point z, belonging to an arbitrary interval $[f(y_i), f(y_{i+1}))$. Since S_i is a minimum capacity s-t separator, applying Menger's theorem yields a family of s-t paths $p_1, \ldots, p_m$ (where $m = \mathrm{cap}(S_i)$) which use no vertex $v \in V$ more than c_v times. We view each of these paths as contributing 1 to the value of $g(z)$, and hence fully accounting for the value $g(z) = \mathrm{cap}(S_i)$. We now consider the contribution of these paths to the functions g_v.

Consider an arbitrary such path p_j. Since it crosses from C_i to $V \setminus C_i$, there must exist two consecutive vertices along the path (say, u and v) such that $u \in C_i$ and $v \in V \setminus C_i$. The fact that f is edge-compatible with cost implies that the union of the balls of u and v covers the interval $[f(u), f(v)]$ that includes the interval $[f(y_i), f(y_{i+1}))$. Hence z is in at least one of these two balls (say, the ball of v), and then we have p_j contribute one unit to $g_v(z)$. Note that the total contribution of the m disjoint paths to $g_v(z)$ can be at most c_v, because v can occur in at most c_v of these paths.

In summary, based on the disjoint paths, we provided a charging mechanism that accounts for all of $g(z)$, and charges at least as much to $\sum_v g_v(z)$ without exceeding the respective values c_v. This completes the proof of Lemma 3.9. ∎

Combining Lemmas 3.8 and 3.9 finishes the proof of Lemma 3.7. ∎

3.5. Analysis of the vector program. We now continue our analysis of the vector program from section 2.3. Recall that $\pi(i)(1 - x_i^2 - y_i^2)$ is the contribution of vertex i to the objective function. For every $i \in V$, define $\mathsf{cost}(i) = 4(1 - x_i^2 - y_i^2)$. We will consider the metric space (V, d) given by $d(i, j) = (x_i - x_j)^2$ (note that this is a metric space precisely because every valid solution to the SDP must satisfy the triangle inequality constraints). The following key proposition allows us to apply the techniques of sections 3.4 and 3.2 to the solution of the vector program.

PROPOSITION 3.10. *For every edge* $(i, j) \in E$, $(x_i - x_j)^2 \le \frac{\mathsf{cost}(i)+\mathsf{cost}(j)}{2}$.

Proof. Since $(i, j) \in E$, we have $x_i \cdot y_j = x_j \cdot y_i = 0$, and recall that $x_i \cdot y_i = x_j \cdot y_j = 0$. It follows that

$$(x_i - x_j)^2 \le 2[(x_i + y_i - v)^2 + (x_j + y_i - v)^2] \le 2[(1 - x_i^2 - y_i^2) + (1 - x_j^2 - y_i^2)].$$

Note that the first inequality above follows from the fact that $(x_i - x_j)^2 = ((x_i + y_i - v) - (x_j + y_i - v))^2$ and the inequality $(x - y)^2 \le 2(x^2 + y^2)$. Substitute $x = x_i + y_i - v$ and $y = x_j + y_i - v$. Then the second inequality follows from the constraints $vx_i = x_i^2$ and $vy_i = y_i^2$.

Putting y_j instead of y_i in the above equation gives $(x_i - x_j)^2 \le 2[(1 - x_i^2 - y_j^2) + (1 - x_j^2 - y_j^2)]$. Summing these two inequalities yields

$$2(x_i - x_j)^2 \le 4[(1 - x_i^2 - y_i^2) + (1 - x_j^2 - y_j^2)] = \mathsf{cost}(i) + \mathsf{cost}(j). \qquad (2)$$ ∎

Now, let $U = \mathrm{supp}(\pi) = \{i \in V : \pi(i) \neq 0\}$, and put $k = |U|$. Finally, let $f : (U, d) \to \mathbb{R}$ be any 1-Lipschitz map, and let $\tilde{f} : V \to \mathbb{R}$ be the 1-Lipschitz extension guaranteed by Lemma 3.4.

Then for any edge $(u, v) \in E$, we have

$$|\tilde{f}(u) - \tilde{f}(v)| \le d(u, v) = (x_u - x_v)^2 \le \frac{\mathsf{cost}(u) + \mathsf{cost}(v)}{2},$$

where the final inequality is from Proposition 3.10. We conclude that $\tilde{f}$ is path-compatible with cost.

Defining a product demand by $\omega(i,j) = \pi(i)\pi(j)$ for every $i, j \in V$ and capacities $c_i = \pi(i)$, we now apply FindCut$(G, \tilde{f})$. If the best separator found has sparsity α_0, then by Lemma 3.7,

$$\begin{aligned}\frac{1}{K}\sum_{i\in V}\pi(i)(1-x_i^2-y_i^2) = \frac{1}{4K}\sum_{i\in V} c_i \cdot \mathsf{cost}(i) &\geq \frac{\alpha_0}{4K}\sum_{i,j\in V}\omega(i,j)\cdot|\tilde{f}(i)-\tilde{f}(j)| \\ &= \frac{\alpha_0}{4K}\sum_{i,j\in U}\omega(i,j)\cdot|f(i)-f(j)| \\ &\geq \frac{\alpha_0}{2}\cdot\frac{\sum_{i,j\in U}\omega(i,j)\cdot|f(i)-f(j)|}{\sum_{i,j\in U}\omega(i,j)\cdot d(i,j)} \\ &= \frac{\alpha_0}{2\cdot avd_\omega(f)}.\end{aligned}$$

It follows that $\tilde{\alpha}_\pi(G) \geq \alpha_0/(2\cdot avd_\omega(f))$. Since the metric (V,d) is of negative type and $\omega(\cdot,\cdot)$ is a product weight, we can achieve $avd_\omega(f) = O(\sqrt{\log k})$ using Theorem 3.3. Using this f, it follows that FindCut$(G,\tilde{f})$ returns a separator (A,B,S) such that $\alpha_\pi(A,B,S) \leq O(\sqrt{\log k})\,\tilde{\alpha}_\pi(G)$, completing the analysis.

Theorem 3.11. *Given a graph $G=(V,E)$ and vertex weights $\pi: V\to\mathbb{R}_+$, there exists a polynomial-time algorithm which computes a vertex separator (A,B,S) for which*

$$\alpha_\pi(A,B,S) \leq O(\sqrt{\log k})\,\alpha_\pi(G),$$

where $k = |\mathrm{supp}(\pi)|$.

In the next section, we extend this theorem to more general weights. This is necessary for some of the applications in section 6.3.

3.6. More general weights. An important generalization of the minimum ratio vertex cut introduced in section 2 is when a pair of weight functions $\pi_1, \pi_2 : V \to \mathbb{R}_+$ is given and one wants to find the vertex separator (A,B,S) which minimizes

$$\alpha_{\pi_1,\pi_2}(A,B,S) = \frac{\pi_1(S)}{\pi_2(A\cup S)\cdot\pi_2(B\cup S)},$$

where, as a convention, $\pi_2(B) \leq \pi_2(A)$. We let $\alpha_{\pi_1,\pi_2}(G)$ denote the minimum value of $\alpha_{\pi_1,\pi_2}(A,B,S)$ in graph G. Under a common interpretation, π_1 denotes vertex capacities, π_2 induces a *demand* (one needs to route $\pi_2(u)\pi_2(v)$ units of flow between vertices u and v), and then the value of $\alpha_{\pi_1,\pi_2}(G)$ serves as an upper bound on the fraction of demand that can be routed subject to the capacity constraints on the vertices.

In analogy to the discussion in section 2, call a separator *trivial* if $\pi_2(B) = 0$ (and, in particular, when B is empty). Unlike the case in section 2, when π_1 differs from π_2 it may happen that $\alpha_{\pi_1,\pi_2}(G)$ is obtained by a trivial separator. Hence in the current section, algorithms that minimize (or approximately minimize) $\alpha_{\pi_1,\pi_2}(A,B,S)$ are allowed to return a trivial separator.

We now explain how our approximation algorithm can be extended to give an $O(\sqrt{\log k})$ approximation for $\alpha_{\pi_1,\pi_2}(G)$, where here $k = |\mathrm{supp}(\pi_2)|$.

Let

$$\tilde{\alpha}_{\pi_1,\pi_2}(A,B,S) = \pi_1(S)/[\pi_2(A)\cdot\pi_2(B\cup S)],$$

where $\pi_2(A) \geq \pi_2(B)$. Also define $\alpha_{\pi_1,\pi_2}(G)$ and $\tilde{\alpha}_{\pi_1,\pi_2}(G)$ as before. By changing the vector program so that K is defined in terms of π_2 and the objective is to minimize $\frac{1}{K}\sum_{i\in V}\pi_1(i)(1-x_i^2-y_i^2)$, it becomes a relaxation for $\tilde{\alpha}_{\pi_1,\pi_2}(G)$. The rounding analysis goes through unchanged to yield a separator (A,B,S) with

$$\alpha_{\pi_1,\pi_2}(A,B,S) \leq O(\sqrt{\log k})\,\tilde{\alpha}_{\pi_1,\pi_2}(G).$$

One difficulty still remains. It may happen that for the optimal separator (A^*,B^*,S^*), $\pi_2(S^*) \geq \pi_2(A^*)$, and then the values $\alpha_{\pi_1,\pi_2}(A^*,B^*,S^*)$ and $\tilde{\alpha}_{\pi_1,\pi_2}(A^*,B^*,S^*)$, are not within a factor of 2 of each other. In this case we show how to output a (possibly trivial) separator that approximates $\alpha_{\pi_1,\pi_2}(G)$ within constant factors. Observe that in this case

$$\frac{\pi_1(S^*)}{\pi_2(S^*)^2} \leq 4\,\alpha_{\pi_1,\pi_2}(G).$$

Hence it suffices to find an approximation for a different problem, that of finding a subset $S \subseteq V$ which minimizes the ratio $\pi_1(S)/\pi_2(S)^2$. This problem can be solved in polynomial time; see section A.3.

THEOREM 3.12. *Given a graph $G=(V,E)$ and vertex weights $\pi_1,\pi_2 : V \to \mathbb{R}_+$, there exists a polynomial-time algorithm which computes a vertex separator (A,B,S) for which*

$$\alpha_{\pi_1,\pi_2}(A,B,S) \leq O(\sqrt{\log k})\,\alpha_{\pi_1,\pi_2}(G),$$

where $k = |\mathrm{supp}(\pi_2)|$.

4. Approximate max-flow/min-vertex-cut theorems. Let $G=(V,E)$ be a graph with capacities $\{c_v\}_{v\in V}$ on vertices and a demand function $\omega : V\times V \to \mathbb{R}_+$. For every pair of distinct vertices $u,v \in V$, let $\mathcal{P}_{uv}$ be the set of all simple u-v paths in G. For $s,t \in V$, an *s-t flow in G* is a mapping $F : \mathcal{P}_{st} \to \mathbb{R}_+$ where for $p \in \mathcal{P}_{st}$, $F(p)$ represents the amount of flow sent from s to t along path p.

For any simple path p in G, let p_0 and p_1 denote the initial and final nodes of p, respectively. By convention, we will assert that for such a flow F and for every $p \in \mathcal{P}_{st}$, the flow path p uses up $\frac{1}{2}F(p)$ of the capacity of p_0 and p_1 and uses up $F(p)$ of the capacity of all other nodes in p. Intuitively, one can think of the loss in capacity for flowing through a vertex to be charged half for entering the vertex and half for exiting; hence the initial and final vertices of a flow path are only charged half. This is made formal in the linear program (LP) that follows. We remark that this choice (as opposed to incurring a full loss of capacity in the initial and final nodes) is only for simplicity in the dual linear program; it is easily seen that all the results in this section hold for the other setting, with a possible loss of a factor of 2. To simplify notation, we also define, for any $p \in \mathcal{P}_{uv}$ and $w \in p$, the number $\kappa_p(w)$ to be 1 if w is an intermediate vertex of p and $\frac{1}{2}$ if w is the initial or final vertex of p.

The *maximum concurrent vertex flow* of the instance $(G,\{c_v\}_{v\in V},\omega)$ is the maximum constant $\epsilon \in [0,1]$ such that one can simultaneously route an ϵ fraction of each u-v demand $\omega(u,v)$ without violating the capacity constraints. For each $p \in \mathcal{P}_{uv}$, let p^{uv} denote the amount of the u-v commodity that is sent from u to v along p. We now write an LP that computes the maximum concurrent vertex flow:

$$\begin{aligned}
&\text{maximize } \epsilon \\
&\text{subject to } \sum_{p\in\mathcal{P}_{uv}} p^{uv} \geq \epsilon\cdot\omega(u,v), && u,v\in V,\\
&\qquad \sum_{u,v\in V}\sum_{p\in\mathcal{P}_{uv}:w\in p} \kappa_p(w)p^{uv} \leq c_w, && w\in V,\\
&\qquad p^{uv}\geq 0, && u,v\in V, p\in\mathcal{P}_{uv}.
\end{aligned}$$

We now write the dual of this LP with variables $\{s_v\}_{v\in V}$ and $\{\ell_{uv}\}_{u,v\in V}$:

$$\begin{aligned}
&\text{minimize} \sum_{v\in V} c_v s_v \\
&\text{subject to} \sum_{w\in p} \kappa_p(w)s_w \geq \ell_{uv}, && p\in\mathcal{P}_{uv}, \text{for all } u,v\in V,\\
&\qquad \sum_{u,v\in V}\omega(u,v)\ell_{uv}\geq 1,\\
&\qquad \ell_{uv}\geq 0, s_v\geq 0, && u,v\in V.
\end{aligned}$$

Finally, define

$$\mathsf{dist}(u,v) = \min_{p\in\mathcal{P}_{uv}} \sum_{w\in p}\kappa_p(w)s_w.$$

By setting $\ell_{uv} = \mathsf{dist}(u,v)$, we see that the above dual LP is equivalent to the following:

$$\begin{aligned}
&\text{minimize} \sum_{v\in V} c_v s_v \\
&\text{subject to} \sum_{u,v}\omega(u,v)\cdot\mathsf{dist}(u,v)\geq 1.
\end{aligned}$$

Remark 4.1. We remark that the distance function $\mathsf{dist}(u,v)$ is a metric which can be alternatively defined as follows: For any $u,v\in V$, $\mathsf{dist}(u,v)$ is precisely the (edge-weighted) shortest-path distance in G between u and v where the weight of the edge $(u,v)\in E$ is $\frac{1}{2}(s_u+s_v)$.

4.1. Rounding to vertex separators. Any vertex separator (A,B,S) yields an upper bound on the maximum concurrent flow in G via the following expression:

$$\frac{\mathrm{cap}(S)}{\sum_{u\in A,v\in B}\omega(u,v)+\sum_{u,v\in S}\omega(u,v)+\frac{1}{2}\sum_{u\in S}\sum_{v\in A\cup B}\omega(u,v)}. \tag{3}$$

The numerator is the capacity of the separator. Every unit of demand served between $u\in A$ and $v\in B$ must consume at least one unit of capacity from S. Likewise, every unit of demand served between $u\in S$ and $v\in S$ must consume at least one unit of capacity from S. Finally, every unit of demand served between $u\in S$ and $v\in A\cup B$ must consume at least half a unit of capacity from S. Hence the denominator is a lower bound on the amount of S's capacity burned by every unit of concurrent flow. We observe that the quantity (3) is bounded between $\alpha^{\mathrm{cap},\omega}(A,B,S)$ and $2\cdot\alpha^{\mathrm{cap},\omega}(A,B,S)$.

We will write $\alpha = \alpha^{\mathrm{cap},\omega}$ if the capacity and demands are clear from context. For a graph G, we will write $\alpha(G)$ for the minimum of $\alpha(A, B, S)$, where this minimum is taken over all vertex separators in G. We wish to study how tight the upper bound of $2 \cdot \alpha(G)$ is. In order to do so, we take the dual of the maximum concurrent-flow LP from the previous section and round it to a vertex separator. The increase in cost incurred by the rounding provides an upper bound on the worst possible ratio between $\alpha(G)$ and the maximum concurrent flow.

We note that the dual LP is a relaxation of the value $2 \cdot \alpha(G)$, since every vertex separator (A, B, S) gives a feasible solution, where $s_v = 1/\lambda$ if $v \in S$ and $s_v = 0$ otherwise. In this case $\mathsf{dist}(u, v) \geq 1/(2\lambda)$ if $u \in A \cup S$ and $v \in B \cup S$ or vice-versa, so that setting $\lambda = \sum_{u \in A \cup S, v \in B \cup S} \omega(u, v)$ yields a feasible solution.

4.2. The rounding. Before presenting our approach for rounding the LP, let us recall a typical rounding approach for the case of *edge-capacitated* flows. In the edge context [38, 8], one observes that the dual LP is essentially integral when $\mathsf{dist}(\cdot, \cdot)$ forms an L_1 metric. To round in the case when $\mathsf{dist}(\cdot, \cdot)$ does not form an L_1 metric, one uses Bourgain's theorem [14] to embed (V, dist) into L_1 (with $O(\log n)$ distortion, which translates to a similar loss in the approximation ratio), and then rounds the resulting L_1 metric (where rounding the L_1 metric does not incur a loss in the approximation ratio). This approach is not as effective in the case of vertex separators, because rounding an L_1 metric does incur a loss in the approximation ratio (as the example below shows), and hence there is not much point in embedding (V, dist) into L_1 and paying the distortion factor.

The discrete cube. Let $G = (V, E)$ be the d-dimensional discrete hypercube $\{0, 1\}^d$. We set $c_v = 1$ for every $v \in V$, and $\omega(u, v) = 1$ for every pair $u, v \in V$. It is well known that $\alpha(G) = \Theta(1/(2^d\sqrt{d}))$ [27]. On the other hand, consider the fractional separator (i.e., dual solution) given by $s_v = 10 \cdot \frac{4^{-d}}{d}$. Note that $\mathsf{dist}(u, v)$ is proportional to the shortest-path metric on the standard cube, and hence $\sum_{u,v} \mathsf{dist}(u, v) \geq 1$, yielding a feasible solution which is a factor $\Theta(\sqrt{d})$ away from $\alpha(G)$. It follows that even when (V, dist) is an L_1 metric, the integrality gap of the dual LP might be as large as $\Omega(\sqrt{\log n})$.

Rounding with line embeddings. The rounding is done as follows. Let $\{s_v\}_{v \in V}$ be an optimal solution to the dual LP, and let $\mathsf{dist}(\cdot, \cdot)$ be the corresponding metric on V. Suppose that the demand function $\omega : V \times V \to \mathbb{R}_+$ is supported on a set S, i.e., $\omega(u, v) > 0$ only if $u, v \in S$, and that $|S| = k$. Let $f : (S, \mathsf{dist}) \to \mathbb{R}$ be the map guaranteed by Theorem 3.1 with $avd_\omega(f) = O(\log k)$, and let $\tilde{f} : (V, \mathsf{dist}) \to \mathbb{R}$ be the 1-Lipschitz extension from Lemma 3.4.

For $v \in V$, define $\mathsf{cost}(v) = s_v$. Then since $\tilde{f}$ is 1-Lipschitz, for any edge $(u, v) \in E$, we have

$$|\tilde{f}(u) - \tilde{f}(v)| \leq \mathsf{dist}(u, v) = \frac{s_u + s_v}{2} = \frac{\mathsf{cost}(u) + \mathsf{cost}(v)}{2};$$

hence $\tilde{f}$ is path-compatible with cost.

We now apply FindCut$(G, \tilde{f})$. If the best separator found has sparsity α_0, then by Lemma 3.7,

$$\begin{aligned}\sum_v c_v s_v = \sum_v c_v \cdot \mathsf{cost}(v) &\geq \alpha_0 \sum_{u,v \in V} \omega(u, v)\, |\tilde{f}(u) - \tilde{f}(v)| \\ &= \alpha_0 \sum_{u,v \in S} \omega(u, v)\, |f(u) - f(v)|\end{aligned}$$

$$\geq \Omega\left(\frac{\alpha_0}{\log k}\right) \sum_{u,v \in V} \omega(u,v)\,\mathsf{dist}(u,v) \geq \Omega\left(\frac{\alpha_0}{\log k}\right).$$

THEOREM 4.1. *For an arbitrary vertex-capacitated flow instance, where the demand is supported on a set of size k, there is an $O(\log k)$-approximate max-flow/min-vertex-cut theorem. In particular, this holds if there are only k commodities.*

4.3. Excluded minor families. Recall that by Remark 4.1, we can view the metric dist arising from the LP dual as an edge-weighted metric on the graph G. A consequence of this is that if the graph G excludes some fixed graph H as a minor, then the metric dist is an H-excluded metric.

It follows that applying Theorem 3.2 yields a better result when G excludes a minor and the demand function $\omega(u,v)$ is uniform on a subset of the vertices. This special case will be needed later when we discuss treewidth and follows from the following theorem (because product demands include as a special case demand functions that are uniform on a subset of the vertices).

THEOREM 4.2. *When G is an H-minor-free graph, there is an $O(|V(H)|^2)$-approximate max-flow/min-vertex-cut theorem with product demands. Additionally, there exists an $O(|V(H)|^2)$ approximation algorithm for finding minimum quotient vertex cuts in G.*

4.4. More integrality gaps for uniform demands.

Expanders. Our analysis for the integrality gap of the dual LP is tight. Just as in the edge case, constant-degree expander graphs provide the bad example. If $G = (V, E)$ is such a graph, with uniform vertex capacities and uniform demands, then $\alpha(G) = 1/\Theta(n)$, while the dual LP has a solution of value $1/\Omega(n \log n)$ (by setting $s_v = 1/\Omega(n^2 \log n)$ for every $v \in V$).

Euclidean metrics. Even if the vertex-weighted distance function returned by the LP is equivalent to a Euclidean metric, up to a universal constant, there may still be an integrality gap of $\Omega(\sqrt{\frac{\log n}{\log\log n}})$. We sketch the argument here. The idea is to take a fine enough "mesh" on a high-dimensional sphere so that the shortest-path distance along the mesh approximates the Euclidean distance. Using standard isoperimetric considerations on high-dimensional spheres, we are able to determine the structure of the near-optimal vertex separators. Here we will only sketch the proof; one may refer to [40] for a more detailed argument along these lines.

Let S^d be the d-dimensional sphere, let $\epsilon = 1/\Theta(d)$, and let V be an ϵ-net on the sphere S^d. (An ϵ-net in a metric space X is a subset $N \subseteq X$ such that $x, y \in N \implies d(x,y) \geq \epsilon$, and $X \subseteq \bigcup_{x \in N} B(x, \epsilon)$.) Standard arguments show that $n = |V| \leq O(d)^d$. Define a graph G with vertex set V and an edge between $u, v \in V$ whenever $\|u - v\|_2 \leq 10\,\epsilon$. We claim the following facts without proof (see [40] for a similar argument).

CLAIM 4.3. *The following three assertions hold true:*

1. $\alpha(G) = 1/\Theta(n\sqrt{d})$.
2. *Setting $s_v = 1/\Theta(n^2 d)$ in the dual LP yields a feasible solution with value $1/\Theta(nd)$.*
3. *The (vertex-weighted) shortest path metric on G with weights given by $\{s_v\}_{v \in V}$ is equivalent (up to a universal constant) to a Euclidean metric (V, d). (Namely, the metric given by $d(u,v) = \|u - v\|_2/n^2$, recalling that $V \subseteq S^d$.)*

It follows that the integrality gap is at least $\Theta(\sqrt{d}) = \Theta(\sqrt{\frac{\log n}{\log\log n}})$.

5. An integrality gap for the vector program. Consider the hypercube graph. Namely, the n vertices of the graph (where n is a power of 2) can be viewed as all vectors in $\{\pm 1\}^{\log n}$, and edges connect two vertices that differ in exactly one coordinate. Every vertex separator (A, B, S) has $\alpha(A, B, S) \geq 1/O(n\sqrt{\log n})$. This follows from standard vertex isoperimetry on the cube [27]. We show a solution to the vector program with value of $O(n/\log n)$, proving an integrality ratio of $\Omega(\sqrt{\log n})$ for the vector program, and implying that our rounding technique achieves the best possible approximation ratio (relative to the vector program), up to constant multiplicative factors.

In the solution to the vector program, we describe for every vertex i the associated vectors x_i and y_i. The vectors s_i will not be described explicitly, but are implicit, using the relation $s_i = v - x_i - y_i$. Note that the exclusion constraints $s_i \cdot x_i = s_i \cdot y_i = 0$ are implied by the exclusion constraints $x_i \cdot y_i = 0$ and the sphere constraints. Each vector will be described as a vector in $1 + n \log n + 2(n-1)$ dimensions (even though n dimensions certainly suffice). Our redundant representation in terms of the number of dimensions helps clarify the structure of the solution.

To describe the vector solution, we introduce two parameters, a and b. Their exact value will be determined later and will turn out to be $a = 1/2 - \Theta(1/\log n)$ and $b = \Theta(1/\sqrt{n \log n})$. We partition the coordinates into three groups of coordinates:

G1. Group 1 contains one coordinate. This coordinate corresponds to the direction of vector v (which has value 1 in this coordinate and 0 elsewhere). All x_i and y_i vectors have value a on this coordinate.

G2. Group 2 contains n identical blocks of $\log n$ coordinates. The coordinates within a block exactly correspond to the structure of the hypercube. Within a block, each x_i is a vector in $\{\pm b\}^{\log n}$ derived by scaling the hypercube label of vertex i (which is a vector in $\{\pm 1\}^{\log n}$) by a factor of b. Vector y_i is the negation of vector x_i on the coordinates of Group 2.

G3. Group 3 contains two identical blocks of $n-1$ coordinates. The coordinates within a block arrange all the x_i vectors as vertices of a simplex. This is done in the following way. Let H_n be the n by n Hadamard matrix with entries ± 1, obtained by taking the $(\log n)$-fold tensor product [16] of the 2 by 2 matrix H_2 that has rows $(1, 1)$ and $(1, -1)$. The inner product of any two rows of H_n is 0, the first column is all 1, and the sum of entries in any other column is 0. Remove the first column to obtain the matrix H'_n. Within a block, let vector x_i be the ith row of H'_n, scaled by a factor of b. Hence within a block, $x_i x_i = b^2(n-1)$, and $x_i x_j = -b^2$ for $i \neq j$. Vector y_i is identical to x_i on the coordinates of Group 3.

We now show that the triangle constraints are satisfied by our vector solution. Recall (see section 2) that there is some flexibility in the choice of which triangle constraints to include in the vector program (and likewise for many other constraints that are valid for 0/1 solutions but are not used in our analysis). We shall address here a subset of the triangle constraints that is larger than that actually used in the analysis of our rounding algorithm.

There are five sets of vectors from which we can take the three vectors that participate in a triangle constraint: X (the x_i vectors), Y (the y_i vectors), S (the s_i vectors), v, and 0. In our analysis we used only triangle constraints over vectors from X. Here we show that all the triangle constraints that involve only vectors from $X \bigcup Y$ are satisfied. All vectors in $X \bigcup Y$ have the identical value a in their first coordinate, and in every other coordinate they take only values from $\pm b$. Hence *every* quadratic constraint that holds for all ± 1 vectors (including, but not limited to, the

triangle constraints) is satisfied on every coordinate separately, which implies that it is satisfied for all x_i and y_i vectors.

We let $K = \sum_{i,j \in V} (x_i - x_j)^2 = \Theta(n^3 b^2 \log n)$. The value of the parameters a and b is governed by the following three constraints:

1. The exclusion constraints imply that
$$a^2 - nb^2 \log n + 2b^2(n-1) = 0.$$
2. The edge constraints (and the fact that edges connect vertices of Hamming distance 1) imply that
$$a^2 - nb^2(\log n - 2) - 2b^2 = 0.$$
3. The sphere constraints imply that
$$a = a^2 + nb^2 \log n + 2b^2(n-1).$$

Hence we have a system of three equalities in two unknowns (a and b). This system is consistent, because the first two equalities are in fact identical (due to our careful choice of number of blocks in each group). They both give

$$a^2 + (-n \log n + 2n - 2)b^2 = 0.$$

By setting $b = a/\sqrt{n \log n - 2n + 2}$ the first two equalities are satisfied. The third equality now reads $a = a^2(2 + \epsilon)$ for some $\epsilon = \Theta(1/\log n)$. This equality is satisfied by taking a roughly equal to $1/2 - \epsilon/4$, which is $1/2 - \Theta(1/\log n)$.

It follows that in the vector solution all $s_i^2 = 1 - x_i^2 - y_i^2$ is $O(1/\log n)$ for every $i \in V$. Hence our vector solution has value

$$\frac{1}{K} \sum_{i \in V} s_i^2 = \frac{1}{\Theta(n \log n)}.$$

Finally, we note that rather than having only one coordinate in Group 1, we can have $(a/b)^2 = n \log n - 2n + 2$ coordinates, and give the x and y vectors values b in these coordinates. Then all x and y vectors become vertices of a $2n \log n$-dimensional hypercube (of side length b). We see that even in this special case, the integrality gap remains $\Omega(\sqrt{\log n})$.

6. Balanced separators and applications.

6.1. Reduction from minimum ratio cuts to balanced separators. In this section, we sketch a pseudoapproximation for finding balanced vertex separators in a graph $G = (V, E)$. Let $W \subseteq V$ be an arbitrary subset of V. For $\delta \in (0, 1)$, we say that a subset $X \subseteq V$ is a *δ-vertex separator (with respect to W)* if every connected component C of $G[V \setminus X]$ has $|C \cap W| \leq \delta |W|$. Our goal in this section is to show that we can find a $\frac{3}{4}$-vertex separator $X \subseteq V$ whose size is within an $O(\beta)$ factor of the optimal $\frac{2}{3}$-vertex separator of G, whenever we can find approximate minimum ratio cuts in G within factor β. This technique is standard (see [36]).

The algorithm. Let $m = |W|$, and for any subset $U \subseteq V$, define $|U|_W = |U \cap W|$. Let $\pi_1(v) = 1$ for every $v \in V$, and $\pi_2(v) = 1$ if $v \in W$ and $\pi_2(v) = 0$ otherwise. These are the weights for the numerator and denominator, respectively; i.e., we assume that we have a β-approximation for $\alpha_{\pi_1, \pi_2}(\cdot)$. We maintain a vertex separator $S \subseteq V$. Initially, $S = \emptyset$. As long as there exists some connected component $U \subseteq V$ in $G[V \setminus S]$

with $|U|_w \geq \frac{3}{4}|W|$, we use our β-approximation to find a minimum ratio vertex cut S' in $G[U]$ which is within β of optimal. We then set $S \leftarrow S \cup S'$ and continue.

The analysis. Let S be the final vertex separator. By construction, it is a $\frac{3}{4}$-vertex separator since every connected component U of $G[V \setminus S]$ has $|U|_w < \frac{3}{4}|W|$. Let $T \subseteq V$ be an optimal $\frac{2}{3}$-vertex separator.

CLAIM 6.1. $|S| \leq O(\beta)|T|$.

Proof. The fact that T is a $\frac{2}{3}$-vertex separator with respect to W implies that the vertices in $V \setminus T$ can be partitioned into two disjoint sets $A_T, B_T \subseteq V$ such that $|A_T \cup T|_w, |B_T \cup T|_w \geq \frac{1}{3}|W|$, with no edges between A_T and B_T. Suppose we are at a step where $|U|_w \geq \frac{3}{4}|W|$. Let (A', B', S') be the vertex separator in $G[U]$ that we find by running our minimum quotient cut algorithm with ratio β, and suppose that $|A'|_w \geq |B'|_w$. We know that

$$\frac{|S'|}{|A' \cup S'|_w \cdot |B' \cup S'|_w} \leq \beta \frac{|T|}{|(A_T \cup T) \cap U|_w \cdot |(B_T \cup T) \cap U|_w} \leq \frac{18\beta|T|}{m^2},$$

where the final inequality follows because $|U|_w \geq \frac{3m}{4}$. It follows that

$$|S'| \leq \frac{18\beta|T|(|B'|_w + |S'|_w)}{m}.$$

To see that $|S| \leq O(\beta)|T|$, it suffices to see that when we sum $|B'|_w + |S'|_w$ over all iterations, the value is at most $O(m)$. But since we throw away the vertices of $B' \cup S'$ in every iteration (and recurse only on A'), the sum is clearly at most m. ∎

6.2. Getting an $O(\sqrt{\log \mathsf{opt}})$ approximation for vertex separators. In this section, we sketch a proof of how one can obtain an $O(\sqrt{\log \mathsf{opt}})$ pseudoapproximation for finding balanced vertex separators. In other words, given a graph G with a $\frac{2}{3}$-vertex separator of size m, we find a $\frac{3}{4}$-vertex separator whose size is at most $(m\sqrt{\log m})$. The method is based on the following enhancement of Theorem 3.3.

THEOREM 6.2. *Let $C > 0$ be a universal constant. Let (X, d) be an n-point metric space of negative type, and let $\omega_0 : X \times X \to \mathbb{R}_+$ be any product weight. If*

$$\frac{\sum_{x,y} \omega_0(x,y)\, d(x,y)}{\sum_{x,y} \omega_0(x,y)} = 1,$$

and there exists an ε-net $N \subseteq X$ with $|N| \leq m$ and $\varepsilon \leq 1/(C\sqrt{\log m})$, then there exists an efficiently computable map $f : X \to \mathbb{R}$ with $avd_{\omega_0}(f) = O(\sqrt{\log m})$.

Proof. Assume that $\omega_0(x,y) = \pi(x)\pi(y)$ for all $x, y \in X$. As in the proof of Theorem 3.3 (see section A.2), if there exists some point $x_0 \in X$ for which $\pi(B(x_0, \frac{1}{4n^2})) \geq \frac{1}{2}\pi(X)$, then we achieve a map $f : X \to \mathbb{R}$ with $avd_{\omega_0}(f) = O(1)$. If no such x_0 exists, then it must be the case (see the proof of [7, Lemma 14]) that there exists a set $S \subseteq X \times X$ of pairs for which $\sum_{(x,y)\in S} \pi(x)\pi(y) \geq \Omega(1) \sum_{x,y} \omega_0(x,y)$, and $d(x,y) \geq \frac{1}{100}$ for $(x,y) \in S$.

We construct a new weight function $\pi^* : N \to \mathbb{R}_+$ on N as follows. Since N is an ε-net, we have $X \subseteq \bigcup_{y\in N} B(y, \varepsilon)$. For every point $x \in X$, put x into a set S_y for some net point $y \in N$ with $d(x,y) \leq \varepsilon$ (so that $\{S_y\}_{y\in N}$ is a partition of X). Define $\pi^*(y) = \sum_{x\in S_y} \pi(x)$ for every $y \in N$.

We now consider the quantity

$$\bar{d}_N = \frac{\sum_{x,y\in N} \pi^*(x)\pi^*(y)\, d(x,y)}{\sum_{x,y\in N} \pi^*(x)\pi^*(y)} = \frac{\sum_{x,y\in N} \sum_{u\in S_x, v\in S_y} \pi(u)\pi(v)\, d(x,y)}{\sum_{x,y} \omega_0(x,y)}.$$

We claim that $\bar{d}_N = \Omega(1)$. But this follows since

$$\begin{aligned}\sum_{x,y\in N}\sum_{u\in S_x, v\in S_y}\pi(u)\pi(v)\,d(x,y) &\geq \sum_{x,y\in N}\sum_{u\in S_x, v\in S_y,(u,v)\in S}\pi(u)\pi(v)\,d(x,y)\\ &\geq \sum_{x,y\in N}\sum_{u\in S_x, v\in S_y,(u,v)\in S}\pi(u)\pi(v)\ (d(u,v)-2\varepsilon)\\ &\geq \frac{1}{2}\sum_{(u,v)\in S}\pi(u)\pi(v)\,d(u,v) = \Omega(1)\sum_{x,y}\omega_0(x,y).\end{aligned}$$

As discussed in section A.2, the techniques of [7] now show that there exist two subsets $L, R \subseteq N$ for which $d(L,R) \geq 1/O(\sqrt{\log m})$ and $\pi^*(L), \pi^*(R) \geq \frac{1}{10}\pi^*(X)$. Construct sets

$$L' = \{x \in X : x \in S_y \text{ for some } y \in L\} \quad \text{and} \quad R' = \{x \in X : x \in S_y \text{ for some } y \in R\}.$$

Note that $\pi(L') = \pi^*(L)$ and $\pi(R') = \pi^*(R)$; hence $\pi(L'), \pi(R') \geq \frac{1}{10}\pi(X)$. Finally, for any points $x_L \in L'$, $x_R \in R'$, let y_L, y_R be such that $x_L \in S_{y_L}$ and $x_R \in S_{y_R}$, and notice that

$$d(x_L, x_R) \geq d(y_L, y_R) - d(x_L, y_L) - d(x_R, y_R) \geq \tfrac{1}{O(\sqrt{\log m})} - 2\varepsilon \geq \tfrac{1}{O(\sqrt{\log m})},$$

where the last inequality holds for $C > 0$ chosen sufficiently large (and hence ε chosen sufficiently small). Now one simply takes the map $f(x) = d(x, L')$, which has $avd_{\omega_0}(f) = O(\sqrt{\log m})$. □

Next, we make an observation about solutions to the SDP of section 2.3.

LEMMA 6.3. *If $\{x_i, y_i\}$ is a solution to the SDP with $W = \sum_{i\in V}(1 - x_i^2 - y_i^2)$, then in the metric space $(\{x_1, \ldots, x_n\}, d)$ where $d(i,j) = (x_i - x_j)^2$, there exists an ε-net $N \subseteq \{x_1, \ldots, x_n\}$ with $|N| \leq O(W/\varepsilon)$.*

Proof. For each $i \in V$, define $w(i) = 1 - x_i^2 - y_i^2$. For a subset $S \subseteq V$, define $w(S) = \sum_{x\in S} w(x)$. Let $G = (V, E)$ be the original graph, and let $d_G(i,j) = \min_{p\in\mathcal{P}_{ij}} w(p)$, where we recall that $\mathcal{P}_{ij}$ is the set of all simple i-j paths. We claim first that $d(i,j) \leq 4\, d_G(i,j)$. Indeed, let $i = i_1, i_2, \ldots, i_k = j$ be a minimum weight path in G; then

$$d(i,j) = (x_i - x_j)^2 \leq \sum_{h=1}^{k-1}(x_{i_h} - x_{i_{h+1}})^2 \tag{4}$$

$$\leq 2\sum_{h=1}^{k-1}\Big((1 - x_{i_h}^2 - y_{i_h}^2) + (1 - x_{i_{h+1}}^2 - y_{i_{h+1}}^2)\Big) \tag{5}$$

$$= 2\sum_{h=1}^{k-1}(w(i_h) + w(i_{h+1}))$$

$$\leq 4\, d_G(i,j),$$

where (4) follows from the squared triangle inequalities, and (5) follows from line (2) in Proposition 3.10.

Thus it will suffice to find an $\varepsilon/4$-net N in the metric d_G, and the rest of the proof refers to this metric on $X = \{x_1, \ldots, x_n\}$. Choose a maximal set $Y \subseteq \{x_1, \ldots, x_n\}$ among all points $x \in X$ for which $w(B_{d_G}(x, \varepsilon/8)) \geq \varepsilon/16$, subject to the constraint that $x, y \in Y \implies d(x,y) > \varepsilon/8$. By construction, the balls $B_{d_G}(x, \varepsilon/8)$ are disjoint for $x \in Y$; hence $|Y| \leq 16W/\varepsilon$, recalling that $W = w(X)$.

So we are done once we prove that Y is an $\varepsilon/4$-net in (X, d_G). For the sake of contradiction, suppose there is a point $x \in X$ with $d_G(x, Y) > \varepsilon/4$. Let $y \in Y$ be such that $d_G(x, y) = d_G(x, Y)$, and consider any shortest-path $x = y_1, \ldots, y_k = y$ in G. Letting $P = \{y_1, \ldots, y_k\}$, we know that $w(P) = d_G(x, y) > \varepsilon/4$. If we set $P' = \{u \in P : d_G(y, u) > \varepsilon/8\}$, then $w(P') > w(P) - \varepsilon/8 \geq \varepsilon/8$, and for every $u \in P'$, we have $d_G(u, Y) > \varepsilon/8$. So if there exists any point $u \in P'$ with $w(u) \geq \varepsilon/16$, then we could add u to Y, contradicting its maximality. Thus we may assume that for every $u \in P'$, we have $w(u) < \varepsilon/16$. But now let $z \in P'$ be the point of P' which is closest to y. Then $d_G(z, x) = w(P') > \varepsilon/8$; hence we know that

$$w(B_{d_G}(z, \varepsilon/8)) \geq w(B_{d_G}(z, \varepsilon/8) \cap P') \geq \varepsilon/16,$$

because the first point along P' not included in $B_{d_G}(z, \varepsilon/8)$ (which must exist) must be further than $\varepsilon/8$ away from z but also have weight at most $\varepsilon/16$. We again conclude that Y is not maximal, completing the proof. □

Combining Theorem 6.2 and Lemma 6.3, along with the analysis of section 3, yields an $O(\sqrt{\log m})$-approximation to vertex sparsest cut where m is the number of vertices in an optimal $\frac{2}{3}$-vertex separator. Now applying the transformation of section 6.1 yields the desired $O(\sqrt{\log \mathsf{opt}})$ pseudoapproximation for finding balanced vertex separators.

6.3. Applications. The notion of treewidth was introduced by Robertson and Seymour [43] and plays an important role in their fundamental work on graph minors. In addition, it has numerous practical applications (see, e.g., [10]). A large amount of effort has been put into determining treewidth, which is NP-complete even when the input graph is severely restricted (see the discussion in [21] for a brief history).

From the approximation viewpoint, Bodlaender et al. [11] gave an $O(\log n)$-approximation algorithm for treewidth on general graphs. Amir [4] improved the approximation factor to $O(\log \mathsf{opt})$, where opt is the actual treewidth of the graph. Constant-factor approximations for treewidth were obtained on asteroidal triple–free (AT-free) graphs [13, 12] and on planar graphs [44]. The approximation for planar graphs is a consequence of the polynomial-time algorithm given by [44] for computing the parameter *branchwidth*, whose value approximates treewidth within a factor of 1.5. Recently, [5] obtained a new approximation algorithm for treewidth in planar graphs with a constant factor slightly worse than 1.5, and the authors of [21] derived a polynomial-time algorithm for approximating treewidth within a factor of 1.5 for single-crossing minor-free graphs and generalizations of planar graphs. A well-known open problem is whether treewidth can be approximated within a constant factor.

Using our new approximation algorithms for vertex separators, we improve the approximation ratio for treewidth, both in general graphs and in some special families of graphs. Our improvements and some of their implications will be presented after we formally define the notion of treewidth.

Treewidth. The notion of *treewidth* involves a representation of a graph as a tree, called a tree decomposition. More precisely, a *tree decomposition* of a graph $G = (V, E)$ is a pair (T, χ) in which $T = (I, F)$ is a tree and $\chi = \{\chi_i \mid i \in I\}$ is a family of subsets of $V(G)$ such that (1) $\bigcup_{i \in I} \chi_i = V$; (2) for each edge $e = \{u, v\} \in E$, there exists an $i \in I$ such that both u and v belong to χ_i; and (3) for all $v \in V$, the set of nodes $\{i \in I \mid v \in \chi_i\}$ forms a connected subtree of T. To distinguish between vertices of the original graph G and vertices of T in the tree decomposition, we call vertices of T *nodes* and their corresponding χ_i's *bags*. The maximum size of a bag in χ minus one is called the *width* of the tree decomposition. The *treewidth* of a graph G,

which we denote by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G. A tree decomposition is called a *path decomposition* if $T = (I, F)$ is a path. The *pathwidth* of a graph G is the minimum width over all possible path decompositions of G.

Now we are ready to state our approximation result for treewidth.

THEOREM 6.4. *There exist polynomial time algorithms that find a tree decomposition of width at most* $O(\sqrt{\log \mathrm{tw}(G)}\,\mathrm{tw}(G))$ *for a general graph* G *and at most* $O(|V(H)|^2\,\mathrm{tw}(G))$ *for an* H*-minor-free graph* G.

Proof. The proof follows by plugging our improved approximation ratios for balanced vertex separators into the known approximation algorithms for treewidth. Specifically, the algorithm of [11] finds a tree decomposition by recursively using a balanced vertex separator algorithm. The vertex separator algorithm is applied to subgraphs of the original graph, in a product demand setting. It turns out that the approximation ratio obtained for treewidth is at most a constant factor worse than that of the underlying vertex separator algorithm. Using our bounds from section 6.2 one obtains the first part of Theorem 6.4, and using Theorem 4.2 one obtains the second part of Theorem 6.4. ∎

Improving the approximation factor of treewidth improves the approximation factor for several other problems. We refer the reader to [11] for a discussion of these implications and the relevant definitions.

COROLLARY 6.5. *There exist* $O(\sqrt{\log \mathsf{opt}})$ *(resp.,* $O(|V(H)|^2)$*) approximation algorithms for branchwidth, minimum front size, and minimum size of a clique in a chordal supergraph of a general (resp.,* H*-minor-free) graph* G*. Additionally, there are* $O(\sqrt{\log \mathsf{opt}\,\log n})$ *(resp.,* $O(|V(H)|^2 \log n)$*) approximation algorithms for pathwidth, minimum height elimination order tree, and search number in a general (resp.,* H*-minor-free) graph* G.

We also note that Theorem 3.12 with general weights π_1, π_2 is useful for certain hypergraph partitioning problems [36]. Improving the approximation factor for treewidth has a direct improvement on the running time of approximation schemes and subexponential fixed parameter algorithms for several NP-hard problems on graph families which exclude a fixed minor. In such algorithms finding the tree decomposition of almost minimum width, on which we can run dynamic programming, plays a very important role. More precisely, Demaine and Hajiaghayi [20, 19] introduced the concept of (contraction/minor) bidimensional parameters for planar graphs and more generally for excluded-minor families. Examples of bidimensional parameters include number of vertices, diameter, and the size of various structures, e.g., feedback vertex set, vertex cover, minimum maximal matching, face cover, a series of vertex-removal parameters, dominating set, edge dominating set, r-dominating set, connected dominating set, connected edge dominating set, connected r-dominating set, and unweighted Traveling Salesman tour (a walk in the graph visiting all vertices).

They show how one can obtain PTASs for almost all bidimensional parameters on planar graphs, single-crossing minor-free graphs, and bounded genus graphs. In fact, as they mentioned, their approach can be extended to work on apex-minor-free graphs for contraction-bidimensional parameters and on H-minor-free graphs, where H is a fixed graph for minor-bidimensional parameters (see [20, 19] for appropriate definitions). However, currently they obtain quasi-polynomial-time approximation schemes for these general settings. The only barrier to obtaining PTASs for these general settings is obtaining a constant-factor polynomial-time approximation algorithm for treewidth of an H-minor-free graph for a fixed H (this is posed as an open

problem in [20]). Using Theorem 6.4, we overcome this barrier and obtain PTASs for contraction-bidimensional parameters in apex-minor-free graphs and for minor-bidimensional parameters in H-minor-free graphs for a fixed H. As an immediate consequence, we obtain the following theorem (see [20, 19] for the exact definitions of the problems mentioned below).

THEOREM 6.6. *There are PTASs for feedback vertex set, vertex cover, minimum maximal matching, and a series of vertex-removal problems in H-minor-free graphs for a fixed H. Also, there are PTASs for dominating set, edge dominating set, r-dominating set, connected dominating set, connected edge dominating set, connected r-dominating set, and clique-transversal set in apex-minor-free graphs.*

Among the problems mentioned above, PTASs for vertex cover and dominating set (but not its other variants) using a different approach were known before (see, e.g., [26]).

Appendix.

A.1. A note about approximating vertex expansion. In the case of *edge cuts*, the value of the sparsest cut (under uniform weights) corresponds to *edge expansion* of the graph G. Thus it is perhaps more natural to consider finding the vertex separator (A, B, S) which minimizes the ratio $|S|/|B|$, where, by convention, $|B| \leq |A|$.

We now show that having the $|S|$ term in the denominator, i.e., $|S|/(|B| + |S|)$, is crucial to obtaining polylogarithmic approximation ratios. We present here an argument (essentially due to Shimon Kogan) that demonstrates this fact.

Consider the problem of a balanced bipartite independent set (BBIS). The input is a bipartite graph $G(U \bigcup V, E)$ with $|U| = |V| = n$, and the goal is to find the maximum value of t and sets $A \subset U$, $B \subset V$ with $|A| = |B| = t$ with no edges between A and B. It is known that when t is small compared to n, approximating this problem (the value of t) within a ratio of n^δ for some $\delta > 0$ will have some major algorithmic consequences [23, 24], including subexponential algorithms for all NP problems [29]. Now modify G by making U into a clique and V into a clique, obtaining a graph G'. The set S of vertices not in the maximum BBIS provides a vertex separator (A, B, S) for G'. The ratio $|S|/|B|$ for this separator is the minimum possible up to constant factors. (For every separator (A', B', S'), side U cannot contain vertices both from A and from B. Hence $|S'| = \Omega(n)$ unless both A' and B' are of size nearly n. When t is known to be small, this implies that $|S'| = \Theta(n)$ for all separators. Hence the ratio $|S'|/|B'|$ of any separator in G' is governed by $|B'|$ rather than by $|S|$. The value of $|B'|$ is maximized by taking the separator (A, B, S).) This implies that for the minimum balanced vertex separator the quantity $|S|/|B|$ cannot be approximated within a ratio of n^δ (unless NP has subexponential algorithms).

Remark. For a set B of vertices, let $N(B)$ denote the set of vertices not in B that are neighbors of vertices in B. Then the expansion of B is $|N(B)|/|B|$. The expansion of a graph is the minimum over all sets B up to a certain size of the ratio $|N(B)|/|B|$. The restriction on the size of B is necessary so as to avoid B being the whole graph, giving expansion 0. For bounded degree graph, one typically requires $|B| \leq n/2$. For graphs of unbounded degree, such a requirement is insufficient, as it always bounds the expansion by 1 (taking B to be half the graph), whereas one would like to allow for much higher expansions. A possible restriction on B in this case is to require it to be the smaller side of an (A, B, S)-vertex separator. Under this definition of vertex expansion, the above argument shows that vertex expansion cannot be approximated within a factor of n^δ unless NP has subexponential algorithms.

A.2. Line embedding theorems. We now sketch how the following three theorems follow from their respective sources. We begin with Bourgain's theorem.

THEOREM A.1 (Bourgain [14]). *If (X,d) is an n-point metric space, then for every weight function $\omega : X \times X \to \mathbb{R}_+$, there exists an efficiently computable map $f : X \to \mathbb{R}$ with $avd_w(f) = O(\log n)$.*

In [14], it is shown that every n-point metric (X,d) space embeds into a Hilbert space with distortion $O(\log n)$, but Bourgain actually shows something stronger. He proves that there exists a probability space (Ω,μ) on random subsets $A_\tau \subseteq X$, $\tau \in \Omega$, satisfying the following property: For every $x, y \in X$,

$$\mathbb{E}_\Omega\left[|d(x,A_\tau) - d(y,A_\tau)|\right] \geq \frac{d(x,y)}{O(\log n)}.$$

To show how this implies the theorem, note that by linearity of expectation

$$\mathbb{E}_\Omega\left[\sum_{x,y\in X} \omega(x,y)\cdot|d(x,A_\tau) - d(y,A_\tau)|\right] \geq \frac{1}{O(\log n)}\sum_{x,y\in X}\omega(x,y)\cdot d(x,y).$$

Hence there must exist some subset $A_\tau \subseteq X$ for which the map $f : X \to \mathbb{R}$ given by $f(x) = d(x,A_\tau)$ has $avd_\omega(f) = O(\log n)$. An efficient randomized algorithm for sampling A_τ is given in [38].

THEOREM A.2 (Rabinovich [41]). *If (X,d) is any metric space supported on a graph which excludes a K_r-minor, then for every* product weight *$\omega_0 : X \times X \to \mathbb{R}_+$, there exists an efficiently computable map $f : X \to \mathbb{R}$ with $avd_{\omega_0}(f) = O(r^2)$.*

In [41], Rabinovich proves precisely this fact, although only for the uniform weight function $\omega_0(x,y) = 1$ for all $x, y \in X$. It is easy to see that we can assume arbitrary product form for ω_0 without loss of generality. Suppose that we have vertex weights $\pi : V \to \mathbb{R}_+$. We can replace X by the pseudometric where each copy of $x \in X$ occurs $\pi(x)$ times. Then applying the analysis of [41] immediately yields the desired result.

THEOREM A.3 (Arora, Rao, and Vazirani [7]). *If (X,d) is an n-point metric of negative type, then for every* product weight *$\omega_0 : X \times X \to \mathbb{R}_+$, there exists an efficiently computable map $f : X \to \mathbb{R}$ with $avd_{\omega_0}(f) = O(\sqrt{\log n})$.*

Assume that $\omega_0(x,y) = \pi(x)\pi(y)$ for all $x, y \in X$. We will "mentally" replace every copy of x by $\pi(x)$ copies, but we will ensure that this increase in the number of points does not affect the quality of our map f. Also, suppose that (by scaling) $\left(\frac{1}{\sum_{x,y}\omega_0(x,y)}\right)\sum_{x,y\in X}\omega_0(x,y)\cdot d(x,y) = 1$.

Suppose there exists some point $x_0 \in X$ for which $\pi(B(x_0,\frac{1}{4})) \geq \frac{1}{2}\pi(X)$. In this case, the map $f(x) = d\left(x, B(x_0,\frac{1}{4})\right)$ has $avd_{\omega_0}(f) = O(1)$ (see, e.g., [7, Lemma 14]).

Otherwise, the techniques of [7] show that there exist two subsets $L, R \subseteq X$ for which $d(L,R) \geq 1/O(\sqrt{\log n})$ and $\pi(L), \pi(R) \geq \frac{1}{10}\pi(X)$. The fact that the number of copies of a point $x \in X$ does not affect the analysis is somewhat technical and relies on the fact that an "(ϵ,δ)-cover" has size which is lower-bounded by the number of *distinct* points that it contains. In this latter case, one simply takes the map $f(x) = d(x,L)$, which has $avd_{\omega_0}(f) = O(\sqrt{\log n})$. A simpler algorithm for computing the map f (which consists of choosing a few random hyperplanes) is given in [34].

A.3. Approximating the "densest subgraph." To orient the reader arriving at this section from section 3.6, let us remark that π and ω below can correspond to π_1 and a product distribution $\pi_2 \times \pi_2$ in section 3.6.

Given a set $V = \{v_1, \ldots, v_n\}$ with a positive rational weight function π on V and a nonnegative rational weight function ω on $V \times V$, we need to find a set $S \subseteq V$ of maximum density, where the density of a set is defined as

$$\Delta(S) \equiv \frac{\sum_{i,j \in S} \omega(i,j)}{\pi(S)}. \tag{6}$$

This is a weighted version of the *densest subgraph* problem and can be solved in polynomial time (see, for example, Chapter 4 in [33]). For completeness, we sketch the algorithm.

Construct a bipartite graph with sides U and W, where U has n vertices labeled $\{u_1, \ldots u_n\}$, and W has n^2 vertices labeled w_{ij} for $1 \leq i, j \leq n$. For every i, connect vertex u_i to the vertices w_{ij} and w_{ji} (for all j). All these edges have infinite capacity. Add two special vertices, s and t, to the graph. For every i, connect vertex u_i to s by an edge of capacity $k\pi(i)$, where k is a parameter whose value will be optimized later. For every $1 \leq i, j \leq n$, connect vertex $w_{i,j}$ to t by an edge of capacity $\omega(i,j)$. Now compute the minimum capacity (s,t)-cut in the resulting capacitated graph (a problem that can be solved in polynomial time by using flow techniques).

We now analyze the above algorithm. Observe first that the minimum (s,t)-cut contains only edges that are connected to either t or s, as other edges have infinite capacity. Furthermore, observe that if the parameter k is sufficiently large, then the minimum (s,t)-cut contains exactly those edges connected to t. (Here we used our assumption that $\pi(i) > 0$ for all i, but we remark that this assumption can be made without loss of generality, because all v_i with $\pi(i) = 0$ can be placed in S.) How low should k be so that the cut also cuts edges connected to s? This may happen only when $k \leq \Delta$ (and will necessarily happen when $k < \Delta$), where $\Delta = \min_S \Delta(S)$. The reason is the following. Cutting a set $S \subset U$ from s costs $k\pi(S)$. This needs to be offset by a gain on the t side, resulting from the fact that edges between t and vertices of W labeled by $S \times S$ no longer need to be cut. This gives a saving of $\sum_{i,j \in S} \omega(i,j)$. The saving equals the cost precisely when $k = \Delta$.

Using the above analysis, it follows that by performing a search over the parameter k, one can find the value of Δ and the densest set S achieving this value.

Acknowledgments. We would like to thank the anonymous reviewers for their very useful comments on an earlier version of this manuscript. The second author would like to thank Erik D. Demaine and Mohammad Ali Safari for helpful comments and discussions.

REFERENCES

[1] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev, *$O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, ACM, New York, 2005, pp. 573–581.

[2] N. Alon, P. Seymour, and R. Thomas, *A separator theorem for graphs with excluded minor and its applications*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (Baltimore, 1990), ACM, New York, 1990, pp. 293–299.

[3] N. Alon, P. Seymour, and R. Thomas, *A separator theorem for nonplanar graphs*, J. AMS, 3 (1990), pp. 801–808.

[4] E. Amir, *Efficient approximation for triangulation of minimum treewidth*, in Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, New York, 2001, pp. 7–15.

[5] E. Amir, R. Krauthgamer, and S. Rao, *Constant factor approximation of vertex-cuts in planar graphs*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 90–99.

[6] S. Arora, J. R. Lee, and A. Naor, *Euclidean distortion and the sparsest cut*, J. Amer. Math. Soc., 21 (2008), pp. 1–21.

[7] S. Arora, S. Rao, and U. Vazirani, *Expander flows, geometric embeddings, and graph partitionings*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 222–231.

[8] Y. Aumann and Y. Rabani, *An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm*, SIAM J. Comput., 27 (1998), pp. 291–301.

[9] S. N. Bhatt and F. T. Leighton, *A framework for solving VLSI graph layout problems*, J. Comput. System Sci., 28 (1984), pp. 300–343.

[10] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998), pp. 1–45.

[11] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks, *Approximating treewidth, pathwidth, frontsize, and shortest elimination tree*, J. Algorithms, 18 (1995), pp. 238–255.

[12] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca, *On treewidth approximations*, in 1st Cologne-Twente Workshop on Graphs and Combinatorial Optimization, Electron. Notes Discrete Math. 8, Elsevier, Amsterdam, 2001.

[13] V. Bouchitté and I. Todinca, *Treewidth and minimum fill-in: Grouping the minimal separators*, SIAM J. Comput., 31 (2001), pp. 212–232.

[14] J. Bourgain, *On Lipschitz embedding of finite metric spaces in Hilbert space*, Israel J. Math., 52 (1985), pp. 46–52.

[15] T. N. Bui and C. Jones, *Finding good approximate vertex and edge partitions is NP-hard*, Inform. Process. Lett., 42 (1992), pp. 153–159.

[16] N. Burbaki, *Elements of Matematics. Algebra*, Springer-Verlag, New York, 2006.

[17] S. Chawla, A. Gupta, and H. Räcke, *Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, Canada, 2005, pp. 102–111.

[18] C. Chekuri, S. Khanna, and B. Shepherd, *Multicommodity flow, well-linked terminals, and routing problems*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, ACM, New York, 2005, pp. 183–192.

[19] E. D. Demaine and M. Hajiaghayi, *Linearity of grid minors in treewidth with applications through bidimensionality*, Combinatorica, to appear. A preliminary version appeared in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005).

[20] E. D. Demaine and M. Hajiaghayi, *Bidimensionality: New connections between FPT algorithms and PTASs*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, Canada, 2005, pp. 590–601.

[21] E. D. Demaine, M. Hajiaghayi, N. Nishimura, P. Ragde, and D. M. Thilikos, *Approximation algorithms for classes of graphs excluding single-crossing graphs as minors*, J. Comput. System Sci., 69 (2004), pp. 166–195.

[22] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[23] U. Feige, *Relations between average case complexity and approximation complexity*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 534–543.

[24] U. Feige and S. Kogan, *Hardness of Approximation of the Balanced Complete Bipartite Subgraph Problem*, Technical report MCS04-04, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 2004.

[25] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan, *A separator theorem for graphs of bounded genus*, J. Algorithms, 5 (1984), pp. 391–407.

[26] M. Grohe, *Local tree-width, excluded minors, and approximation algorithms*, Combinatorica, 23 (2003), pp. 613–632.

[27] L. H. Harper, *Optimal numberings and isoperimetric problems on graphs*, J. Combin. Theory, 1 (1966), pp. 385–393.

[28] J. A. Kelner, *Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus*, SIAM J. Comput., 35 (2006), pp. 882–902.

[29] S. Khot, *Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique*, SIAM J. Comput., 36 (2006), pp. 1025–1071.

[30] S. Khot and N. Vishnoi, *The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into L_1*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 53–62.

[31] P. N. Klein, S. A. Plotkin, and S. Rao, *Excluded minors, network decomposition, and multicommodity flow*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 682–690.

[32] R. Krauthgamer, J. R. Lee, M. Mendel, and A. Naor, *Measured descent: A new embedding method for finite metrics*, Geom. Funct. Anal., 15 (2005), pp. 839–858.

[33] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[34] J. R. Lee, *On distance scales, embeddings, and efficient relaxations of the cut cone*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, Canada, 2005, pp. 92–101.

[35] F. T. Leighton, *Complexity Issues in VLSI: Optimal Layout for the Shuffle-Exchange Graph and Other Networks*, MIT Press, Cambridge, MA, 1983.

[36] T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.

[37] C. Leiserson, *Area-efficient graph layouts (for VLSI)*, in Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1980, pp. 270–280.

[38] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.

[39] R. J. Lipton and R. E. Tarjan, *Applications of a planar separator theorem*, SIAM J. Comput., 9 (1980), pp. 615–627.

[40] J. Matoušek and Y. Rabinovich, *On dominated l_1 metrics*, Israel J. Math., 123 (2001), pp. 285–301.

[41] Y. Rabinovich, *On average distortion of embedding metrics into the line and into L_1*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 456–462.

[42] S. Rao, *Small distortion and volume preserving embeddings for planar and Euclidean metrics*, in Proceedings of the 15th Annual Symposium on Computational Geometry, ACM, New York, 1999, pp. 300–306.

[43] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[44] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

[45] D. B. West, *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, NJ, 1996.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 658–701

TREE-WALKING AUTOMATA DO NOT RECOGNIZE ALL REGULAR LANGUAGES*

MIKOŁAJ BOJAŃCZYK† AND THOMAS COLCOMBET‡

Abstract. Tree-walking automata are a natural sequential model for recognizing tree languages. It is well known that every tree language recognized by a tree-walking automaton is regular. We show that the converse does not hold.

Key words. tree automata, formal languages, tree-walking automata

AMS subject classifications. 68Q45, 03D05

DOI. 10.1137/050645427

1. Introduction. A tree-walking automaton is a natural type of finite automaton for trees. At every moment in a run, a tree-walking automaton is located in one node of the tree. In one step, the automaton moves to a neighboring node and changes its state according to the transition relation. The step depends on the current state of the automaton and some local information: the label of the current node, whether or not it is the root, a leaf, a left child, or a right child. The tree is accepted if one of the accepting states is reached. For instance, a tree-walking automaton can check whether all nodes of the tree have the same label by doing a depth-first search.

Even though tree-walking automata were introduced in the early 1970s by Aho and Ullman [1], not much is known about this model.

This situation is different from the "usual" tree automata—branching tree automata—which are well-understood objects. In particular, top-down, bottom-up, and two-way nondeterministic branching tree automata recognize the same class of tree languages. The tree languages of this class are called *regular*, the name being so chosen because the class enjoys many nice properties of regular word languages. A comprehensive introduction to the standard theory of tree automata can be found in [4].

As tree-walking automata are a particular case of two-way branching automata, tree-walking automata recognize regular tree languages. Closure under union and intersection is also simple. Until recently, however, other fundamental questions pertaining to tree-walking automata remained unanswered:

1. Is every regular tree language recognized by a tree-walking automaton?
2. Can tree-walking automata be determinized?
3. Are tree-walking automata closed under complementation?

Much research has been dedicated to tree-walking automata. There are nondefinability results for weakened models of tree-walking automata [9, 11, 2], as well as definability results for strengthened models of tree-walking automata [9, 5, 7]. A line of research is dedicated to logical characterizations of tree-walking automata [11]

*Received by the editors November 17, 2005; accepted for publication (in revised form) October 9, 2007; published electronically May 23, 2008. This work was partially supported by the EU Research training network "Games and Automata for Synthesis and Validation."
http://www.siam.org/journals/sicomp/38-2/64542.html
†Institute of Informatics, Warsaw University, 02-097 Banacha 2, Warszawa, Poland (bojan@mimuw.edu.pl).
‡IRISA/CNRS, Campus de Beaulieu, F-35042 Rennes Cedex, France (Thomas.Colcombet@irisa.fr).

and their pebble extensions [6]. There has also been some research on tree-walking automata with an output tape—which define tree-to-word transductions [1]—and on expressiveness issues concerning this model [8].

Question 2 has been answered negatively in [3]. Question 3 is still open, the only known result being closure under complementation of deterministic tree-walking automata [10]. The contribution of this paper is to give a negative answer to the first question.

2. Preliminaries and the separating tree language. In this section we define the basic concepts and state our main result.

2.1. Basic definitions. The trees in this paper are finite, binary trees labeled by a given finite alphabet Σ. A Σ-*tree* is a partial mapping $t : \{0,1\}^* \to \Sigma$ of finite, nonempty, and prefix-closed domain $dom(t)$. Elements of this domain are called *nodes* of the tree; the *label* of a node u is the value $t(u)$. Additionally we assume that if $v0$ is a node of the tree, then so is $v1$, and vice versa. Nodes are partially ordered by the prefix relation; when a node x is the prefix of a node y, we say that x is *above* y, or y is *below* x. The least node ε is called the *root*, and maximal nodes are called *leaves*. The nodes are also ordered lexicographically; we say that x is *to the left* (resp., *right*) of y if x and y are incomparable by the prefix relation, and x is lexicographically before y (resp., after y). Given a node u, the *subtree of* t *rooted in* u—we simply say the *subtree of* u when the tree t is clear from the context—is the Σ-tree t' of domain $\{v : uv \in dom(t)\}$ defined for all nodes v of t' by $t'(v) = t(uv)$. The *depth of a node* u is $|u| + 1$, where $|u|$ is the length of u as a word. The *depth of a tree* is the maximal depth of its nodes. A *balanced tree* is one where all leaves are at the same depth.

A set of trees over a given alphabet is called a *tree language*. A *regular* tree language is a tree language recognized by a bottom-up branching tree automaton. We assume the reader to be familiar with branching automata; see [4] for further reading. We denote by REG the class of regular tree languages.

We now define (nondeterministic) tree-walking automata. The *type* of a node says whether the node is a leaf and whether it is the root. There are four possible types; we denote the set of types by Types.

DEFINITION 1. *A* tree-walking automaton *is a tuple* $\mathcal{A} = (Q, \Sigma, I, F, \delta)$*, where* Q *is a finite set of* states*;* $I, F \subseteq Q$ *are, respectively, the sets of* initial *and* accepting *states; and* δ *is the* transition relation *of the form*

$$\delta \subseteq (Q \times \text{Types} \times \Sigma \times \{\varepsilon, 0, 1\})^2.$$

A *configuration* is a pair of a state and a node. The automaton can go in one step from a configuration (q_1, v_1) to a configuration (q_2, v_2) if δ contains a transition

$$(q_1, t_1, a_1, d_1, q_2, t_2, a_2, d_2)$$

such that the type and label of v_i are t_i, a_i and there is a node u such that $v_i = u \cdot d_i$ for $i = 1, 2$. A *run* is a nonempty sequence of configurations $c_1, \ldots, c_n$ in which every two consecutive configurations are consistent with the transition relation. We say that such a run is *from* c_1 *to* c_n; if both configurations c_1 and c_n are located at the same node u, then the run is called a *loop in* u. A run is *accepting* if it starts and ends in the root of the tree, the first state is initial, and the last state is accepting. The automaton *accepts* a Σ-tree if it has an accepting run in that tree. The set of Σ-trees accepted is called the tree language *recognized* by the automaton. We use TWA to denote the class of tree languages recognized by some tree-walking automaton.

The reader may be surprised by our definition of tree-walking automata. In other texts, the transition relation is of the form

$$\delta \subseteq Q \times \{\text{root, left child, right child}\} \times \{\text{leaf, nonleaf}\} \times \Sigma \times Q \times \{\uparrow, 0, 1, \varepsilon\}.$$

In this definition—which can easily be shown to be equivalent to the one we use—the second coordinate extends Types by saying whether a node is a left or right child, while the $\{\uparrow, 0, 1, \varepsilon\}$ causes the automaton to move to the parent, left child, or right child of the current node (or not move at all). The point of using our slightly more verbose definition is that it allows us to easily define "reversed" automata, which visit the tree in a chronologically (or spatially) opposite manner. We will comment on these reversed automata in the next section.

We would like to point out here that testing whether a node is a left or right child is an essential feature of a tree-walking automaton. Indeed, Kamimura and Slutzki show in [9] that tree-walking automata that do not have access to this information cannot even test whether all nodes in a tree have the same label.

Symmetry principles. As pointed out before, our definition of tree-walking automata—in particular in their nondeterministic form—easily adapts itself to symmetry arguments, which are very convenient in the proofs. There are two symmetries: *time symmetry* and *space symmetry*. Their formal definition is in terms of transformations of a tree-walking automaton; namely, each automaton has a time-reversed and a space-reversed variant. The *time-reversed automaton of* $\mathcal{A}$, denoted $\mathcal{A}^{-T}$, is obtained from $\mathcal{A}$ by replacing each transition

$$(q_1, t_1, a_1, d_1, q_2, t_2, a_2, d_2) \in \delta$$

with the transition

$$(q_2, t_2, a_2, d_2, q_1, t_1, a_1, d_1).$$

On the other hand, the *space-reversed automaton of* $\mathcal{A}$, denoted $\mathcal{A}^{-S}$, is obtained by replacing each transition

$$(q_1, t_1, a_1, d_1, q_2, t_2, a_2, d_2) \in \delta$$

with the transition

$$(q_1, t_1, a_1, s(d_1), q_2, t_2, a_2, s(d_2)),$$

where $s(\varepsilon) = \varepsilon$, $s(0) = 1$, and $s(1) = 0$. One can easily see that

$$(\mathcal{A}^{-S})^{-T} = (\mathcal{A}^{-T})^{-S}.$$

We extend the space symmetry $s : \{\varepsilon, 0, 1\} \to \{\varepsilon, 0, 1\}$ mapping to nodes ($s : \{0,1\}^* \to \{0,1\}^*$) and trees in the natural manner. The following obvious fact encapsulates the properties of the reversed automata.

FACT 2.1. *Let* $(q_1, v_1), \ldots, (q_n, v_n)$ *be a run of* $\mathcal{A}$ *in a tree* t.

- $(q_n, v_n), \ldots, (q_1, v_1)$ *is a run of* $\mathcal{A}^{-T}$ *in* t.
- $(q_1, s(v_1)), \ldots, (q_n, s(v_n))$ *is a run of* $\mathcal{A}^{-S}$ *in* $s(t)$.

We say that an automaton is *isomorphic* to another if there exists a one-to-one mapping i from the states of the first onto the states of the second such that there exists a transition

$$(q_1, t_1, a_1, d_1, q_2, t_2, a_2, d_2)$$

in the first automaton if and only if there exists a transition

$$(i(q_1), t_1, a_1, d_1, i(q_2), t_2, a_2, d_2)$$

in the second automaton. In particular, the isomorphism notion does not involve the initial or the accepting states of the automata.

We say an automaton is *self-symmetric* if it is isomorphic to its time-reversed and also to its space-reversed automaton. By adding (unreachable) states to an automaton, any tree-walking automaton can be made self-symmetric. In what follows, self-symmetric automata will be very convenient in reducing the number of cases to be analyzed. Assume, for instance, that we have proved a statement of the form "for every two states p, q, any run from p in the root to q in a leaf can be transformed into one without loops." If the automaton for which we proved this statement was self-symmetric, we would also get the following statement for free: "for every two states p, q, any run from q in a leaf to p in the root can be transformed into one without loops." This is because the time-reversal of a run from the root to a leaf is a run from a leaf to the root, and being loop-free is invariant under time-reversals.

2.2. The separating language. As mentioned in the introduction, it is well known that all tree languages recognized by tree-walking automata are regular:

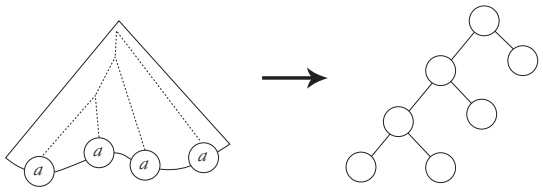
$$\text{TWA} \subseteq \text{REG}. \tag{2.1}$$

Whether this inclusion is strict has long been an open question. Engelfriet, Hoogeboom, and Van Best conjectured that this is indeed the case [7]. The aim of this paper is to establish this conjecture.

In this section we describe a tree language L that is regular, but not accepted by any tree-walking automaton, and therefore witnesses the strictness of the inclusion (2.1).

We consider trees with two possible labels: **a** and **b**. Moreover, **a** is allowed only in the leaves. We sometimes refer to the symbol **b** as the *blank symbol*. Trees containing only the blank symbol are called *blank trees*. In a blank tree, only the set of nodes is important.

Let t be a nonblank tree with **a** occurring only in the leaves. We will now define the *branching structure* $bs(t)$ of t, which is a blank tree (since only the nodes of $bs(t)$, and not their labels, are relevant). We say a node u is a *branching node* of t either if it is an **a**-labeled leaf of t or if both left and right successors of u have **a**-labeled leaves in their subtrees. We define the *branching structure* of t as the (unique) blank tree $bs(t)$ such that there is a bijection between the branching nodes of t and the nodes of $bs(t)$ that preserves the prefix relation and the lexicographic ordering of nodes. The following drawing illustrates this definition with an example:



Let K be the set of blank trees where all leaves are at even depth. The separating tree language L mentioned at the beginning of this section is $bs^{-1}(K)$, i.e., the set

of trees whose branching structures have all leaves at even depth. We now state the main result of this paper as follows.

THEOREM 2. *The tree language L is regular, but is not recognized by any tree-walking automaton.*

Showing that L is regular is not difficult: it is recognized by a bottom-up deterministic automaton with three states recognizing, respectively, the set of blank trees, the set of trees whose branching structure has only branches of even length, and the set of trees whose branching structure has only branches of odd length. For completeness, a fourth "error" state can be added.

Fact 2.2 below shows that a stronger result holds.

FACT 2.2. *The tree language L is definable in first-order logic with the prefix relation and the left and right successor relations.*

Proof. First note that there is a first-order logic formula with one free variable, which is true in exactly the branching nodes of a tree. Therefore, the nodes of the branching structure $bs(t)$ can be interpreted as the branching nodes of t. (Note that for a node interpreted by v, its left successor is interpreted as "first branching node lexicographically after or at the left successor of v," and similarly for the right successor.) It follows that if K is a property of branching structures defined by a first-order formula φ, then $bs^{-1}(K)$ is also defined by a first-order formula. The first-order formula for $bs^{-1}(K)$ is obtained from φ by restricting quantification to branching nodes and replacing the left/right successors by their abovementioned interpretations (the prefix relation is not changed).

Therefore, the statement of the fact will follow if we establish that the language K is definable in first-order logic. This latter result is Lemma 5.1.8 in [12]. For completeness, we provide a proof for it below.

The main idea is that first-order logic can express whether a leaf in $(01)^*(\varepsilon + 0)$ is at even depth or not. We will refer to such a leaf as the *middle leaf* of the tree. A first-order formula can detect the middle leaf by checking that each node above it is either the leaf itself, the father of the leaf, the right child of a left child, the left child of a right child, the left child of the root, or the root itself. The *middle parity* of a tree is defined to be the parity of the depth of the middle leaf; it is definable in first-order logic since the middle leaf is at even depth if and only if it is a left child. The *middle parity* of a node is defined to be the middle parity of the subtree rooted at this node.

Let M be the set of trees whose middle parity is even, and for which all children of any internal node have the same middle parity. We claim that $K = M$. According to the previous remarks, this implies that K is definable in first-order logic.

The inclusion $K \subseteq M$ is obvious. For the other direction, let t be a tree outside K. If all leaves in t have the same depth parity, then the middle parity is odd and $t \notin M$. Otherwise, consider a node in t of maximal depth whose subtree has leaves of both even and odd depth. But then by maximality, the middle parities of this node's children must be different and $t \notin M$. □

The hard part in the proof of Theorem 2 remains: we need to show that the tree language L is not recognized by any tree-walking automaton. The remainder of this paper is devoted to proving this result.

In the next section, we define *patterns*; these are the same as those used in [3]. A pattern is a particular type of tree with distinguished nodes, called ports. As in [3], we consider three particular patterns (the *basic patterns*) that confuse a tree-walking automaton. Then, in section 4, to every blank tree t we associate a tree made out of basic patterns (its *pattern expansion*) whose branching structure—where ports

are considered as **a**-leaves—is t and is confusing for the automaton. We then study throughout sections 5 and 6 the possible runs of the automaton in expansions and their images in the original tree t. This study results in a precise understanding of the behavior of the automaton in expansions: it can perform only a fixed number of simple behaviors—such as left-to-right depth-first search, a move back to the root, or nondeterministic search of a leaf—and can only (nondeterministically) switch between these behaviors a bounded number of times. In section 7, we use this knowledge about the tree-walking automaton for proving that it cannot recognize L. In particular, we show that a simple local transformation, called the *rotation*, applied to a sufficiently big tree cannot be detected by the automaton while it transforms a tree in the language into one which is not in the language.

A more detailed overview of the proof is found in section 4.2, after the concepts of pattern and pattern expansion have been introduced.

We fix for the remainder of the paper a tree-walking automaton

$$\mathcal{A} = (Q, \{\mathbf{a}, \mathbf{b}\}, I, F, \delta).$$

Eventually, we will show that $\mathcal{A}$ cannot recognize the tree language L. As noted in the section on symmetry, we assume without loss of generality that the automaton $\mathcal{A}$ is self-symmetric. We also assume without loss of generality that the automaton has at least two states.

3. Patterns. In this section we define patterns, develop a pumping argument for them, and then study its consequences for the automaton.

Patterns are fragments of trees with holes (called ports) in them. There are two types of ports: leaf ports, which are in the leaves, and the root, which is also called a port. Patterns can be assembled by gluing the root port of one pattern to a leaf port of another pattern. A tree-walking automaton naturally induces an equivalence relation on such patterns: two patterns (with the same number of ports) are equivalent if in any context the automaton cannot detect the difference when one pattern is replaced by another. This equivalence relation (technically, a slightly finer one, which speaks of states of the automaton) is the key notion in the study of patterns.

3.1. Patterns and pattern equivalence. A *pattern* is an $\{\mathbf{a}, \mathbf{b}, *\}$-tree where the labels **a** and $*$ are found only in the leaves. For technical reasons we require that a pattern have at least two nodes and that all $*$-labeled leaves be left children. A *blank pattern* is any pattern with no **a**-labeled leaf. The ith $*$-labeled leaf (numbered from left to right, starting from 0) is called the ith *port*. We number the ports from 0 to be consistent with the usual tree terminology, where a left successor is denoted by 0 and a right successor by 1. *Port* ε stands for the root. The number of leaf ports is called the *arity* of the pattern. In particular, patterns of arity 0 are $\{\mathbf{a}, \mathbf{b}\}$-trees. See Figure 3.1 for an illustration. Given an n-ary pattern Δ and n patterns $\Delta_0, \ldots, \Delta_{n-1}$, the *composition* $\Delta[\Delta_0, \ldots, \Delta_{n-1}]$ is the pattern obtained from Δ by simultaneously substituting each pattern Δ_i for the ith port. We also allow some Δ_i's to be $*$. In this case, nothing is changed for the corresponding ports. We write $\Delta[\Delta_i/i]$ in the particular case where all Δ_j's but Δ_i are $*$; i.e., a single substitution is performed at port i. Given a set P of patterns, we denote by $\mathcal{C}(P)$ the least set of patterns which contains P and is closed under composition.

A run in a pattern is defined just as a run in a tree, except that the ports (both root and leaf) are treated as being nonleaf left children with the blank label. The latter assumption is for technical reasons; it will allow us to compose runs in larger

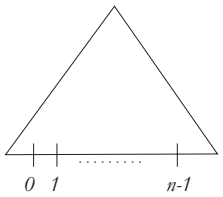


FIG. 3.1. *A pattern of arity n.*

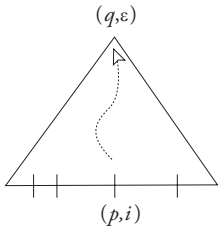


FIG. 3.2. *A pattern* Δ *with* (p, i, q, ε) *in* δ_Δ.

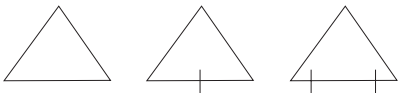


FIG. 3.3. *The patterns* Δ_0, Δ_1, *and* Δ_2.

patterns from runs in smaller ones. Moreover, we require that a run in a pattern visit ports at most twice: a port may occur only in the first and last configurations. In the following definition, illustrated by Figure 3.2, we show how to describe a transition relation corresponding to a pattern for the automaton.

DEFINITION 3. *The* automaton's transition relation over an n-ary pattern Δ,

$$\delta_\Delta \subseteq (Q \times \{\varepsilon, 0, \ldots, n-1\})^2,$$

contains (p, i, q, j) *if in* Δ *there is a run from state* p *in port* i *to state* q *in port* j.

From the point of view of the automaton, the relation δ_Δ sums up all important properties of a pattern, and we consider two patterns *equivalent* if they induce the same δ relation; i.e., patterns Δ and Δ' are equivalent if $\delta_\Delta = \delta_{\Delta'}$. This equivalence relation is a congruence with respect to composition of patterns, thanks to the technical assumptions. The essence of this equivalence is that if one replaces a subpattern by an equivalent one, the automaton is unable to see the difference. Here, we only consider contexts that are consistent with our technical assumptions: the root of the pattern corresponds to a left child, and the nodes plugged into the leaf ports are not leaves and have the blank label.

The following lemma was shown in [3, Lemma 9].

LEMMA 3.1. *There are blank patterns* $\Delta_0, \Delta_1, \Delta_2$*—of respective arities* 0, 1, *and* 2*—such that any pattern in* $\mathcal{C}(\{\Delta_0, \Delta_1, \Delta_2\})$ *of arity* $i = 0, 1, 2$ *is equivalent to* Δ_i.

These patterns will be used a lot in the constructions below. To keep the drawings uncluttered, we omit specifying the names Δ_0, Δ_1, and Δ_2, as this information can be reconstructed from the number of leaves; see Figure 3.3.

Note that the lemma may fail for $i = 3$ when nondeterministic automata are involved; see [3]. The patterns Δ_0, Δ_1, and Δ_2 are the key to our proof. In a sense, their construction encapsulates all of the pumping arguments that we will do with respect to the automaton $\mathcal{A}$. For instance, the pattern Δ_1 is equivalent to a composition of any number of copies of Δ_1 patterns. In particular, if the automaton can go from the leaf port of Δ_1 to the root port, then there must be a state that is used twice along the way. We write $\mathcal{C}_\mathcal{A}$ to denote the set $\mathcal{C}(\{\Delta_0, \Delta_1, \Delta_2\})$; from now on almost all patterns considered will be taken from $\mathcal{C}_\mathcal{A}$.

3.2. Inner loops. Although simply defined, the relation δ_Δ is rather cumbersome to work with. The automaton may do some redundant moves, such as going one step down and then one step up, without any apparent purpose (a phenomenon called oscillation in [3]). It is convenient to eliminate this obfuscating phenomenon. This is the purpose of the inner loop relation introduced in the next definition.

First, however, we state Fact 3.2, which is a consequence of Lemma 3.1. In this statement and elsewhere, by the expression *plugging the Δ_1 pattern into some/any port of a pattern Δ*, we refer to one of the patterns $\Delta_1[\Delta], \Delta[\Delta_1/1], \ldots, \Delta[\Delta_1/n]$, where n is the arity of the pattern Δ. Similarly, the pattern obtained by *plugging Δ_1 into all ports of a pattern Δ* represents the pattern $\Delta_1[\Delta[\Delta_1, \ldots, \Delta_1]]$.

FACT 3.2. *Plugging the Δ_1 pattern into some port of a pattern in $\mathcal{C}_\mathcal{A}$ yields an equivalent pattern.*

Proof. The proof is by induction on the structure of the pattern, using Lemma 3.1 as the basis of the induction. □

Consider now a composition of two patterns $\Delta[\Delta'/i]$ and the junction of these patterns, i.e., the node v that corresponds to port i in Δ. By Fact 3.2, we may well assume that v is on the junction of two Δ_1 patterns: one plugged into the leaf port i of Δ and one plugged into the root port of Δ'. In particular, any loop that can be done on the junction of two Δ_1 patterns can be replicated in $\Delta[\Delta'/i]$. Hence the importance of such loops; we call them "inner loops" in the following definition (see also Figure 3.4 for an illustration).

DEFINITION 4. *The* inner loop relation *over states is the least transitive and reflexive relation $\to_\varepsilon$ over states such that $p \to_\varepsilon q$ holds whenever $(p, \varepsilon, q, \varepsilon)$ or $(p, 0, q, 0)$ belongs to δ_{Δ_1}.*

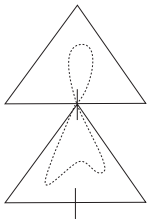


FIG. 3.4. *An inner loop.*

The following lemma formalizes the comments preceding the introduction of the inner loop relation. It shows how $\to_\varepsilon$ describes all possible loops on interfaces between patterns from $\mathcal{C}_\mathcal{A}$.

LEMMA 3.3. *Let $\Delta, \Delta' \in \mathcal{C}_\mathcal{A}$ be patterns of nonzero arity, and let v be the junction node corresponding to the leaf port i of Δ and the root of Δ'. There is an inner loop $p \to_\varepsilon q$ if and only if there is a run from (p, v) to (q, v) in $\Delta[\Delta'/i]$.*

Before proceeding with the proof, we would like to comment on the relevance of this lemma. Recall that by our definition of runs in patterns, the loop from (p, v) to (q, v) is not allowed to visit any of the ports. Therefore, the relation $\to_\varepsilon$ tells us what possible loops can be done on the interface of two patterns without visiting any ports. In particular, the possible types of such loops do not depend on the two patterns Δ and Δ', as long as they are from $\mathcal{C}_\mathcal{A}$.

Another important consequence of this lemma is that it gives us a sort of normal form of runs through patterns in $\mathcal{C}_\mathcal{A}$. Any loop on a junction between patterns can be replaced by the $\to_\varepsilon$ relation; therefore a run through a pattern in $\mathcal{C}_\mathcal{A}$ can be seen as going directly from the source to the target, with all the loops being represented by the $\to_\varepsilon$ relation.

Proof. Assume that $p \to_\varepsilon q$. By definition of $\to_\varepsilon$, there is a run from (p, w) to (q, w) in $\Delta_1[\Delta_1]$, where w is at the junction of the two Δ_1 patterns. But this run can be reused within $\Delta[\Delta'/i]$, since by Fact 3.2 we may assume without loss of generality that both Δ and Δ' have Δ_1 plugged into all their ports.

Reciprocally, assume that there is a run from (p, v) to (q, v) in the pattern $\Delta[\Delta'/i]$. This run can be reused in the same pattern where a Δ_0 has been substituted for all ports except for some leaf port—say port 0—of Δ'. By Lemma 3.1, this new pattern is equivalent to the composition of two Δ_1 patterns. It then follows by definition that $p \to_\varepsilon q$ holds. □

DEFINITION 5. *For a pattern Δ, the relation γ_Δ is the set of tuples (p, i, q, j) such that $p \to_\varepsilon p'$ and $q' \to_\varepsilon q$ for some p', q' satisfying $(p', i, q', j) \in \delta_\Delta$.*

Observe that a consequence of the definition above is that if $p \to_\varepsilon q$, then (p, i, q, i) belongs to γ_Δ for all ports i of Δ.

The γ relation has nicer closure properties than δ; hence from now on we will be using it—and not the δ relation—to describe runs in patterns. For instance, γ satisfies the following "swallowing" property:

$$(p, \varepsilon, q, 0), (q, \varepsilon, r, 0), (r, 0, s, \varepsilon) \in \gamma_{\Delta_1} \qquad \text{implies} \qquad (p, \varepsilon, s, 0) \in \gamma_{\Delta_1}.$$

This is because $(q, \varepsilon, r, 0), (r, 0, s, \varepsilon) \in \gamma_{\Delta_1}$ implies $q \to_\varepsilon s$. Another useful property of the γ relation—resulting from the equivalence of Δ_1 and $\Delta_1[\Delta_1]$—is as follows:

$$(p, \varepsilon, q, 0) \in \gamma_{\Delta_1} \qquad \text{iff} \qquad (p, \varepsilon, r, 0), (r, \varepsilon, q, 0) \in \gamma_{\Delta_1} \text{ for some } r.$$

Note that the left-to-right implication fails for the δ relation, since the state r may require some loops on the junction between the Δ_1 patterns before the run reaches $(q, 0)$.

Obviously, if two patterns are equivalent, then they have the same γ relations. Let us remark also that a form of the converse also holds: if two patterns in $\mathcal{C}_\mathcal{A}$ have the same γ relations, then they are equivalent. However, this fact is of no use in the remainder of the proof, and we need not establish it.

4. Pattern expansions and the proof strategy. In this section we introduce pattern expansions and then give an overview of our proof strategy.

4.1. Pattern expansions. The *pattern pre-expansion* of a blank tree t is the pattern obtained by replacing every inner node of t with the pattern Δ_2 and replacing every leaf with a port $*$. Thus, the pattern pre-expansion has as many leaf ports as t has leaves.

The *pattern expansion* Δ_t of t is the pattern obtained by plugging Δ_1 into all ports of the pattern pre-expansion (see Figure 4.1). Note that the expansion and

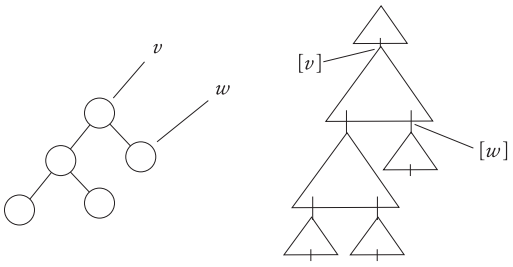


FIG. 4.1. *A blank tree t and its pattern expansion Δ_t.*

the pre-expansion are equivalent as patterns. With every node v of t we associate a node $[v]$ in the pattern Δ_t in the natural way (see Figure 4.1); this node does not depend on t. A *junction node* in a pattern expansion is any node of the form $[v]$; it is called a *junction leaf* when v is a leaf of t. Note that a junction leaf is not a leaf in the pattern Δ_t, since it has Δ_1 as its subtree. The Δ_1 patterns plugged into the pattern expansion are used so that every junction node is on the interface of two patterns of nonzero arity in $\mathcal{C}_{\mathcal{A}}$. In particular, junction nodes are a suitable place for using Lemma 3.3.

We denote by $\Delta_{\mathbf{a}}$ a fixed pattern of arity 0 equal to $\Delta_1[\Delta'_{\mathbf{a}}]$, where $\Delta'_{\mathbf{a}}$ is some zero arity pattern containing exactly one $\mathbf{a}$-labeled leaf. The particular form of $\Delta'_{\mathbf{a}}$ is not important, but we can fix it to be a two-leaf tree with $\mathbf{a}$ in the left leaf and $\mathbf{b}$ in the other nodes. The only two important points concerning $\Delta_{\mathbf{a}}$ are first, that it contains a single leaf labeled by $\mathbf{a}$, and second, that $\Delta_{\mathbf{a}}$ is equivalent to $\Delta_1[\Delta_{\mathbf{a}}]$. This last point is obtained by remarking that $\Delta_{\mathbf{a}}$ equals $\Delta_1[\Delta'_{\mathbf{a}}]$ which is equivalent to $\Delta_1[\Delta_1[\Delta'_{\mathbf{a}}]]$, because $\Delta_1[\Delta_1]$ is equivalent to Δ_1 and equivalence is a congruence with respect to composition.

Given a blank tree t, the tree $\Delta_t^{\mathbf{a}}$ is obtained by plugging $\Delta_{\mathbf{a}}$ into all leaf ports of Δ_t; i.e., $\Delta_t^{\mathbf{a}} = \Delta_t[\Delta_{\mathbf{a}}, \ldots, \Delta_{\mathbf{a}}]$. Clearly the branching structure of $\Delta_t^{\mathbf{a}}$ is t. If the tree-walking automaton were to accept the tree language L, it would have to accept every tree $\Delta_t^{\mathbf{a}}$ for $t \in K$ and reject every tree $\Delta_t^{\mathbf{a}}$ for $t \notin K$. We will eventually show that this is impossible, due to the way tree-walking automata get lost in pattern expansions.

In order to avoid confusion we remark here that $\Delta_t^{\mathbf{a}}$ is treated as a tree and not a pattern of zero arity. Therefore, a run over $\Delta_t^{\mathbf{a}}$ is allowed to visit the root several times, as opposed to runs over patterns of zero arity.

A *junction configuration* is defined to be a configuration of the form $(q, [v])$ for some node $v \in \{0,1\}^*$. We will write such a configuration as $[q, v]$. If v is a node of a blank tree t, then $[q, v]$ can be interpreted as a configuration in either the pattern Δ_t or the tree $\Delta_t^{\mathbf{a}}$. In either case, $[q, v]$ is a configuration whose node is a junction node. Moreover, if v is a leaf of t (i.e., $[v]$ is a junction leaf), the junction configuration is also called a *leaf configuration* (this, of course, is relative to the tree t). We use square brackets for junction configurations; these describe configurations in the branching structure t. On the other hand, normal configurations are written with round brackets; these describe configurations in the pattern expansion Δ_t or in the tree $\Delta_t^{\mathbf{a}}$.

The following two lemmas show that the $\rightarrow_\varepsilon$ relation and the γ_{Δ_2} relation describe the way our fixed tree-walking automaton can move in a pattern expansion, from a junction node to itself or to a neighboring junction node, respectively.

LEMMA 4.1. *Let t be a blank tree and v a node of t. The following statements*

are equivalent for any states p and q:

1. *In the pattern expansion Δ_t there is a run from $[p, v]$ to $[q, v]$.*
2. $p \to_\varepsilon q$.
3. *In the pattern expansion Δ_t there is a run from $[p, v]$ to $[q, v]$ that does not visit any other junction node $[w]$, $w \neq v$.*

Proof. From 1 to 2. By cutting the pattern expansion Δ_t at the junction node $[v]$, we can decompse Δ_t as $\Delta[\Delta'/i]$ for two patterns Δ and Δ' in $\mathcal{C}_\mathcal{A}$ of nonzero arity, and i the number of $[v]$ as a leaf port of Δ. Applying Lemma 3.3 to this decomposition, we get $p \to_\varepsilon q$.

From 2 to 3. Let Δ and Δ' be either Δ_1 or Δ_2, and let i be a leaf port of Δ. Let v' be the port i of Δ. By using $p \to_\varepsilon q$ and applying Lemma 3.3 to $\Delta[\Delta'/i]$, we obtain that there is a run from (v', p) to (v', q) which does not visit the ports of $\Delta[\Delta'/i]$. By definition of the pattern expansion Δ_t, the junction node $[v]$ appears either as port 0 of a Δ_1 pattern or as port 0 or 1 of a Δ_2 pattern. Similarly it is also the root port of a Δ_1 or a Δ_2 pattern. Hence $[v]$ can be identified with node v' in a pattern $\Delta[\Delta'/i]$ above. We can transfer the run we had witnessed on $\Delta[\Delta'/i]$ to Δ_t, obtaining a run from $[p, v]$ to $[q, v]$ that does not visit any other junction node $[w]$, $w \neq v$.

From 3 to 1. The proof is straightforward. □

LEMMA 4.2. *Let t be a blank tree and $v \cdot a, v \cdot b$ nodes of t, with $v \in \{0, 1\}^*$ and $a, b \in \{\varepsilon, 0, 1\}$. The following statements are equivalent for any states p and q:*

1. *In the pattern expansion Δ_t there is a run from $[p, v \cdot a]$ to $[q, v \cdot b]$.*
2. $(p, a, q, b) \in \gamma_{\Delta_2}$.
3. *In the pattern expansion Δ_t there is a run from $[p, v \cdot a]$ to $[q, v \cdot b]$ that does not visit any other junction node $[w]$, $w \notin \{v \cdot a, v \cdot b\}$.*

Proof. From 1 to 2. Assume there is a run in Δ_t from configuration $[p, v \cdot a]$ to configuration $[q, v \cdot b]$. Let us first treat the case $a = b$. In this case $p \to_\varepsilon q$ by Lemma 4.1. Hence $(p, a, q, b) \in \gamma_{\Delta_2}$ (recall the observation following Definition 5). Now let $a \neq b$, and set $c \in \{\varepsilon, 0, 1\}$ to be different from a and b. Let p' be the last state assumed by the run while visiting the junction node $[v \cdot a]$, and let q' be the first state assumed at the junction node $[v \cdot b]$ after crossing $[v \cdot a]$ for the last time. If the corresponding subrun from $[p', v \cdot a]$ to $[q', v \cdot b]$ does not visit $[v \cdot c]$, then (p', a, q', b) belongs to δ_{Δ_2}. Otherwise it visits $[v \cdot c]$, and we let p'' and q'' be the first and last state, respectively, assumed by that subrun at $[v \cdot c]$. Then $p'' \to_\varepsilon q''$, by Lemma 4.1. Also, by construction, (p', a, p'', c) and (q'', c, q', b) are in δ_{Δ_2}. This shows (by plugging Δ_1 into port c of Δ_2) that (p', a, q', b) belongs to δ_{Δ_2}. Moreover, by Lemma 4.1, we have $p \to_\varepsilon p'$ and $q' \to_\varepsilon q$. By definition of γ_{Δ_2}, $(p, a, q, b) \in \gamma_{\Delta_2}$ follows.

From 2 to 3. Since $(p, a, q, b) \in \gamma_{\Delta_2}$, there exist two states p' and q' such that

$$p \to_\varepsilon p', \quad (p', a, q', b) \in \delta_{\Delta_2}, \quad \text{and} \quad q' \to_\varepsilon q.$$

By Lemma 4.1, these three properties provide a run in the pattern expansion Δ_t that successively passes through the configurations

$$[p, v \cdot a],\ [p', v \cdot a],\ [q', v \cdot b],\ [q, v \cdot b]$$

without visiting any junction node other than $[v \cdot a]$ and $[v \cdot b]$.

From 3 to 1. The proof is straightforward. □

The above lemma shows that runs of the automaton between neighboring junction nodes in pattern expansions can be assumed to have a very particular form. Take, for instance, a blank tree t and two nodes v and w, with v above w. If there is a run

in Δ_t that goes from $[v]$ to $[w]$, then, by Lemma 4.2, there is a run that does this by going directly from v to w using the shortest path. This means performing only a series of "steps" of the form $(p, \varepsilon, q, 0), (p, \varepsilon, q, 1) \in \gamma_{\Delta_2}$. A similar characterization holds when v and w are incomparable: the automaton first goes directly from $[v]$ in the up direction, then does one of the steps $(p, 0, q, 1), (p, 1, q, 0) \in \gamma_{\Delta_2}$ (a "go to the sibling" move), and then goes directly downward to $[w]$. This principle is formalized in Lemma 6.1.

4.2. The proof strategy. We are now ready to give an overview of the proof strategy. Recall that our aim is to find trees $s \in L$ and $s' \notin L$ such that any accepting run in s can be transferred to s'. In fact, the trees s, s' will be, respectively, of the form $\Delta_t^{\mathbf{a}}$ and $\Delta_{t'}^{\mathbf{a}}$ for some blank trees $t \in K$ and $t' \notin K$. Therefore, we need to develop a good understanding of runs within trees of the form $\Delta_t^{\mathbf{a}}$.

The remainder of this paper is divided into three sections, which correspond to ever larger parts of a run over a tree of the form $\Delta_t^{\mathbf{a}}$. Such a run can be analyzed on three scales.

The greatest scale is analyzed in section 7. Fix a tree $\Delta_t^{\mathbf{a}}$. In this greatest scale, we will be most interested in runs that connect leaf configurations to one another, without passing through the root of the tree. (This is because, without loss of generality, we may assume the root is visited at most $|Q|$ times.) Consider such a run that goes from one leaf configuration $[p, v]$ to another leaf configuration $[q, w]$. Within such a run, we can isolate all the intermediate leaf configurations:

$$[p, v] = [r_1, u_1], \ldots, [r_n, u_n] = [q, w].$$

Since no leaf configurations are visited in the meantime, a run from $[r_i, u_i]$ to $[r_{i+1}, u_{i+1}]$ corresponds to either (a) a loop in the root of the pattern $\Delta_1[\Delta_{\mathbf{a}}]$, or (b) a run from one junction leaf to another in the pattern Δ_t. Case (a) can be treated as a sort of ε-transition for leaf configurations. The interesting case is (b).

In section 6, we treat the runs of type (b), which correspond to the intermediate scale. These runs are in $\Delta_t^{\mathbf{a}}$, but since no $\mathbf{a}$-labeled leaf is visited during those runs, they are also runs in the pattern expansion Δ_t. We first show that whether or not there is a run of type (b) from $[r_i, u_i]$ to $[r_{i+1}, u_{i+1}]$ does not depend on the tree t but only on the nodes u_i and u_{i+1}. This allows us to consider the notation $[p, v] \to [q, w]$, meaning that there is a run from $[p, v]$ to $[q, w]$ in some (equivalently, every) pattern expansion Δ_t for which v, w are nodes of t. A type of run that realizes $[p, v] \to [q, w]$ is called a *move*; a classification of the possible types of move is the subject of section 6.

As preparation, in section 5, we consider the smallest scale: the pattern Δ_2. By Lemma 4.2, any move within a pattern expansion can be decomposed into a certain number of traversals of the pattern Δ_2. Hence the need for an investigation of the relation γ_{Δ_2}.

Before proceeding, we describe in general terms the results of these investigations in sections 5, 6, and 7.

The main result of section 5 is Proposition 5.10, which gives a characterization of the possible ways the automaton can go from the leaf port to the root port of Δ_1. Generally speaking, this characterization says that the automaton either gets completely lost or must do some sort of depth-first search. Even though stated in terms of the Δ_1 pattern, these results can also be applied to the Δ_2 pattern. Indeed, by Lemma 3.1, any run from a leaf port to the root port in Δ_2 can also be transfered to Δ_1, by simply plugging the unused leaf port with Δ_0.

The main result of section 6 is Proposition 6.10. This proposition roughly says that there are only eleven types of interesting moves between junction leaves in a

pattern expansion. The interesting moves—called elementary moves—are moves such as: "go to the next junction leaf to the left" or "go to any junction leaf to the left." Proposition 6.10 states that if a move is not elementary, then it contains a "shift," a phenomenon of inherent confusion for the automaton.

Finally, in section 7, we show that tree-walking automata cannot detect a properly placed rotation, which concludes the proof. Given a blank tree T and a node x, the *rotation of T with the pivot x* is the tree T' defined as follows: we move the subtrees of $x \cdot 00$, $x \cdot 01$, and $x \cdot 1$ to the new positions $x \cdot 0$, $x \cdot 10$, and $x \cdot 11$ (see Figure 7.1). Clearly doing a rotation in a tree with all leaves at even depth creates a leaf at odd depth. We will show, however, that given a very large balanced blank tree T, one can find a pivot x such that $\Delta^{\mathbf{a}}_{T'}$ cannot be distinguished from $\Delta^{\mathbf{a}}_{T}$ by our fixed tree-walking automaton $\mathcal{A}$.

5. The pattern Δ_2. In this section we investigate the γ relations of the patterns Δ_0, Δ_1, and Δ_2. The main result, Proposition 5.10, uncovers by a case distinction the possible ways the tree-walking automaton can cross the pattern Δ_1. This is important for our analysis of pattern expansions, since by Lemma 4.2 every path through a pattern expansion corresponds to a sequence of traversals of Δ_2 patterns.

$$
\begin{array}{rlll}
p \circlearrowleft q & \text{if} & (p,\varepsilon,q,\varepsilon) \in \gamma_{\Delta_0} & p \nwarrow\!\searrow q \quad \text{if} \quad (p,1,q,\varepsilon) \in \gamma_{\Delta_2} \\
 & & & p \nearrow\!\swarrow q \quad \text{if} \quad (p,0,q,\varepsilon) \in \gamma_{\Delta_2} \\
p \circlearrowleft_{\mathbf{a}} q & \text{if} & (p,\varepsilon,q,\varepsilon) \in \gamma_{\Delta_{\mathbf{a}}} & p \searrow\!\nwarrow q \quad \text{if} \quad (p,\varepsilon,q,1) \in \gamma_{\Delta_2} \\
 & & & p \swarrow\!\nearrow q \quad \text{if} \quad (p,\varepsilon,q,0) \in \gamma_{\Delta_2} \\
p \uparrow q & \text{if} & (p,0,q,\varepsilon) \in \gamma_{\Delta_1} & p \curvearrowleft q \quad \text{if} \quad (p,1,q,0) \in \gamma_{\Delta_2} \\
p \downarrow q & \text{if} & (p,\varepsilon,q,0) \in \gamma_{\Delta_1} & p \curvearrowright q \quad \text{if} \quad (p,0,q,1) \in \gamma_{\Delta_2}
\end{array}
$$

$$
\begin{array}{lll}
p \nwarrow\!\swarrow q & \text{if } p \nwarrow\!\searrow q \text{ and not } p \nearrow\!\swarrow q & \\
p \nwarrow\!\!\uparrow q & \text{if } p \nearrow\!\swarrow q \text{ and not } p \nwarrow\!\searrow q & p \uparrow\!\!\uparrow q \text{ if } p \uparrow r \nearrow\!\swarrow r \nwarrow\!\searrow r \uparrow q \text{ for some } r \\
p \downarrow\!\!\swarrow q & \text{if } p \swarrow\!\nearrow q \text{ and not } p \searrow\!\nwarrow q & p \downarrow\!\!\downarrow q \text{ if } p \downarrow r \searrow\!\nwarrow r \swarrow\!\nearrow r \downarrow q \text{ for some } r \\
p \swarrow\!\downarrow q & \text{if } p \searrow\!\nwarrow q \text{ and not } p \swarrow\!\nearrow q &
\end{array}
$$

FIG. 5.1. *Graphical notation for $\gamma_{\Delta_0}, \gamma_{\Delta_{\mathbf{a}}}, \gamma_{\Delta_1}, \gamma_{\Delta_2}$.*

From now on, instead of the γ_{Δ_0}, $\gamma_{\Delta_{\mathbf{a}}}$, γ_{Δ_1}, and γ_{Δ_2} relations, we will be using the more graphical notation depicted in Figure 5.1. Note that the $p \curvearrowleft q$ notation may be somewhat misleading: we *start* with state p in port 1 and *end* in state q in port 0. The left state is chronologically before the right one, although the movement is in the left direction.

Due to the equivalences in Lemma 3.1, the relations $\gamma_{\Delta_1}, \gamma_{\Delta_2}$ satisfy properties such as the following, which we call swallowing rules:

$$
(p,0,q,1) \in \gamma_{\Delta_2},\ (q,\varepsilon,r,0) \in \gamma_{\Delta_1} \qquad \text{imply} \qquad (p,0,r,1) \in \gamma_{\Delta_2}.
$$

Using our graphical notation, this can be rewritten into the first property among the following ones:

$$
\begin{array}{lll}
p \curvearrowright q \downarrow r & \text{implies} & p \curvearrowright r, \\
p \uparrow q \curvearrowleft r & \text{implies} & p \curvearrowleft r, \\
p \curvearrowleft q \downarrow r & \text{implies} & p \curvearrowleft r, \\
p \uparrow q \curvearrowright r & \text{implies} & p \curvearrowright r.
\end{array} \tag{5.1}
$$

We will now illustrate how time symmetry can be used to show the second implication; the third and fourth are then obtained using space symmetry.

Let then p, q, and r be such that $p \uparrow q \curvearrowleft r$ holds. As the reader may recall, our tree-walking automaton is self-symmetric; i.e., there is an isomorphism $i : Q \to Q$, which maps the automaton onto its time-reversed variant. Let $i(p)$, $i(q)$, and $i(r)$ be the time-reversed counterparts of p, q, and r. By Fact 2.1, every run from p to q can be reversed to obtain a run from $i(q)$ to $i(p)$; likewise for q and r. If we reverse a run witnessing $p \uparrow q$, we obtain a run witnessing $i(q) \downarrow i(p)$. In the same way, there is a run witnessing $i(r) \curvearrowright i(q)$. In particular, we have

$$i(r) \curvearrowright i(q) \downarrow i(p).$$

Now we can apply the already shown first implication in (5.1), to obtain $i(r) \curvearrowright i(p)$. Since the isomorphism i is its own inverse, we obtain the desired $p \curvearrowleft r$.

In a similar way, using space symmetry, we can derive the last two statements of (5.1) from the first two. Later on, we will be using this type of reasoning a lot, omitting the details of the argumentation.

The following lemma shows that the $\circlearrowleft$ and $\circlearrowleft_{\mathbf{a}}$ notation is not misleading in suggesting a loop.

LEMMA 5.1. *The relations $\circlearrowleft$ and $\circlearrowleft_{\mathbf{a}}$ are transitive.*

Proof. We do the proof only for $\circlearrowleft$. We first claim that $p \circlearrowleft q$ holds if and only if either $p \to_\varepsilon q$ holds, or $p \downarrow p' \circlearrowleft^* q' \uparrow q$ holds for some states p', q' (where $\circlearrowleft^*$ is the transitive closure of $\circlearrowleft$).

The left-to-right implication of the claim is shown as follows. If $p \circlearrowleft q$ holds, then there exist p'', q'' such that

$$p \to_\varepsilon p'', \quad (p'', \varepsilon, q'', \varepsilon) \in \delta_{\Delta_0}, \quad \text{and} \quad q'' \to_\varepsilon q.$$

Let us analyze the run corresponding to $(p'', \varepsilon, q'', \varepsilon) \in \delta_{\Delta_0}$ in the pattern $\Delta_1[\Delta_0]$ (which is equivalent to Δ_0), the junction node being v. If this run does not visit v, then we have $p'' \to_\varepsilon q''$, and consequently $p \to_\varepsilon q$. Otherwise, there exist states p' and q' such that $(p'', \varepsilon, p', 0)$ and $(q', 0, q'', \varepsilon)$ belong to δ_{Δ_1}, and there is a path from configuration (p', v) to configuration (q', v) in the pattern $\Delta_1[\Delta_0]$. From this path we deduce $p' \circlearrowleft^* q'$. Hence $p \downarrow p' \circlearrowleft^* q' \uparrow q$.

The right-to-left implication of the claim is shown as follows. If $p \to_\varepsilon q$, we obviously have $p \circlearrowleft q$. Otherwise, assume that there exist states p', q' such that $p \downarrow p' \circlearrowleft^* q' \uparrow q$ holds. Then there are states p'', p''', q''', q'' such that

$$\begin{array}{lll} p \to_\varepsilon p'', & (p'', \varepsilon, p''', 0) \in \delta_{\Delta_1}, & p''' \to_\varepsilon p', \\ q' \to_\varepsilon q''', & (q''', 0, q'', \varepsilon) \in \delta_{\Delta_1}, & \text{and} \quad q'' \to_\varepsilon q. \end{array}$$

From $p''' \to_\varepsilon p' \circlearrowleft^* q' \to_\varepsilon q'''$ we obtain $p''' \circlearrowleft^* q'''$. Together with $(p'', \varepsilon, p''', 0) \in \delta_{\Delta_1}$ and $(q''', 0, q'', \varepsilon) \in \delta_{\Delta_1}$ we obtain a run from state p'' in the root to q'' in the root in $\Delta_1[\Delta_1[\Delta_0]]$ (which is equivalent to Δ_0). Hence $(p'', \varepsilon, q'', \varepsilon)$ belongs to δ_{Δ_0}. Together with $p \to_\varepsilon p''$ and $q'' \to_\varepsilon q$ we obtain $p \circlearrowleft q$. This concludes the proof of the claim.

Let us now show the transitivity of $\circlearrowleft$. Assume $p \circlearrowleft q \circlearrowleft r$. If either $p \to_\varepsilon q$ or $q \to_\varepsilon r$ holds, then we have $p \circlearrowleft r$ by definition of $\circlearrowleft$. Otherwise, according to the claim above, there exist states p', q', q'', r' such that

$$p \downarrow p' \circlearrowleft^* q' \uparrow q \downarrow q'' \circlearrowleft^* r' \uparrow r.$$

Since $q' \uparrow q \downarrow q''$ implies $q' \to_\varepsilon q''$, we obtain $p' \circlearrowleft^* r'$ by transitivity of $\circlearrowleft^*$. Using the other direction of the claim, we get $p \circlearrowleft r$.

For $\circlearrowleft_{\mathbf{a}}$, the proof is identical. The only property required from the pattern $\Delta_{\mathbf{a}}$ is that it is equivalent to $\Delta_1[\Delta_{\mathbf{a}}]$ (and hence to $\Delta_1[\Delta_1[\Delta_{\mathbf{a}}]]$); this fact was observed above while defining the pattern $\Delta_{\mathbf{a}}$. □

5.1. Depth-first search. In this section we define the key concept of depth-first search (DFS). The main result, Lemma 5.5, states that $p \nwarrow p$ can only be realized using a DFS (similarly for $\nearrow$, $\swarrow$, and $\searrow$).

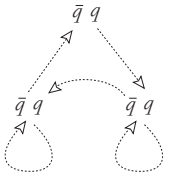


FIG. 5.2. *A right-to-left DFS.*

DEFINITION 6. *A state pair* $(q, \bar{q})$ *is a* right-to-left DFS *(see Figure* 5.2*) if*

$$q \searrow q, \qquad q \circlearrowleft \bar{q}, \qquad \bar{q} \curvearrowleft q, \quad \text{and} \quad \bar{q} \nearrow \bar{q}.$$

The pair is a left-to-right DFS *if*

$$q \swarrow q, \qquad q \circlearrowleft \bar{q}, \qquad \bar{q} \curvearrowright q, \quad \text{and} \quad \bar{q} \nwarrow \bar{q}.$$

Throughout the paper, we will try to keep the convention that if two states $\bar{q}$ and q appear simultaneously, then $\bar{q}$ is a state that is going up in the tree and q is a state that is going down in the tree.

We now illustrate the way a left-to-right DFS allows $\mathcal{A}$ to walk through a pattern expansion. Consider a pattern expansion Δ_t and a left-to-right DFS $(q, \bar{q})$. Using $q \swarrow q$ repeatedly, the automaton can go from q in $[\varepsilon]$ to q in the leftmost junction leaf (all this reasoning is done using Lemma 4.2). If v and w are successive leaves of t, then the automaton can go from $[\bar{q}, v]$ to $[q, w]$. This is done by using a sequence of steps $\bar{q} \nwarrow \bar{q}$, then doing a step of the form $\bar{q} \curvearrowright q$, and then doing a sequence of $q \swarrow q$ steps. Finally, using $\bar{q} \nwarrow \bar{q}$, the automaton can go from $\bar{q}$ in the rightmost junction leaf to $\bar{q}$ in $[\varepsilon]$. Moreover, if we plug Δ_0 into every leaf port of Δ_t, then $q \circlearrowleft \bar{q}$ together with the above observations can be used to obtain a left-to-right DFS of all the junction nodes in the pattern $\Delta_t[\Delta_0, \dots, \Delta_0]$.

In Lemma 5.4, we will also show the converse: without doing a DFS, the automaton cannot systematically visit all junction nodes in a pattern of the form $\Delta_t[\Delta_0, \dots, \Delta_0]$. First, however, we provide some preparatory results. In the following lemma, we say that a run *omits* a node if it never crosses it.

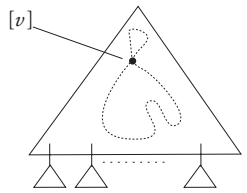
LEMMA 5.2. *Let t be a blank tree, with nodes v, w. Let ρ be a run in $\Delta_t[\Delta_0, \dots, \Delta_0]$ from configuration $[p, v]$ to configuration $[q, w]$ for some states p, q. Then the following hold:*

1. *If $v = w$, then $p \circlearrowleft q$.*
2. *Assume $v = w$, and let u be a node strictly below v. If ρ omits $[u]$, then $p \to_\varepsilon q$.*
3. *Assume v is strictly above w, and let u be a node strictly below w. If ρ omits $[u]$, then $p \downarrow q$.*

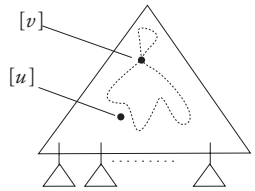
4. *Assume v is strictly above w, let u be a node strictly below w, and let u' be a node to the right of w and strictly below v. If ρ omits $[u]$ and $[u']$, then $p \swarrow q$.*
5. *Assume v is to the left of w, and let u, u' be nodes strictly below v, w, respectively. If ρ omits $[u]$ and $[u']$, then $p \curvearrowright q$.*

Proof. We would like to clarify that $\Delta_t[\Delta_0, \ldots, \Delta_0]$ is treated here as a pattern of arity zero and not a tree. Therefore, by definition of runs in patterns, ρ never visits the root port. Below, we successively treat the five cases. Each explanation is followed by a drawing illustrating the situation.

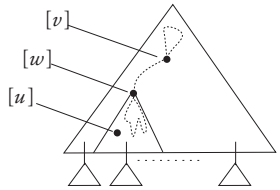
1. By transitivity of $\circlearrowleft$ (Lemma 5.1), it suffices to consider runs ρ where v is visited only in the first and last configurations. If the run never goes below $[v]$, then, by putting a port in the node $[v]$, the run can be replicated in a pattern equivalent to Δ_1, yielding $(p, 0, q, 0) \in \delta_{\Delta_1}$ and hence $p \to_\varepsilon q$. Otherwise, the run visits only nodes below $[v]$ and is therefore a root-to-root run in a pattern equivalent to Δ_0.



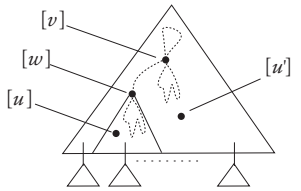
2. Again, by transitivity of the $\to_\varepsilon$ relation, it suffices to consider the case where v is visited only in the first and last configurations. There are two cases. Either the run never goes below $[v]$ and we can reuse the argument of item 1, or the run visits only nodes below $[v]$ but not the node $[u]$. Therefore, if we put a port into node $[u]$, ρ becomes a root-to-root loop in a pattern equivalent to Δ_1, witnessing $(p, \varepsilon, q, \varepsilon) \in \delta_{\Delta_1}$; consequently $p \to_\varepsilon q$.



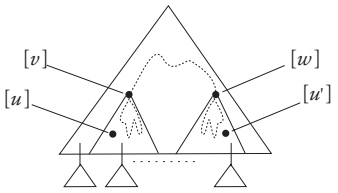
3. First we show that, without loss of generality, we may assume that ρ visits $[v], [w]$ only in the first and last configurations. Indeed, if we take the longest prefix of ρ that is a loop in $[v]$, this prefix satisfies the assumptions of item 2, and can therefore be replaced by an inner loop $\to_\varepsilon$. The same can then be done for the suffix, removing additional visits to $[w]$. Once ρ is assumed to visit $[v]$ and $[w]$ only once, it is easily seen to witness $(p, \varepsilon, q, 0) \in \delta_{\Delta_1}$ and hence $p \downarrow q$: if we put the root port in $[v]$ and a leaf port in $[w]$, we get a pattern equivalent to Δ_1.



4. As in the previous point, we may assume that ρ visits $[v], [w]$ only in the first and last configurations (we use the assumption of the node $[u]$ being omitted and below both $[v]$ and $[w]$). If we put one leaf port (the left port) in $[w]$, one leaf port (the right port) in $[u']$, and the root port in node $[v]$, we get a pattern equivalent to Δ_2. Then the run ρ witnesses $(p, \varepsilon, q, 0) \in \delta_{\Delta_2}$, and hence $p \swarrow q$.



5. Again, we assume that ρ visits $[v], [w]$ only in the first and last configurations (this time, we need to use both $[u]$ and $[u']$). We put the left port in node $[v]$ and the right port in node $[w]$ (the root port stays unchanged).



□

We now need a simple combinatorial result concerning labeling of trees.

LEMMA 5.3. *Let Σ be an alphabet, and consider a balanced Σ-tree t of depth at least $|\Sigma| + 1$. There exist three nodes w, w_0, w_1 with the same label such that w_0 is to the left of w_1 and w is above both w_0 and w_1.*

Proof. Induction on $|\Sigma|$. The base case of $|\Sigma| = 1$ is obvious. Otherwise let a be the root label of t. If both the subtrees of nodes 0 and 1 contain a's, we are done. Otherwise one of these is a $\Sigma \setminus \{a\}$-tree, and the induction hypothesis can be applied to it. □

The following lemma is the first important characterization of runs on patterns. It says that there are only two types of root-to-root runs over the pattern Δ_0: either a run that does not visit anything and only does an inner loop $\rightarrow_\varepsilon$, or a systematic DFS traversal.

LEMMA 5.4. *Let $q, \bar{q}$ be such that $q \circlearrowleft \bar{q}$. Then either $q \rightarrow_\varepsilon \bar{q}$ or there is a (left-to-right or right-to-left) DFS $(r, \bar{r})$ such that $q \downarrow r$ and $\bar{r} \uparrow \bar{q}$.*

Proof. Let t be the blank balanced tree of depth $|Q|^2 + 2$. Let Γ be the pattern obtained from Δ_t by substituting Δ_0 for all leaf ports; i.e., $\Gamma = \Delta_t[\Delta_0, \ldots, \Delta_0]$. By definition of expansions, the pattern Γ can be rewritten as $\Delta_1[\Gamma']$ in which, by Lemma 3.1, the pattern Γ' is equivalent to Δ_0.

By $q \circlearrowleft \bar{q}$, there exists a run in Γ from $[q, \varepsilon]$ to $[\bar{q}, \varepsilon]$. First we show that we can furthermore enforce the following property (*) of ρ: every subrun starting and ending at the same junction node $[v]$ for v a nonleaf node of t visits only junction nodes below $[v]$. This is proved by induction on the number of junction nodes in the run. Indeed, take a minimal loop in some junction node $[v]$ that visits junction nodes not below $[v]$. By minimality, the loop never visits nodes strictly below $[v]$. Hence, by Lemma 5.2 (item 2) and Lemma 4.1, this loop can be replaced by another, with the

same initial and final configurations, which does not visit any junction node other than $[v]$.

Then let ρ be a run from $[q, \varepsilon]$ to $[\bar{q}, \varepsilon]$ that satisfies property (*).

If ρ does not visit some junction node, then by Lemma 5.2, $q \rightarrow_\varepsilon \bar{q}$ holds.

Otherwise, given a node v of t, let $\mathit{first}(v)$ be the state in which the junction node $[v]$ is visited for the first time in the run ρ. Similarly we define $\mathit{last}(v)$. By Lemma 5.3, there are three nonleaf nodes w, w_0, w_1 of t with the same values of *first* and *last*, and such that w is above both w_0 and w_1, and w_0 is to the left of w_1 (see Figure 5.3).

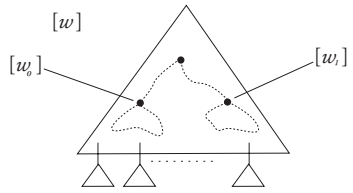


FIG. 5.3. *The nodes* $[w]$, $[w_0]$, *and* $[w_1]$.

First consider the case where $[w_0]$ is visited before $[w_1]$. Let r be $\mathit{first}(w)$ and $\bar{r}$ be $\mathit{last}(w)$. By Lemma 5.2, $r \circlearrowleft \bar{r}$. From property (*) we derive that the run cannot visit $[w_0]$, then $[w_1]$, and then again $[w_0]$. Hence we can apply Lemma 5.2 to configurations $[\bar{r}, w_0]$ and $[r, w_1]$ and obtain $\bar{r} \curvearrowright r$. Also, from the definition of *first* and *last* and Lemma 5.2, we obtain $r \swarrow r$ and $\bar{r} \nwarrow \bar{r}$. Overall $(r, \bar{r})$ is a left-to-right DFS. Furthermore, by Lemma 5.2 (item 2 or 3), either $q \rightarrow_\varepsilon r$ or $q \downarrow r$. In the first case, in combination with $r \downarrow r$, we also obtain $q \downarrow r$. We similarly have $\bar{r} \uparrow \bar{q}$.

If $[w_1]$ is visited before $[w_0]$, a similar argument gives a right-to-left DFS. □

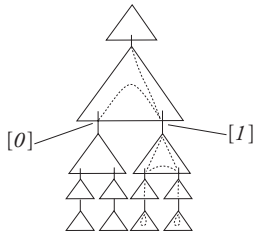
We now proceed to another one of our DFS characterizations: the relation $\leftrightarrows$ can be realized only by doing a DFS.

LEMMA 5.5. *If* $\bar{q} \leftrightarrows \bar{q}$, *then* $(q, \bar{q})$ *is a left-to-right DFS for some state* q.

Proof. Unraveling the definition of $\bar{q} \leftrightarrows \bar{q}$, we have that $\bar{q} \nwarrow \bar{q}$ holds, but $\bar{q} \nearrow \bar{q}$ does not. Let t be the blank balanced binary tree of depth 3 (i.e., with four leaves). Let Γ be the pattern $\Delta_t[\Delta_0, \Delta_0, \Delta_0, \Delta_0]$. Since $\bar{q} \nwarrow \bar{q}$ implies $\bar{q} \uparrow \bar{q}$, and the pattern

$$\Delta_2[*, \Delta_2[\Delta_1[\Delta_0], \Delta_1[\Delta_0]]]$$

is equivalent to Δ_1, there is a run in this pattern from $[\bar{q}, 0]$ to $[\bar{q}, \varepsilon]$:



This run has to visit the junction node [1] since otherwise Lemma 5.2 would give $\bar{q} \nearrow \bar{q}$, a contradiction. Let p be the first state assumed by this run at the junction node [1], and let $\bar{p}$ be the last.

By Lemma 5.2, we have $\bar{q} \curvearrowright p$, $p \circlearrowleft \bar{p}$, and $\bar{p} \nwarrow \bar{q}$. We cannot have $p \rightarrow_\varepsilon \bar{p}$, since we would otherwise get $\bar{q} \nearrow \bar{q}$. By Lemma 5.4, we obtain states $r, \bar{r}$ such that $(r, \bar{r})$

is a DFS and $p \downarrow r$, $\bar{r} \uparrow \bar{p}$. By swallowing, we obtain $\bar{q} \curvearrowright r$, $\bar{r} \nwarrow \bar{q}$ (and we can forget about states p and $\bar{p}$).

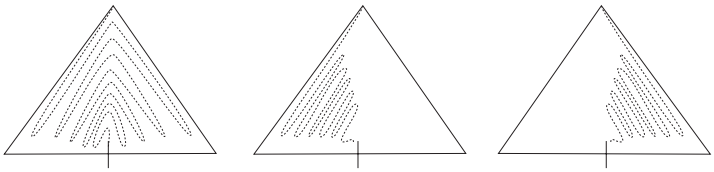
Two cases have to be considered depending on the orientation of the DFS $(r, \bar{r})$.

- Assume first that $(r, \bar{r})$ is a left-to-right DFS. We have $r \downarrow r$ and $\bar{r} \uparrow \bar{q}$. Thus, $r \circlearrowleft \bar{q}$ follows from $r \downarrow r \circlearrowleft \bar{r} \uparrow \bar{q}$. We obtain that $(r, \bar{q})$ is a left-to-right DFS as well.
- Otherwise, $(r, \bar{r})$ is a right-to-left DFS. By $r \searrow r \circlearrowleft \bar{r} \nwarrow \bar{q}$, we get $r \to_\varepsilon \bar{q}$. But then $\bar{q} \curvearrowright r \to_\varepsilon \bar{q} \nwarrow \bar{q}$ gives $\bar{q} \nearrow \bar{q}$, a contradiction. ∎

5.2. Subtree omission. This section is devoted to showing the following proposition.

PROPOSITION 5.6. *For all p and q, $p \uparrow q$ if and only if $p \nwarrow q$ or $p \nearrow q$.*

The right-to-left implication is obvious; the remainder of this section is devoted to showing the left-to-right implication. The intuitive idea is illustrated in the picture below: whenever there is a run as on the left, there is also an equivalent run as in the middle or on the right.



We first show the following intermediate result.

LEMMA 5.7. *If $\bar{q} \uparrow \bar{q}$, then*

- *$\bar{q} \nearrow \bar{q}$ or $\bar{q} \nwarrow \bar{q}$, or*
- *there is a right-to-left DFS $(r, \bar{r})$ such that $\bar{q} \curvearrowright r$ and $\bar{r} \uparrow \bar{q}$.*

Proof. As in the proof of Lemma 5.5, we obtain that $\bar{q} \nearrow \bar{q}$ holds or there are two states $r, \bar{r}$ such that $(r, \bar{r})$ is a DFS and both $\bar{q} \curvearrowright r$ and $\bar{r} \uparrow \bar{q}$ hold. The first case as well as the second when the DFS is right-to-left are conclusions of the lemma.

In the remaining case, $(r, \bar{r})$ is a left-to-right DFS. But then by $\bar{q} \curvearrowright r \circlearrowleft \bar{r} \nwarrow \bar{r}$ we obtain $\bar{q} \uparrow \bar{r}$. Combining this with $\bar{r} \nwarrow \bar{r} \uparrow \bar{q}$, we obtain the desired $\bar{q} \nwarrow \bar{q}$. ∎

A variant symmetric to the one above can be obtained, where $(r, \bar{r})$ is a left-to-right DFS and $\bar{q} \curvearrowleft r$ and $\bar{r} \uparrow \bar{q}$ hold.

LEMMA 5.8. *If $p \uparrow q$, then $p \uparrow r \uparrow r \uparrow q$ for some state r.*

Proof. This results from a pumping argument. Since $\Delta_1[\Delta_1]$ is equivalent to Δ_1, we can expand the Δ_1 pattern into the composition of n times itself for any n. This means that there are states $r_1, \ldots, r_n$ such that $p = r_1 \uparrow \cdots \uparrow r_n = q$.

If n is large enough, some state r is repeated, and the result follows by transitivity of $\uparrow$. ∎

We will now prove Proposition 5.6. Since we have the implication

$$p \uparrow r \nwarrow r \uparrow q \qquad \text{implies} \qquad p \nwarrow q$$

and its symmetric counterpart for $\nearrow$, Lemma 5.8 allows us to restrict our attention to the case where $p = q$. That is, we need to show that $p \uparrow p$ implies $p \nwarrow p$ or $p \nearrow p$.

If in either Lemma 5.7 or its symmetric variant the first case holds, we are done. Otherwise there are states q, $\bar{q}$, r, and $\bar{r}$ such that

$$\begin{array}{llllll} p \curvearrowright q, & q \searrow q, & q \circlearrowleft \bar{q}, & \bar{q} \curvearrowleft q, & \bar{q} \nearrow \bar{q}, & \bar{q} \uparrow p, \\ p \curvearrowleft r, & r \swarrow r, & r \circlearrowleft \bar{r}, & \bar{r} \curvearrowright r, & \bar{r} \nwarrow \bar{r}, & \bar{r} \uparrow p. \end{array}$$

By $p \curvearrowleft r \circlearrowleft \bar{r} \curvearrowright r$ we get $p \to_\varepsilon r$. Together with $\bar{q} \uparrow p$ and $r \downarrow r$, this gives $\bar{q} \to_\varepsilon r$. Together with $q \circlearrowleft \bar{q}$ and $r \circlearrowleft \bar{r}$, this yields $q \circlearrowleft \bar{r}$ by Lemma 5.1. Then $p \curvearrowright q \circlearrowleft \bar{r} \nwarrow \bar{r}$ shows $p \uparrow \bar{r}$. Finally, we combine this with $\bar{r} \nwarrow \bar{r} \uparrow p$ and obtain $p \nwarrow p$.

5.3. A characterization of moves over Δ_1. In this section, we present a classification of the possible ways the automaton can go in Δ_1 from the leaf port to the root port. This is the main result of section 5. Before we proceed with Proposition 5.10, we show a certain "denseness" property of the relations $\upharpoonleft$, $\upharpoonleft$, and $\updownarrow$.

LEMMA 5.9. *For any states p, q and $R = \upharpoonleft, \upharpoonleft, \updownarrow$,*

$$p \; R \; q \qquad \textit{implies} \qquad p \; R \; r \; R \; r \; R \; q \quad \textit{for some state } r.$$

Proof. The case of $\updownarrow$ follows straight from the definition. We do only $\upharpoonleft$; the other case is done symmetrically. If $p \upharpoonleft q$, then $p \nearrow q$, and thus also $p \uparrow q$. By Lemma 5.8, there must be some state r such that $p \uparrow r \uparrow r \uparrow q$. By Proposition 5.6, we must have at least one of $r \nearrow r$, $r \nwarrow r$. But we cannot have $r \nwarrow r$, since this would yield $p \nwarrow q$ and contradict $p \upharpoonleft q$; hence $r \upharpoonleft r$. For similar reasons $p \upharpoonleft r$ and $r \upharpoonleft q$ must also hold. ∎

Note that the converse implication may fail. This is because $p \upharpoonleft q$ requires $p \nwarrow q$ to fail, while there may be some other state s satisfying $p \uparrow s \nwarrow s \uparrow q$.

PROPOSITION 5.10. *If $p \uparrow q$, then*

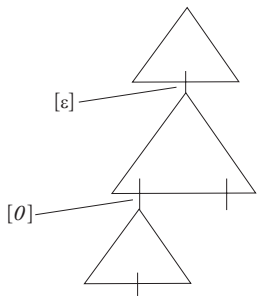
1. *$p \updownarrow q$, or*
2. *$p \upharpoonright q$, or*
3. *$p \upharpoonleft q$, or*
4. *for some states r_1, r_2*
   (a) *$p \uparrow r_1 \upharpoonright r_1 \uparrow q$ and $p \uparrow r_2 \upharpoonleft r_2 \uparrow q$, or*
   (b) *$p \uparrow r_1 \upharpoonright r_1 \nearrow r_2 \upharpoonright r_2 \uparrow q$, or*
   (c) *$p \uparrow r_1 \upharpoonleft r_1 \nwarrow r_2 \upharpoonleft r_2 \uparrow q$.*

Proof. If neither case 2 nor 3 holds, then by Proposition 5.6 we must have both $p \nearrow q$ and $p \nwarrow q$. Let $R^\uparrow(p,q)$ be the set of states r such that $p \uparrow r \uparrow r \uparrow q$ holds; this set is nonempty by Lemma 5.8. Let $R^\nearrow(p,q) \subseteq R^\uparrow(p,q)$ be the set of those states r in $R^\uparrow(p,q)$ such that $r \nearrow r$. Similarly we define $R^\nwarrow(p,q)$. Note that by Proposition 5.6,

$$R^\uparrow(p,q) = R^\nearrow(p,q) \cup R^\nwarrow(p,q).$$

Now a case analysis proves the lemma:

- If $R^\nearrow(p,q) \cap R^\nwarrow(p,q)$ is nonempty, then item 1 holds.
- $R^\nearrow(p,q)$ is empty. By $p \nearrow q$, in the pattern $\Delta_1[\Delta_2[\Delta_1, *]]$, there is a run from state p' in port 0 to state q' in port ε that does not visit port 1, where $p \to_\varepsilon p'$ and $q' \to_\varepsilon q$. This run uses two states q_1, q_2 at the intermediate junction nodes $[\varepsilon]$ and $[0]$. These states satisfy $p \uparrow q_1 \nearrow q_2 \uparrow q$:



The set $R^{\nearrow}(p, q_1)$ is empty, since it is included in $R^{\nearrow}(p, q)$. Furthermore, since $p \uparrow q_1$ holds, the set $R^{\uparrow}(p, q_1)$ must be nonempty, and consequently $R^{\nwarrow}(p, q_1)$ is nonempty by Proposition 5.6. Therefore we can choose r_1 in $R^{\nwarrow}(p, q_1)$ which is not in $R^{\nearrow}(p, q_1)$. This means

$$p \uparrow r_1 \downarrow r_1 \uparrow q_1.$$

Similarly, there is some state r_2 such that

$$q_2 \uparrow r_2 \downarrow r_2 \uparrow q.$$

Since $q_1 \nearrow q_2$, we have $r_1 \nearrow r_2$, and therefore item (b) holds.

- $R^{\nwarrow}(p, q)$ is empty. By reasoning as above, we have item (c).
- Finally, if $R^{\nearrow}(p, q) \cap R^{\nwarrow}(p, q)$ is empty, yet both $R^{\nearrow}(p, q)$ and $R^{\nwarrow}(p, q)$ are nonempty, then clearly item (a) holds. □

The point of characterizing $\uparrow$ and $\downarrow$ is that these are the most basic types of moves the automaton can make in a pattern expansion. Indeed, by Lemma 4.2, in order to move from one junction node to another, the automaton needs to traverse the Δ_2 pattern. Since the pattern Δ_2 can be seen as having Δ_1 plugged into each of its ports, each such traversal must employ one of the moves $\uparrow$ or $\downarrow$. But then we can use Proposition 5.10 in order to uncover other possible moves of the automaton.

When put together, Proposition 5.10 and Lemmas 5.9 and 5.5 give us some idea of how a tree-walking automaton can move upward within a pattern expansion: it may get completely lost (by allowing a move from a node to any node above it, case 1 in Proposition 5.10), allow a DFS in some fixed direction and nothing else (cases 2 and 3), or, finally, do some DFSs coupled with moves in opposing directions (case 4).

6. Moves. In the previous section, we analyzed the way an automaton can move through single instances of the basic patterns Δ_0, Δ_1, and Δ_2. In this section, we consider runs through larger objects built as compositions of Δ_1 and Δ_2 patterns, i.e., pattern expansions. We are especially interested in the way the tree-walking automaton can go from one junction leaf of such an expansion to another. Recall Lemma 4.1, which states that any loop in a junction node can be replaced by the $\rightarrow_\varepsilon$ relation and hence swallowed by the γ relations. This means that any run between two junction nodes in a pattern expansion can be assumed to be a nonlooping sequence of steps consistent with the γ relation. In a tree, a nonlooping path is the shortest possible path.

Note that all patterns considered in this section and the previous ones use only the blank symbol. The **a** label will be introduced only in the final section, section 7. From this perspective, the sections leading up to section 7 can be seen as an analysis of runs that never see the **a** label.

6.1. Pattern paths and moves. Before proceeding with a classification of possible moves, we introduce a more convenient syntax for describing runs between junction nodes within a pattern expansion. Essentially, by Lemma 4.2, such a run can be decomposed as a sequence of moves taken from γ_{Δ_2}. Moreover, by closure properties of γ_{Δ_2}, the runs can be assumed to have a certain normal form.

A *pattern path* (path for short) is a word over the alphabet $\{\varepsilon, 0, 1\} \times \{\varepsilon, 0, 1\}$. A pattern path can be used to go from one junction node to another in a pattern expansion in the following manner. An empty pattern path can stay in the same junction node, while the pattern path $\pi \cdot (a, b)$ can go from $[v]$ to $[u \cdot b]$ if its prefix π can go from $[v]$ to $[u \cdot a]$. We write $v \rightarrow_\pi w$ when π can go from $[v]$ to $[w]$.

The $\to_\pi$ relation can also be annotated with states of the automaton. Given states p, q, and a pattern path $\pi = (a_1, b_1) \cdots (a_n, b_n)$, we write $p \to_\pi q$ if there are states $p = r_1, \ldots, r_{n+1} = q$ such that for all $i = 1, \ldots, n$ the tuple (r_i, a_i, r_{i+1}, b_i) belongs to γ_{Δ_2}. In the special case of $\pi = \varepsilon$, we require $p \to_\varepsilon q$. Given two states p, q and two nodes v, w, we write $[p, v] \to_\pi [q, w]$ if both $p \to_\pi q$ and $v \to_\pi w$ hold.

A pattern path is called *normalized* if it is the shortest path between two junction nodes. For having more understandable normalized paths, we use the following abbreviations: $\swarrow = (\varepsilon, 0)$, $\searrow = (\varepsilon, 1)$, $\nearrow = (0, \varepsilon)$, $\nwarrow = (1, \varepsilon)$, $\curvearrowright = (0, 1)$, and $\curvearrowleft = (1, 0)$. Let us define the following languages:

$$\begin{aligned} \mathrm{Up} &= (\nearrow + \nwarrow)^+, & \mathrm{Down} &= (\swarrow + \searrow)^+, \\ \mathrm{Left} &= (\mathrm{Up} + \varepsilon) \curvearrowleft (\mathrm{Down} + \varepsilon), & \mathrm{Right} &= (\mathrm{Up} + \varepsilon) \curvearrowright (\mathrm{Down} + \varepsilon), \\ \mathrm{Side} &= \varepsilon + \mathrm{Left} + \mathrm{Right}. \end{aligned}$$

The sets Up, Down, Left, Right, Side are called, respectively, the sets of *upward*, *downward*, *left*, *right*, and *sideways* paths. A pattern path is normalized if and only if it belongs to Up + Down + Side. Given nodes of a tree v and w, $\pi(v, w)$ denotes the unique normalized path such that $v \to_{\pi(v,w)} w$. As expected, for nodes v and w, $\pi(v, w) \in \mathrm{Up}$ if and only if w is strictly above v; $\pi(v, w) \in \mathrm{Down}$ if and only if w is strictly below v; $\pi(v, w) \in \mathrm{Left}$ if and only if w is to the left of v; and $\pi(v, w) \in \mathrm{Right}$ if and only if w is to the right of v. A set of normalized pattern paths is called a *move*, and we write vMw if $\pi(v, w) \in M$.

We will now show some results about the possible paths that the automaton can use when going from one node to another; these were mentioned after Lemma 4.2. We begin with the following lemma, which shows how paths correspond to runs of the automaton, at least as far as junction nodes are concerned.

LEMMA 6.1. *The following are equivalent for nodes v, w in a blank tree t.*

1. *There is a run in Δ_t from $[p, v]$ to $[q, w]$.*
2. *$[p, v] \to_{\pi(v,w)} [q, w]$ holds.*
3. *There is a run in Δ_t from $[p, v]$ to $[q, w]$ which visits only junction nodes $[u]$ such that $v \to_\pi u$ for some prefix π of $\pi(v, w)$.*

Proof. This is a generalization of Lemma 4.2, and the same proof works: any loop appearing in a run can be contracted using the $\to_\varepsilon$ relation. □

Since the normalized path connecting v and w does not depend on the tree t but only on the nodes v, w, we obtain the following corollary.

COROLLARY 6.2. *Let v, w be nodes of a blank tree t. Whether or not there is a run from $[p, v]$ to $[q, w]$ in Δ_t depends only on $\pi(v, w)$ and not on t.*

The above corollary justifies the notation $[p, v] \to [q, w]$, where no particular path or tree is mentioned; it is equivalent to $p \to_{\pi(v,w)} q$. We will use this notation often in what follows.

DEFINITION 7. *For states p and q, we define $U(p, q)$ to be the set of upward paths π such that $p \to_\pi q$. Similarly, we define $D(p, q)$, $L(p, q)$, $R(p, q)$, and $S(p, q)$ for downward, left, right, and sideways paths, respectively.*

In particular, a direct consequence of this definition is that for two distinct nodes v and w and states p and q, $[p, v] \to [q, w]$ if and only if

$$\pi(v, w) \in U(p, q) \cup D(p, q) \cup S(p, q).$$

The following lemma will be used several times; it transfers some properties of γ_{Δ_2} to equalities on the sets U, D, L, R. Its meaning is natural, but the statement as well as the proof are slightly clouded by the case of the normalized path ε.

LEMMA 6.3. *The move $R(p,q)$ is the union of $(U(p,p')+\varepsilon) \curvearrowright (D(q',q)+\varepsilon)$ for states p',q' satisfying $p \uparrow p' \curvearrowright q' \downarrow q$. A similar statement holds for L.*

Proof. We do only the case of R. We begin with the right-to-left inclusion. Let p',q' be states such that $p \uparrow p' \curvearrowright q' \downarrow q$, and let

$$\pi \in (U(p,p')+\varepsilon) \curvearrowright (D(q',q)+\varepsilon).$$

We can write π as $\pi_1 \curvearrowright \pi_2$ with $\pi_1 \in U(p,p')+\varepsilon$ and $\pi_2 \in D(q',q)+\varepsilon$.

The first case is when both π_1, π_2 are empty and therefore π is $\curvearrowright$. By assumption, we have $p \uparrow p' \curvearrowright q' \downarrow q$ and hence $p \curvearrowright q$, by swallowing. Consequently $p \to_\pi q$ and $\pi \in R(p,q)$. If neither of π_1, π_2 is empty, then $\pi_1 \in U(p,p')$ and $\pi_2 \in D(q',q)$, which gives the desired result. The remaining cases where only one of π_1 or π_2 is empty are treated by combining the two first cases.

We now treat the left-to-right inclusion. Let π be in $R(p,q)$. By definition of Right, π can be written as $\pi_1 \curvearrowright \pi_2$ with $\pi_1 \in \mathrm{Up}+\varepsilon$ and $\pi_2 \in \mathrm{Down}+\varepsilon$. As above, we first consider the case when π_1, π_2 are both empty. In this case, we have $p \curvearrowright q$. But then, by looking at the path from port 0 to port 1 in the pattern $\Delta_2[\Delta_1, \Delta_1]$ which is equivalent to Δ_2, we can find states p',q' such that $p \uparrow p' \curvearrowright q' \downarrow q$, which completes the proof.

Assume now that both π_1, π_2 are nonempty. Let p',q' be the states such that $p \to_{\pi_1} p' \curvearrowright q' \to_{\pi_2} q$. We need to show that $p \uparrow p'$ and $q' \downarrow q$. But this follows from transitivity of $\downarrow, \uparrow$ and the inclusions $\swarrow, \searrow \subseteq \downarrow$ and $\nwarrow, \nearrow \subseteq \uparrow$. The remaining cases where only one of π_1 or π_2 is empty are treated by combining the two first cases. ∎

Furthermore, there exists a strong link between the set of upward moves (and by time symmetry, downward moves) and the behaviors of the automaton exhibited in the previous section; this is the subject of the next lemma.

LEMMA 6.4. *If $p \updownarrow q$, then $U(p,q) = \mathrm{Up}$. The analogous results hold for $\downdownarrows$, $\uparrow\!\!\!\!\nearrow$, $\uparrow\!\!\!\!\swarrow$, $\downarrow\!\!\!\!\searrow$, and $\downdownarrows$, the corresponding moves being, respectively,* Down, $\nwarrow^+$, $\nearrow^+$, $\swarrow^+$, *and* $\searrow^+$.

Proof. We treat the case $\uparrow\!\!\!\!\nearrow$. If $p \uparrow\!\!\!\!\nearrow q$, then by Lemma 5.9, there is a state r such that $p \uparrow\!\!\!\!\nearrow r \uparrow\!\!\!\!\nearrow r \uparrow\!\!\!\!\nearrow q$. This shows $\nwarrow^+ \subseteq U(p,q)$. The opposite inclusion must also hold, since otherwise we would get $p \nearrow q$. ∎

The next lemma gives other required facts about $U(p,q)$ and $S(p,q)$.

LEMMA 6.5. *If $p \uparrow q$, then $\nwarrow^+ \subseteq U(p,q)$ or $\nearrow^+ \subseteq U(p,q)$. If $p \nearrow q$, then $\nwarrow^* \nearrow \subseteq U(p,q)$ or $\nearrow^+ \subseteq U(p,q)$. If $p \uparrow\!\!\!\!\swarrow p' \curvearrowright q$, then $\varepsilon \in S(p,q)$.*

Proof. First statement: for some r, $p \uparrow r \nwarrow r \uparrow q$ or $p \uparrow r \nearrow r \uparrow q$ by Lemma 5.8 and Proposition 5.6. Second statement: for some p', $p \uparrow p' \nearrow q$ must hold; then use the first statement. Third statement: by Lemmas 5.9 and 5.5, there exist $r, \bar{r}$ such that $p \uparrow \bar{r} \curvearrowleft r \downarrow r \circlearrowleft \bar{r} \uparrow p'$. Hence (by swallowing) $p \curvearrowleft r \downarrow r \circlearrowleft \bar{r} \uparrow p' \curvearrowright q$, and so $p \to_\varepsilon q$, i.e., $\varepsilon \in S(p,q)$. ∎

As hinted by Proposition 5.10 and Lemmas 6.3, 6.4, and 6.5, there are not so many ways that a sideways move can be done. The following eleven moves will play a special role below.

DEFINITION 8. *An* elementary move *is any one of the eleven moves in Figure* 6.1.

6.2. Move offsets. This section is devoted to moves that depend only on the number of junction leaves between the source and destination, i.e., moves that do not really depend on the structure of the tree.

We number the leaves of a tree t from left to right, starting from 0. Formally, given a blank tree t and a leaf v of t, we denote by $\#_t(v)$ the number of leaves in the tree t that are lexicographically before v. For v and w leaves of t, we denote

$$
\begin{array}{ll}
\mathit{Stay} = \varepsilon & \\
\rightleftarrows = \nwarrow^* \curvearrowright \swarrow^* & \leftrightarrows = \nearrow^* \curvearrowleft \searrow^* \\
\rightarrow\!\!\!\!\!\!\rightarrow = (\nearrow + \nwarrow)^* \curvearrowright \swarrow^* & \leftarrow\!\!\!\!\!\!\leftarrow = (\nearrow + \nwarrow)^* \curvearrowleft \searrow^* \\
\rightleftarrows = \nwarrow^* \curvearrowright (\swarrow + \searrow)^* & \leftrightarrows = \nearrow^* \curvearrowleft (\swarrow + \searrow)^* \\
\rightarrow\!\!\!\!\!\!\rightarrow = \varepsilon + (\nearrow + \nwarrow)^* \curvearrowright \searrow^* & \leftarrow\!\!\!\!\!\!\leftarrow = \varepsilon + (\nearrow + \nwarrow)^* \curvearrowleft \swarrow^* \\
\rightleftarrows = \varepsilon + \nearrow^* \curvearrowright (\swarrow + \searrow)^* & \leftrightarrows = \varepsilon + \nwarrow^* \curvearrowleft (\swarrow + \searrow)^*
\end{array}
$$

FIG. 6.1. *Elementary moves.*

by $\#_t(v, w)$ the offset from v to w within t, i.e., the difference $\#_t(w) - \#_t(v)$. This number is positive when v is to the left of w. If v or w is not a leaf of t, then $\#_t(v, w)$ is not defined.

DEFINITION 9. *A* move offset *of two states p, q is an integer i such that for every tree t and leaves v and w of t, $\#_t(v, w) = i$ implies $[p, v] \to [q, w]$. We write* $\mathrm{moff}(p, q)$ *for the set of move offsets of p, q. We say that p, q admit a* shift *if* $\mathrm{moff}(p, q)$ *contains two successive integers from $\{-2, -1, 0, 1, 2\}$.*

The next lemma shows how move offsets can be described in terms of paths.

LEMMA 6.6. *For every pair of states p, q,*

$$
\begin{array}{rll}
0 \in \mathrm{moff}(p, q) & \textit{iff} & \varepsilon \in S(p, q), \\
1 \in \mathrm{moff}(p, q) & \textit{iff} & \nwarrow^* \curvearrowright \swarrow^* \subseteq S(p, q), \\
2 \in \mathrm{moff}(p, q) & \textit{iff} & \nwarrow^* \nearrow \nwarrow^* \curvearrowright \swarrow^* + \nwarrow^* \curvearrowright \swarrow^* \searrow \swarrow^* \subseteq S(p, q), \\
-1 \in \mathrm{moff}(p, q) & \textit{iff} & \nearrow^* \curvearrowleft \searrow^* \subseteq S(p, q), \\
-2 \in \mathrm{moff}(p, q) & \textit{iff} & \nearrow^* \nwarrow \nearrow^* \curvearrowleft \searrow^* + \nearrow^* \curvearrowleft \searrow^* \swarrow \searrow^* \subseteq S(p, q).
\end{array}
$$

Proof. The case of 0 follows straight from the definition: if $\#_t(v, w) = 0$, then $v = w$ and therefore $\pi(v, w) = \varepsilon$ (and vice versa). The remaining cases follow by listing the paths that can connect nodes separated by $1, 2, -1, -2$ leaves, respectively. ∎

In particular, directly from the definition of elementary moves, we deduce the following corollary.

COROLLARY 6.7. *Every elementary move has an offset among $-1, 0, 1$.*

A typical example of a move offset of 1 is the DFS.

LEMMA 6.8. *If $(p, \bar{p})$ is a left-to-right DFS, then 1 is a move offset of $\bar{p}, p$. If $(p, \bar{p})$ is a right-to-left DFS, then -1 is a move offset of $\bar{p}, p$.*

Proof. If $(p, \bar{p})$ is a left-to-right DFS, then $\bar{p} \nwarrow \bar{p} \curvearrowright p \swarrow p$ holds. Thus the move $S(\bar{p}, p)$ contains $\nwarrow^* \curvearrowright \swarrow^*$, and the move offset 1 follows by Lemma 6.6. The right-to-left case is the same. ∎

The following lemma gathers a number of sufficient conditions for move offsets.

LEMMA 6.9. *For all states $\bar{p}$, $\bar{r}$, and q the following hold:*

1. *If $\bar{p} \leftarrow\!\!\!\!\!\!\leftarrow \bar{p} \curvearrowright q$, then 0 is a move offset of $\bar{p}, q$.*
2. *If $\bar{p} \rightleftarrows \bar{p} \curvearrowright q$, then 1 is a move offset of $\bar{p}, q$.*
3. *If $\bar{p} \rightleftarrows \bar{p} \nearrow \bar{r} \rightleftarrows \bar{r} \curvearrowright q$, then both 1 and 2 are move offsets of $\bar{p}, q$.*
4. *If $\bar{p} \leftarrow\!\!\!\!\!\!\leftarrow \bar{p} \nwarrow \bar{r} \leftarrow\!\!\!\!\!\!\leftarrow \bar{r} \curvearrowright q$, then both -1 and 0 are move offsets of $\bar{p}, q$.*
5. *If $\bar{p} \leftarrow\!\!\!\!\!\!\leftarrow \bar{p} \curvearrowright q \leftrightarrows q$, then both -1 and 0 are move offsets of $\bar{p}, q$.*

Proof.

1. The proof is immediate by Lemmas 6.5 and 6.6.

2. By Lemma 5.5, there is a state p such that $(p, \bar{p})$ is a left-to-right DFS. By Lemma 6.8, $\bar{p}, p$ has offset 1. From $p \swarrow p \circlearrowleft \bar{p} \curvearrowright q$, we have $p \downarrow q$. Using Lemma 6.6, it follows by swallowing that $\bar{p}, q$ also has offset 1.
3. The move offset 1 follows from the previous case, since $\bar{p} \nwarrow\!\!\!\uparrow \bar{p} \curvearrowright q$. By Lemma 6.6, the move offset 2 will follow once we show that $S(\bar{p}, q)$ contains both

$$\nwarrow^* \nearrow \nwarrow^* \curvearrowright \swarrow^* \quad \text{and} \quad \nwarrow^* \curvearrowright \swarrow^* \searrow \swarrow^*.$$

By Lemma 5.5, there are states p, r such that $(p, \bar{p})$ and $(r, \bar{r})$ are left-to-right DFSs. By $r \swarrow r \circlearrowleft \bar{r} \curvearrowright q$, we obtain $r \downarrow q$. Then by

$$\bar{p} \nwarrow \bar{p} \nearrow \bar{r} \nwarrow \bar{r} \curvearrowright r \swarrow r \downarrow q$$

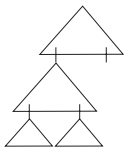
we obtain that $S(\bar{p}, q)$ contains $\nwarrow^* \nearrow \nwarrow^* \curvearrowright \swarrow^*$.
It remains to show that $S(\bar{p}, q)$ contains $\nwarrow^* \curvearrowright \swarrow^* \searrow \swarrow^*$. This will follow once we prove $p \searrow r$, by

$$\bar{p} \nwarrow \bar{p} \curvearrowright p \swarrow p \searrow r \swarrow r \downarrow q.$$

It thus remains to show $p \searrow r$. By $p \swarrow p \circlearrowleft \bar{p} \nearrow \bar{r}$, we get $p \to_\varepsilon \bar{r}$. Finally, if we consider the path in $\Delta_2[\Delta_1, *]$ witnessed by $p \swarrow p \to_\varepsilon \bar{r} \curvearrowright r$, we get the desired $p \searrow r$.
4. By item 1, there is a move offset of 0.
By Lemma 5.5, there are states p, r such that both $(p, \bar{p})$ and $(r, \bar{r})$ are right-to-left DFSs. Using the space-symmetric variant of the proof of $p \searrow r$ in item 3, we obtain $p \swarrow r$. Now, by $p \swarrow r \circlearrowleft \bar{r} \curvearrowright q$, we have $p \downarrow q$. Consequently $\bar{p} \swarrow\!\!\!\uparrow \bar{p} \curvearrowleft p \downarrow q$, and, applying the space-symmetric version of item 2, we obtain an offset of -1 for $\bar{p}, p$.
5. The offset 0 follows from item 1. By Lemma 5.5, there are states $p, \bar{q}$ such that both $(p, \bar{p})$ and $(q, \bar{q})$ are right-to-left DFSs.
Let us show that $\bar{p} \uparrow \bar{q}$. For this, we trace the run

$$\bar{p} \curvearrowleft p \searrow p \circlearrowleft \bar{p} \curvearrowleft p \circlearrowleft \bar{p} \curvearrowright q \circlearrowleft \bar{q} \curvearrowleft q \circlearrowleft \bar{q} \nearrow \bar{q} \nearrow \bar{q}$$

that goes from state $\bar{p}$ in port 0 to state $\bar{q}$ in port ε of the following pattern, which is equivalent to Δ_1:



Now by $\bar{p} \uparrow \bar{q} \nearrow \bar{q} \curvearrowleft q \searrow q$, we obtain that $\nearrow^* \curvearrowleft \searrow^* \subseteq S(\bar{p}, q)$. By Lemma 6.6, we obtain offset -1. □

6.3. Classification of moves. In this section we state and prove Proposition 6.10, which says that if $S(p, q)$ is nonempty, then either it is a union of some of the eleven elementary moves, or there is a shift. This separation of cases is at the core of the argumentation of section 7.

PROPOSITION 6.10. *If $S(p, q)$ is nonempty, then p, q have a move offset in $\{-1, 0, 1\}$. Furthermore, either p, q admit a shift or $S(p, q)$ is a union of elementary moves.*

Proof. By Lemma 6.3, the move $S(p,q)$ is a finite union of sets of the form $\{\varepsilon\}$, which is an elementary move, or

$$(U(p,p')+\varepsilon) \curvearrowright (D(q',q)+\varepsilon), \qquad \text{where } p',q' \text{ satisfy } p \uparrow p' \curvearrowright q' \downarrow q, \qquad \text{or}$$
$$(U(p,p')+\varepsilon) \curvearrowleft (D(q',q)+\varepsilon), \qquad \text{where } p',q' \text{ satisfy } p \uparrow p' \curvearrowleft q' \downarrow q.$$

Letting $M \subseteq S(p,q)$ be a move of one of those forms, we prove that either $S(p,q)$ contains a shift or there is an elementary move E such that $M \subseteq E \subseteq S(p,q)$. If this holds for all such moves M, we directly obtain the statement of the proposition, using Corollary 6.7 for the offset of elementary moves.

By symmetry, we consider only the case $M = (U(p,p')+\varepsilon) \curvearrowright (D(q',q)+\varepsilon)$. We now apply Proposition 5.10 to $p \uparrow p'$. This proposition distinguishes six cases, namely, $\upuparrows$, $\upharpoonright$, $\upharpoonleft$, (a), (b), and (c). Similarly, we can do the time-reversed reasoning for $q' \downarrow q$ and also consider six cases.

If we have (b) or (c) for $p \uparrow p'$, then by items 3 and 4 of Lemma 6.9, respectively, we get a shift. In the case of (a), we get a shift by using items 1 and 2. By the time-space-reverse variant of Lemma 6.9, the same happens if (a), (b), or (c) holds for $q' \downarrow q$.

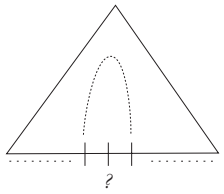
Only the other cases remain. There are nine possibilities:

- If $p \upharpoonright p'$ and $q' \upharpoonleft q$, we get $M = \sqcup\!\!\rightarrow$ by Lemma 6.4. Similarly, if $p \upharpoonright p'$ and $q' \downdownarrows q$, we get $M = \sqcup\!\!\downarrow$, and when $p \upuparrows p'$ and $q' \upharpoonleft q$, we have $M = \sqcap\!\!\rightarrow$.
- If $p \upharpoonleft p'$ and $q' \downdownarrows q$, we have $M \subseteq \sqcap\!\!\downarrow$ by Lemma 6.4. Furthermore, using Lemmas 6.4 and 6.5, we get $\sqcap\!\!\downarrow \subseteq S(p,q)$. Similarly, if $p \upuparrows p'$ and $q' \upharpoonright q$, we obtain $M \subseteq \sqcap\!\!\rightarrow \subseteq S(p,q)$.
- If $p \upharpoonright p'$ and $q' \upharpoonright q$, then p,q admit a shift. Indeed, 1 is a move offset of p,q: by Lemma 5.9, $p \uparrow p'' \upharpoonright p'' \uparrow p'$, and item 2 of Lemma 6.9 gives offset 1. Offset 0 is obtained by the time-space-reverse variant of Lemma 6.5 (i.e., if $\bar{p} \curvearrowright q' \upharpoonright q$, then $\epsilon \in S(p,q)$). The results are similar for $p \upharpoonleft p'$ and $q' \upharpoonleft q$.
- If $p \upharpoonleft p'$ and $q' \upharpoonright q$, then p,q admit a shift by item 5 of Lemma 6.9.
- If $p \upuparrows p'$ and $q' \downdownarrows q$, then p,q clearly admit a shift; 1 and 2 are move offsets. ∎

6.4. Right-skipping moves. The result of this section concerns right-skipping moves. A move is called *right-skipping* if it contains an element of

$$\text{Right} \setminus \sqcup\!\!\rightarrow.$$

A right-skipping move can go to the right in a pattern while omitting (skipping) the junction leaf immediately to the right:



We say that states p,q are *right-skipping* if the move $R(p,q)$ is right-skipping. A *left-skipping* move is defined in the same fashion.

LEMMA 6.11. *Let p,q be states with a maximal move offset k. Let u,v be leaves of a tree t, with $\#_t(u,v) > k$. If $[p,u] \to [q,v]$, then p,q are right-skipping.*

Proof. Note that if the offset k can be arbitrarily large, i.e., there is no maximal offset k, then p,q are right-skipping straight from the definition, thanks to any move

offset larger than 1. By the first clause of Proposition 6.10, $k \geq -1$, and so $\#_t(u,v)$ is at least 0. Furthermore, it cannot be 0, since otherwise we would have $p \to_\varepsilon q$ and therefore $0 \in \mathrm{moff}(p,q)$; this would give $k \geq 0$, a contradiction with $\#_t(u,v) > k$.

If $\#_t(u,v)$ is at least 2, then p, q are right-skipping by definition; the remaining case is 1. The path $\pi(u,v)$ witnesses $p \curvearrowright q$ and, consequently, the existence of p', q' such that $p \uparrow p' \curvearrowright q' \downarrow q$. By Proposition 5.6, either $p \nwarrow p'$ or $p \nearrow p'$ must hold. If $p \nearrow p'$, then $\nearrow\!\!\curvearrowright \in R(p,q)$ is a witness for p, q being right-skipping. Otherwise, $p \upharpoonleft p' \curvearrowright q$. By Lemma 5.9 and item 2 of Lemma 6.9, p, q has offset 1, which gives $k \geq 1$; a contradiction with $\#_t(u,v) > k$. ∎

7. The rotation. We have now gathered enough information about runs of the automaton that never see the label **a**. In this section we consider runs that do see **a**, and conclude the proof of Theorem 2. We will show that the tree-walking automaton $\mathcal{A}$ cannot detect a well-placed rotation in a large balanced tree.

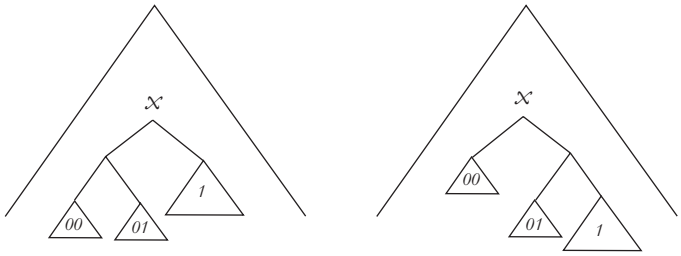


FIG. 7.1. *Rotating at node x.*

We proceed as follows. We start with a blank balanced binary tree T of large even depth. Clearly all leaves of T are at even depth, and therefore $\mathcal{A}$ must accept the tree $\Delta^{\mathbf{a}}_T$. We then choose a pivot node x in T and perform a rotation at that node. Rotation is the operation depicted in Figure 7.1; it moves the subtrees rooted in $x00$, $x01$, and $x1$ to the new positions $x0$, $x10$, and $x11$. One can easily see that the resulting tree T' has some leaves at odd depth, and hence $\mathcal{A}$ should not accept $\Delta^{\mathbf{a}}_{T'}$. We will, however, show that, for a carefully chosen pivot, $\mathcal{A}$ does accept this tree, thereby completing the proof of Theorem 2.

PROPOSITION 7.1. *The tree $\Delta^{\mathbf{a}}_{T'}$ is accepted by $\mathcal{A}$.*

First we describe how to properly choose the height of the tree T and the pivot in it. The remainder of the paper is then devoted to showing Proposition 7.1.

7.1. The pivot. The goal of this section is to construct a tree T, an accepting run of $\mathcal{A}$ over $\Delta^{\mathbf{a}}_T$, and a node x of T (the pivot), such that the properties of Definition 10 are satisfied. Essentially, these properties say that the tree is balanced and large and the path leading to the pivot contains a zigzag. Furthermore, some undesirable parts of the accepting run do not use nodes below the pivot. These properties will be used in the remainder of the paper in order to prove that doing a rotation in the pivot x on the tree T gives a tree T' such that $\Delta^{\mathbf{a}}_{T'}$ is accepted by $\mathcal{A}$. We begin by defining the "undesirable" parts of the run. After that, we state Definition 10 and then show that the tree and pivot can indeed be found, in Lemma 7.3.

Let t be a blank tree. Recall the definitions of junction and leaf configurations from section 4.2. By distinguishing all the configurations whose node is either a junction leaf or the root of $\Delta^{\mathbf{a}}_t$, every accepting run in $\Delta^{\mathbf{a}}_t$ can be decomposed into a sequence of the following form:

$$(q_1, v_1), \ldots, (q_n, v_n), \tag{7.1}$$

where each v_i $(i = 1, \ldots, n)$ is either the root ε or $[u_i]$ for some leaf u_i of t, and in between two such configurations, no junction leaf or the root is visited. In this case the run linking (q_i, v_i) to (q_{i+1}, v_{i+1}) is either

(a) a run visiting at least once the root of $\Delta_t^{\mathbf{a}}$ (and no $\mathbf{a}$-labeled leaf of $\Delta_t^{\mathbf{a}}$);
(b) a loop inside the $\Delta_1[\Delta_{\mathbf{a}}]$ subtree rooted in v_i, which is equivalent to $\Delta_{\mathbf{a}}$ (hence $q_i \circlearrowleft_{\mathbf{a}} q_{i+1}$); or
(c) a run from a junction leaf to another junction leaf in the pattern Δ_t (hence by Lemma 6.1, $[q_i, u_i] \to [q_{i+1}, u_{i+1}]$ holds, where $v_i = [u_i]$ and $v_{i+1} = [u_{i+1}]$).

Such a sequence is called a *rooted leaf run in* $\Delta_t^{\mathbf{a}}$. An *unrooted leaf run* is one that never uses a step of the form (a). By shortcutting each part of the run starting and ending in the root with the same state, we can safely assume that every rooted leaf run uses at most $2|Q|$ steps of the form (a); hence the greater part of a rooted leaf run is unrooted. Since an unrooted leaf run uses only leaf configurations, it can be written as $[q_1, u_1], \ldots, [q_n, u_n]$.

For junction configurations $[p, v]$ and $[q, w]$, we write $[p, v] \Rightarrow_t [q, w]$ if in the tree $\Delta_t^{\mathbf{a}}$ there is a run from $[p, v]$ to $[q, w]$ that does not visit the root. Note that when v, w are leaves, this means that there is an unrooted leaf run in $\Delta_t^{\mathbf{a}}$ from $[p, v]$ to $[q, w]$. As opposed to the relation $[p, v] \to [q, w]$, this run may depend on the tree t and not only on the nodes v and w.

We say that one state q is *leaf reachable* from another state p if they can be connected by an unrooted leaf run in some tree; i.e., $[p, v] \Rightarrow_t [q, w]$ for some tree t and leaves v, w. Equivalently, q is leaf reachable from p if there exist $p = p_1, \ldots, p_n = q$ such that for every $i = 1, \ldots, n-1$ either $S(p_i, p_{i+1})$ is nonempty (which corresponds to case (c)) or $p_i \circlearrowleft_{\mathbf{a}} p_{i+1}$ holds (corresponding to case (b); recall the definition of $\circlearrowleft_{\mathbf{a}}$ from Figure 5.1). (The right-to-left part of this equivalence is a consequence of the existence of a move offset in $\{-1, 0, 1\}$ shown in Proposition 6.10 and is not a priori obvious). A *component* of the automaton is a maximal set of pairwise leaf reachable states; in other words, it is a strongly connected component of the directed graph with Q as nodes and an edge from p to q if and only if $S(p, q)$ is nonempty or $p \circlearrowleft_{\mathbf{a}} q$.

Let us consider a rooted leaf run ρ as in (7.1), which witnesses the acceptance of $\Delta_t^{\mathbf{a}}$ by $\mathcal{A}$. The main point in choosing the pivot is to restrict our attention to fragments of ρ that are unrooted leaf runs and only use states from one component. We say the run *changes components* below a node y of t if it contains two successive leaf configurations (q_i, v_i), (q_{i+1}, v_{i+1}) such that q_i and q_{i+1} are in different components and at least one of v_i, v_{i+1} is below $[y]$. The run is *rooted below* y if there is some i such that $v_i = \varepsilon$ and either v_{i-1} or v_{i+1} exists and is below $[y]$.

DEFINITION 10. *We define the following properties for a node x in a blank balanced tree t with respect to a rooted leaf run of $\mathcal{A}$ in $\Delta_t^{\mathbf{a}}$:*

1. *the subtrees rooted in x and the children of x are $\log_2(|Q|)$-fractal (see below);*
2. *the subtree of x has depth larger than $4 + |Q| + 2\log_2(|Q|)$;*
3. *the node x is below the node* 01010101;
4. *the run does not change components below x;*
5. *the run is not rooted below x.*

Note that since the tree t is balanced, the number of leaves below a node v depends only on its depth. Then let v be a node of t, whose depth $|v| + 1$ is at most $|x| + 2$. From condition 2, it follows that the number of leaves in the subtree of v exceeds all the following thresholds (the constants D, E defined below will be used in the subsequent proofs):

$$|Q|, \quad D = |Q|(|Q|+1), \quad E = 3D + |Q|. \tag{7.2}$$

We will refer to the above property later in the paper.

We will now proceed to show that such a run and a node (called the pivot) can be found (Lemma 7.3) as long as t is a sufficiently large balanced tree of even depth. Before we do so, however, we need to define what a fractal tree is.

Within a tree t, we distinguish five *characteristic types* of nodes: (1) the root, (2) the leftmost leaf, (3) the rightmost leaf, (4) the remaining leaves, and—for the sake of completeness—(5) the remaining nodes. We say a tree t' *simulates* a tree t if for every two junction configurations in t that satisfy $[p, v] \Rightarrow_t [q, w]$ one can find two configurations satisfying $[p, v'] \Rightarrow_{t'} [q, w']$ such that v and v' have the same characteristic type, as well as w and w'. Given a natural number m, a tree is called *m-fractal* if it contains a proper subtree that simulates it and has depth larger than m.

LEMMA 7.2. *For every natural number m, all balanced binary trees of sufficiently large depth are m-fractal.*

Proof. For a tree t and states p, q, we can calculate the characteristic types of nodes v, w satisfying $[p, v] \Rightarrow_t [q, w]$. This information is sufficient to see whether one tree simulates another. Moreover, it can be calculated by a deterministic bottom-up finite tree automaton. In the case of a balanced binary tree, this information is a (regular) property of its depth and is thus ultimately periodic. This means that there exist constants N and k such that every balanced blank tree of depth $n \geq N$ is simulated by the balanced blank tree of depth $n - k$. Hence every balanced tree of depth at least $\max\{m + k, N\}$ is m-fractal. ∎

LEMMA 7.3. *There exist a blank balanced binary tree T, an accepting run ρ of $\mathcal{A}$ in $\Delta_T^{\mathbf{a}}$, and a pivot x such that x satisfies properties* 1–5 *of Definition* 10 *with respect to ρ.*

Proof. Let N be more than the minimum depth of balanced tree obtained from Lemma 7.2 for $m = \log_2(|Q|)$ and also larger than the depth stated in condition 2 of Definition 10. Let K be more than $\log_2(4|Q|) + 1$. Finally, let T be a balanced blank tree of even depth larger than $N + K + 8$. The tree $\Delta_T^{\mathbf{a}}$ is accepted by $\mathcal{A}$ by hypothesis on $\mathcal{A}$.

Consider now an accepting run and its representation as in (7.1),

$$(q_1, v_1), \ldots, (q_n, v_n),$$

where only the root and junction leaf configurations are displayed. We assume that in this run, the root of $\Delta_T^{\mathbf{a}}$ is seen at most $|Q|$ times. If this is not the case, some state appears twice in the root, and hence a configuration is seen twice in the run; one can then shortcut the corresponding part of the run.

Let V_{root} be all those junction leaves v_i such that at least one of v_{i-1}, v_{i+1} is the root ε. One can easily see that the run is rooted below a node y if and only if $[y]$ is above some node in V_{root}. Likewise, let V_{cc} be all those junction leaves v_i such that the state q_i is in a component other than either q_{i-1} or q_{i+1} (or both). Again, the run changes component below a node y if and only if $[y]$ is above some node in V_{cc}. Combining the above, a node x satisfies properties 4 and 5 from Definition 10 if and only if x is not above some node in one of $V_{\text{root}}, V_{\text{cc}}$. By assumption on the run not visiting the root more than $|Q|$ times, the sets $V_{\text{root}}, V_{\text{cc}}$ have together at most $4|Q|$ nodes.

Let us count the number of nodes at depth $K + 8$ in T that are below 01010101. There are 2^{K-1} such nodes, i.e., more than $4|Q|$ by construction of K. In particular,

there is a node x at depth $K+8$ that satisfies conditions 3, 4, and 5. Furthermore, since the whole tree has depth at least $N+K+8$, the subtree of x has depth at least N, and therefore x satisfies conditions 1 and 2. ∎

Detection of the rotation. Throughout the remainder of the paper, the tree T, the run ρ, and the pivot x are fixed according to Lemma 7.3. Let T' be the tree obtained from T by doing a rotation in the pivot x; this tree clearly contains leaves at odd depth. Our objective is to show that $\Delta^{\mathbf{a}}_{T'}$ is accepted by $\mathcal{A}$. For this, we have to show that the run can in some sense be replicated after the rotation. We will use properties 4 and 5 from Definition 10 to show that only unrooted leaf runs that do not change components need be considered.

We say that a component $\Gamma \subseteq Q$ of the automaton $\mathcal{A}$ *cannot detect the rotation* if for every two leaf configurations $[p, v]$, $[q, w]$ with p, q in Γ and v, w not below the pivot

$$[p, v] \Rightarrow_T [q, w] \qquad \text{implies} \qquad [p, v] \Rightarrow_{T'} [q, w]. \tag{7.3}$$

Observe that it makes sense to speak about the configurations $[p, v]$ and $[q, w]$ in the tree T' since the leaves v and w are not below the pivot and hence are not affected by the rotation.

Consequence for Proposition 7.1. Let us show that, under the assumption that no component can detect the rotation, we have Proposition 7.1.

Consider the run ρ and its representation as in (7.1):

$$(q_1, v_1), \ldots, (q_n, v_n).$$

Recall how we classified each of the runs linking (q_i, v_i) to (q_{i+1}, v_{i+1}), for $i = 1, \ldots, n-1$, according to their form, (a), (b), or (c). Also define u_i to be the node of T such that $v_i = [u_i]$, when possible, for $i = 1, \ldots, n$.

We claim the following: for $i < j$ in $\{1, \ldots, n\}$ such that both v_i and v_j are not below $[x]$, there is a run of the automaton $\mathcal{A}$ from (q_i, v_i) to (q_j, v_j) in $\Delta^{\mathbf{a}}_{T'}$. Since neither $v_1 = \varepsilon$ nor $v_n = \varepsilon$ is below $[x]$, our claim yields a run in $\Delta^{\mathbf{a}}_{T'}$ that goes from (q_1, ε) to (q_n, ε). Since the original run from (q_1, ε) to (q_n, ε) in $\Delta^{\mathbf{a}}_{T}$ was accepting, we know that q_1 is initial and q_n is final, and therefore the tree $\Delta^{\mathbf{a}}_{T'}$ is accepted by the automaton.

The proof of the claim is by induction on $j - i$. The induction step is obvious and follows by concatenating runs. The nontrivial case is the base case of the induction, when for every k with $i < k < j$ the node v_k is below $[x]$. We now prove the claim for this case.

Consider first the case when $i + 1 = j$. We now look at the form of the subrun that goes from (q_i, v_i) to (q_{i+1}, v_{i+1}). If this subrun is of the form (b), it can be directly replicated on $\Delta^{\mathbf{a}}_{T'}$ without change. If the subrun from (q_i, v_i) to (q_{i+1}, v_{i+1}) is of the form (c), then we use Corollary 6.2, which shows that a similar run can be used in $\Delta^{\mathbf{a}}_{T'}$ from configuration (q_i, v_i) to (q_{i+1}, v_{i+1}). The last remaining case is when the subrun from (q_i, v_i) to (q_{i+1}, v_{i+1}) is of the form (a), and in particular either $v_i = \varepsilon$, or $v_j = \varepsilon$, or both hold. If $v_i = \varepsilon$, but $v_j \neq \varepsilon$, we decompose the run from (q_i, ε) to (q_j, v_j) into a run from (q_i, ε) to $[q', \varepsilon]$, which does not visit more than once the nodes ε and $[\varepsilon]$, followed by a run from $[q', \varepsilon]$ to $[q_j, u_j]$, which does not visit ε or a junction leaf other than $[u_j]$. This second piece of run can be reused in $\Delta^{\mathbf{a}}_{T'}$ according to Corollary 6.2. Hence there is a run in $\Delta^{\mathbf{a}}_{T'}$ from (q_i, v_i) to (q_j, v_j). The case $v_i \neq \varepsilon$ and $v_j = \varepsilon$ is obtained by time symmetry. Finally, when $v_i = v_j = \varepsilon$,

either the run does not visit $[\varepsilon]$ and can be directly reused in $\Delta^{\mathbf{a}}_{T'}$, or it visits $[\varepsilon]$. In this latter case, let q' be the first and q'' be the last state assumed by the run when visiting $[\varepsilon]$. The piece of run between $[q', \varepsilon]$ and $[q'', \varepsilon]$ does not visit any junction leaf, and consequently by Lemma 4.1, there is a similar run which does not visit any junction node other than $[\varepsilon]$. This run can be reused in $\Delta^{\mathbf{a}}_{T'}$ to obtain a run from (q_i, v_i) to (q_j, v_j).

Otherwise, we have $i + 1 < j$. For all k with $i < k < j$, the node v_k is below $[x]$, which together with condition 5 of Definition 10 gives that no subruns of form (a) happen between (q_i, v_i) and (q_j, v_j). Consequently, all the nodes v_k are junction leaves, and the run from (q_i, v_i) to (q_j, v_j) is an unrooted leaf run:

$$[q_i, u_i] \Rightarrow_T [q_j, u_j].$$

Furthermore, by condition 4 of Definition 10, the states q_i and q_j belong to the same component Γ of the automaton. Also, by hypothesis on i and j, neither u_i nor u_j is below x. It follows that we can apply our assumption that no component can detect the rotation, and get a run corresponding to

$$[q_i, u_i] \Rightarrow_{T'} [q_j, u_j].$$

This completes the proof of the claim and, consequently, of Proposition 7.1. What remains to be done is to establish that no component of the automaton can detect the rotation. The remainder of this paper is dedicated to establishing this.

Using Proposition 6.10, we divide all components into two categories: components with a shift, i.e., containing two states that admit a shift, and components without a shift. Proposition 7.1 is then proved in sections 7.2 and 7.3 for each of the two categories.

7.2. Components with a shift cannot detect the rotation. In this section we fix a component Γ with a shift and prove that it cannot detect the rotation. In order to do this, we extend the definition of move offsets to *run offsets*, where more than one move can be used. The idea is that the shift in the component can be exploited to allow the automaton to move around the tree in an almost arbitrary fashion, independently of the structure of T.

A run offset between state p and state q is defined similarly to a move offset.

DEFINITION 11. *Given a natural number $m \geq 0$ called the* safety margin*, an m-*run offset *of states p, q is an integer i such that $[p, v] \Rightarrow_t [q, w]$ holds for every two leaves v, w of a tree t where $\#_t(v, w) = i$ and v, w both have at least m leaves to their left and right. We write $\mathrm{roff}_m(p, q)$ for the set of m-run offsets of p, q.*

We remark here, slightly ahead of time, that a consequence of x being below 01010101 is that all nodes below the pivot, and even some nodes not below the pivot, have at least m leaves to their left and right for fairly large values of m. This means that those nodes are suitable for using run offsets.

The differences between move offsets and run offsets are that (1) we replace $\to$ with $\Rightarrow_t$ (which depends on t and can read the label $\mathbf{a}$) and (2) the leaves v, w must have a "safety margin" of at least m leaves to their left and to their right; see Figure 7.2.

We now list some basic properties of roff, which hold for any given states p, q, r. First, $\mathrm{moff}(p, q)$ is included in $\mathrm{roff}_0(p, q)$. Furthermore, if $p \circlearrowleft_{\mathbf{a}} q$ holds, then 0 belongs to $\mathrm{roff}_0(p, q)$. Also, if both $i, j \leq 0$ or both $i, j \geq 0$, then

$$i \in \mathrm{roff}_m(p, q),\ j \in \mathrm{roff}_m(q, r) \qquad \Rightarrow \qquad i + j \in \mathrm{roff}_m(p, r).$$

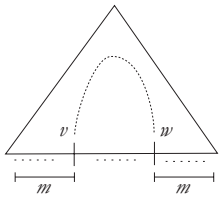


FIG. 7.2. *An m-run offset.*

Finally, if $ij \leq 0$ (i.e., if i, j are of opposite sign), then

$$i \in \mathrm{roff}_m(p,q),\ j \in \mathrm{roff}_m(q,r) \qquad \Rightarrow \qquad i+j \in \mathrm{roff}_{m+\min(|i|,|j|)}(p,r).$$

In particular, if $j \in \{-1,0,1\}$, then $i+j \in \mathrm{roff}_{m+1}(p,r)$.

A consequence of these properties, together with Proposition 6.10, is given in the following lemma.

LEMMA 7.4. *For every component Γ and states p, q in Γ, $\mathrm{roff}_{|Q|-2}(p,q)$ is nonempty and contains a value k with $|k| < |Q|$.*

Proof. For $p = q$, we naturally have $0 \in \mathrm{roff}_0(p,q)$, and hence $0 \in \mathrm{roff}_{|Q|-2}(p,q)$. Now assume $p \neq q$. Since p and q are in the same component, there exists a sequence of states $p = r_1, \ldots, r_n = q$ such that for all $i = 1, \ldots, n-1$ either $r_i \circlearrowleft_{\mathbf{a}} r_{i+1}$ holds or the move $S(r_i, r_{i+1})$ is nonempty. Without loss of generality, we assume that no state is seen twice in this sequence, and therefore $2 \leq n \leq |Q|$.

By induction on $i = 2, \ldots, n$, we will show that $\mathrm{roff}_{i-2}(r_1, r_i)$ contains a value k with $|k| < i$. For $i = n \leq |Q|$, the statement of the lemma follows. For $i = 2$, using the properties described above, if $r_1 \circlearrowleft_{\mathbf{a}} r_2$, then $0 \in \mathrm{roff}_0(r_1, r_2)$, and if $S(r_1, r_2)$ is nonempty, then $\mathrm{moff}(r_1, r_2)$ contains an offset $j \in \{-1, 0, 1\}$ by Proposition 6.10, and hence $j \in \mathrm{roff}_0(r_1, r_2)$. For $i > 2$, by the induction hypothesis, the set $\mathrm{roff}_{i-3}(r_1, r_{i-1})$ contains a value k with $|k| < i-1$. If $r_{i-1} \circlearrowleft_{\mathbf{a}} r_i$ holds, then $\mathrm{roff}_{i-3}(r_1, r_i)$ also contains k, and therefore so does $\mathrm{roff}_{i-2}(r_1, r_i)$. On the other hand, if $S(r_{i-1}, r_i)$ is nonempty, then $\mathrm{moff}(r_{i-1}, r_i)$ contains an offset $j \in \{-1, 0, 1\}$ by Proposition 6.10. In particular, the offset j belongs to $\mathrm{roff}_0(r_{i-1}, r_i)$. Using the properties described above, we obtain that $k + j$ belongs to $\mathrm{roff}_{i-2}(r_1, r_i)$. This concludes the induction step, since $|k + j| < i$. ∎

For a safety margin m and a natural number d called the *threshold*, a pair of states p, q is called an (m,d)-*right-teleport* if $\mathrm{roff}_m(p,q)$ contains all integers not smaller than d. An (m,d)-*left-teleport* is when $\mathrm{roff}_m(p,q)$ contains all the integers not larger than $-d$. An m-*full-teleport* corresponds to $\mathrm{roff}_m(p,q)$ containing all integers. Observe that if p, q is an (m,d)-right-teleport and q, r is an (m,d)-left-teleport, then p, r is an $(m+d)$-full-teleport.

Recall the constant $D = |Q|(|Q|+1)$ defined after Definition 10.

LEMMA 7.5. *If a component Γ contains a shift, either all pairs of states in Γ are $(2|Q|, D)$-right-teleports or all are $(2|Q|, D)$-left-teleports.*

Proof. Let q_1, q_2 be states in the component Γ that admit a shift, i.e., have two consecutive move offsets $i, i+1 \in \{-2, -1, 0, 1, 2\}$. By adding $i, i+1$ to the offset obtained by applying Lemma 7.4 to the pair q_2, q_1, we infer that q_1, q_1 admits two consecutive $|Q|$-run offsets k and $k+1$ with $|k|, |k+1| \leq |Q|+1$.

Without loss of generality, let us assume that $0 \leq k \leq |Q|$. Let

$$m \geq |Q|(|Q|-1) \geq k(k-1).$$

We will show that m belongs to $\mathrm{roff}_{|Q|}(q_1, q_1)$. For $k = 0$, we have $k + 1 = 1 \in \mathrm{roff}_{|Q|}(q_1, q_1)$ and hence, using this run offset m times, $m \in \mathrm{roff}_{|Q|}(q_1, q_1)$. For $k \geq 1$, the number m can be written as $\alpha k + \beta$ with $\beta \in \{0, \ldots, k-1\}$. In particular,

$$m = (\alpha - \beta)k + \beta(k + 1).$$

Let us remark that since $m \geq k(k-1)$, we have that $\alpha \geq k - 1$ and consequently $\alpha - \beta \geq 0$. Hence by using $\alpha - \beta$ times the run offset k, and β times the run offset $k + 1$, one obtains

$$m \in \mathrm{roff}_{|Q|}(q_1, q_1).$$

This proves that q_1, q_1 is a $(|Q|, |Q|(|Q| - 1))$-right-teleport.

Once one pair of states q_1, q_1 is a right-teleport, the same can be shown for all other state pairs in the component Γ. Indeed, we will show that any two states p, q in Γ are a $(2|Q|, |Q|(|Q| + 1))$-right-teleport, which completes the proof of the lemma. We need to show that the automaton can go from $[p, v]$ to $[q, w]$. First, using Lemma 7.4, the automaton can go from $[p, v]$ to a configuration of the form $[q_1, u_1]$, where u_1 is a leaf separated from v by at most $|Q|$ leaves (note that u_1 may be to the left or to the right of v). In the same way, there is a leaf u_2, separated from w by at most $|Q|$ leaves, such that the automaton can go from $[q_1, u_2]$ to $[q, w]$. If the leaves v, w have safety margins of $2|Q|$, then the leaves u_1, u_2 have safety margins of at least $|Q|$. Furthermore, if w is at least

$$|Q|(|Q| - 1) + 2|Q| = |Q|(|Q| + 1) = D$$

leaves to the right of v, then u_2 is at least $|Q|(|Q| - 1)$ leaves to the right of u_1. Therefore the $(|Q|, |Q|(|Q| - 1))$-right-teleport q_1, q_1 can be used to go from $[q_1, u_1]$ to $[q_1, u_2]$. This completes the proof that p, q is a $(2|Q|, D)$-right-teleport.

The left-teleport is obtained in the case when k is negative. □

For the remainder of this section (section 7.2), we assume that the second case in the above lemma holds, i.e., that all pairs of states are $(2|Q|, D)$-left-teleports. The case of right-teleports is symmetric. We now proceed to show that the component Γ cannot detect the rotation, i.e., that the implication (7.3) holds for any two leaves v, w not below the pivot and any two states p, q of the component Γ.

Consider all the leaf configurations of the unrooted leaf run from $[p, v]$ to $[q, w]$:

$$[p, v] = [r_0, u_0] \Rightarrow_T [r_1, u_1] \Rightarrow_T \cdots \Rightarrow_T [r_n, u_n] \Rightarrow_T [r_{n+1}, u_{n+1}] = [q, w]. \tag{7.4}$$

Without loss of generality we can assume that all the leaves $u_1, \ldots, u_n$ are below the pivot; the other parts of the run can be easily replicated in T' (as explained at the end of section 7.1). Note that since Γ is a component, all the states $r_0, \ldots, r_{n+1}$ belong to Γ.

We will do a case analysis. We say the leaf run from $[p, v]$ to $[q, w]$ satisfies property (*) if for some $0 \leq i < j \leq n + 1$ the leaf u_j is at least $|Q|$ leaves to the right of u_i, i.e., $\#_T(u_i, u_j) \geq |Q|$. We do the proof first for leaf runs that do not satisfy this property and then for those that do.

7.2.1. Leaf runs not satisfying (*). Recall that we assume that Γ is a component such that every pair of states in it is a left-teleport. We make a case distinction depending on the relative position of v and w with respect to the pivot.

If v is to the left of the pivot and w is to its right, then all the leaves below the pivot separate v from w. By (7.2), there are more than $|Q|$ of these leaves, which contradicts our assumption on property (*) failing.

Consider now the case when v is to the right of the pivot and w is to the left. If v has more than $2|Q|$ leaves to its right and w has more than $2|Q|$ leaves to its left, we can use the $(2|Q|, D)$-left-teleport and go from $[p, v]$ to $[q, w]$ independently of the rotation, since there are at least D leaves below the pivot x, thanks to (7.2). Let us assume now that v has less than $2|Q|$ leaves to its right and w has less than $2|Q|$ leaves to its left. By (7.2), node 11 has more than $2|Q|$ leaves in its subtree. In particular, if there are less than $2|Q|$ leaves to the right of v, then v must be below 11. Since u_1 is below the pivot x, hence below 01, the path going from $[p, v]$ to $[r_1, u_1]$ is of the form $\pi \nwarrow \curvearrowleft \searrow \pi' \in S(p, r_1)$. It follows that $\pi \curvearrowleft \pi'$ also belongs to $S(p, r_1)$, by swallowing. Hence, the automaton can go from $[p, v]$ to $[r_1, v']$ for some v' below 10. In the same way, if w has less than $2|Q|$ leaves to its left, one shows that there exists a leaf w' below 001 such that the automaton can go in one move from $[r_n, w']$ to $[q, w]$. We are in the situation where v' has more than $2|Q|$ leaves to its right, w' has more than $2|Q|$ leaves to its left, and there are more than D leaves between w' and v'. Now we can use the $(2|Q|, D)$-left-teleport to go from $[r_1, v']$ to $[r_n, w']$. Furthermore, since v', w' are not below the pivot and the teleport is used to jump over the leaves below the pivot, the resulting run cannot detect the rotation. The other cases are a combination of the two previous ones.

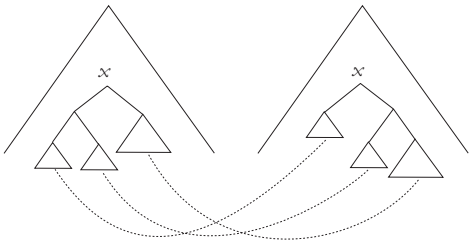


FIG. 7.3. *The bijection f.*

Next we consider the case for which both v and w are to the right of the pivot. Since condition (*) is not satisfied, all leaves $u_1, \ldots, u_n$ are at most $|Q|$ leaves away from w. In particular all these leaves are below $x1$ since the subtree rooted in $x1$ contains at least $|Q|$ leaves by (7.2). Let f be the unique bijection between the leaves of T and the leaves of T' that preserves the leaf numbering (i.e., $\#_T(v) = \#_{T'}(f(v))$; see Figure 7.3). Note that f is the identity function on leaves not below the pivot; in particular $f(v) = v$ and $f(w) = w$. We claim that the unrooted leaf run from $[p, v]$ to $[q, w]$ can be replicated in the tree T' as follows:

$$[p, v] \Rightarrow_{T'} [r_1, f(u_1)] \Rightarrow_{T'} \cdots \Rightarrow_{T'} [r_n, f(u_n)] \Rightarrow_{T'} [q, w].$$

Since all leaves $u_1, \ldots, u_n$ are located below $x1$, the path connecting $f(u_i)$ to $f(u_{i+1})$ is identical to the one connecting u_i to u_{i+1} for $i = 1, \ldots, n-1$. It follows that the run from $[r_1, f(u_1)]$ to $[r_n, f(u_n)]$ is valid in T'. Only the first and last steps remain to be considered. We do only the first one, the other being time-symmetric. Consider the first step in the leaf run, when the automaton goes from $[p, v]$ to $[r_1, u_1]$. Since v is to the right of the pivot and u_1 is below the pivot, the path from v to u_1 is of the

form

$$\pi \ \curvearrowleft \ \pi_1\pi_2 \qquad \text{with} \quad \pi \in \{\nwarrow, \nearrow\}^* \quad \text{and } \pi_1, \pi_2 \in \{\swarrow, \searrow\}^*.$$

Here π_1 is chosen so that $\pi \curvearrowleft \pi_1$ leads from v to the pivot, while π_2 leads from the pivot to u_1. Since u_1 is at a distance at most $|Q|$ from v, it is separated by at most $|Q|$ leaves from the rightmost leaf below the pivot. Therefore, the only left turns $\swarrow$ in π_2 happen in its last $\log_2(|Q|)$ letters. It follows from condition 2 of Definition 10 that π_2 has a prefix $\searrow^k$ with $k \geq |Q|$. Hence some state r has to be used twice in the prefix. Since $\searrow$ is transitive, we deduce that

$$\pi \ \curvearrowleft \ \pi_1 \ \searrow \ \pi_2$$

also belongs to the move $S(p, r_1)$. But this path is the one linking v to $f(u_1)$ in T', proving that $[p, v] \Rightarrow_{T'} [r_1, f(u_1)]$.

The last remaining case is when both v and w are to the left of the pivot. This time we show that from a run of the form $\pi \curvearrowright \pi_1 \swarrow \pi_2$ one can deduce a run of the form $\pi \curvearrowright \pi_1\pi_2$, which can be used after the rotation. This case is, in fact, simpler since Definition 10 need not be invoked, but rather swallowing is used.

7.2.2. Leaf runs satisfying (*). In this case we will show that the component Γ contains a right-skipping move (or a right-teleport). After combining this with the left-teleports from our assumption, we will show that any two leaf configurations with states from Γ and nodes below the pivot can be reached one from the other, which implies that Γ cannot detect the rotation. For this, we will use the assumption that the pivot is below 01010101.

Recall the constant $E = 3D + |Q|$ defined after Definition 10.

LEMMA 7.6. *Either all pairs of states in* Γ *are* E*-full-teleports, or some states* r, r' *in* Γ *are right-skipping.*

Proof. Recall that we are analyzing a run as in (7.4). For $i = 0, \ldots, n$, let k_i be the offset $\#_T(u_i, u_{i+1})$. If some k_i exceeds 1, the corresponding move $R(r_i, r_{i+1})$ is by definition right-skipping and we are done. Furthermore, if some k_i exceeds the maximal move offset of r_i, r_{i+1} (if that exists), then the corresponding move is right-skipping by Lemma 6.11. We assume that neither case happens.

Since (*) is satisfied, there are $i < j$ such that $\#_T(u_i, u_j) \geq |Q|$. We will inspect the run from u_i to u_j and find in it a state used twice, the first configuration involved being to the left of the second one. Since $k_i, \ldots, k_{j-1} \leq 1$, we can assume that $\#_T(u_i, u_j)$ is exactly $|Q|$. Furthermore, if we choose i, j so that $j - i$ is minimal, all the leaves $u_{i+1}, \ldots, u_{j-1}$ are to the right of u_i and to the left of u_j. Since, without loss of generality, we can assume that no leaf is visited more than $|Q|$ times, and using the fact that the automaton has at least two states, we obtain $j - i < |Q|^2$.

Claim. Let $k = 1, \ldots, |Q|$. If $g < h$ in $\{i, \ldots, j\}$ are such that the sequence $r_g, \ldots, r_h$ contains at most k distinct states, and, furthermore, $\#_T(u_g, u_h) \geq k$, then there are $g' < h'$ in $\{g, \ldots, h\}$ such that $r_{g'} = r_{h'}$ and $\#_T(u_{g'}, u_{h'}) \geq 1$.

Proof. The proof is by induction on k. For $k = 1$ the statement is obvious. Consider now $k > 1$. We take a longest suffix $u_m, \ldots, u_h$ of $u_g, \ldots, u_h$ where the leaf u_g is not visited anymore (in particular, all leaves $u_m, \ldots, u_h$ are to the right of u_g). By assumption on all moves going at most one position to the right, the leaf u_m must be the leaf immediately to the right of u_g, and therefore $\#_T(u_m, u_h) \geq k - 1$. If the states $r_m, \ldots, r_h$ do not contain the state r_g, then we apply the induction hypothesis. Otherwise, the position among $m, \ldots, h$ where state r_g is used gives us the desired g' and h', since all nodes $u_m, \ldots, u_h$ are to the right of u_g. ∎

Using this claim with $g = i$ and $h = j$, together with $j - i < |Q|^2$, we find two indices $i' < j'$ such that $r_{i'} = r_{j'} = r$, $j' - i' < |Q|^2$, and $\#_T(u_{i'}, u_{j'}) \geq 1$. We will use this to show that there is also a right-teleport in the component. When combined with the left-teleport from our assumption on Γ, we will get a full-teleport.

By assumption on the values k_l, each pair r_l, r_{l+1} has some move offset $m_l \geq k_l$. (Here we use the convention that there is a move offset $m_l = 0 \geq k_l$ when the run from $[r_l, u_l]$ to $[r_{l+1}, u_{l+1}]$ is of type (b), i.e., when $r_l \circlearrowleft_{\mathbf{a}} r_{l+1}$ holds.) By Proposition 6.10 each nonempty move has a move offset in $\{-1, 0, 1\}$, and we have $m_l \geq -1$. Furthermore, if $m_l > 1$, then r_l, r_{l+1} would be right-skipping; hence we have $-1 \leq m_l \leq 1$. Therefore,

$$1 \leq \#_T(u_{i'}, u_{j'}) = k_{i'} + \cdots + k_{j'-1} \leq m_{i'} + \cdots + m_{j'-1} < |Q|^2.$$

(The last inequality is due to $j' - i' < |Q|^2$ and $m_l \leq 1$.) However, since the sum

$$m = m_{i'} + \cdots + m_{j'-1}$$

is composed only of move offsets, it must belong to $\mathrm{roff}_{|Q|^2}(r, r)$ and therefore also to $\mathrm{roff}_D(r, r)$, since

$$D = |Q|(|Q| + 1) > |Q|^2.$$

We will now show that $\mathrm{roff}_{3D}(r, r)$ contains all integers, and therefore r, r is a $3D$-full-teleport. Indeed, let v and w be two leaves in a tree t, both with safety margin at least $3D$. We will show that the automaton can go from $[r, v]$ to $[r, w]$. Using $m \in \mathrm{roff}_D(r, r)$ (recall that $m \geq 1$), starting from $[r, v]$, we successively move to the right by steps of m leaves. We stop as soon we have reached a configuration $[r, u]$ with u located at least D leaves to the right of w, possibly stopping immediately. Observe that if $u \neq v$ (i.e., u is to the right of v), then the leaf u is numbered at most $\#_T(w) + D + m - 1$, and thus u has at least $3D - (D + m - 1)$ leaves to its right. Since $m < D$, this value is larger than D. Hence, such a leaf u can indeed be reached using $m \in \mathrm{roff}_D(r, r)$. Since both u and v have safety margin at least $D > 2|Q|$, we can therefore use the $(2|Q|, D)$-left-teleport—that all state pairs in Γ have by assumption—to go from $[r, u]$ to $[r, w]$.

Once one state pair r, r has been shown to be a full-teleport, the same can be argued for the other state pairs in Γ. This is done in the same way as in the last part of the proof of Lemma 7.5. As in that lemma, the safety margin must be increased by $|Q|$; hence the value $E = 3D + |Q|$ in the statement of this lemma. □

LEMMA 7.7. *For v, w leaves below the pivot in T', and p, q in Γ, $[p, v] \Rightarrow_{T'} [q, w]$.*

Before we proceed with the proof, we show how this implies that Γ cannot detect the rotation. Indeed, consider the leaf run

$$[p, v] \Rightarrow_T [r_1, u_1] \Rightarrow_T \cdots \Rightarrow_T [r_n, u_n] \Rightarrow_T [q, w]$$

that goes from $[p, v]$ to $[q, w]$. As mentioned before, we assume that all $u_1, \ldots, u_n$ are below the pivot, and hence only the first and last moves cross the pivot. Thanks to the yet unproved Lemma 7.7, it suffices to connect $[p, v]$ with some leaf configuration in T' below the pivot and also connect some leaf configuration in T' below the pivot with $[q, w]$. We will therefore show that there are leaves u'_1 and u'_n in T' below the pivot such that

$$[p, v] \Rightarrow_{T'} [r_1, u'_1] \qquad \text{and} \qquad [r_n, u'_n] \Rightarrow_{T'} [q, w].$$

First consider the path from v to u_1 in T. This path first goes to the pivot, arriving there in some state s, and then moves down to the configuration $[r_1, u_1]$. This implies $s \downarrow r_1$. Hence, by the time-symmetric variant of Lemma 6.5, one of the sets $\swarrow^+$ or $\searrow^+$ is included in $D(s, r_1)$. In the first case we pick u'_1 to be the leftmost leaf below the pivot in T', while in the second case we take the rightmost one. The symmetric reasoning also works for u'_n.

Only the proof of Lemma 7.7 remains. Let v, w be leaves below the pivot, and let p, q belong to Γ. We need to show that $[p, v] \Rightarrow_{T'} [q, w]$ holds.

By Lemma 7.6, either all pairs of states r, r' in Γ are E-full-teleports, or there exists a pair of states r, r' in Γ that is right-skipping. In the first case, there are more than E leaves to the left and to the right of the pivot by (7.2) and hence to the left and to the right of both v and w. The E-full-teleport p, q is usable, and the statement of the lemma follows.

We now treat the case when some state pair r, r' in Γ is right-skipping. By assumption, all state pairs in the component Γ are $(2|Q|, D)$-left-teleports. Our strategy is to combine the left-teleports with the right-skipping move r, r'. First, we use the left-teleport to go from $[p, v]$ to $[r, u]$, with u being a specially chosen leaf to the left of the pivot. We then use the right-skipping move and the properties of u in order to move to $[r', u']$, with u' being a specially chosen leaf to the right of the pivot. Finally, we use the left-teleport to reach the configuration $[q, w]$. This process is illustrated in Figure 7.4.

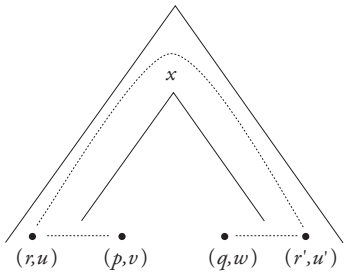


FIG. 7.4. *The leaf run witnessing* $[p, v] \Rightarrow_{T'} [q, w]$.

We need to find leaves u and u' such that the above strategy works. This is the goal of the claim below. The first property in the statement allows us to perform the right-skip, while the last three allow us to use the left-teleport.

Claim. There exist leaves u, u' in T' such that

- the path between u and u' belongs to $S(r, r')$ (r, r' taken from Lemma 7.6);
- there are at least D leaves between u and any node below the pivot;
- there are at least D leaves between any node below the pivot and u'; and
- there are at least $2|Q|$ leaves to the left and right of both u, u'.

Proof. The states r, r' are right-skipping by assumption. Hence, by definition of a right-skipping move and Lemma 6.3, one can find states s, s' with

$$r \uparrow s \curvearrowright s' \downarrow r'$$

such that either $U(r, s) \setminus \nwarrow^+$ or $D(s', r') \setminus \swarrow^+$ is nonempty. First consider the case when $U(r, s) \setminus \nwarrow^+$ is nonempty. This means that $r \nearrow s$ holds. Hence, by Lemma 6.5, $U(r, s)$ contains either $\nwarrow^* \nearrow$ or $\nearrow^+$, and $D(s', r')$ contains either $\swarrow^+$ or $\searrow^+$. By

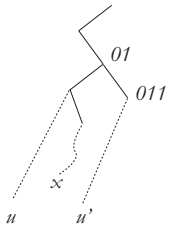


FIG. 7.5. *The move from u to u'.*

time-symmetry, in the case when $D(s', r') \setminus \swarrow^+$ is nonempty, $D(s', r')$ contains either $\searrow^+$ or $\searrow\swarrow^*$, and $U(r, s)$ contains either $\nwarrow^+$ or $\nearrow^+$.

Altogether, we obtain that $R(r, r')$ contains at least one of the following five moves (using swallowing, we have simplified $\nwarrow^*\nearrow$ into $\nwarrow^*$, and $\searrow\swarrow^*$ into $\swarrow^*$):

$$\nearrow^*\curvearrowright\swarrow^*, \quad \nwarrow^*\curvearrowright\searrow^*, \quad \nearrow^*\curvearrowright\searrow^*, \quad \nwarrow^*\nearrow\curvearrowright\swarrow^*, \quad \text{or } \nwarrow^*\curvearrowright\searrow\swarrow^*.$$

We treat each of these cases separately. For $\nearrow^*\curvearrowright\swarrow^*$, take u to be the leftmost leaf below 01 and u' to be the leftmost leaf below 011 (see Figure 7.5). Clearly the path from u to u' belongs to $\nearrow^*\curvearrowright\swarrow^*$; hence the first property of the statement holds. By changing the leaves u and u', we obtain similarly the other cases: for $\nwarrow^*\curvearrowright\searrow^*$, take u to be the rightmost leaf below 0100 and u' to be the rightmost leaf below 010; for $\nearrow^*\curvearrowright\searrow^*$, take u to be the leftmost leaf below 01 and u' to be the rightmost leaf below 010; for the case $\nwarrow^*\nearrow\curvearrowright\swarrow^*$, take u to be the rightmost leaf below 0100 and u' to be the leftmost leaf below 011; for $\nwarrow^*\curvearrowright\searrow\swarrow^*$, take u to be the rightmost leaf below 0100 and u' to be the leftmost leaf below 01011. In each case, the path from u to u' belongs to $S(r, r')$, and therefore the first item of the statement is satisfied.

For the second item of the statement, note that in all five cases above, the leaf u is located below 0100. Using condition 3 of Definition 10, we also know that the pivot is below the node 01010101. Hence, all the leaves below 010100 are to the right of u and to the left of the pivot. Furthermore, by (7.2), there are more than D such leaves. The third item is similar: in all five cases, u' is below 011 or 01011. Hence, the leaves below 0101011 are to the right of the pivot and to the left of u'. And there are more than D of them.

The fourth point is obtained by the same kind of arguments. The leaves below 00 are all to the left of u, and the leaves below 1 are all to the right of u'. And in each case there are more than $2|Q|$ of them. This completes the proof of the claim. □

This completes the proof that no component with a shift can detect the rotation.

7.3. Components without a shift cannot detect the rotation. In this section we consider a component Γ without shifts. This is the second and last case to be considered in the proof of Proposition 7.1. According to Proposition 6.10, every nonempty move $S(p, q)$ with p, q in Γ is a union of the elementary moves

$$\textit{Stay}, \ \sqcup\!\!\downarrow, \ \downarrow\!\sqcup, \ \sqcap\!\!\downarrow, \ \downarrow\!\sqcap, \ \sqcup\!\!\uparrow, \ \uparrow\!\sqcup, \ \sqcap\!\!\uparrow, \ \uparrow\!\sqcap, \ \sqcup\!\sqcap, \text{ and } \sqcap\!\sqcup$$

(see Figure 6.1). Our strategy is as follows. First, we distinguish some elementary moves, called "adjacency moves." Then we show that all other moves can be simulated using adjacency moves. Finally, we show that a component where all moves are adjacency moves cannot detect the rotation.

Two paths in Right are called *right adjacency similar* if one can be obtained from the other by replacing one fragment in $\sqcup\!\!\rightarrow$ by another. More formally, two paths are right adjacency similar if they can be decomposed as

$$y \nwarrow^k \curvearrowright \swarrow^l z \quad \text{and} \quad y \nwarrow^m \curvearrowright \swarrow^n z, \quad \text{where } k,l,m,n \in \mathbb{N},\ y \in \mathrm{Up} + \varepsilon,\ z \in \mathrm{Down} + \varepsilon.$$

Left adjacency similarity is defined in the same way by replacing $\nwarrow$, $\curvearrowright$, and $\swarrow$ by $\nearrow$, $\curvearrowleft$, and $\searrow$, respectively. Two paths are *adjacency similar* if they are either left or right adjacency similar.

Definition 12. *An* adjacency move *is an elementary move closed under adjacency similarity.*

The following simple fact is given without further proof.

Fact 7.8. *Stay,* $\sqcup\!\!\rightarrow$, $\leftarrow\!\!\sqcup$, $\sqcap\!\!\rightarrow$, $\leftarrow\!\!\sqcap$, $\sqcup\!\!\downarrow$, *and* $\downarrow\!\!\sqcup$ *are adjacency moves.*

Adjacency moves are going to be used in conjunction with fractality (see condition 1 of Definition 10). The following lemma presents a typical example of such an argument (fractality is not explicitly mentioned, but the lemma refers to characteristic types and by consequence can be used with fractality).

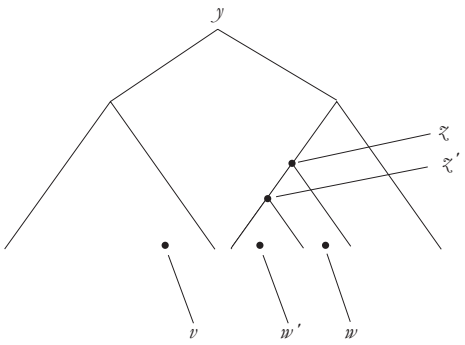


Fig. 7.6. *The nodes* y, v, z, z', w, *and* w'.

Lemma 7.9. *Fix a blank tree* t, *a node* y, *and two nodes* z, z' *on the leftmost branch below* $y1$. *Furthermore, let* w *be a leaf in the subtree of* z *and let* w' *be a leaf in the subtree of* z', *both with the same characteristic types within the subtrees of* z *and* z', *respectively (see Figure* 7.6*). Then, for any given leaf* v *below* $y0$ *and any adjacency move* M,

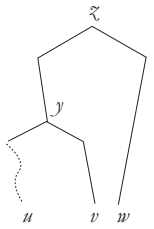
$$\textit{if} \quad vMw, \quad \textit{then} \quad vMw'.$$

Proof. Assume vMw. Since M is an adjacency move and hence also an elementary move, it is of the form $U \curvearrowright D$ with $\pi(y1, w) \in D$. It is sufficient to prove that $\pi(y1, w') \in D$ also holds. Since M is an adjacency move, D is of the form either $\swarrow^*$ or $(\swarrow + \searrow)^*$. If D is $\swarrow^*$, this means that w is the leftmost leaf below $y1$ and consequently also the leftmost leaf below z. Since w' has the same characteristic type (w.r.t. z') as w (w.r.t. z), this means that w' is the leftmost leaf below z'. By consequence $w = w'$, and we have $\pi(y1, w') \in D$. Otherwise D is of the form $(\swarrow + \searrow)^*$. Since w' is below $y1$, this implies $\pi(y1, w') \in D$. □

We will now eliminate the moves $\sqcap\!\!\rightarrow$, $\leftarrow\!\!\sqcap$, $\sqcap\!\!\downarrow$, and $\downarrow\!\!\sqcap$, which are not adjacency moves. This is done by simulating them by a sequence of adjacency moves. The following lemma treats the case of $\sqcap\!\!\rightarrow$, which is simulated by $\sqcap\!\!\rightarrow \leftarrow\!\!\sqcap$. The other cases are symmetric.

LEMMA 7.10. *Let t be a tree and let u, v be two leaves of t. If $u \sqcap\!\!\downarrow v$ and v is not the rightmost leaf of t, then there exists a leaf w such that $u \sqcap\!\!\rightarrow w \sqsubset\!\!\downarrow v$. If there exists a leaf w such that $u \sqcap\!\!\rightarrow w \sqsubset\!\!\downarrow v$, then $u \sqcap\!\!\downarrow v$.*

Proof. We prove here only the first implication; both implications can be seen in the following picture:



Let w be the next leaf in t after v. Since v is not the rightmost leaf, w exists. By the choice of w, we have $w \sqsubset\!\!\downarrow v$. Let us show $u \sqcap\!\!\rightarrow w$.

Let y be the deepest node above both u and v. Since $u \sqcap\!\!\downarrow v$ holds, v is the rightmost leaf below y. Let z be the deepest node above both y and w. Since w is the next leaf after v—which is the rightmost leaf below y—the leaf w is the leftmost one below $z1$. But then we can use $\sqcap\!\!\rightarrow$ to go from u to w (the appropriate path doing a $\curvearrowright$ from $z0$ to $z1$). □

In the remainder of the proof, we use only the fact that for each $p, q \in \Gamma$ the move $S(p, q)$ is a union of elementary moves. We then use Lemma 7.10 in the following manner. We enrich the automaton by allowing (after a nondeterministic choice) each move $\sqcap\!\!\downarrow$ to be possibly replaced by the sequence of $\sqcap\!\!\rightarrow$ followed by $\sqsubset\!\!\downarrow$, likewise for the other moves $\sqcup\!\!\downarrow$, $\sqcup\!\!\rightarrow$, and $\sqcup\!\!\uparrow$, by using time-, space-, and time-space-symmetric variants of this operation. (Note that this transformation requires the use of extra states.) According to the second implication of Lemma 7.10, the resulting automaton is equivalent to $\mathcal{A}$. Furthermore, any unrooted leaf run of the original automaton that uses only states from Γ can be transformed—using the first implication of Lemma 7.10—into an unrooted leaf run of the modified automaton where all moves happening below the pivot (i.e., the source and target leaves of the move are below the pivot) are adjacency moves. For this reason, from now on, we assume that all moves happening below the pivot are adjacency moves.

We proceed to show that Γ cannot detect the rotation. We have to show that

$$[p, v] \Rightarrow_T [q, w] \qquad \text{implies} \qquad [p, v] \Rightarrow_{T'} [q, w]$$

for any states $p, q \in \Gamma$ and nodes v, w not below the pivot. As before (in section 7.2), since T and T' are equal over nodes not below the pivot, it suffices to establish the lemma for unrooted leaf runs where all positions but the initial and final ones are below the pivot. In other words, the first move of the unrooted leaf run is used to enter the subtree of the pivot, the last move is used to exit it, and in between all moves are below the pivot. In this run all moves but the initial and final ones are used between leaf configurations below the pivot. Hence, according to the comment above, we can assume that all the moves used in this unrooted leaf run are adjacency moves, except possibly the first and last ones.

Recall from section 7.2.1 the bijection f that assigns to every leaf in T a leaf in T' and preserves the numbering. Let V_1, V_2, and V_3 be the sets of leaves of T below $x00$, $x01$, and $x1$, respectively. Let W_1, W_2, and W_3 be the sets of leaves of T' below

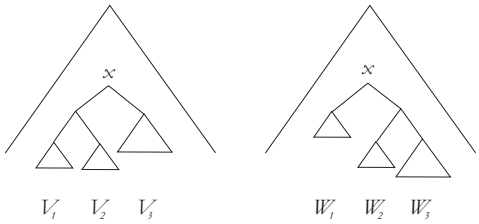


FIG. 7.7. *The trees T and T'.*

$x0$, $x10$, and $x11$, respectively (see Figure 7.7 for an illustration). By definition of rotation, $f(V_i) = W_i$ for $i = 1, 2, 3$. We say that two leaves $v \in V_i$ and $w \in V_j$ are *neighbors* if $|i-j| \leq 1$. If v, w are neighbors, then the path linking v to w and the path linking $f(v)$ to $f(w)$ are adjacency similar. In particular, whenever the automaton can go from v to w in one adjacency move, then it also can do this from $f(v)$ to $f(w)$. Therefore, a leaf run that does only moves between neighbor nodes is mapped by f onto a valid leaf run in the tree T'.

This remark is the key to our proof. The idea is that we will transform the run from $[p, v]$ to $[q, w]$ into one where all moves below the pivot are between neighbor leaves. Let the leaf run corresponding to $[p, v] \Rightarrow_T [q, w]$ be

$$[p, v] = [r_0, u_0], [r_1, u_1], \ldots, [r_n, u_n], [r_{n+1}, u_{n+1}] = [q, w].$$

We will transform it into the following leaf run in T':

$$[p, v] = [r_0, u_0], [r_1, f(u_1)], \ldots, [r_n, f(u_n)], [r_{n+1}, u_{n+1}] = [q, w].$$

For this construction to work, i.e., for this sequence to be a valid leaf run over T', it is sufficient to verify that the step from $[r_0, u_0]$ to $[r_1, u_1]$ can be transformed into a step from $[r_0, u_0]$ to $[r_1, f(u_1)]$ (similarly for the last step) and, furthermore, that the following property (*) holds: for every $1 \leq i < n$, the leaves u_i and u_{i+1} are neighbors. We first establish the first property; then we will show that every run can be transformed into one where (*) holds. (Property (*) may not hold for the original run.)

First step of the leaf run. The move from $[r_0, u_0]$ to $[r_1, u_1]$ is an elementary move, though perhaps not an adjacency move. Hence, there are three ways of entering the subtree of the pivot: by going to the leftmost leaf below the pivot (using one of $\sqcap\!\!\downarrow$, $\sqcap\!\!\rightarrow$, $\sqcap\!\!\circlearrowright$), to the rightmost one (using one of $\sqcap\!\!\downarrow$, $\sqcap\!\!\leftarrow$, $\sqcap\!\!\circlearrowleft$), or anywhere (using one of $\sqcap\!\!\downarrow$, $\sqcap\!\!\downarrow$, $\sqcap\!\!\downarrow$, $\sqcap\!\!\downarrow$). In each of those cases, using the same argument as in Lemma 7.9, it can easily be shown that the same move goes from $[r_0, u_0]$ to $[r_1, f(u_1)]$. The proof for the last step of the leaf run is the same.

Proof of (*). According to the remark above, there are three ways to enter the subtree of the pivot. By time symmetry, there are also three ways to exit this subtree. This results in nine possibilities.

Case leftmost-leftmost. Consider first the case where the automaton enters in the leftmost node of V_1 and leaves by the same node. This means that $u_1 = u_n$ is the leftmost leaf below x.

Since the subtree of the pivot is fractal (condition 1 of Definition 10), it contains a proper subtree that simulates it. Since all subtrees of T are complete binary trees, we may as well assume that there is a node u on the leftmost branch below $x0$ whose

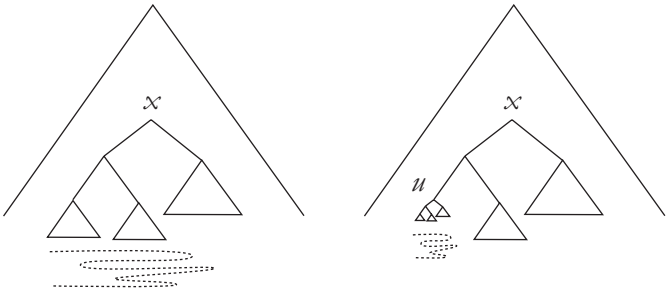


FIG. 7.8. *Moving the leaf run to* V_1.

subtree simulates the subtree of x. Since the leftmost node is one of the characteristic types (from the definition of fractality), the leaf run that went from the leftmost node below the pivot back to this leftmost node can be assumed to visit only nodes below u (see Figure 7.8). Such a leaf run satisfies property (*), since it never leaves $V_1 \cup V_2$.

All other cases are solved using the same argument, except for two: when the automaton enters in the leftmost leaf below the pivot and leaves in the rightmost one, and when the automaton enters in the rightmost leaf below the pivot and leaves in the leftmost one. The first of these is treated in the next item, and the other follows by time symmetry.

Case leftmost-rightmost. This time u_1 is the leftmost leaf of V_1 and u_n the rightmost leaf of V_3. We are going to construct a similar leaf run satisfying (*).

As an intermediate step, we first construct a similar leaf run satisfying the different property (#): once a position in V_3 is encountered during the run, no position in V_1 is visited anymore. Let us prove that we can transform the unrooted leaf run into one that satisfies (#). Let $1 < i \leq j < n$ be positions in the run that witness a violation of (#): the leaves u_{i-1} and u_{j+1} belong to V_3, the leaves $u_i, \ldots, u_j$ belong to $V_1 \cup V_2$, and at least one of $u_i, \ldots, u_j$ belongs to V_1. We will replace this violating subrun with one that does not visit V_1. By iterating this operation, we get a run where property (#) is satisfied.

By condition 1 of Definition 10, we can find on the rightmost branch below $x01$ a node u such that the subtree of u simulates the subtree of $x0$. Hence, one can find leaves u'_i, u'_j below u such that the characteristic type of u_i (resp., u_j) in the subtree of $x0$ is the same as the characteristic type of u'_i (resp., u'_j) in the subtree of u, and there is a run from $[r_i, u'_i]$ to $[r_j, u'_j]$ that uses only leaves below u. By choice of u, all these leaves are in V_2. Therefore, in order to remove the violation of (#) witnessed by i and j, it remains for us to connect $[r_{i-1}, u_{i-1}]$ with $[r_i, u'_i]$ and $[r_j, u'_j]$ with $[r_{j+1}, u_{j+1}]$. This is a direct application of Lemma 7.9 (and its time- and space-reverse variants); note that $[r_{i-1}, u_{i-1}] \to [r_i, u_i]$ and hence $u_{i-1} M u_i$ for some adjacency move $M \subseteq S(r_{i-1}, r_i)$.

Thanks to the above argument, we may assume that the leaf run satisfies property (#). If it already has property (*), then the problem is over. Otherwise, there is some moment in the leaf run where two consecutive leaf configurations are not neighboring. Since after visiting V_3 we never come back to V_1, this can happen at most once, where the source configuration $[r_i, u_i]$ is in V_1 and the target configuration $[r_{i+1}, u_{i+1}]$ is in V_3. Moreover, the only way to go from a position in V_1 to a position in V_3 via an adjacency move is by using $\sqcap\!\!\searrow$. In particular, u_{i+1} is the leftmost leaf

in V_3. However, if we want to use the move $\sqcap\!\!\searrow$ from $[r_i, u_i]$ and satisfy property (*), the only place we can go to is the leftmost leaf of V_2.

In order to complete the proof, we will construct a new leaf run that satisfies property (*) and goes from state r_{i+1} in the leftmost leaf of V_2 to state r_i in some leaf u of V_2. Then we can reuse the move $r_i \sqcap\!\!\searrow r_{i+1}$ to go from u to u_{i+1}, since $\sqcap\!\!\searrow$ does not care about the position of the leaf u within $V_1 \cup V_2$.

The leaf run that goes from r_{i+1} in the leftmost leaf of V_2—call this leaf v—to r_i in some leaf u of V_2 is constructed in two stages. In the first stage, we show that there is some leaf u' in V_2 such that the configuration $[r_n, u']$ can be reached from $[r_{i+1}, v]$. Furthermore, the leaf u' has at least $|Q|$ leaves to the left and to the right that belong to V_2. In the second stage, we use this latter assumption to go from $[r_n, u']$ to state r_i without leaving V_2. This is done simply by using Lemma 7.4 (observe that this lemma does not use any assumption on the component). The node where we arrive is going to be u.

Therefore, in order to complete the proof of Proposition 7.1—and therefore also the proof of Theorem 2—it remains to find some leaf u' in V_2 such that the configuration $[r_n, u']$ can be reached from $[r_{i+1}, v]$ and such that u' has $|Q|$ leaves to the left and right inside V_2. This process is illustrated in Figure 7.9.

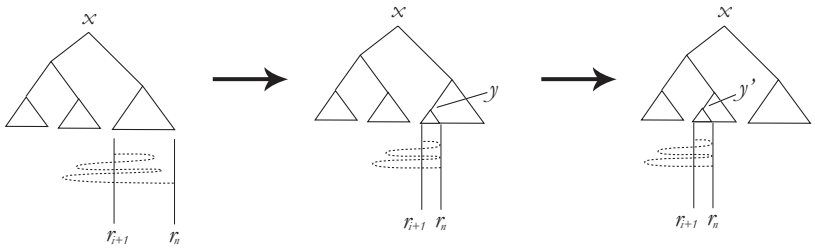


FIG. 7.9. *Transforming the run from r_{i+1} to r_n.*

Recall our assumption that the nodes $u_{i+1}, \ldots, u_n$ are all in $V_2 \cup V_3$. By condition 1 of Definition 10, there is a node y on the leftmost branch below $x10$ whose subtree simulates the subtree of $x1$ and has more than $|Q|$ leaves. Using the same proof as for property (#), we can transform the unrooted leaf run from $[r_{i+1}, u_{i+1}]$ to $[r_n, u_n]$ into an unrooted leaf run from $[r_{i+1}, u_{i+1}]$ to $[r_n, u'_n]$, where only leaves from V_2 or below y are used, and where u'_n is the rightmost leaf below y. (This run is illustrated in the middle picture of Figure 7.9.) Let y' be the node on the leftmost branch below $x01$ such that the subtree of y' is the same as the subtree of y. By using the properties of adjacency moves, one can shift—by $|V_2|$ leaves to the left—the run that goes from $[r_{i+1}, u_{i+1}]$ to $[r_n, u'_n]$ into a run that goes from $[r_{i+1}, v]$ to $[r_n, u']$, where u' is the rightmost leaf below y', which concludes the proof of Proposition 7.1. Note that in V_2, there are more than $|Q|$ leaves to the left and to the right of u', by construction of y.

Acknowledgments. We would like to thank the anonymous referees for their valuable comments. Special thanks are due to one of the referees, whose detailed and insightful comments totalled over fifty pages, all of them valid and used here. The first author's work was undertaken at LIAFA (Paris, France) and at Warsaw University (Poland). The second author's work was undertaken at Warsaw University (Poland) and IRISA/CNRS (Rennes, France).

REFERENCES

[1] A. V. Aho and J. D. Ullman, *Translations on a context-free grammar*, Inform. and Control, 19 (1971), pp. 439–475.

[2] M. Bojańczyk, 1-*bounded TWA cannot be determinized*, in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 2914, Springer-Verlag, Berlin, 2003, pp. 62–73.

[3] M. Bojańczyk and T. Colcombet, *Tree-walking automata cannot be determinized*, Theoret. Comput. Sci., 350 (2006), pp. 164–173.

[4] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, *Tree Automata Techniques and Applications*, http://www.grappa.univ-lille3.fr/tata (October 1, 2002).

[5] J. Engelfriet and H. J. Hoogeboom, *Tree-walking pebble automata*, in Jewels Are Forever, Contributions to Theoretical Computer Science in Honor of Arto Salomaa, J. Karhumaki, H. Maurer, G. Paun, and G. Rozenberg, eds., Springer-Verlag, Berlin, 1999, pp. 72–83.

[6] J. Engelfriet and H. J. Hoogeboom, *Automata with nested pebbles capture first-order logic with transitive closure*, Log. Methods Comput. Sci., 3 (2) (2007), paper 3.

[7] J. Engelfriet, H. J. Hoogeboom, and J. P. Van Best, *Trips on trees*, Acta Cybernet., 14 (1999), pp. 51–64.

[8] J. Engelfriet, G. Rosenberg, and G. Slutzki, *Tree transducers, L systems, and two-way machines*, J. Comput. System Sci., 20 (1980), pp. 150–202.

[9] T. Kamimura and G. Slutzki, *Parallel two-way automata on directed ordered acyclic graphs*, Inform. and Control, 49 (1981), pp. 10–51.

[10] A. Muscholl, M. Samuelides, and L. Segoufin, *Complementing deterministic tree-walking automata*, Inform. Process. Lett., 99 (2006), pp. 33–39.

[11] F. Neven and T. Schwentick, *On the power of tree-walking automata*, Inform. and Comput., 183 (2003), pp. 86–103.

[12] A. Potthoff, *Logische Klassifizierung regulärer Baumsprachen*, Ph.D. thesis, Institut für Informatik und Praktische Mathematik, Universität Kiel, Kiel, Germany, 1994.

SIAM J. COMPUT.
Vol. 38, No. 2, pp. 702–752

NEW AND IMPROVED CONSTRUCTIONS OF NONMALLEABLE CRYPTOGRAPHIC PROTOCOLS*

RAFAEL PASS† AND ALON ROSEN‡

Abstract. We present a new constant-round protocol for nonmalleable zero-knowledge. Using this protocol as a subroutine, we obtain a new constant-round protocol for nonmalleable commitments. Our constructions rely on the existence of (standard) collision-resistant hash functions. Previous constructions either relied on the existence of trapdoor permutations and hash functions that are collision resistant against subexponential-sized circuits or required a superconstant number of rounds. Additional results are the first construction of a nonmalleable commitment scheme that is statistically hiding (with respect to opening) and the first nonmalleable commitments that satisfy a strict polynomial-time simulation requirement. Our approach differs from the approaches taken in previous works in that we view nonmalleable zero-knowledge as a building block rather than an end goal. This gives rise to a modular construction of nonmalleable commitments and results in a somewhat simpler analysis.

Key words. cryptography, zero-knowledge, nonmalleability, man-in-the-middle, round-complexity, nonblack-box simulation

AMS subject classification. 94A60

DOI. 10.1137/060671553

1. Introduction.

Consider the execution of two-party protocols in the presence of an adversary that has full control of the communication channel between the parties. The adversary has the power to omit, insert, or modify messages at its choice. It also has full control over the scheduling of the messages. The honest parties are not necessarily aware of the existence of the adversary and are not allowed to use any kind of trusted setup (such as a common reference string).

The above kind of attack is often referred to as a *man-in-the-middle* attack. It models a natural scenario whose investigation is well motivated. Protocols that retain their security properties in the face of a man in the middle are said to be *nonmalleable* [14]. Due to the hostile environment in which they operate, the design and analysis of nonmalleable protocols is a notoriously difficult task. The task becomes even more challenging if the honest parties are not allowed to use any kind of trusted setup. Indeed, only a handful of such protocols have been constructed so far.

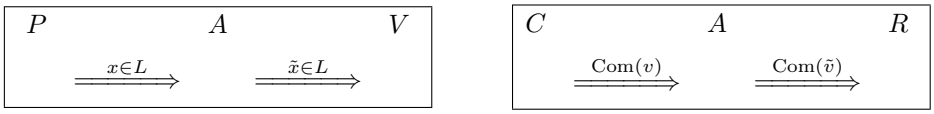
The rigorous treatment of two-party protocols in the man-in-the-middle setting has been initiated in the seminal paper by Dolev, Dwork, and Naor [14]. The paper contains definitions of security for the tasks of nonmalleable commitment and nonmalleable zero-knowledge. It also presents protocols that meet these definitions. The protocols rely on the existence of one-way functions, and require $O(\log n)$ rounds of interaction, where $n \in N$ is a security parameter.

*Received by the editors October 5, 2006; accepted for publication (in revised form) February 1, 2008; published electronically May 23, 2008. A preliminary version appeared in *Proceedings of the 37th ACM Symposium on Theory of Computing*, 2005, pp. 533–542.
http://www.siam.org/journals/sicomp/38-2/67155.html
†Department of Computer Science, Cornell University, Ithaca, NY 14853 (rafael@cs.cornell.edu). Part of this author's work was done while at CSAIL, MIT, Cambridge, MA.
‡Center for Research on Computation and Society (CRCS), DEAS, Harvard University, Cambridge, MA 02138 (alon@eecs.harvard.edu). Part of this author's work was done while at CSAIL, MIT, Cambridge, MA.

A more recent result by Barak presents constant-round protocols for nonmalleable commitment and nonmalleable zero-knowledge [2]. This is achieved by constructing a coin-tossing protocol that is secure against a man in the middle and then using the outcome of this protocol to instantiate known constructions for nonmalleable commitment and zero-knowledge in the common reference string model. The proof of security makes use of nonblack-box techniques and is highly complex. It relies on the existence of trapdoor permutations and hash functions that are collision resistant against subexponential-sized circuits.

In this paper we continue the line of research initiated by the above papers. We will be interested in constructions of new constant-round protocols for nonmalleable commitment and nonmalleable zero-knowledge and will not rely on any kind of set-up assumption.

1.1. Nonmalleable protocols. In accordance with the above discussion, consider a man-in-the-middle adversary A that is simultaneously participating in two executions of a two-party protocol. These executions are called the *left* and the *right* interaction. Besides controlling the messages that it sends in the left and right interactions, A has control over the scheduling of the messages. In particular, it may delay the transmission of a message in one interaction until it receives a message (or even multiple messages) in the other interaction.



(a) (b)

FIG. 1. *The man-in-the-middle adversary.* (a) *Interactive proofs.* (b) *Commitments.*

The adversary is trying to take advantage of its participation in the left interaction in order to violate the security of the protocol executed in the right interaction, where the exact interpretation of the term "violate the security" depends on the specific task at hand.

A two-party protocol is said to be nonmalleable if the left interaction does not "help" the adversary in violating the security of the right interaction. Following the simulation paradigm [24, 25, 21, 22], this is formalized by defining appropriate "real" and "idealized" executions.

In the real execution, called the *man-in-the-middle* execution, the adversary participates in both the left and the right interactions. In the idealized execution, called the *stand-alone* execution, the adversary is participating only in a single interaction. Security is defined by requiring that the adversary cannot succeed better in the man-in-the-middle execution than he could have in the stand-alone execution. In the specific instances of zero-knowledge and string commitment, the definition of security takes the following forms.

Nonmalleable zero-knowledge [14]. Let $\langle P, V\rangle$ be an interactive proof system. In the left interaction the adversary A is verifying the validity of a statement x by interacting with an honest prover P. In the right interaction A proves the validity of a statement $\tilde{x} \neq x$ to the honest verifier V (see Figure 1(a)). The objective of the adversary is to convince the verifier in the right interaction that $\tilde{x} \in L$. Nonmalleability of $\langle P, V\rangle$ is defined by requiring that, for any man-in-the-middle adversary A, there exists a stand-alone prover S that manages to convince the verifier with essentially

the same probability as A. The interactive proof $\langle P, V\rangle$ is said to be *nonmalleable zero-knowledge* if it is nonmalleable and (stand-alone) zero-knowledge.

Nonmalleable commitments [14]. Let $\langle C, R\rangle$ be a commitment scheme. In the left interaction the adversary A is receiving a commitment to a value v from the committer C. In the right interaction A is sending a commitment to a value $\tilde{v}$ to the receiver R (see Figure 1(b)). The objective of the adversary is to succeed in committing in the right interaction to a value $\tilde{v} \neq v$ that satisfies $\mathcal{R}(v, \tilde{v}) = 1$ for some poly-time computable relation $\mathcal{R}$. Nonmalleability of $\langle C, R\rangle$ is defined by requiring that, for any man-in-the-middle adversary A, there exists a stand-alone committer S that manages to commit to the related $\tilde{v}$ with essentially the same probability as A.

Schemes that satisfy the above definition are said to be nonmalleable *with respect to commitment*. In a different variant, called nonmalleable commitment *with respect to opening* [17], the adversary is considered to have succeeded only if it manages to *decommit* to a related value $\tilde{v}$.

1.2. Our contributions. Our main result is the construction of a new constant-round protocol for nonmalleable $\mathcal{ZK}$. The proof of security relies on the existence of (ordinary) collision resistant hash functions and does not rely on any set-up assumption.

THEOREM 1 (nonmalleable $\mathcal{ZK}$). *Suppose that there exists a family of collision-resistant hash functions. Then there exists a constant-round nonmalleable $\mathcal{ZK}$ argument for every $L \in \mathcal{NP}$.*

Theorem 1 is established using the notion of *simulation extractability*. A protocol is said to be simulation extractable if, for any man-in-the-middle adversary A, there exists a simulator-extractor that can simulate the views of both the left and the right interactions for A while outputting a witness for the statement proved by A in the right interaction. Any protocol that is simulation extractable is also nonmalleable $\mathcal{ZK}$. The main reason for using simulation extractability (which is more technical in flavor than nonmalleability) is that it is easier to work with.

Using our new simulation-extractable protocols as a subroutine, we construct constant-round protocols for nonmalleable string commitment. One of our constructions achieves statistically binding commitments that are nonmalleable with respect to commitment, and the other achieves statistically hiding commitments that are nonmalleable with respect to opening.

THEOREM 2 (statistically binding nonmalleable commitment). *Suppose that there exists a family of collision-resistant hash functions. Then there exists a constant-round statistically binding commitment scheme that is nonmalleable with respect to commitment.*

THEOREM 3 (statistically hiding nonmalleable commitment). *Suppose that there exists a family of collision-resistant hash functions. Then there exists a constant-round statistically hiding commitment scheme that is nonmalleable with respect to opening.*

Underlying cryptographic assumptions. The main quantitative improvement of our construction over the constant-round protocols in [2] is in the underlying cryptographic assumption. Our constructions rely on the existence of ordinary collision-resistant hash functions. The protocols in [2] relied on the existence of both trapdoor permutations and hash functions that are collision resistant against subexponential-sized circuits. The constructions in [14] assumed only the existence of one-way functions but had a superconstant number of rounds.

Statistically hiding nonmalleable commitments. Theorem 3 gives the first construction of a nonmalleable commitment scheme that is statistically hiding and that

does not rely on set-up assumptions. We mention that the existence of collision-resistant hash functions is the weakest assumption currently known to imply constant-round statistically hiding commitment schemes (even those that are not of the nonmalleable kind) [33, 10].

Strict vs. liberal nonmalleability. The notion of nonmalleability that has been considered so far in all works allows the stand-alone adversary S to run in expected polynomial time. A stronger ("tighter") notion of security, called *strict nonmalleability* [14], requires S to run in strict polynomial time. In the context of strict nonmalleability, we have the following result.

THEOREM 4 (strict nonmalleability). *Suppose that there exists a family of collision-resistant hash functions. Then there exists a constant-round statistically binding commitment scheme that is strictly nonmalleable with respect to commitment.*

1.3. Techniques and new ideas. Our protocols rely on nonblack-box techniques used by Barak to obtain the constant-round public-coin $\mathcal{ZK}$ argument for $\mathcal{NP}$ [1] (in a setting where no man in the middle is considered). They are closely related to previous works by Pass [35] and Pass and Rosen [36] that appeared in the context of bounded-concurrent two-party and multiparty computation; in particular our protocols rely and further explore the technique from [35] of using *message lengths* to obtain nonmalleability. Our techniques are different from the ones used by Barak in the context of nonmalleable coin tossing [2].

The approach we follow in this work is fundamentally different from the approach used in [14]. Instead of viewing nonmalleable commitments as a tool for constructing nonmalleable $\mathcal{ZK}$ protocols, we reverse the roles and use nonmalleable $\mathcal{ZK}$ protocols in order to construct nonmalleable commitments. Our approach is also different from the one taken by [2], where a coin-tossing protocol is used to instantiate constructions that rely on the existence of a common reference string.

Our approach gives rise to a modular and natural construction of nonmalleable commitments. This construction emphasizes the role of nonmalleable $\mathcal{ZK}$ as a building block for other nonmalleable cryptographic primitives. In proving the security of our protocols, we introduce the notion of *simulation extractability*, which is a convenient form of nonmalleability (in particular, it enables a more modular construction of proofs). A generalization of simulation extractability, called one-many simulation extractability, has already been found to be useful in constructing commitment schemes that retain their nonmalleability properties even if executed concurrently an unbounded (polynomial) number of times [37].

In principle, our definitions of nonmalleability are compatible with the ones appearing in [14]. However, the presentation is more detailed and somewhat different (see section 3). Our definitional approach, as well as our construction of nonmalleable $\mathcal{ZK}$, highlights a distinction between the notions of nonmalleable interactive proofs and nonmalleable $\mathcal{ZK}$. This distinction was not present in the definitions given in [14].

1.4. Related work. Assuming the existence of a common random string, Di Crescenzo, Ishai, and Ostrovsky [13], and Di Crescenzo et al. [12] construct nonmalleable commitment schemes. Sahai [38] and De Santis et al. [11] construct a noninteractive nonmalleable $\mathcal{ZK}$ protocol under the same assumption. Fischlin and Fischlin [17] and Damgård and Groth [9] construct nonmalleable commitments assuming the existence of a common reference string. We note that the nonmalleable commitments constructed in [13] and [17] satisfy nonmalleability only with respect to opening [17]. Canetti and Fischlin [7] construct a universally composable commitment assuming a common random string. Universal composability implies nonmalleability.

However, it is impossible to construct universally composable commitments without making set-up assumptions [7].

Goldreich and Lindell [20] and Nguyen and Vadhan [34] consider the task of session-key generation in a setting where the honest parties share a password that is taken from a relatively small dictionary. Their protocols are designed having a man-in-the-middle adversary in mind and require only the usage of a "mild" set-up assumption (namely the existence of a "short" password).

1.5. Future and subsequent work. Our constructions (and even moreso the previous ones) are quite complex. A natural question is whether they can be simplified. A somewhat related question is whether nonblack-box techniques are necessary for achieving constant-round nonmalleable $\mathcal{ZK}$ or commitments. Our constructions rely on the existence of collision-resistant hash functions, whereas the nonconstant-round construction in [14] relies on the existence of one-way functions. We wonder whether the collision resistance assumption can be relaxed.

Another interesting question (which has already been addressed in subsequent work—see below) is whether it is possible to achieve nonmalleability under concurrent executions. The techniques used in this paper do not seem to extend to the (unbounded) concurrent case, and new ideas seem to be required. Advances in that direction might shed light on the issue of concurrent composition of general secure protocols.

In subsequent work [37], we show that (a close variant of) the commitments presented here will retain their nonmalleability even if executed concurrently an unbounded (polynomial) number of times. We note that besides using an almost identical protocol, the proof of this new result heavily relies on a generalization of simulation extractability (called "one-many" simulation extractability). This notion has proved itself very useful in the context of nonmalleability, and we believe that it will find more applications in scenarios where a man-in-the-middle adversary is involved. We additionally mention that the presentation of some of the results in this version of the paper incorporate simplification developed by us in [37].

2. Preliminaries.

2.1. Basic notation. We let N denote the set of all integers. For any integer $m \in N$, denote by $[m]$ the set $\{1, 2, \ldots, m\}$. For any $x \in \{0,1\}^*$, we let $|x|$ denote the size of x (i.e., the number of bits used in order to write it). For two machines M, A, we let $M^A(x)$ denote the output of machine M on input x and given oracle access to A. The term *negligible* is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from nonnegative integers to reals is called *negligible* if, for every constant $c > 0$ and all sufficiently large n, it holds that $\nu(n) < n^{-c}$.

2.2. Witness relations. We recall the definition of a witness relation for an $\mathcal{NP}$ language [18].

DEFINITION 2.1 (witness relation). *A* witness relation *for a language $L \in \mathcal{NP}$ is a binary relation R_L that is polynomially bounded and polynomial-time recognizable and characterizes L by*

$$L = \{x : \exists y \text{ so that } (x, y) \in R_L\}.$$

We say that y is a witness for the membership $x \in L$ if $(x, y) \in R_L$ (also denoted $R_L(x, y) = 1$). We will also let $R_L(x)$ denote the set of witnesses for the membership

$x \in L$, i.e.,

$$R_L(x) = \{y : (x, y) \in L\}.$$

In the following, we assume a fixed witness relation $R_L(x, y)$ for each language $L \in \mathcal{NP}$.

2.3. Probabilistic notation. Denote by $x \stackrel{R}{\leftarrow} X$ the process of uniformly choosing an element x in a set X. If $B(\cdot)$ is an event depending on the choice of $x \stackrel{R}{\leftarrow} X$, then $\Pr_{x \leftarrow X}[B(x)]$ (alternatively, $\Pr_x[B(x)]$) denotes the probability that $B(x)$ holds when x is chosen with probability $1/|X|$. Namely,

$$\Pr_{x \leftarrow X}[B(x)] = \sum_x \frac{1}{|X|} \cdot \chi(B(x)),$$

where χ is an indicator function so that $\chi(B) = 1$ if event B holds and equals zero otherwise. We denote by U_n the uniform distribution over the set $\{0, 1\}^n$.

2.4. Computational indistinguishability and statistical closeness. The following definition of (computational) indistinguishability originates in the seminal paper of Goldwasser and Micali [24].

Let X be a countable set of strings. A *probability ensemble indexed by* X is a sequence of random variables indexed by X. Namely, any $A = \{A_x\}_{x \in X}$ is a random variable indexed by X.

DEFINITION 2.2 ((computational) indistinguishability). *Let X and Y be countable sets. Two ensembles $\{A_{x,y}\}_{x \in X, y \in Y}$ and $\{B_{x,y}\}_{x \in X, y \in Y}$ are said to be* computationally indistinguishable over X *if, for every probabilistic "distinguishing" machine D whose running time is polynomial in its first input, there exists a negligible function $\nu(\cdot)$ so that for every $x \in X, y \in Y$*

$$|\Pr[D(x, y, A_{x,y}) = 1] - \Pr[D(x, y, B_{x,y}) = 1]| < \nu(|x|);$$

$\{A_{x,y}\}_{x \in X, y \in Y}$ and $\{B_{x,y}\}_{x \in X, y \in Y}$ are said to be statistically close *over X if the above condition holds for all (possibly unbounded) machines D.*

2.5. Interactive proofs, zero-knowledge, and witness indistinguishability. We use the standard definitions of interactive proofs (and interactive Turing machines) [25, 18] and arguments [6]. Given a pair of interactive Turing machines, P and V, we denote by $\langle P, V \rangle(x)$ the random variable representing the (local) output of V when interacting with machine P on common input x, when the random input to each machine is uniformly and independently chosen.

DEFINITION 2.3 (interactive proof system). *A pair of interactive machines $\langle P, V \rangle$ is called an interactive proof system for a language L if machine V is polynomial time and the following two conditions hold with respect to some negligible function $\nu(\cdot)$:*

- Completeness: *For every $x \in L$,*

$$\Pr[\langle P, V \rangle(x) = 1] \geq 1 - \nu(|x|).$$

- Soundness: *For every $x \notin L$, and every interactive machine B,*

$$\Pr[\langle B, V \rangle(x) = 1] \leq \nu(|x|).$$

In case that the soundness condition is required to hold only with respect to a computationally bounded prover, the pair $\langle P, V \rangle$ is called an interactive argument system.

Zero-knowledge. An interactive proof is said to be *zero-knowledge* ($\mathcal{ZK}$) if it yields nothing beyond the validity of the assertion being proved. This is formalized by requiring that the view of every probabilistic polynomial-time adversary V^* interacting with the honest prover P can be simulated by a probabilistic polynomial-time machine S (a.k.a. the *simulator*). The idea behind this definition is that whatever V^* might have learned from interacting with P, he could have actually learned by himself (by running the simulator S).

The notion of $\mathcal{ZK}$ was introduced by Goldwasser, Micali, and Rackoff [25]. To make $\mathcal{ZK}$ robust in the context of protocol composition, Goldreich and Oren [23] suggested augmenting the definition so that the above requirement also holds with respect to all $z \in \{0,1\}^*$, where both V^* and S are allowed to obtain z as auxiliary input. The verifier's *view* of an interaction consists of the common input x, followed by its random tape and the sequence of prover messages the verifier receives during the interaction. We denote by $\text{view}^P_{V^*}(x,z)$ a random variable describing $V^*(z)$'s view of the interaction with P on common input x.

DEFINITION 2.4 (zero-knowledge). *Let $\langle P,V\rangle$ be an interactive proof system. We say that $\langle P,V\rangle$ is zero-knowledge if, for every probabilistic polynomial-time interactive machine V^*, there exists a probabilistic polynomial-time algorithm S such that the ensembles $\{\text{view}^P_{V^*}(x,z)\}_{x\in L,z\in\{0,1\}^*}$ and $\{S(x,z)\}_{x\in L,z\in\{0,1\}^*}$ are computationally indistinguishable over L.*

A stronger variant of zero-knowledge is one in which the output of the simulator is statistically close to the verifier's view of real interactions. We focus on *argument* systems, in which the soundness property is guaranteed to hold only with respect to polynomial-time provers.

DEFINITION 2.5 (statistical zero-knowledge). *Let $\langle P,V\rangle$ be an interactive argument system. We say that $\langle P,V\rangle$ is statistical zero-knowledge if, for every probabilistic polynomial-time V^*, there exists a probabilistic polynomial-time S such that the ensembles $\{\text{view}^P_{V^*}(x,z)\}_{x\in L,z\in\{0,1\}^*}$ and $\{S(x,z)\}_{x\in L,z\in\{0,1\}^*}$ are statistically close over L.*

In case that the ensembles $\{\text{view}^P_{V^*}(x,z)\}_{x\in L,z\in\{0,1\}^*}$ and $\{S(x,z)\}_{x\in L,z\in\{0,1\}^*}$ are identically distributed, the protocol $\langle P,V\rangle$ is said to be *perfect* zero-knowledge.

Witness indistinguishability. An interactive proof is said to be *witness indistinguishable* ($\mathcal{WI}$) if the verifier's view is "computationally independent" (resp., "statistically independent") of the witness used by the prover for proving the statement (the notion of statistical $\mathcal{WI}$ will be used in section 4.2). In this context, we focus our attention on languages $L \in \mathcal{NP}$ with a corresponding witness relation R_L. Namely, we consider interactions in which on common input x the prover is given a witness in $R_L(x)$. By saying that the view is computationally (resp., statistically) independent of the witness, we mean that, for any two possible $\mathcal{NP}$ witnesses that could be used by the prover to prove the statement $x \in L$, the corresponding views are computationally (resp., statistically) indistinguishable.

Let V^* be a probabilistic polynomial-time adversary interacting with the prover, and let $\text{view}^P_{V^*}(x,w,z)$ denote V^*'s view of an interaction in which the witness used by the prover is w (where the common input is x and V^*'s auxiliary input is z).

DEFINITION 2.6 (witness indistinguishability). *Let $\langle P,V\rangle$ be an interactive proof system for a language $L \in \mathcal{NP}$. We say that $\langle P,V\rangle$ is witness indistinguishable for R_L if, for every probabilistic polynomial-time interactive machine V^* and for every two sequences $\{w^1_x\}_{x\in L}$ and $\{w^2_x\}_{x\in L}$ such that $w^1_x, w^2_x \in R_L(x)$ for every $x \in L$, the probability ensembles $\{\text{view}^P_{V^*}(x,w^1_x)\}_{x\in L,z\in\{0,1\}^*}$ and $\{\text{view}^P_{V^*}(x,w^2_x)\}_{x\in L,z\in\{0,1\}^*}$ are com-*

putationally indistinguishable over L.

When the ensembles $\{\text{view}^P_{V^*}(x, w^1_x)\}_{x\in L, z\in\{0,1\}^*}$ and $\{\text{view}^P_{V^*}(x, w^2_x)\}_{x\in L, z\in\{0,1\}^*}$ are statistically close over L, the proof system $\langle P, V\rangle$ is said to be *statistically witness indistinguishable*. If the ensembles are identically distributed, the proof system is said to be *witness independent*.

2.6. Proofs of knowledge. Informally an interactive proof is a proof of knowledge if the prover convinces the verifier not only of the validity of a statement but also that it possesses a witness for the statement. This notion is formalized by the introduction of an machine E, called a *knowledge extractor*. As the name suggests, the extractor E is supposed to extract a witness from any malicious prover P^* that succeeds in convincing an honest verifier. More formally, we have the following definition.

DEFINITION 2.7. *Let (P, V) be an interactive proof system for the language L with witness relation R_L. We say that (P, V) is a proof of knowledge if there exist a polynomial q and a probabilistic oracle machine E such that, for every probabilistic polynomial-time interactive machine P^*, there exists some negligible function $\mu(\cdot)$ such that, for every $x \in L$ and every $y, r \in \{0,1\}^*$ such that $\Pr[\langle P^*_{x,y,r}, V(x)\rangle = 1] > 0$, where $P^*_{x,y,r}$ denotes the machine P^* with common input fixed to x, auxiliary input fixed to y, and random tape fixed to r, the following hold:*

1. *The expected number of steps taken by $E^{P^*_{x,y,r}}$ is bounded by*

$$\frac{q(|x|)}{\Pr[\langle P^*_{x,y,r}, V(x)\rangle = 1]},$$

   where $E^{P^*_{x,y,r}}$ denotes the machine E with oracle access to $P^*_{x,y,r}$.
2. *Furthermore,*

$$\Pr[\langle P^*_{x,y,r}, V(x)\rangle = 1 \wedge E^{P^*_{x,y,r}} \notin R_L(x)] \le \mu(|x|).$$

The machine E is called a (knowledge) extractor.

We remark that as our definition considers only computationally bounded provers, we get only a "computationally convincing" notion of a proof of knowledge (a.k.a *argument of knowledge*) [6]. In addition, our definition is slightly different from the definition of [4] in that we require that the expected running time of the extractor is always bounded by $\text{poly}(|x|)/p$, where p denotes the success probability of P^*, whereas [4] allows for some additional slackness in the running time. On the other hand, whereas [4] requires the extractor to always output a valid witness, we instead allow the extractor to fail with some negligible probability. We will rely on the following theorem.

THEOREM 2.8 (see [5, 6]). *Assume the existence of claw-free permutations. Then there exists a constant-round public-coin witness independent argument of knowledge for $\mathcal{NP}$.*

Indeed, standard techniques can be used to show that the parallelized version of the protocol of [5], using perfectly hiding commitments, is an argument of knowledge (as defined above). As usual, the knowledge extractor E proceeds by feeding new "challenges" to the prover P^* until it gets two accepting transcripts. If the two accepting challenges contain the same challenge, or if the prover manages to open up a commitment in two different ways, the extractor outputs `fail`; otherwise it can extract a witness.

2.7. Universal arguments. Universal arguments (introduced in [3] and closely related to the notion of CS-proofs [30]) are used in order to provide "efficient" proofs to statements of the form $y = (M, x, t)$, where y is considered to be a true statement if M is a nondeterministic machine that accepts x within t steps. The corresponding language and witness relation are denoted $L_{\mathcal{U}}$ and $R_{\mathcal{U}}$, respectively, where the pair $((M, x, t), w)$ is in $R_{\mathcal{U}}$ if M (viewed here as a two-input deterministic machine) accepts the pair (x, w) within t steps. Notice that every language in $\mathcal{NP}$ is linear time reducible to $L_{\mathcal{U}}$. Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all $\mathcal{NP}$-statements. In fact, a proof system for $L_{\mathcal{U}}$ enables us to handle languages that are presumably "beyond" $\mathcal{NP}$, as the language $L_{\mathcal{U}}$ is $\mathcal{NE}$-complete (hence the name universal arguments).[1]

DEFINITION 2.9 (universal argument). *A pair of interactive Turing machines (P, V) is called a universal argument system if it satisfies the following properties:*

- Efficient verification: *There exists a polynomial p such that, for any $y = (M, x, t)$, the total time spent by the (probabilistic) verifier strategy V, on common input y, is at most $p(|y|)$. In particular, all messages exchanged in the protocol have length smaller than $p(|y|)$.*
- Completeness by a relatively efficient prover: *For every $((M, x, t), w)$ in $R_{\mathcal{U}}$,*

$$\Pr[(P(w), V)(M, x, t) = 1] = 1.$$

  Furthermore, there exists a polynomial q such that the total time spent by $P(w)$, on common input (M, x, t), is at most $q(T_M(x, w)) \leq q(t)$, where $T_M(x, w)$ denotes the running time of M on input (x, w).
- Computational soundness: *For every polynomial size circuit family $\{P_n^*\}_{n\in N}$, and every triplet $(M, x, t) \in \{0,1\}^n \setminus L_{\mathcal{U}}$,*

$$\Pr[(P_n^*, V)(M, x, t) = 1] < \nu(n),$$

  where $\nu(\cdot)$ is a negligible function.
- Weak proof of knowledge: *For every positive polynomial p there exist a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every polynomial-sized circuit family $\{P_n^*\}_{n\in N}$, and every sufficiently long $y = (M, x, t) \in \{0,1\}^*$, if $\Pr[(P_n^*, V)(y) = 1] > 1/p(|y|)$, then*

$$\Pr[\exists w = w_1, \ldots w_t \in R_{\mathcal{U}}(y) \text{ so that } \forall i \in [t],\ E_r^{P_n^*}(y, i) = w_i] > \frac{1}{p'(|y|)},$$

  where $R_{\mathcal{U}}(y) \stackrel{\text{def}}{=} \{w : (y, w) \in R_{\mathcal{U}}\}$ and $E_r^{P_n^*}(\cdot, \cdot)$ denotes the function defined by fixing the random tape of E to equal r and providing the resulting E_r with oracle access to P_n^*.

2.8. Commitment schemes. Commitment schemes are used to enable a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver* (this property is called *hiding*). Furthermore, the commitment is *binding*, and thus in a later stage, when the commitment is opened, it is guaranteed that the "opening" can yield only a single value determined in the committing phase. Commitment schemes come in two different flavors, *statistically binding* and *statistically hiding*. We sketch the properties of each one of these flavors. Full definitions can be found in [18].

[1] Furthermore, every language in $\mathcal{NEXP}$ is polynomial-time (but not linear-time) reducible to $L_{\mathcal{U}}$.

Statistically binding: In statistically binding commitments, the binding property holds against unbounded adversaries, while the hiding property holds only against computationally bounded (nonuniform) adversaries. The statistical-binding property asserts that, with overwhelming probability over the coin tosses of the receiver, the transcript of the interaction fully determines the value committed to by the sender. The computational-hiding property guarantees that the commitments to any two different values are computationally indistinguishable.

Statistically hiding: In statistically hiding commitments, the hiding property holds against unbounded adversaries, while the binding property holds only against computationally bounded (nonuniform) adversaries. Loosely speaking, the statistical-hiding property asserts that commitments to any two different values are statistically close (i.e., have negligible statistical distance). In case the statistical distance is 0, the commitments are said to be *perfectly hiding*. The computational-binding property guarantees that no polynomial-time machine is able to open a given commitment in two different ways.

Noninteractive statistically binding commitment schemes can be constructed using any 1-1 one-way function (see section 4.4.1 of [18]). Allowing some minimal interaction (in which the receiver first sends a single random initialization message), statistically binding commitment schemes can be obtained from any one-way function [31, 26]. We will think of such commitments as a *family* of noninteractive commitments, where the description of members in the family will be the initialization message. Perfectly hiding commitment schemes can be constructed from any one-way permutation [32]. However, *constant-round* schemes are known to exist only under stronger assumptions, specifically, assuming the existence of a collection of certified claw-free functions [19].

3. Nonmalleable protocols. The notion of nonmalleability was introduced by Dolev, Dwork, and Naor [14]. In this paper we focus on nonmalleability of zero-knowledge proofs and of string commitment. The definitions are stated in terms of interactive proofs, though what we actually construct are nonmalleable argument systems. The adaptation of the definitions to the case of arguments can be obtained by simply replacing the word "proof" with "argument," whenever it appears.

In principle, our definitions are compatible with the ones appearing in [14]. However, the presentation is more detailed and somewhat different (see section 3.4 for a discussion on the differences between our definition and previous ones).

3.1. Nonmalleable interactive proofs. Let $\langle P, V\rangle$ be an interactive proof. Consider a scenario where a man-in-the-middle adversary A is simultaneously participating in two interactions. These interactions are called the *left* and the *right* interaction. In the left interaction the adversary A is verifying the validity of a statement x by interacting with an honest prover P. In the right interaction A proves the validity of a statement $\tilde{x}$ to the honest verifier V. The statement $\tilde{x}$ is chosen by A, possibly depending on the messages it receives in the left interaction.

Besides controlling the messages sent by the verifier in the left interaction and by the prover in the right interaction, A has control over the scheduling of the messages. In particular, it may delay the transmission of a message in one interaction until it receives a message (or even multiple messages) in the other interaction. Figure 2 describes two representative scheduling strategies.

The interactive proof $\langle P, V\rangle$ is said to be *nonmalleable* if, whenever $x \neq \tilde{x}$, the left interaction does not "help" the adversary in convincing the verifier in the right

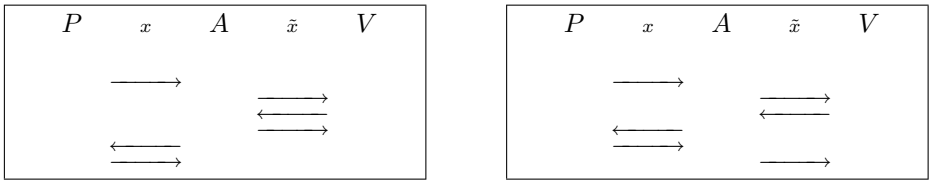


FIG. 2. *Two scheduling strategies.*

interaction.[2] Following the simulation paradigm [24, 25, 21], this is formalized by defining appropriate "real" and "idealized" executions. In the real execution, called the *man-in-the-middle* execution, the adversary participates in both the left and the right interactions with common inputs x and $\tilde{x}$, respectively. In the idealized execution, called the *stand-alone* execution, the adversary is playing the role of the prover in a single interaction with common input $\tilde{x}$. Security is defined by requiring that the adversary cannot succeed better in the man-in-the middle execution than he could have done in the stand-alone execution. More formally, we consider the following two executions.

Man-in-the-middle execution. The man-in-the-middle execution consists of the scenario described above. The input of P is an instance-witness pair (x, w), and the input of V is an instance $\tilde{x}$. A receives x and an auxiliary input z. Let $\mathsf{mim}_V^A(x, w, z)$ be a random variable describing the output of V in the above experiment when the random tapes of P, A, and V are uniformly and independently chosen. In case that $x = \tilde{x}$, the view $\mathsf{mim}_V^A(x, w, z)$ is defined to be $\perp$.

Stand-alone execution. In the stand-alone execution only one interaction takes place. The stand-alone adversary S interacts directly with the honest verifier V. As in the man-in-the-middle execution, V receives as input an instance $\tilde{x}$. S receives instances $x, \tilde{x}$ and auxiliary input z. Let $\mathsf{sta}_V^S(x, \tilde{x}, z)$ be a random variable describing the output of V in the above experiment when the random tapes of S and V are uniformly and independently chosen.

DEFINITION 3.1 (nonmalleable interactive proof). *An interactive proof $\langle P, V\rangle$ for a language L is said to be nonmalleable if, for every probabilistic polynomial-time man-in-the-middle adversary A, there exist a probabilistic expected polynomial-time stand-alone prover S and a negligible function $\nu : N \to N$ such that, for every $(x, w) \in L \times R_L(x)$, every $\tilde{x} \in \{0,1\}^{|x|}$ so that $\tilde{x} \neq x$, and every $z \in \{0,1\}^*$,*

$$\Pr\left[\mathsf{mim}_V^A(x, \tilde{x}, w, z) = 1\right] \;<\; \Pr\left[\mathsf{sta}_V^S(x, \tilde{x}, z) = 1\right] \;+\; \nu(|x|).$$

Nonmalleability with respect to tags. Definition 3.1 rules out the possibility that the statement proved on the right interaction is identical to the one on the left. Indeed, if the same protocol is executed on the left and on the right, this kind of attack cannot be prevented, as the man-in-the-middle adversary can always copy messages between the two executions (cf. the chess-master problem [14]). Still, in many situations it might be important to be protected against an attacker that attempts to prove even the same statement. In order to deal with this problem, one could instead consider a "tag-based" variant of nonmalleability (see [28] for a definition of tag-based nonmalleability in the context of encryption).

[2]Notice that requiring that $x \neq \tilde{x}$ is necessary, since otherwise the adversary can succeed in convincing the verifier in the right interaction by simply forwarding messages back and forth between the interactions.

We consider a family of interactive proofs, where each member of the family is labeled with a tag string $\text{TAG} \in \{0,1\}^m$, and $t = t(n)$ is a parameter that potentially depends on the length of the common input (security parameter) $n \in N$. As before, we consider a MIM adversary A that is simultaneously participating in a left and a right interaction. In the left interaction, A is verifying the validity of a statement x by interacting with a prover P_{TAG} while using a protocol that is labeled with a string TAG. In the right interaction A proves the validity of a statement $\tilde{x}$ to the honest verifier $V_{\text{T}\tilde{\text{A}}\text{G}}$ while using a protocol that is labeled with a string $\text{T}\tilde{\text{A}}\text{G}$. Let $\mathsf{mim}_V^A(\text{TAG}, \text{T}\tilde{\text{A}}\text{G}, x, \tilde{x}, w, z)$ be a random variable describing the output of V in the man-in-the-middle experiment. The stand-alone execution is defined as before with the only difference being that, in addition to their original inputs, the parties also obtain the corresponding tags. Let $\mathsf{sta}_V^S(\text{TAG}, \text{T}\tilde{\text{A}}\text{G}, x, \tilde{x}, z)$ be a random variable describing the output of V in the stand-alone experiment.

The definition of nonmalleability with respect to tags is essentially identical to Definition 3.1. The only differences in the definition is that, instead of requiring nonmalleability (which compares the success probability of $\mathsf{mim}_V^A(\text{TAG}, \text{T}\tilde{\text{A}}\text{G}, x, \tilde{x}, w, z)$ and $\mathsf{sta}_V^S(\text{TAG}, \text{T}\tilde{\text{A}}\text{G}, x, \tilde{x}, z)$) whenever $x \neq \tilde{x}$, we will require nonmalleability whenever $\text{TAG} \neq \text{T}\tilde{\text{A}}\text{G}$. For convenience, we repeat the definition.

DEFINITION 3.2 (tag-based nonmalleable interactive proofs). *A family of interactive proofs $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ for a language L is said to be nonmalleable with respect to tags of length m if, for every probabilistic polynomial-time man-in-the-middle adversary A, there exist a probabilistic expected polynomial-time stand-alone prover S and a negligible function $\nu : N \to N$ such that, for every $(x, w) \in L \times R_L(x)$, every $\tilde{x} \in \{0,1\}^{|x|}$, every $\text{TAG}, \text{T}\tilde{\text{A}}\text{G} \in \{0,1\}^m$ so that $\text{TAG} \neq \text{T}\tilde{\text{A}}\text{G}$, and every $z \in \{0,1\}^*$,*

$$\Pr\left[\mathsf{mim}_V^A(\text{TAG}, \text{T}\tilde{\text{A}}\text{G}, x, \tilde{x}, w, z) = 1\right] \;<\; \Pr\left[\mathsf{sta}_V^S(\text{TAG}, \text{T}\tilde{\text{A}}\text{G}, x, \tilde{x}, z) = 1\right] \;+\; \nu(|x|).$$

Tags vs. statements. A nonmalleable interactive proof can be turned into a tag-based one by simply concatenating the tag to the statement being proved. On the other hand, an interactive proof that is nonmalleable with respect to tags of length $t(n) = n$ can be turned into a nonmalleable interactive proof by using the statement $x \in \{0,1\}^n$ as the tag.

The problem of constructing a tag-based nonmalleable interactive proof is already nontrivial for tags of length, say $t(n) = O(\log n)$ (and even for $t(n) = O(1)$), but is still potentially easier than for tags of length n. This opens up the possibility of reducing the construction of interactive proofs that are nonmalleable with respect to long tags into interactive proofs that are nonmalleable with respect to shorter tags. Even though we do not know whether such a reduction is possible in general, our work follows this path and demonstrates that in specific cases such a reduction is indeed possible.

Nonmalleability with respect to other protocols. Our definitions of nonmalleability refer to protocols that are nonmalleable *with respect to themselves*, since the definitions consider a setting where the same protocol is executed in the left and the right interaction. In principle, one could consider two different protocols that are executed on the left and on the right which are nonmalleable *with respect to each other*. Such definitions are not considered in this work.

3.2. Nonmalleable zero-knowledge. Nonmalleable $\mathcal{ZK}$ proofs are nonmalleable interactive proofs that additionally satisfy the $\mathcal{ZK}$ property (as stated in Definition 2.4).

DEFINITION 3.3 (nonmalleable zero-knowledge). *A family* $\{\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle\}_{\text{TAG}\in\{0,1\}^*}$ *of interactive proofs is said to be nonmalleable zero-knowledge if it is both nonmalleable and zero-knowledge.*

3.3. Nonmalleable commitments. Informally, a commitment scheme is nonmalleable if a man-in-the-middle adversary that receives a commitment to a value v will not be able to "successfully" commit to a related value $\tilde{v}$. The literature discusses two different interpretations of the term "success."

Nonmalleability with respect to commitment [14]. The adversary is said to succeed if it manages to commit to a related value, even without being able to later decommit to this value. This notion makes sense only in the case of statistically binding commitments.

Nonmalleability with respect to opening [13]. The adversary is said to succeed only if it is able to both commit and decommit to a related value. This notion makes sense both in the case of statistically binding and statistically hiding commitments.

As in the case of nonmalleable zero-knowledge, we formalize the definition by comparing a man-in-the-middle and a stand-alone execution. Let $n \in N$ be a security parameter. Let $\langle C, R\rangle$ be a commitment scheme, and let $\mathcal{R} \subseteq \{0,1\}^n \times \{0,1\}^n$ be a polynomial-time computable irreflexive relation (i.e., $\mathcal{R}(v,v) = 0$). As before, we consider man-in-the-middle adversaries that are simultaneously participating in a left and a right interaction in which a commitment scheme is taking place. The adversary is said to succeed in mauling a left commitment to a value v if he is able to come up with a right commitment to a value $\tilde{v}$ such that $\mathcal{R}(v,\tilde{v}) = 1$. Since we cannot rule out copying, we will be interested only in relations where copying is not considered success, and we therefore require that the relation $\mathcal{R}$ is irreflexive. The man-in-the-middle and the stand-alone executions are defined as follows.

The man-in-the-middle execution. In the man-in-the-middle execution, the man-in-the-middle adversary A is simultaneously participating in a left and a right interaction. In the left interaction the man-in-the-middle adversary A interacts with C receiving a commitment to a value v. In the right interaction A interacts with R attempting to commit to a related value $\tilde{v}$. Prior to the interaction, the value v is given to C as local input. A receives an auxiliary input z, which in particular might contain a priori information about v.[3] The success of A is defined using the following two Boolean random variables:

- $\mathsf{mim}^A_{com}(\mathcal{R}, v, z) = 1$ if and only if A produces a valid commitment to $\tilde{v}$ such that $\mathcal{R}(v,\tilde{v}) = 1$.
- $\mathsf{mim}^A_{open}(\mathcal{R}, v, z) = 1$ if and only if A decommits to a value $\tilde{v}$ such that $\mathcal{R}(v,\tilde{v}) = 1$.

The stand-alone execution. In the stand-alone execution only one interaction takes place. The stand-alone adversary S interacts directly with R. As in the man-in-the-middle execution, the value v is chosen prior to the interaction and S receives some a priori information about v as part of its an auxiliary input z. S first executes the commitment phase with R. Once the commitment phase has been completed, S receives the value v and attempts to decommit to a value $\tilde{v}$. The success of S is defined using the following two Boolean random variables:

[3] The original definition by Dolev, Dwork, and Naor [14] accounted for such a priori information by providing the adversary with the value $\mathsf{hist}(v)$, where the function $\mathsf{hist}(\cdot)$ is a polynomial-time computable function.

- $\mathsf{sta}^S_{com}(\mathcal{R}, v, z) = 1$ if and only if S produces a valid commitment to $\tilde{v}$ such that $\mathcal{R}(v, \tilde{v}) = 1$.
- $\mathsf{sta}^S_{open}(\mathcal{R}, v, z) = 1$ if and only if A decommits to a value $\tilde{v}$ such that $\mathcal{R}(v, \tilde{v}) = 1$.

DEFINITION 3.4 (nonmalleable commitment). *A commitment scheme $\langle C, R\rangle$ is said to be nonmalleable with respect to commitment if, for every probabilistic polynomial-time man-in-the-middle adversary A, there exist a probabilistic expected polynomial-time stand-alone adversary S and a negligible function $\nu : N \to N$ such that, for every irreflexive polynomial-time computable relation $\mathcal{R} \subseteq \{0,1\}^n \times \{0,1\}^n$, every $v \in \{0,1\}^n$, and every $z \in \{0,1\}^*$, it holds that*

$$\Pr\left[\mathsf{mim}^A_{com}(\mathcal{R}, v, z) = 1\right] \;<\; \Pr\left[\mathsf{sta}^S_{com}(\mathcal{R}, v, z) = 1\right] \;+\; \nu(n).$$

Nonmalleability with respect to opening is defined in the same way while replacing the random variables $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$ and $\mathsf{sta}^S_{com}(\mathcal{R}, v, z)$ with $\mathsf{mim}^A_{open}(\mathcal{R}, v, z)$ and $\mathsf{sta}^S_{open}(\mathcal{R}, v, z)$.

Content-based vs. tag-based commitments. Similar to the definition of interactive proofs nonmalleable with respect to statements, the above definitions require only that the adversary should not be able to commit to a value that is related, *but different*, from the value it receives a commitment from. Technically, the above fact can be seen from the definitions by noting that the relation $\mathcal{R}$, which defines the success of the adversary, is required to be irreflexive. This means that the adversary is said to fail if it is able only to produce a commitment to the same value.[4] Indeed, if the same protocol is executed in the left and the right interaction, the adversary can always copy messages and succeed in committing to the same value on the right as it receives a commitment from, on the left. To cope with this problem, the definition can be extended to incorporate tags, in analogy with the definition of interactive proofs nonmalleable with respect to tags. The extension is straightforward and therefore omitted.

We note that any commitment scheme that satisfies Definition 3.4 can easily be transformed into a scheme which is tag-based nonmalleable by prepending the tag to the value before committing. Conversely, in analogy with nonmalleable interactive proof, commitment schemes that are nonmalleable with respect to tags of length $t(n) = \text{poly}(n)$ can be transformed into commitment schemes nonmalleable with respect to content in a standard way (see, e.g., [14, 28]).

3.4. Comparison with previous definitions. Our definitions of nonmalleability essentially follow the original definitions by Dolev, Dwork, and Naor [14]. However, whereas the definitions by Dolev, Dwork, and Naor quantify the experiments over all distributions D of inputs for the left and the right interaction (or just left interaction in the case of commitments), we instead quantify over all possible input values $x, \tilde{x}$ (or in the case of commitments over all possible input values v for the left interaction). Our definitions can thus be seen as nonuniform versions of the definitions of [14].

Our definition of nonmalleability with respect to opening is, however, different from the definition of [13] in the following ways: (1) The definition of [13] does not take into account possible a priori information that the adversary might have about

[4] Potentially, one could consider a slightly stronger definition, which also rules out the case when the adversary is able to construct a *different* commitment to the *same* value. Nevertheless, here we adhere to the standard definition of nonmalleable commitments which allows the adversary to produce a different commitment to the same value.

the commitment, while ours (following [14]) does. (2) In our definition of the stand-alone execution the stand-alone adversary receives the value v after having completed the commitment phase and is thereafter supposed to decommit to a value related to v. The definition of [13] does not provide the simulator with this information.

In our view, the "a priori information" requirement is essential in many situations, and we therefore present a definition that satisfies it. (Consider, for example, a setting where the value v committed to is determined by a different protocol, which "leaks" some information about v.) In order to be able to satisfy this stronger requirement we relax the definition of [13] by allowing the stand-alone adversary to receive the value v before decommitting.

3.5. Simulation extractability. A central tool in our constructions of nonmalleable interactive proofs and commitments is the notion of *simulation extractability*. Loosely speaking, an interactive protocol is said to be simulation extractable if, for any man-in-the-middle adversary A, there exists a probabilistic polynomial-time machine (called the simulator-extractor) that can simulate both the left and the right interaction for A, while outputting a witness for the statement proved by the adversary in the right interaction.

Simulation extractability can be thought of as a technical (and stronger) variant of nonmalleability. The main reason for introducing this notion is that it enables a more modular analysis (and in particular is easier to work with). At the end of this section, we argue that any protocol that is simulation extractable is also a nonmalleable zero-knowledge proof of knowledge. In section 6 we show how to use simulation-extractable protocols in order to obtain nonmalleable commitments.

Let A be a man-in-the middle adversary that is simultaneously participating in a left interaction of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ while acting as verifier and a right interaction of $\langle P_{\tilde{\text{TAG}}}, V_{\tilde{\text{TAG}}} \rangle$ while acting as prover.

Let $view_A(x, z, \text{TAG})$ denote the *joint* view of $A(x, z)$ and the honest verifier $V_{\tilde{\text{TAG}}}$ when A is verifying a left proof of the statement x, using identity TAG, and proving on the right a statement $\tilde{x}$ using identity $\tilde{\text{TAG}}$. (The view consists of the messages sent and received by A in both the left and the right interaction and the random coins of A, and $V_{\tilde{\text{TAG}}}$.)[5] Both $\tilde{x}$ and $\tilde{\text{TAG}}$ are chosen by A. Given a function $t = t(n)$ we use the notation $\{\cdot\}_{z,x,\text{TAG}}$ as shorthand for $\{\cdot\}_{z \in \{0,1\}^*, x \in L, \text{TAG} \in \{0,1\}^{t(|x|)}}$.

DEFINITION 3.5 (simulation-extractable protocol). *A family* $\{\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle\}_{\text{TAG} \in \{0,1\}^*}$ *of interactive proofs is said to be* simulation extractable *with tags of length* $t = t(n)$ *if, for any man-in-the-middle adversary* A*, there exists a probabilistic expected poly-time machine* $\mathcal{S}$ *such that the following hold:*

1. *The probability ensembles* $\{\mathcal{S}_1(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ *and* $\{view_A(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ *are statistically close over* L*, where* $\mathcal{S}_1(x, z, \text{TAG})$ *denotes the first output of* $\mathcal{S}(x, z, \text{TAG})$.
2. *Let* $x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^{t(|x|)}$*, and let* $(view, w)$ *denote the output of* $\mathcal{S}(x, z, \text{TAG})$ *(on input some random tape). Let* $\tilde{x}$ *be the right-execution statement appearing in view, and let* $\tilde{\text{TAG}}$ *denote the right-execution tag. Then, if the right execution in view is accepting AND* $\text{TAG} \neq \tilde{\text{TAG}}$*, then* $R_L(\tilde{x}, w) = 1$.

We note that the above definition refers to protocols that are simulation extractable *with respect to themselves*. A stronger variant (which is not considered in

[5] Since the messages sent by A are fully determined given the code of A and the messages it receives, including them as part of the view is somewhat redundant. The reason we have chosen to do so is for convenience of presentation.

the current work) would have required simulation extractability even in the presence of protocols that do not belong to the family.

We next argue that in order to construct nonmalleable zero-knowledge protocols, it will be sufficient to come up with a protocol that is simulation extractable. To do so, we prove that any protocol that is simulation extractable (and has an efficient prover strategy) is also nonmalleable zero-knowledge (i.e., it satisfies Definitions 3.1 and 3.3).

PROPOSITION 3.6. *Let $\{\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle\}_{\text{TAG}\in\{0,1\}^*}$ be a family of simulation-extractable protocols with tags of length $t = t(n)$ (with respect to the language L and the witness relation R_L) with an efficient prover strategy. Then $\{\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle\}_{\text{TAG}\in\{0,1\}^*}$ is also a nonmalleable zero-knowledge (with tags of length t).*

Proof. Let $\{\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle\}_{\text{TAG}\in\{0,1\}^*}$ be a family of simulation-extractable protocols with tags of length t with respect to the language L and the witness relation R_L. We argue that $\{\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle\}_{\text{TAG}\in\{0,1\}^*}$ is both a nonmalleable interactive proof and stand-alone zero-knowledge.

Nonmalleability. Assume for contradiction that there exist a probabilistic poly-time man-in-the-middle adversary A and a polynomial $p(n)$ such that, for infinitely many n, there exist $x, \tilde{x} \in \{0,1\}^n$, $w, z \in \{0,1\}^*$, and $\text{TAG}, \tilde{\text{TAG}} \in \{0,1\}^{t(n)}$ such that $(x, w) \in L \times R_L(x)$, $\text{TAG} \neq \tilde{\text{TAG}}$, and

$$(3.1)\quad \Pr\left[\mathsf{mim}_V^A(\text{TAG}, \tilde{\text{TAG}}, x, \tilde{x}, w, z) = 1\right] \;\geq\; \Pr\left[\mathsf{sta}_V^S(\text{TAG}, \tilde{\text{TAG}}, x, \tilde{x}, z) = 1\right] \;+\; \frac{1}{p(n)}.$$

By Definition 3.5, there exists a probabilistic polynomial-time machine $\mathcal{S}$ for A that satisfies the definition's conditions. We show how to use $\mathcal{S}$ in order to construct a stand-alone prover S for $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$. On input $\text{TAG}, \tilde{\text{TAG}}, x, \tilde{x}, z$, the machine S runs the simulator-extractor $\mathcal{S}$ on input x, z, TAG and obtains the view *view* and witness $\tilde{w}$. In the event that the *view* contains an accepting right execution of the statement $\tilde{x}$ using tag TAG, S executes the honest prover strategy P_{TAG} on input x and the witness w.

It follows directly from the simulation property of $\mathcal{S}$ that the probability that *view* contains an accepting right-execution proof of $\tilde{x}$ using tag $\tilde{\text{TAG}}$ is negligibly close to

$$p_A = \Pr\left[\mathsf{mim}_V^A(\text{TAG}, \tilde{\text{TAG}}, x, \tilde{x}, w, z) = 1\right].$$

Since $\mathcal{S}$ always outputs a witness when the right execution is accepting and the tag of the right execution is different from the tag of the left execution, we conclude that the success probability of S is also negligibly close to p_A (since $\text{TAG} \neq \tilde{\text{TAG}}$). This contradicts (3.1).

Zero-knowledge. Consider any probabilistic poly-time verifier V^*. Construct the man-in-the-middle adversary A that internally incorporates V and relays its left execution unmodified to V^*. In the right execution, A simply outputs $\perp$. By the simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$, there exists a simulator-extractor $\mathcal{S}$ for A. We describe a simulator S for V^*.

On input x, z, TAG, S runs $\mathcal{S}$ on input x, z, TAG to obtain (view, w). Given the view *view*, S outputs the view of V^* in *view* (which is a subset of *view*). It follows directly from the simulation property of $\mathcal{S}$, and from the fact that S outputs an (efficiently computable) subset of *view*, that the output of S is indistinguishable from the view of V^* in an honest interaction with a prover. ∎

4. A simulation-extractable protocol. We now turn to describing our construction of simulation-extractable protocols. At a high level, the construction proceeds in two steps:

1. For any $n \in N$, construct a family $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle\}_{\mathsf{tag}\in[2n]}$ of simulation-extractable arguments with tags of length $t(n) = \log n + 1$.
2. For any $n \in N$, use the family $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle\}_{\mathsf{tag}\in[2n]}$ to construct a family $\{\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle\}_{\text{TAG}\in\{0,1\}^n}$ of simulation-extractable arguments with tags of length $t(n) = 2^n$.

The construction of the family $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle\}_{\mathsf{tag}\in[2n]}$ relies on Barak's nonblack-box techniques for obtaining the constant-round public-coin $\mathcal{ZK}$ argument for $\mathcal{NP}$ [1], and they are very similar in structure to the $\mathcal{ZK}$ protocols used by Pass in [35]. We start by reviewing the ideas underlying Barak's protocol. We then proceed to presenting our protocols.

4.1. Barak's nonblack-box protocol. Barak's protocol is designed to allow the simulator access to "trapdoor" information that is not available to the prover in actual interactions. Given this "trapdoor" information, the simulator will be able to produce convincing interactions even without possessing a witness for the statement being proved. The high-level idea is to enable the usage of the verifier's code as a "fake" witness in the proof. In the case of the honest verifier V (which merely sends over random bits), the code consists of the verifier's random tape. In the case of a malicious verifier V^*, the code may also consist of a program that generates the verifier's messages (based on previously received messages).

Since the actual prover does not have a priori access to V's code in real interactions, this will not harm the soundness of the protocol. The simulator, on the other hand, will be always able to generate transcripts in which the verifier accepts since, by definition, it obtains V^*'s code as input.

Let $n \in N$, and let $T : N \to N$ be a "nice" function that satisfies $T(n) = n^{\omega(1)}$. To make the above ideas work, Barak's protocol relies on a "special" $\mathbf{NTIME}(T(n))$ relation. It also makes use of a witness-indistinguishable universal argument ($WIUARG$) [16, 15, 27, 30, 3]. We start by describing a variant of Barak's relation, which we denote by R_{sim}. Usage of this variant will facilitate the presentation of our ideas in later stages.

Let $\{\mathcal{H}_n\}_n$ be a family of hash functions where a function $h \in \mathcal{H}_n$ maps $\{0,1\}^*$ to $\{0,1\}^n$ (cf. [29, 8]), and let Com be a statistically binding commitment scheme for strings of length n, where, for any $\alpha \in \{0,1\}^n$, the length of $\text{Com}(\alpha)$ is upper bounded by $2n$. The relation R_{sim} is described in Figure 3.

Instance: A triplet $\langle h, c, r\rangle \in \mathcal{H}_n \times \{0,1\}^n \times \{0,1\}^{\text{poly}(n)}$.

Witness: A program $\Pi \in \{0,1\}^*$, a string $y \in \{0,1\}^*$, and a string $s \in \{0,1\}^{\text{poly}(n)}$.

Relation: $\mathbf{R}_{sim}(\langle h, c, r\rangle, \langle \Pi, y, s\rangle) = 1$ if and only if the following hold:

1. $|y| \le |r| - n$.
2. $c = \text{Com}(h(\Pi); s)$.
3. $\Pi(y) = r$ within $T(n)$ steps.

FIG. 3. *R_{sim}: a variant of Barak's relation.*

Remark 4.1 (simplifying assumptions). The relation presented in Figure 3 is slightly oversimplified and will make Barak's protocol work only when $\{\mathcal{H}_n\}_n$ is

collision resistant against "slightly" superpolynomial-sized circuits [1]. To make it work assuming collision resistance against polynomial-sized circuits, one should use a "good" error-correcting code ECC (i.e., with constant distance and with polynomial-time encoding and decoding) and replace the condition $c = \mathrm{Com}(h(\Pi); s)$ with $c = \mathrm{Com}(h(\mathtt{ECC}(\Pi)); s)$ [3]. We also assume that Com is a one-message commitment scheme. Such schemes can be constructed based on any 1-1 one-way function. At the cost of a small complication, the one-message scheme could have been replaced by the two-message commitment scheme of [31], which can be based on "ordinary" one-way functions [26].

Let L be any language in $\mathcal{NP}$, let $n \in N$, and let $x \in \{0,1\}^n$ be the common input for the protocol. The idea is to have the prover claim (in a witness-indistinguishable fashion) that either $x \in L$, or that $\langle h, c, r\rangle$ belongs to the language L_{sim} that corresponds to R_{sim}, where $\langle h, c, r\rangle$ is a triplet that is jointly generated by the prover and the verifier. As will turn out from the analysis, no polynomial-time prover will be able to make $\langle h, c, r\rangle$ belong to L_{sim}. The simulator, on the other hand, will use the verifier's program in order to make sure that $\langle h, c, r\rangle$ is indeed in L_{sim} (while also possessing a witness for this fact).

A subtle point to be taken into consideration is that the verifier's running time (program size) is not a priori bounded by any specific polynomial (this is because the adversary verifier might run in arbitrary polynomial time). This imposes a choice of $T(n) = n^{\omega(1)}$ in R_{sim} and implies that the corresponding language does not lie in $\mathcal{NP}$ (but rather in $\mathbf{NTIME}(n^{\omega(1)})$). Such languages are beyond the scope of the "traditional" witness-indistinguishable proof systems (which were originally designed to handle "only" $\mathcal{NP}$ languages) and will thus require the usage of a witness-indistinguishable universal argument. Barak's protocol is described in Figure 4.

Common Input: An instance $x \in \{0,1\}^n$.

Security parameter: 1^n.

Stage 1:

$V \to P$: Send $h \overset{\mathrm{R}}{\leftarrow} \mathcal{H}_n$.
$P \to V$: Send $c = \mathrm{Com}(0^n)$.
$V \to P$: Send $r \overset{\mathrm{R}}{\leftarrow} \{0,1\}^{3n}$.

Stage 2 (Body of the proof):

$P \Leftrightarrow V$: A *WI UARG* $\langle P_{UA}, V_{UA}\rangle$ proving the OR of the following statements:
1. $\exists\ w \in \{0,1\}^{\mathrm{poly}(|x|)}$ so that $R_L(x, w) = 1$.
2. $\exists\ \langle \Pi, y, s\rangle$ so that $R_{sim}(\langle h, c, r\rangle, \langle \Pi, y, s\rangle) = 1$.

FIG. 4. *Barak's ZK argument for* $\mathcal{NP}$: $\langle P_B, V_B\rangle$.

Soundness. The idea behind the soundness of $\langle P_B, V_B\rangle$ is that any program Π (be it efficient or not) has only one output for any given input. This means that Π, when fed with an input y, has probability 2^{-3n} to "hit" a string $r \overset{\mathrm{R}}{\leftarrow} \{0,1\}^{3n}$. Since the prover sends c before actually receiving r, and since R_{sim} imposes $|y| \le |r| - n = n$, then it is not able to "arrange" that both $c = \mathrm{Com}(h(\Pi))$ and $\Pi(y) = r$ with probability significantly greater than $2^{2n} \cdot 2^{-3n} = 2^{-n}$ (as $|c| \le 2n$). The only way for a prover to make the honest verifier accept in the *WIUARG* is thus to use a witness w for R_L. This guarantees that, whenever the verifier is convinced, it is indeed the case that $x \in L$.

Zero-knowledge. Let V^* be the program of a potentially malicious verifier. The $\mathcal{ZK}$ property of $\langle P_B, V_B\rangle$ follows by letting the simulator set $\Pi = V^*$ and $y = c$. Since $|c| \leq 2n \leq |r| - n$, and since, by definition, $V^*(c)$ always equals r, the simulator can set $c = \mathrm{Com}(h(V^*); s)$ in Stage 1 and use the triplet $\langle V^*, c, s\rangle$ as a witness for R_{sim} in the $WIUARG$. This enables the simulator to produce convincing interactions, even without knowing a valid witness for $x \in L$. The $\mathcal{ZK}$ property then follows (with some work) from the hiding property of Com and the $\mathcal{WI}$ property of $\langle P_{UA}, V_{UA}\rangle$.

4.2. A "special-purpose" universal argument. Before we proceed with the construction of our new protocol, we will need to present a universal argument that is specially tailored for our purposes. The main distinguishing features of this universal argument, which we call the *special-purpose* argument, are the following: (1) it is *statistically* witness indistinguishable; and (2) it will enable us to prove that our protocols satisfy the proof of knowledge property of Definition 2.7.[6]

Let **Com** be a statistically hiding commitment scheme for strings of length n. Let $\mathbf{R}_{sim}$ be a variant of the relation R_{sim} (from Figure 3) in which the statistically binding commitment Com is replaced with the commitment **Com**, let $\langle P_{sWI}, V_{sWI}\rangle$ be a statistical witness-indistinguishable argument of knowledge and let $\langle P_{UA}, V_{UA}\rangle$ be a four-message, public-coin universal argument where the length of the messages is upper bounded by n.[7] The special-purpose $UARG$, which we denote by $\langle P_{sUA}, V_{sUA}\rangle$, handles statements of the form $(x, \langle h, c_1, c_2, r_1, r_2\rangle)$, where the triplets $\langle h, c_1, r_1\rangle$ and $\langle h, c_2, r_2\rangle$ correspond to instances for $\mathbf{R}_{sim}$. The protocol $\langle P_{sUA}, V_{sUA}\rangle$ is described in Figure 5.

Parameters: Security parameter 1^n.
Common Input: $x \in \{0,1\}^n$, $\langle h, c_1, c_2, r_1, r_2\rangle$, where, for $i \in \{1,2\}$, $\langle h, c_i, r_i\rangle$ is an instance for $\mathbf{R}_{sim}$.
Stage 1 (Encrypted UARG):
$V \to P$: Send $\alpha \stackrel{R}{\leftarrow} \{0,1\}^n$.
$P \to V$: Send $\widehat{\beta} = \mathbf{Com}(0^n)$.
$V \to P$: Send $\gamma \stackrel{R}{\leftarrow} \{0,1\}^n$.
$P \to V$: Send $\widehat{\delta} = \mathbf{Com}(0^n)$.
Stage 2 (Body of the proof):
$P \leftrightarrow V$: A statistical $WIAOK$ $\langle P_{sWI}, V_{sWI}\rangle$ proving the OR of the following statements:
1. $\exists\, w \in \{0,1\}^{\mathrm{poly}(|x|)}$ so that $R_L(x, w) = 1$.
2. $\exists\, \langle \beta, \delta, s_1, s_2\rangle$ so that the following hold:
   - $\widehat{\beta} = \mathbf{Com}(\beta; s_1)$.
   - $\widehat{\delta} = \mathbf{Com}(\delta; s_2)$.
   - $(\alpha, \beta, \gamma, \delta)$ is an accepting transcript for $\langle P_{UA}, V_{UA}\rangle$ proving the following statement:
     – $\exists\, \langle i, \Pi, y, s\rangle$ so that $\mathbf{R}_{sim}(\langle h, c_i, r_i\rangle, \langle \Pi, y, s\rangle) = 1$.

FIG. 5. *A special-purpose universal argument* $\langle P_{sUA}, V_{sUA}\rangle$.

[6] The "weak" proof of knowledge property of a universal argument (as defined in [3]) is not sufficient for our purposes. Specifically, while in a weak proof of knowledge it is required that the extractor succeeds with probability that is polynomially related to the success probability of the prover, in our proof of security we will make use of an extractor that succeeds with probability negligibly close to the success probability of the prover.

[7] Both statistical witness-indistinguishable arguments of knowledge and four-message, public-coin, universal arguments can be constructed assuming a family $\mathcal{H}_n$ of standard collision-resistant hash functions (cf. [18] and [27, 30, 3]).

4.3. A family of $2n$ protocols. We next present $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle\}_{\mathsf{tag}\in[2n]}$, a family of protocols with tags of length $t(n) = \log n + 1$.[8] The protocols are based on $\langle P_B, V_B\rangle$, and are a variant of the $\mathcal{ZK}$ protocols introduced by Pass [35]. There are two key differences between $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$ and $\langle P_B, V_B\rangle$: in the $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$ protocol (1) the prover (simulator) is given two opportunities to guess the verifier's "next message," and (2) the *lengths* of the verifier's "next messages" depend on the tag of the protocol. We mention that the idea of using a multiple slot version of $\langle P_B, V_B\rangle$ already appeared in [36, 35], and the message-length technique appeared in [35]. However, our protocols $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle\}_{\mathsf{tag}\in[2n]}$ differ from the protocol of [35] in two aspects: our protocols are required to satisfy (1) a *statistical* secrecy property and (2) a *proof of knowledge* property. Towards this end, we replace the statistically binding commitments, Com, used in the presentation of $\langle P_B, V_B\rangle$ with statistically hiding commitments and replace the use of a $WIUARG$ with the use of a "special-purpose" $UARG$. Let **Com** be a statistically hiding commitment scheme for strings of length n, where, for any $\alpha \in \{0,1\}^n$, the length of $\mathbf{Com}(\alpha)$ is upper bounded by $2n$. Let $\mathbf{R}_{sim}$ be the statistical variant of the relation R_{sim}, and let $\langle P_{sUA}, V_{sUA}\rangle$ be the special-purpose universal argument (both $\mathbf{R}_{sim}$ and $\langle P_{sUA}, V_{sUA}\rangle$ are described in section 4.2). Protocol $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$ is described in Figure 6.

Common Input: An instance $x \in \{0,1\}^n$.

Parameters: Security parameter 1^n, length parameter $\ell(n)$.

Tag String: $\mathsf{tag} \in [2n]$.

Stage 0 (Setup):

$V \to P$: Send $h \overset{\mathrm{R}}{\leftarrow} \mathcal{H}_n$.

Stage 1 (Slot 1):

$P \to V$: Send $c_1 = \mathbf{Com}(0^n)$.

$V \to P$: Send $r_1 \overset{\mathrm{R}}{\leftarrow} \{0,1\}^{\mathsf{tag}\cdot\ell(n)}$.

Stage 1 (Slot 2):

$P \to V$: Send $c_2 = \mathbf{Com}(0^n)$.

$V \to P$: Send $r_2 \overset{\mathrm{R}}{\leftarrow} \{0,1\}^{(2n+1-\mathsf{tag})\cdot\ell(n)}$.

Stage 2 (Body of the proof):

$P \Leftrightarrow V$: A special-purpose $UARG$ $\langle P_{sUA}, V_{sUA}\rangle$ proving the OR of the following statements:

1. $\exists\ w \in \{0,1\}^{\mathrm{poly}(|x|)}$ so that $R_L(x,w) = 1$.
2. $\exists\ \langle \Pi, y, s\rangle$ so that $\mathbf{R}_{Sim}(\langle h, c_1, r_1\rangle, \langle \Pi, y, s\rangle) = 1$.
3. $\exists\ \langle \Pi, y, s\rangle$ so that $\mathbf{R}_{Sim}(\langle h, c_2, r_2\rangle, \langle \Pi, y, s\rangle) = 1$.

FIG. 6. *Protocol* $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$.

Note that the only difference between two protocols $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$ and $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}}\rangle$ is the length of the verifier's "next messages": in fact, the length of those messages in $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$ is a parameter that depends on tag (as well as on the length parameter $\ell(n)$). This property will be crucial for the analysis of these protocols in the man-in-the-middle setting.

Using similar arguments to the ones used for $\langle P_B, V_B\rangle$, it can be shown that

[8] A closer look at the construction will reveal that it will in fact work for any $t(n) = O(\log n)$. The choice of $t(n) = \log n + 1$ is simply made for the sake of concreteness (as in our constructions it is the case that $\mathsf{tag} \in [2n]$).

$\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ is computationally sound. The main difference to be taken into consideration is the existence of multiple slots in Stage 1 (see Lemma A.1 for a proof of an even stronger statement).

The $\mathcal{ZK}$ property of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ is proved exactly as in the case of $\langle P_B, V_B \rangle$ by letting the simulator pick either $i = 1$ or $i = 2$ and using $\langle V^*, c_i, s_i \rangle$ as the witness for $\langle h, c_i, r_i \rangle \in L_{sim}$ (where L_{sim} is the language that corresponds to $\mathbf{R}_{sim}$). Since for every $\mathsf{tag} \in [m]$, $|r_i| - |c_i| \geq \ell(n) - 2n$, we have that, as long as $\ell(n) \geq 3n$, the protocol $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ is indeed $\mathcal{ZK}$.

We wish to highlight some useful properties of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$. These properties will turn out to be relevant when dealing with a man in the middle.

Freedom in the choice of slot: The simulator described above has the freedom to choose which $i \in \{1, 2\}$ it will use in order to satisfy the relation $\mathbf{R}_{sim}$. In particular, for the simulation to succeed, it is sufficient that $\langle h, c_i, r_i \rangle \in L_{sim}$ for *some* $i \in \{1, 2\}$.

Using a longer y in the simulation: The stand-alone analysis of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ requires only $\ell(n) \geq 3n$. Allowing larger values of $\ell(n)$ opens the possibility of using a longer y in the simulation. This will turn out to be useful if the verifier is allowed to receive "outside" messages that do not belong to the protocol (as occurs in the man-in-the-middle setting).

Statistical secrecy: The output of the simulator described above is *statistically* close to real interactions (whereas the security guaranteed in $\langle P_B, P_B \rangle$ is only computational). A related property will turn out to be crucial for the use of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ as a subroutine in higher-level applications (such as nonmalleable commitments).

Proof of knowledge: $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ is a proof of knowledge. That is, for any prover P^* and for any $x \in \{0,1\}^n$, if P^* convinces the honest verifier V that $x \in L$ with nonnegligible probability, then one can extract a witness w that satisfies $R_L(x, w) = 1$ in (expected) polynomial time.

4.4. A family of 2^n protocols. The protocol family $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle\}_{\mathsf{tag} \in [2n]}$ is replied upon to show how to construct a family $\{\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle\}_{\text{TAG} \in \{0,1\}^n}$ with tags of length $t(n) = n$. The protocols are *constant round* and involve n parallel executions of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$, with appropriately chosen tags. This new family of protocols is denoted $\{\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle\}_{\text{TAG} \in \{0,1\}^n}$ and is described in Figure 7.

Common Input: An instance $x \in \{0,1\}^n$.

Parameters: Security parameter 1^n, length parameter $\ell(n)$.

Tag String: $\text{TAG} \in \{0,1\}^n$. Let $\text{TAG} = \text{TAG}_1, \ldots, \text{TAG}_n$.

The protocol:

$P \leftrightarrow V$: For all $i \in \{1, \ldots, n\}$ (in parallel) do the following:
1. Set $\mathsf{tag}_i = (i, \text{TAG}_i)$.
2. Run $\langle P_{\mathsf{tag}_i}, V_{\mathsf{tag}_i} \rangle$ with common input x and length parameter $\ell(n)$.

V: Accept if and only if all runs are accepting.

FIG. 7. *Protocol* $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$.

Notice that $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ has a constant number of rounds (since each $\langle P_{\mathsf{tag}_i}, V_{\mathsf{tag}_i} \rangle$ is constant round). Also notice that, for $i \in [n]$, the length of $\mathsf{tag}_i = (i, \text{TAG}_i)$ is

$$|i| + |\text{TAG}_i| = \log n + 1 = \log(2n).$$

Viewing (i, TAG_i) as elements in $[2n]$ we infer that the length of verifier messages in $\langle P_{\mathsf{tag}_i}, V_{\mathsf{tag}_i} \rangle$ is upper bounded by $2n\ell(n)$. Hence, as long as $\ell(n) = \text{poly}(n)$, the length of verifier messages in $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ is $2n^2\ell(n) = \text{poly}(n)$.

We now turn to showing that, for any $\text{TAG} \in 2^n$, the protocol $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ is an interactive argument. In fact, what we show is a stronger statement. Namely, the protocols $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ are proofs (actually arguments) of knowledge (as in Definition 2.7). For simplicity of exposition, we will show how to prove the above assuming a family of hash functions that is collision resistant against $T(n) = n^{\omega(1)}$-sized circuits. As mentioned in Remark 4.1, by slightly modifying $\mathbf{R}_{sim}$, one can prove the same statement under the more standard assumption of collision resistance against polynomial-sized circuits.

PROPOSITION 4.2 (argument of knowledge). *Let $\langle P_{sWI}, V_{sWI} \rangle$ and $\langle P_{UA}, V_{UA} \rangle$ be the protocols used in the construction of $\langle P_{sUA}, V_{sUA} \rangle$. Suppose that $\{\mathcal{H}_n\}_n$ is collision resistant for $T(n)$-sized circuits, that* **Com** *is statistically hiding, that $\langle P_{sWI}, V_{sWI} \rangle$ is a statistical witness-indistinguishable argument of knowledge, and that $\langle P_{UA}, V_{UA} \rangle$ is a universal argument. Then, for any $\text{TAG} \in \{0,1\}^n$, $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ is an interactive argument of knowledge.*

Similar arguments to the ones used to prove Proposition 4.2 have already appeared in the works of Barak [1] and Barak and Goldreich [3]. While our proof builds on these arguments, it is somewhat more involved. For the sake of completeness, the full proof appears in the appendix.

5. Proving simulation extractability. Our central technical lemma states that the family of protocols $\{\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle\}_{\text{TAG} \in \{0,1\}^n}$ is simulation extractable. As shown in Proposition 3.6 this implies that these protocols are also nonmalleable zero-knowledge.

LEMMA 5.1 (simulation extractability). *Suppose that* **Com** *are statistically hiding, that $\{\mathcal{H}_n\}_n$ is a family of collision-resistant hash functions, that $\langle P_{UA}, V_{UA} \rangle$ is a special-purpose $WIUARG$, and that $\ell(n) \geq 2n^2 + 2n$. Then $\{\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle\}_{\text{TAG} \in \{0,1\}^n}$ is simulation extractable.*

The proof of Lemma 5.1 is fairly complex. To keep things manageable, we first give an overview of the proof, describing the key ideas used for establishing the simulation extractability of the family $\{\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle\}_{\mathsf{tag} \in [2n]}$. This is followed by a full proof for the case of $\{\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle\}_{\text{TAG} \in \{0,1\}^n}$.

5.1. Proof overview. Consider a man-in-the-middle adversary A that is playing the role of the verifier of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ in the left interaction while simultaneously playing the role of the prover of $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}} \rangle$ in the right interaction. Recall that in order to prove simulation extractability we have to show that, for any such A, there exists a combined simulator-extractor $\mathcal{S} = (\mathsf{SIM}, \mathsf{EXT})$ that is able to simulate both the left and the right interactions for A while simultaneously extracting a witness to the statement $\tilde{x}$ proved in the right interaction.

Towards this goal, we will construct a simulator S that is able to "internally" generate P_{tag} messages for the left interaction of A, even if the messages in the right interaction are forwarded to A from an "external" verifier $V_{\tilde{\mathsf{tag}}}$. The simulator S is almost identical to the simulator of [35] and exploits the difference in message lengths between the protocols $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ and $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}} \rangle$. As the analysis will demonstrate, the left view produced by the simulator S is statistically indistinguishable from A's actual interactions with an honest left prover P_{tag}. Furthermore, we show the following:

1. It will be possible to construct a procedure SIM that faithfully simulates A's view in a man-in-the-middle execution. To do so, we will honestly play the role of $V_{\tilde{\mathsf{tag}}}$ in the right interaction and use S to simulate A's left interaction with P_{tag} (pretending that the messages from the right interaction came from an external $V_{\tilde{\mathsf{tag}}}$).
2. It will be possible to construct a procedure EXT that extracts witnesses for the statements $\tilde{x}$ proved in the right interactions of the views generated by the above SIM. To do so, we will use S to transform A into a stand-alone prover $P^*_{\tilde{\mathsf{tag}}}$ for the statement $\tilde{x}$. This will be done by having $P^*_{\tilde{\mathsf{tag}}}$ internally emulate A's execution while forwarding A's messages to an external honest verifier $V_{\tilde{\mathsf{tag}}}$ and using S to simulate A's left interaction with P_{tag}. We can then invoke the knowledge extractor that is guaranteed by the (stand-alone) proof of knowledge property of $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}}\rangle$ and obtain a witness for $\tilde{x} \in L$.

It is important to have both SIM and EXT use the same simulator program S (with same random coins) in their respective executions. Otherwise, we are not guaranteed that the statement $\tilde{x}$ appearing in the output of SIM is the same one EXT extracts a witness from.[9]

The execution of S (with one specific scheduling of messages) is depicted in Figure 8. In order to differentiate between the left and right interactions, messages m in the right interaction are labeled as $\tilde{m}$. Stage 2 messages in the left and right interactions are denoted u and $\tilde{u}$, respectively.

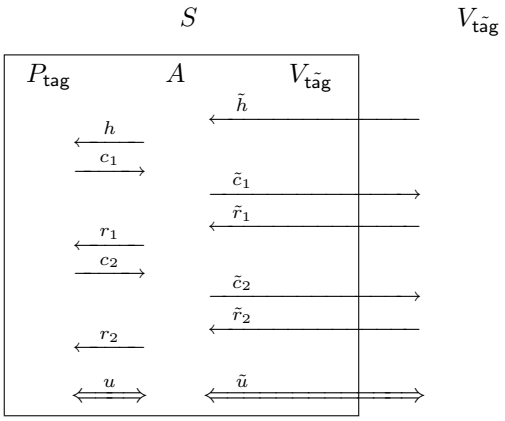


FIG. 8. *The simulator S.*

The main hurdle in implementing S is in making the simulation of the left interaction work. The problem is that the actual code of the verifier whose view we are simulating is only partially available to S. This is because the messages sent by A in the left interaction also depend on the messages A receives in the right interaction. These messages are sent by an "external" $V_{\tilde{\mathsf{tag}}}$, and $V_{\tilde{\mathsf{tag}}}$'s code (randomness) is not available to S.

Technically speaking, the problem is implied by the fact that the values of the r_i's do not necessarily depend only on the corresponding c_i but rather may also depend on the "external" right messages $\tilde{r_i}$. Thus, setting $\Pi = A$ and $y = c_i$ in the simulation

[9]The statement $\tilde{x}$ will remain unchanged because $\tilde{x}$ occurs prior to any message in the right interaction (and hence does not depend on the external messages received by $P^*_{\tilde{\mathsf{tag}}}$).

(as done in section 4.3) will not be sufficient, since in some cases it is simply not true that $r_i = A(c_i)$.

Intuitively, the most difficult case to handle is the one in which $\tilde{r_1}$ is contained in Slot 1 of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ and $\tilde{r_2}$ is contained in Slot 2 of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ (as in Figure 8). In this case $r_i = A(c_i, \tilde{r_i})$, and so $r_i = A(c_i)$ does not hold for either $i \in \{1, 2\}$. As a consequence, the simulator will not be able to produce views of convincing Stage 2 interactions with A. In order to overcome the difficulty, we will use the fact that, for a given instance $\langle h, c_i, r_i \rangle$, the string c_i is short enough to be "augmented" by $\tilde{r_i}$ while still satisfying the relation $\mathbf{R}_{sim}$.

Specifically, as long as $|c_i| + |\tilde{r_i}| \leq |r_i| - n$, the relation $\mathbf{R}_{sim}$ can be satisfied by setting $y = (c_i, \tilde{r_i})$. This guarantees that indeed $\Pi(y) = r_i$. The crux of the argument lies in the following fact.

FACT 5.2. *If* $\mathsf{tag} \neq \tilde{\mathsf{tag}}$, *then there exists* $i \in \{1, 2\}$ *so that* $|\tilde{r_i}| \leq |r_i| - \ell(n)$.

By setting $y = (c_i, \tilde{r_i})$ for the appropriate i, the simulator is thus always able to satisfy $\mathbf{R}_{sim}$ for some $i \in \{1, 2\}$. This is because the "auxiliary" string y used in order to enable the prediction of r_i is short enough to pass the inspection at condition 1 (see Figure 3) of $\mathbf{R}_{sim}$ (i.e., $|y| = |c_i| + |\tilde{r_i}| \leq |r_i| - n$).[10] Once $\mathbf{R}_{sim}$ can be satisfied, the simulator is able to produce views of convincing interactions that are computationally indistinguishable from real left interactions.[11]

The extension of the above analysis to the case of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ has to take several new factors into consideration. First, each execution of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ consists of n parallel executions of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ (and not only one). This imposes the constraint $\ell(n) \geq 2n^2 + 2n$ and requires a careful specification of the way in which the left simulation procedure handles the verifier challenges in $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$. Second, and more importantly, the simulation procedure will not be able to handle a case in which $\langle P_{\tilde{\text{TAG}}}, V_{\text{TAG}} \rangle$ messages of the right interaction are forwarded from an external verifier $V_{\tilde{\text{TAG}}}$ (because these messages are too long for the simulation to work).

While this does not seem to pose a problem for the SIM procedure, it suddenly becomes unclear how to construct a stand-alone prover $P^*_{\tilde{\text{TAG}}}$ for the EXT procedure (since this involves forwarding messages from $V_{\tilde{\text{TAG}}}$). The way around this difficulty will be to construct a stand-alone prover $P^*_{\tilde{\mathsf{tag}}}$ for a *single* subprotocol $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}} \rangle$ instead. This will guarantee that the only messages that end up being forwarded are sent by an external verifier $V_{\tilde{\mathsf{tag}}}$, whose messages are short enough to make the simulation work. Once such a $P^*_{\tilde{\mathsf{tag}}}$ is constructed it is possible to use the knowledge extractor for $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}} \rangle$ in order to obtain a witness for $\tilde{x}$.

5.2. Many-to-one simulation extractabiity. We now proceed with the proof of Lemma 5.1. To establish the simulation extractability of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$, we first consider what happens when a man-in-the-middle adversary is simultaneously involved in the verification of *many* different (parallel) executions of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ on the left while proving a *single* interaction $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}} \rangle$ on the right. As it turns out, as long as the number of left executions is bounded in advance, we can actually guarantee simulation extractability even in this scenario.

[10] This follows from the fact that $\ell(n) \geq 3n$ and $|c_i| = 2n$.

[11] In the above discussion we have been implicitly assuming that $\tilde{h}, \tilde{u}$ are not contained in the two slots of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ (where $\tilde{h}$ denotes the hash function in the right interaction and $\tilde{u}$ denotes the sequence of messages sent in the right $WIUARG$). The case in which $\tilde{h}, \tilde{u}$ are contained in the slots can be handled by setting $\ell(n) \geq 4n$ and by assuming that both $|\tilde{h}|$ and the total length of the messages sent by the verifier in the $WIUARG$ are at most n. We mention that the latter assumption is reasonable and is indeed satisfied by known protocols (e.g., the $WIUARG$ of [3]).

For any $\text{TAG} = (\mathsf{tag}_1, \ldots, \mathsf{tag}_n) \in [2n]^n$ we consider a *left* interaction in which the protocols $\langle P_{\mathsf{tag}_1}, V_{\mathsf{tag}_1}\rangle, \ldots, \langle P_{\mathsf{tag}_n}, V_{\mathsf{tag}_n}\rangle$ are executed in parallel with common input $x \in \{0,1\}^n$ and a *right* interaction in which $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}}\rangle$ is executed with common input $\tilde{x} \in \{0,1\}^n$. The strings $\tilde{\mathsf{tag}}$ and $\tilde{x}$ are chosen adaptively by the man-in-the-middle adversary A. The witness used by the prover in the left interaction is denoted by w, and the auxiliary input used by A is denoted by z.

PROPOSITION 5.3. *Let A be a MIM adversary as above, and suppose that $\ell(n) \geq 2n^2 + 2n$. Then there exists a probabilistic expected polynomial time $\mathcal{S}$ such that the following conditions hold:*

1. *The probability ensembles $\{\mathcal{S}_1(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ and $\{view_A(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ are statistically close over L, where $\mathcal{S}_1(x, z, \text{TAG})$ denotes the first output of $\mathcal{S}(x, z, \text{TAG})$.*
2. *Let $x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^{t(|x|)}$, and let $(view, w)$ denote the output of $\mathcal{S}(x, z, \text{TAG})$ (on input some random tape). Let $\tilde{x}$ be the right-execution statement appearing in view, and let $\tilde{\mathsf{tag}}$ denote the right-execution tag. Then, if the right execution in view is accepting AND $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$ for all $j \in [n]$, then $R_L(\tilde{x}, w) = 1$.*

Proof. As discussed in section 5.1, we construct a "many-to-one" simulator S that internally generates a left view of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle = (\langle P_{\mathsf{tag}_1}, V_{\mathsf{tag}_1}\rangle, \ldots, \langle P_{\mathsf{tag}_n}, V_{\mathsf{tag}_n}\rangle)$ for A while forwarding messages from the right interaction to an external honest verifier $V_{\tilde{\mathsf{tag}}}$. This simulator is essentially identical to the simulator of [35].[12] We then show how to use S to construct the procedures $(\mathsf{SIM}, \mathsf{EXT})$.

5.2.1. The many-to-one simulator. The many-to-one simulator S invokes A as a subroutine. It attempts to generate views of the left and right interactions that are indistinguishable from A's view in real interactions. Messages in the right interaction are forwarded by S to an "external" honest verifier $V_{\tilde{\mathsf{tag}}}$ for $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}}\rangle$, whose replies are then fed back to A. Messages in the left interaction are handled by n "subsimulators" $S_1, \ldots, S_n$, where each S_j is responsible for generating the messages of the subprotocol $\langle P_{\mathsf{tag}_j}, V_{\mathsf{tag}_j}\rangle$. The execution of the simulator is depicted in Figure 9 (for simplicity, we ignore the messages $h^1, \ldots, h^n, u^1, \ldots, u^n$ and $\tilde{h}, \tilde{u}$).

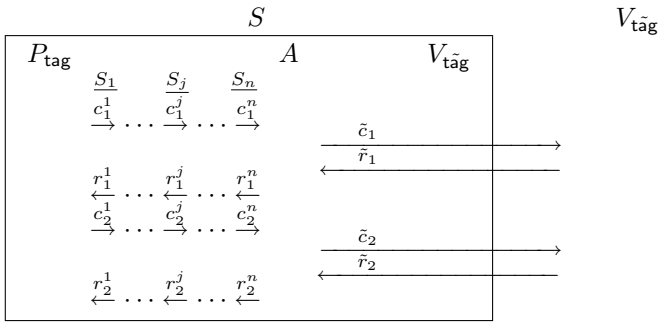


FIG. 9. *The "many-to-one" simulator S.*

The specific actions of a subsimulator S_j depend on the scheduling of Stage 1 messages as decided by A. The scheduling of left and right messages is divided into

[12] In fact, the simulator presented here is somewhat simplified in that we consider only n *parallel* executions of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$, whereas [35] also shows a simulator for n *concurrent* executions of the protocols.

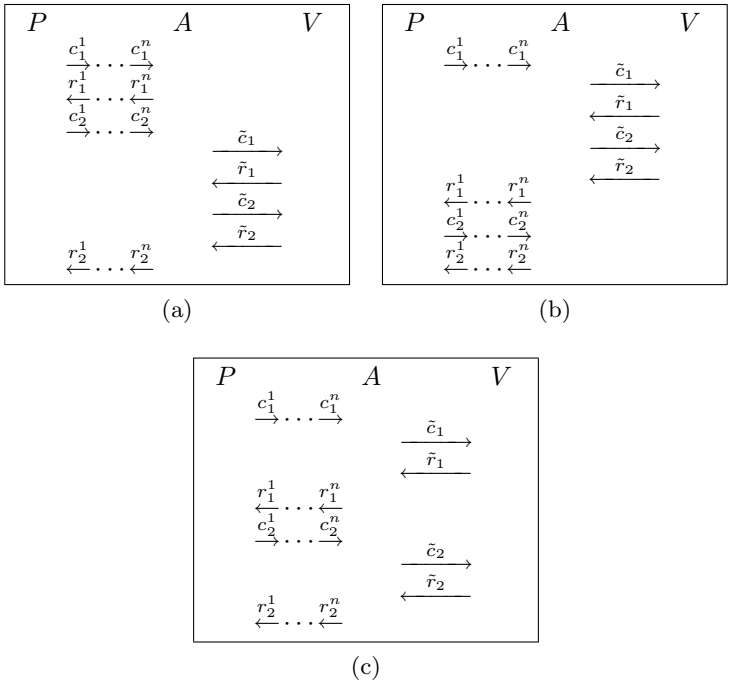


FIG. 10. *Three "representative" schedulings.*

three separate cases which are depicted in Figure 10. In all three cases we make the simplifying assumption that $\tilde{h}$ is scheduled in the right interaction before $h^1 \ldots, h^n$ are scheduled in the left interaction. We also assume that the $\langle P_{sUA}, V_{sUA} \rangle$ messages $\tilde{u}$ in the right interaction are scheduled after the $\langle P_{sUA}, V_{sUA} \rangle$ messages $u^1, \ldots, u^n$ in the left interaction. We later argue how these assumptions can be removed.

Let A_j be a program that acts exactly like A, but for any $i \in \{1, 2\}$ instead of outputting $r_i^1, \ldots, r_i^n$ it outputs only r_i^j. Given a string $\alpha \in \{0,1\}^*$, let $A(\alpha, \cdot)$ denote the program obtained by "hardwiring" α into it (i.e., $A(\alpha, \cdot)$ evaluated on β equals $A(\alpha, \beta)$). We now describe S_j's actions in each of the three cases.

None of $\tilde{r}_1, \tilde{r}_2$ is contained in Slot 1 *of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$:* Assume for simplicity that $\tilde{c_1}, \tilde{r_1}, \tilde{c_2}, \tilde{r_2}$ are all contained in Slot 2 of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ (Figure 10(a)). The simulator S_j sets $c_1 = \text{Com}(h(\Pi_1); s_1)$ and $c_2 = \text{Com}(0^n; s_2)$, where $\Pi_1 = A_j(x, \cdot)$. It then sets the triplet $\langle \Pi_1, (c_1^1, \ldots, c_1^n), s_1 \rangle$ as witness for $\langle h^j, c_1^j, r_1^j \rangle \in L_{sim}$.

None of $\tilde{r}_1, \tilde{r}_2$ is contained in Slot 2 *of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$:* Assume for simplicity that $\tilde{c_1}, \tilde{r_1}, \tilde{c_2}, \tilde{r_2}$ are all contained in Slot 1 of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ (Figure 10(b)). The simulator S_j sets $c_1 = \text{Com}(0^n; s_1)$ and $c_2 = \text{Com}(h(\Pi_2); s_2)$, where $\Pi_2 = A_j(x, c_1^1, \ldots, c_1^n, \tilde{r_1}, \tilde{r_2}, \cdot)$. It then sets the triplet $\langle \Pi_2, (c_2^1, \ldots, c_2^n), s_2 \rangle$ as witness for $\langle h^j, c_2^j, r_2^j \rangle \in L_{sim}$.

$\tilde{r}_1$ is contained in Slot 1 *and $\tilde{r}_2$ is contained in Slot* 2 *of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$:* In this case $\tilde{c_1}, \tilde{r_1}$ are both contained in Slot 1 of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$, and $\tilde{c_2}, \tilde{r_2}$ are both contained in Slot 2 of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ (Figure 10(c)). The simulator S_j sets $c_1 = \text{Com}(h(\Pi_1); s_1)$ and $c_2 = \text{Com}(h(\Pi_2); s_2)$, where $\Pi_1 = A_j(x, \cdot)$ and $\Pi_2 = A_j(x, c_1^1, \ldots, c_1^n, \tilde{r_1}, \cdot)$. Then the following hold:

- If $\mathsf{tag}_j > \tilde{\mathsf{tag}}$, the simulator sets $\langle \Pi_1, (c_1^1, \ldots, c_1^n, \tilde{r_1}), s_1 \rangle$ as witness for

$\langle h^j, c_1^j, r_1^j\rangle \in L_{sim}$.

- If $\mathsf{tag}_j < \tilde{\mathsf{tag}}$, the simulator sets $\langle \Pi_2, (c_2^1, \ldots, c_2^n, \tilde{r_2}), s_2\rangle$ as witness for $\langle h^j, c_2^j, r_2^j\rangle \in L_{sim}$.

In all cases, combining the messages together results in a Stage 1 transcript $\tau_1^j = \langle h^j, c_1^j, r_1^j, c_2^j, r_2^j\rangle$. By definition of $\langle P_{\mathsf{tag}_j}, V_{\mathsf{tag}_j}\rangle$, the transcript τ_1^j induces a Stage 2 special-purpose $WIUARG$ with common input $(x, \langle h^j, c_1^j, r_1^j\rangle, \langle h^j, c_2^j, r_2^j\rangle)$. The sub-simulator S_j now follows the prescribed prover strategy P_{sUA} and produces a Stage 2 transcript τ_2^j for $\langle P_{\mathsf{tag}_j}, V_{\mathsf{tag}_j}\rangle$.

Remark 5.4 (handling $\tilde{h}$ and $\tilde{u}$). To handle the case in which either $\tilde{h}$ or $\tilde{u}$ is contained in one of the slots, we set $\ell(n) \geq 2n^2+2n$ and let the simulator append either $\tilde{h}$ or $\tilde{u}$ to the auxiliary string y (whenever necessary). This will guarantee that the program committed to by the simulator indeed outputs the corresponding "challenge" r_i^j, when fed with y as input. The crucial point is that, even after appending $\tilde{h}$ or $\tilde{u}$ to y, it will still be the case that $|y| \leq |r_i^j| - n$. This just follows from the fact that the total length of $\tilde{h}$ and the messages $\tilde{u}$ sent in $\langle P_{sUA}, V_{sUA}\rangle$ are upper bounded by, say, n and that the gap between $|r_i^j|$ and the "original" $|y|$ (i.e., before appending $\tilde{h}$ or $\tilde{u}$ to it) is guaranteed to be at least n (this follows from the requirement $\ell(n) \geq 2n^2+2n$).

Output of S. To generate its output, which consists of a verifier view of a $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ interaction, S combines all the views generated by the S_j's. Specifically, letting $\sigma_1^j = (c_1^j, c_2^j)$ be the verifier's view of τ_1^j, and σ_2^j be the verifier's view of τ_2^j, the output of S consists of $(\sigma_1, \sigma_2) = ((\sigma_1^1, \ldots, \sigma_1^n), (\sigma_2^1, \ldots, \sigma_2^n))$.

5.2.2. The simulator-extractor. Using S, we construct the simulator-extractor $\mathcal{S} = (\mathsf{SIM}, \mathsf{EXT})$. We start with the machine SIM. In the right interaction SIM's goal is to generate messages by a verifier $V_{\tilde{\mathsf{tag}}}$. This part of the simulation is quite straightforward and is performed by simply playing the role of an honest verifier in the execution of the protocol (with the exception of cases in which $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ for some $j \in [n]$—see below for details). In the left interaction, on the other hand, SIM is supposed to act as a prover P_{TAG}, and this is where S is invoked.

The machine SIM. On input (x, z, TAG), and given a man-in-the-middle adversary A, SIM starts by constructing a man-in-the-middle adversary A' that acts as follows:

Internal messages: Pick random $M' = (\tilde{h}, \tilde{r}_1, \tilde{r}_2, \tilde{u})$ verifier messages for the right interaction.

Right interaction: The statement $\tilde{x}$ proved is the same as the one chosen by A. If there exists $j \in [n]$ so that $\mathsf{tag}_j = \tilde{\mathsf{tag}}$, use the messages in M' in order to internally emulate a right interaction for A (while ignoring external $V_{\tilde{\mathsf{tag}}}$ messages M). Otherwise, forward A's messages in the right interaction to an external $V_{\tilde{\mathsf{tag}}}$ and send back his answers M to A.

Left interaction: As induced by the scheduling of messages by A, forward the messages sent by A in the left interaction to an external prover P_{TAG}, and send back his answers to A.

Figure 11(a) describes the behavior of A' in case $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$ for all $j \in [n]$, whereas Figure 11(b) describes its behavior otherwise. The purpose of constructing such an A' is to enable us to argue that for all "practical purposes" the man-in-the-middle adversary never uses a $\tilde{\mathsf{tag}}$ that satisfies $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ for some $j \in [n]$ (as in such cases A' ignores all messages M in the right interaction anyway).

Given the new adversary A', the machine SIM picks random $V_{\tilde{\mathsf{tag}}}$ messages M and invokes the simulator S with random coins $\bar{s}$. The simulator's goal is to generate a view of a left interaction for an A' that receives messages M in the right interaction.

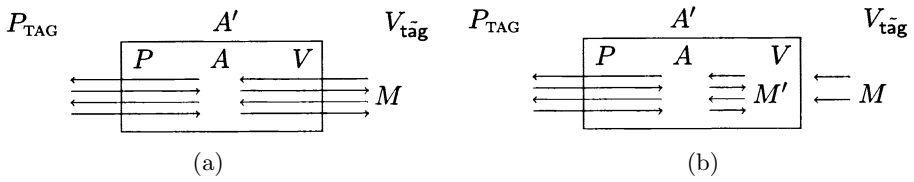


FIG. 11. *The adversary A'.*

Let (σ_1, σ_2) be the view generated by S for the left $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ interaction, and let $\tilde{\mathsf{tag}}$ be the tag sent by A in the right interaction when $(\sigma_1 \sigma_2)$ is its view of the left interaction. If there exists $j \in [n]$ so that $\mathsf{tag}_j = \tilde{\mathsf{tag}}$, then SIM outputs M' as the right view of A. Otherwise, it outputs M. SIM always outputs (σ_1, σ_2) as a left view for A.

The machine EXT. The machine EXT starts by sampling a random execution of SIM using random coins $\bar{s}, M', M$. Let $\tilde{x}$ be the right-hand side common input that results from feeding the output of SIM to A. Our goal is to extract a witness to the statement $\tilde{x}$. At a high level, EXT acts in the following way:

1. If the right session was *not* accepting or $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ for some $j \in [n]$, EXT will assume that no witness exists for the statement $\tilde{x}$ and will refrain from extraction.
2. Otherwise, EXT constructs a stand-alone prover $P^*_{\tilde{\mathsf{tag}}}$ for the right interaction $\langle P_{\tilde{\mathsf{tag}}_i}, V_{\tilde{\mathsf{tag}}_i} \rangle$, from which it will later attempt to extract the witness (see Figure 12).
   In principle, the prover $P^*_{\tilde{\mathsf{tag}}}$ will follow SIM's actions using the same random coins $\bar{s}$ used for initially sampling the execution of SIM. However, $P^*_{\tilde{\mathsf{tag}}}$'s execution will differ from SIM's execution in that it will not use the messages M in the right interaction of A but rather will forward messages receives from an external verifier $V_{\tilde{\mathsf{tag}}}$.

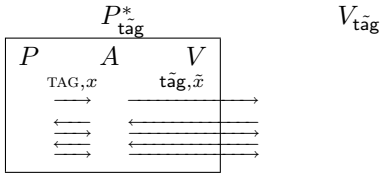


FIG. 12. *The prover $P^*_{\tilde{\mathsf{tag}}}$.*

3. Once $P^*_{\tilde{\mathsf{tag}}}$ is constructed, EXT can apply the knowledge extractor, guaranteed by the proof of knowledge property of $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}} \rangle$, and extract a witness w to the statement $\tilde{x}$. In the unlikely event that the extraction failed, EXT outputs `fail`. Otherwise, it outputs w.

Remark 5.5. It is important to have a prover $P^*_{\tilde{\mathsf{tag}}}$ for the *entire* protocol $\langle P_{\tilde{\mathsf{tag}}_i}, V_{\tilde{\mathsf{tag}}_i} \rangle$ (and not just for $\langle P_{sUA}, V_{sUA} \rangle$). This is required in order to argue that the witness extracted is a witness for $\tilde{x}$ and not a witness to $\langle \tilde{h}, \tilde{c}_1, \tilde{r}_1 \rangle \in L_{sim}$ or to $\langle \tilde{h}, \tilde{c}_2, \tilde{r}_2 \rangle \in L_{sim}$ (which could indeed be the case if we fixed the messages $\langle \tilde{h}, \tilde{c}_1, \tilde{r}_1, \tilde{c}_2, \tilde{r}_2 \rangle$ in advance).

Output of simulator-extractor $\mathcal{S}$. The combined simulator-extractor $\mathcal{S}$ runs EXT and outputs `fail` whenever EXT does so. Otherwise, it outputs the view output by SIM (in the execution by EXT) followed by the output of EXT.

5.2.3. Correctness of simulation extraction. We start by showing that the view of A in the simulation by SIM is statistically close to its view in an actual interaction with P_{TAG} and $V_{\tilde{\mathsf{tag}}}$.

LEMMA 5.6. *The probability ensembles* $\{\mathcal{S}_1(x,z,\text{TAG})\}_{x\in L,z\in\{0,1\}^*,\text{TAG}\in\{0,1\}^m}$ *and* $\{view_A(x,z,\text{TAG})\}_{x\in L,z\in\{0,1\}^*,\text{TAG}\in\{0,1\}^m}$ *are statistically close over* L*, where* $\mathcal{S}_1(x,z,\text{TAG})$ *denotes the first output of* $\mathcal{S}(x,z,\text{TAG})$.

Proof. Recall that $\mathcal{S}$ proceeds by first computing a joint view $\langle(\sigma_1,\sigma_2),M\rangle$ (by running SIM) and then outputting this view only if EXT does not output `fail`. Below we show that the output of SIM is statistically close to the view of A in real interactions. This concludes that, since EXT outputs `fail` only in the event that the extraction fails, and since by the proof of knowledge property of $\langle P_{\tilde{\mathsf{tag}}}, V_{\tilde{\mathsf{tag}}}\rangle$ the extraction fails only with negligible probability, the first output of $\mathcal{S}$ is also statistically close to the view of A.

Let the random variable $\{\mathsf{SIM}(x,z,\text{TAG})\}$ denote the view $\langle(\sigma_1,\sigma_2),M\rangle$ output by $\{\mathsf{SIM}(x,z,\text{TAG})\}$ in the execution by $\mathcal{S}$.

CLAIM 5.7. *The probability ensembles* $\{\mathsf{SIM}(x,z,\text{TAG})\}_{x\in L,z\in\{0,1\}^*,\text{TAG}\in\{0,1\}^m}$ *and* $\{view_A(x,z,\text{TAG})\}_{x\in L,z\in\{0,1\}^*,\text{TAG}\in\{0,1\}^m}$ *are statistically close over* L.

Proof. Recall that the output of SIM consists of the tuple $\langle(\sigma_1,\sigma_2),M\rangle$, where (σ_1,σ_2) is the left view generated by the simulator S and $M=(\tilde{h},\tilde{r}_1,\tilde{r}_2,\tilde{u})$ are uniformly chosen messages that are fed to A during the simulation. In other words,

$$\{\mathsf{SIM}(x,z,\text{TAG})\}_{x,z,\text{TAG}}=\{(S(x,z,\text{TAG}),U_{|M|})\}_{x,z,\text{TAG}}.$$

Let $x\in L,\text{TAG}\in[2n]^n$ and $z\in\{0,1\}^*$. To prove the claim, we will thus compare the distribution $(S(x,z,\text{TAG}),U_{|M|})$ with real executions of the man-in-the-middle adversary $A(x,z,\text{TAG})$.

We start by observing that, whenever M is chosen randomly, a distinguisher between real and simulated views of the left interaction of A yields a distinguisher between $S(x,z,\text{TAG})$ and $view_A(x,z,\text{TAG})$. This follows from the following two facts (both facts are true regardless of whether $\mathsf{tag}_j=\tilde{\mathsf{tag}}$ for some $j\in[n]$):

1. The messages M (resp., M') appearing in its output are identically distributed to messages in a real right interaction of A with $V_{\tilde{\mathsf{tag}}}$ (by construction of SIM).
2. The simulation of the left interaction in SIM is done with respect to an A whose right-hand side view consists of the messages M (resp., M').

In particular, to distinguish whether a tuple $(\sigma_1,\sigma_2),M$ was drawn according to $S(x,z,\text{TAG})$ or according to $view_A(x,z,\text{TAG})$, one could simply take M, hardwire it into A, and invoke the distinguisher for the resulting stand-alone verifier for $\langle P_{\text{TAG}},V_{\text{TAG}}\rangle$ (which we denote by $V^*_{\tilde{\mathsf{tag}}}$). Thus, all we need to prove is the indistinguishability of real and simulated views of an arbitrary stand-alone verifier V^*_{TAG} (while ignoring the messages M). We now proceed with the proof of the claim.

Consider a random variable (σ_1,σ_2) that is distributed according to the output of $S(x,z,\text{TAG})$ and a random variable (π_1,π_2) that is distributed according to the verifier view of $\langle P_{\text{TAG}},V_{\text{TAG}}\rangle$ in $\{view_A(x,z,\text{TAG})\}_{z,x,\text{TAG}}$ (where π_1,π_2 are A's respective views of Stages 1 and 2). We will show that both (σ_1,σ_2) and (π_1,π_2) are statistically close to the hybrid distribution (σ_1,π_2). This distribution is obtained by considering a hybrid simulator that generates σ_1 exactly as S does but uses the witness w for $x\in L$ in order to produce π_2.

SUBCLAIM 5.8. *The distribution* (π_1,π_2) *is statistically close to* (σ_1,π_2).

Proof. The claim follows from the (parallel) statistical-hiding property of **Com**. Specifically, suppose that there exists a (possibly unbounded) D that distinguishes

between the two distributions with probability ϵ. Consider a distinguisher D' that has the witness w for $x \in L$ and $h = V^*_{\text{TAG}}(x)$ hardwired and acts as follows. Whenever D' gets an input $(\bar{c}_1, \bar{c}_2)$, it starts by generating $\bar{r}_1 = V^*_{\text{TAG}}(x, \bar{c}_1)$ and $\bar{r}_2 = V^*_{\text{TAG}}(x, \bar{c}_1, \bar{c}_2)$. It then emulates a Stage 2 interaction between $V^*_{\text{TAG}}(x, \bar{c}_1, \bar{c}_2)$ and the honest provers P_{sUA}, where $(x, \langle h, c_1^j, r_1^j\rangle, \langle h, c_2^j, r_2^j\rangle)$ is the common input for the jth interaction, and P_{sUA} is using w as a witness for $x \in L$. Let π_2 denote the resulting verifier view. D' outputs whatever D outputs on input $(\bar{c}_1, \bar{c}_2, \pi_2)$.

Notice that if $(c_1^j, c_2^j) = (\mathbf{Com}(0^n), \mathbf{Com}(0^n))$ for all $j \in [n]$, then the input fed to D is distributed according to (π_1, π_2), whereas if $(c_1^j, c_2^j) = (\mathbf{Com}(h(\Pi_1^j)), \mathbf{Com}(h(\Pi_2^j)))$, then the input fed to D is distributed according to (σ_1, π_2). Thus, D' has advantage ϵ in distinguishing between two tuples of n committed values. Hence, if ϵ is nonnegligible, we reach contradiction to the statistical-hiding property of **Com**. □

SUBCLAIM 5.9. *The distribution (σ_1, σ_2) is statistically close to (σ_1, π_2).*

Proof. The claim follows from the (parallel) statistical witness-indistinguishability property of $\langle P_{sWI}, V_{sWI}\rangle$ and the (parallel) statistical-hiding property of **Com** (both used in $\langle P_{sUA}, V_{sUA}\rangle$). Let $\sigma_2 = (\sigma_{2,1}, \sigma_{2,2})$, where $\sigma_{2,1}$ corresponds to a simulated view of Stage 1 of $\langle P_{sUA}, V_{sUA}\rangle$ and $\sigma_{2,2}$ corresponds to a simulated view of Stage 2 of $\langle P_{sUA}, V_{sUA}\rangle$. Similarly, let $\pi_2 = (\pi_{2,1}, \pi_{2,2})$ correspond to the real views of Stages 1 and 2 of $\langle P_{sUA}, V_{sUA}\rangle$. We will show that both (σ_1, σ_2) and (σ_1, π_2) are statistically close to the hybrid distribution $(\sigma_1, (\sigma_{2,1}, \pi_{2,2}))$.

Suppose that there exists a (possibly unbounded) algorithm D that distinguishes between (σ_1, σ_2) and $(\sigma_1, (\sigma_{2,1}, \pi_{2,2}))$ with probability ϵ. Then there must exist a Stage 1 view $(\bar{c}_1, \bar{c}_2) = (c_1^1, \ldots, c_1^n, c_2^1, \ldots, c_2^n)$ for $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ and a Stage 1 view $(\bar{\beta}, \bar{\delta}) = (\widehat{\beta}^1, \ldots, \beta^n, \widehat{\delta}^1, \ldots, \widehat{\delta}^n)$ for the subprotocol $\langle P_{sUA}, V_{sUA}\rangle$ so that D has advantage ϵ in distinguishing between $\langle(\sigma_1, \sigma_2)\rangle$ and $\langle(\sigma_1, \pi_2)\rangle$ conditioned on $(\sigma_1, \sigma_{2,1}) = ((\bar{c}_1, \bar{c}_2), (\bar{\beta}, \bar{\delta}))$.

Consider a Stage 2 execution of $\langle P_{sUA}, V_{sUA}\rangle$ with $V^*_{sWI} = V^*_{\text{TAG}}(x, \bar{c}_1, \bar{c}_2, \bar{\beta}, \bar{\delta}, \cdot)$ as verifier. Then a distinguisher $D(\bar{c}_1, \bar{c}_2, \bar{\beta}, \bar{\delta}, \cdot)$ (i.e., D with $(\bar{c}_1, \bar{c}_2, \bar{\beta}, \bar{\delta})$ hardwired as part of its input) has advantage ϵ in distinguishing between an interaction of V^*_{sWI} with n honest P_{sWI} provers that use accepting $\langle P_{UA}, V_{UA}\rangle$ transcripts $(\alpha, \beta, \gamma, \delta)$ and an interaction of V^*_{sUA} with honest P_{sUA} provers that use w as witness.[13] Thus, if ϵ is nonnegligible, we reach contradiction to the (parallel) statistical witness indistinguishability of $\langle P_{sWI}, V_{sWI}\rangle$.

Now suppose that there exists a (possibly unbounded) algorithm D that distinguishes between (σ_1, π_2) and $(\sigma_1, (\sigma_{2,1}, \pi_{2,2}))$ with probability ϵ. Consider a distinguisher D' that has the witness w for $x \in L$ and $(h, \bar{c}_1, \bar{c}_2)$ hardwired and acts as follows. Whenever D' gets an input $(\bar{\beta}, \bar{\delta})$, it starts by generating $\bar{\alpha} = V^*_{\text{TAG}}(x, \bar{c}_1, \bar{c}_2)$ and $\bar{\gamma} = V^*_{\text{TAG}}(x, \bar{c}_1, \bar{c}_2, \bar{\beta})$. It then emulates a Stage 2 interaction between the V_{sWI} verifiers $V^*_{\text{TAG}}(x, \bar{c}_1, \bar{c}_2, \bar{\beta}, \bar{\delta})$ and the honest provers P_{sWI}, where $(x, \langle h, c_1^j, r_1^j\rangle, \langle h, c_2^j, r_2^j\rangle, \langle \alpha, \bar{\beta}, \gamma, \bar{\delta}\rangle)$ is the common input for the jth interaction, and P_{sWI} is using w as a witness for $x \in L$. Let $\pi_{2,2}$ denote the resulting verifier view. D' outputs whatever D outputs on input $(\bar{c}_1, \bar{c}_2, \bar{\beta}, \bar{\delta}, \pi_{2,2})$.

Notice that if $(\widehat{\beta}^j, \widehat{\delta}^j) = (\mathbf{Com}(0^n), \mathbf{Com}(0^n))$ for all $j \in [n]$, then the input fed to D is distributed according to $(\sigma_1, (\pi_{2,1}, \pi_{2,2})) = (\sigma_1, \pi_2)$, whereas if $(\widehat{\beta}^j, \widehat{\delta}^j) = \mathbf{Com}(\beta^j), \mathbf{Com}(\delta^j)\rangle$, for some β^j, δ^j for which $(\alpha^j, \beta^j, \gamma^j, \delta^j)$ is an accepting $\langle P_{UA}, V_{UA}\rangle$, then the input fed to D is distributed according to $(\sigma_1, (\sigma_{2,1}, \pi_{2,2}))$.

[13] In accordance with the specification of $\langle P_{sUA}, V_{sUA}\rangle$, the transcripts $(\alpha, \beta, \gamma, \delta)$ are generated using programs Π_i^j as witnesses, where Π_i^j is the program chosen by the simulator S_j.

Thus, D' has advantage ϵ in distinguishing between two tuples of n committed values. Hence, if ϵ is nonnegligible, we reach contradiction to the statistical-hiding property of **Com**. □

Combining Subclaims 5.9 and 5.8 we conclude that $\{\mathsf{SIM}(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ and $\{view_A(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ are statistically close. □

This completes the proof of Lemma 5.6. □

LEMMA 5.10. *Let $x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^m$, and let $(view, w)$ denote the output of $\mathcal{S}(x, z, \text{TAG})$ (on input some random tape). Let $\tilde{x}$ be the right-execution statement appearing in view, and let $\tilde{\mathsf{tag}}$ denote the right-execution tag. Then, if the right execution in view is accepting AND $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$ for all $j \in [n]$, then $R_L(\tilde{x}, w) = 1$.*

Proof. We start by noting that, since $\mathcal{S}$ outputs `fail` whenever the extraction by EXT fails, the claim trivially holds in the event that the extraction by EXT fails.

Observe that the right-hand side input-tag pair $(\tilde{x}, \tilde{\mathsf{tag}})$ used in EXT is exactly the same as the one generated by SIM. This follows from the following two reasons: (1) Both EXT and SIM use the same random coins $\bar{s}$ in the simulation. (2) The input-tag pair $\tilde{x}, \tilde{\mathsf{tag}}$ is determined before any external message is received in the right interaction. In particular, the pair $(\tilde{x}, \tilde{\mathsf{tag}})$ is *independent* of the messages M (which is the only potential difference between the executions of SIM and EXT).

Since whenever the properties described in the hypothesis hold EXT performs extraction, and since the extraction by EXT proceeds until a witness is extracted (or until the extraction fails, in which case we are already done), we infer that $\mathcal{S}$ always outputs a witness to the statement $\tilde{x}$ proved in the right interaction in the view output. □

We conclude the proof by bounding the running time of the combined simulator-extractor $\mathcal{S}$.

LEMMA 5.11. *$\mathcal{S}$ runs in expected polynomial time.*

Proof. We start by proving that the running time of SIM is polynomial. Recall that the SIM procedure invokes the simulator S with the adversary A'. Thus, we need to show that S runs in polynomial time. To this end, it will be sufficient to show that every individual subsimulator S_j runs in polynomial time. We first do so assuming that $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$. We then argue that, by construction of the adversary A', this will be sufficient to guarantee polynomial running time even in cases where $\mathsf{tag}_j = \tilde{\mathsf{tag}}$.

CLAIM 5.12. *Suppose that $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$. Then S_j completes the simulation in polynomial time.*

Proof. We start by arguing that, in each of the three cases specified in the simulation, the witness used by the simulator indeed satisfies the relation $\mathbf{R}_{sim}$. A close inspection of the simulator's actions in the first two cases reveals that the simulator indeed commits to a program Π_i that on input $y = (c_i^1, \ldots, c_i^n)$ outputs the corresponding r_i. Namely, the following hold:

- $\Pi_1(y) = A_j(x, c_1^1, \ldots, c_1^n) = r_1^j$.
- $\Pi_2(y) = A_j(x, c_1^1, \ldots, c_1^n, \tilde{r_1}, \tilde{r_2}, c_2^1, \ldots, c_2^n) = r_2^j$.

Since in both cases $|y| = n|c_j^i| = 2n^2 \leq \ell(n) - n \leq |r_i^j| - n$ it follows that $\mathbf{R}_{sim}$ is satisfied. As for the third case, observe that, for both $i \in \{1, 2\}$, if S_j sets $y = (c_i^1, \ldots, c_i^n, \tilde{r}_i)$, then the following holds:

- $\Pi_i(y) = A_j(c_i^1, \ldots, c_i^n, \tilde{r}_i) = r_i^j$.

Since $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$ we can use Fact 5.2 and infer that there exists $i \in \{1, 2\}$ so that $|\tilde{r}_i| \leq |r_i^j| - \ell(n)$. This means that for every $j \in [n]$ the simulator S_j will choose the

$i \in \{1, 2\}$ for which

$$\begin{aligned} |y| &= |c_i^1| + \cdots + |c_i^n| + |\tilde{r}_i^i| \\ &= 2n^2 + |\tilde{r}_i^i| \\ &\leq 2n^2 + |r_i^j| - \ell(n) \qquad (5.1) \\ &\leq |r_i^j| - n, \qquad (5.2) \end{aligned}$$

where (5.1) follows from $|\tilde{r}_i| \leq |r_i^j| - \ell(n)$ and (5.2) follows from the fact that $\ell(n) \geq 2n^2 + n$. Thus, $\mathbf{R}_{sim}$ can always be satisfied by S_j.

Since the programs Π_i are of size poly(n) and satisfy $\Pi_i(y) = r_i^j$ in poly(n) time (because A_j does), the verification time of $\mathbf{R}_{sim}$ on the instance $\langle h, c_i^j, r_i^j \rangle$ is polynomial in n. By the perfect completeness and relative prover efficiency of $\langle P_{sUA}, V_{sUA} \rangle$, it then follows that the simulator is always able to make a verifier accept in polynomial time. □

Remark 5.13 (handling the case $\mathsf{tag}_j = \tilde{\mathsf{tag}}$). When invoked by SIM, the simulator S will output an accepting left view even if A chooses $\tilde{\mathsf{tag}}$ so that $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ for some $j \in [n]$.[14] This is because in such a case the A' whose view S needs to simulate ignores all right-hand side messages and feeds the messages M' to A internally. In particular, no external messages will be contained in either Slot 1 or Slot 2 of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$. A look at the proof of Claim 5.12 reveals that in such cases the simulator can indeed always produce an accepting conversation (regardless of whether $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ or not).

It now remains to bound the *expected* running time of EXT. Recall that EXT uses the view sampled by SIM and proceeds to extracting a witness only if the right interaction is accepting and $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$ for all $j \in [n]$. Using Claim 5.12, we know that the simulation internally invoked by the stand-alone prover $P^*_{\tilde{\mathsf{tag}}}$ will always terminate in polynomial time (since $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}$ for all $j \in [n]$ and A is a poly-time machine). We now argue that the extraction of the witness from $P^*_{\tilde{\mathsf{tag}}}$ conducted by EXT will terminate in expected polynomial time.

Let p denote the probability that A produces an accepting proof in the right execution in the simulation by SIM. Let p' denote the probability that A produces an accepting proof in the right execution in the internal execution of $P^*_{\tilde{\mathsf{tag}}}$ (constructed in EXT). By the POK property of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ it holds that the expected running time of the knowledge extractor is bounded by

$$\frac{\text{poly}(n)}{p'}.$$

Since the probability of invoking the extraction procedure is p, the expected number of steps used to extract a witness is

$$p\frac{\text{poly}(n)}{p'}.$$

Now in both SIM and EXT the left view is generated by S, and the right view is uniformly chosen. This in particular means that $p = p'$. It follows that the expected

[14] Note that this is not necessarily true in general. For example, when $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ for some $j \in [n]$, and the messages that A sees are forwarded from an external source (e.g., when S is used by EXT in order to construct the stand alone prover $P^*_{\tilde{\mathsf{tag}}}$), we cannot guarantee anything about the running time of S. Indeed, the definition of simulation extractability does not require EXT to output a witness when $\mathsf{tag}_j = \tilde{\mathsf{tag}}$ for some $j \in [n]$.

number of steps used to extract a witness is

$$p\frac{\text{poly}(n)}{p} = \text{poly}(n).$$

This completes the proof of Lemma 5.11. □

This completes the proof of "many-to-one" simulation extractability (Proposition 5.3). □

5.3. "Full-fledged" simulation extractability. Let $\text{TAG} \in \{0,1\}^m$, let $x \in \{0,1\}^n$, and let A be the corresponding MIM adversary. We consider a left interaction in which $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ is executed with common input $x \in \{0,1\}^n$ and a right interaction in which $\langle P_{\tilde{\text{TAG}}}, V_{\tilde{\text{TAG}}}\rangle$ is executed with common input $\tilde{x} \in \{0,1\}^n$. The strings $\tilde{\mathsf{tag}}$ and $\tilde{x}$ are chosen adaptively by the man-in-the-middle adversary A. The witness used by the prover in the left interaction is denoted by w, and the auxiliary input used by the adversary is denoted by z.

PROPOSITION 5.14. *Let A be a MIM adversary as above, and suppose that $\ell(n) \geq 2n^2 + 2n$. Then there exists a probabilistic expected polynomial-time machine $\mathcal{S}$ such that the following conditions hold:*

1. *The probability ensembles $\{\mathcal{S}_1(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ and $\{view_A(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ are statistically close over L, where $\mathcal{S}_1(x, z, \text{TAG})$ denotes the first output of $\mathcal{S}(x, z, \text{TAG})$.*
2. *Let $x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^{t(|x|)}$, and let $(view, w)$ denote the output of $\mathcal{S}(x, z, \text{TAG})$ (on input some random tape). Let $\tilde{x}$ be the right-execution statement appearing in view, and let $\tilde{\text{TAG}}$ denote the right-execution tag. Then, if the right execution in view is accepting AND $\text{TAG} \neq \tilde{\text{TAG}}$, then $R_L(\tilde{x}, w) = 1$.*

Proof. The construction of the simulator-extractor $\mathcal{S}$ proceeds in two phases and makes use of the many-to-one simulator-extractor guaranteed by Proposition 5.3. In the first phase, the adversary A is used in order to construct a many-to-one adversary A' with the protocol $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ on its left and with one of the subprotocols $\langle P_{\tilde{\mathsf{tag}}_i}, V_{\tilde{\mathsf{tag}}}\rangle$ on its right. In the second phase, the many-to-one simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ is used in order to generate a view for A along with a witness for the statement $\tilde{x}$ appearing in the simulation.

The many-to-one adversary. On input (x, z, TAG), and given a man-in-the-middle adversary A, the many-to-one adversary A' acts as follows:

Internal messages: For all $j \in [n]$, pick random messages $(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)$ for the right interaction.

Right interaction: The statement $\tilde{x}$ proved is the same as the one chosen by A. If there exists $i \in [n]$ so that $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}_i$ for all $j \in [n]$, forward A's messages in the ith right interaction to an external $V_{\tilde{\mathsf{tag}}_i}$ and send back his answers to A. Use the messages $\{(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)\}_{j\neq i}$ to internally emulate all other right interactions $\{\langle P_{\tilde{\mathsf{tag}}_j}, V_{\tilde{\mathsf{tag}}_j}\rangle\}_{j\neq i}$.

Otherwise (i.e., if for all $i \in [n]$ there exists $j \in [n]$ such that $\mathsf{tag}_j = \tilde{\mathsf{tag}}_i$), pick an arbitrary $i \in [n]$, forward A's messages in the ith right interaction to an external $V_{\tilde{\mathsf{tag}}_i}$, and send back his answers to A. Use the messages $\{(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)\}_{j\neq i}$ to internally emulate all other right interactions $\{\langle P_{\tilde{\mathsf{tag}}_j}, V_{\tilde{\mathsf{tag}}_j}\rangle\}_{j\neq i}$.

Left interaction: As induced by the scheduling of messages by A, forward the messages sent by A in the left interaction to an external prover P_{TAG} and send back his answers to A.

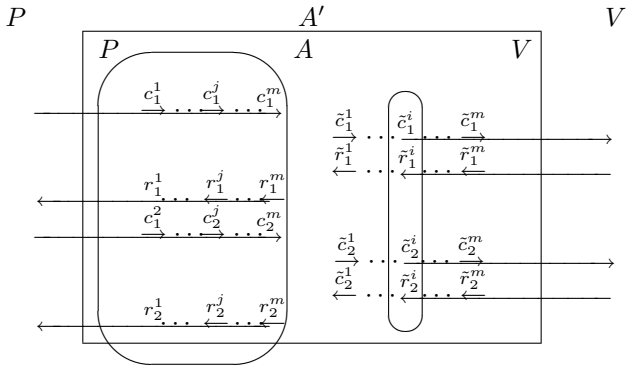


FIG. 13. *The "many-to-one" MIM adversary A'.*

The many-to-one adversary A' is depicted in Figure 13. Messages from circled subprotocols are the ones that get forwarded externally.

The simulator-extractor. By Proposition 5.3 there exists a simulator $\mathcal{S}'$ that produces a view that is statistically close to the real view of A' and outputs a witness, provided that the right interaction is accepting and $\tilde{\mathsf{tag}}$ is different from all the left-side tags $\mathsf{tag}_1, \ldots, \mathsf{tag}_n$.

The simulator-extractor $\mathcal{S}(x, z, \text{TAG})$ for A invokes $\mathcal{S}'(x, z, \text{TAG})$. If $\mathcal{S}'$ outputs `fail`, so does $\mathcal{S}$. Otherwise, let $view', w'$ denote the output of $\mathcal{S}'$. $\mathcal{S}$ now outputs $view, w'$, where $view$ is the view $view'$ (output by $\mathcal{S}'$) augmented with the right-hand side messages $\{(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)\}_{j \neq i}$ that were used in the internal emulation of A'.

CLAIM 5.15. *$\mathcal{S}$ runs in expected polynomial time.*

Proof. Notice that whenever A is polynomial time, then so is A'. By Lemma 5.11, this implies that $\mathcal{S}'$ runs in expected polynomial time, and hence so does $\mathcal{S}$. □

CLAIM 5.16. *The probability ensembles $\{\mathcal{S}_1(x, z, \text{TAG})\}_{x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^m}$ and $\{view_A(x, z, \text{TAG})\}_{x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^m}$ are statistically close over L, where $\mathcal{S}_1(x, z, \text{TAG})$ denotes the first output of $\mathcal{S}(x, z, \text{TAG})$.*

Proof. Given a distinguisher D between ensembles $\{\mathcal{S}_1(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ and $\{view_A(x, z, \text{TAG})\}_{z,x,\text{TAG}}$, we construct a distinguisher D' between $\{\mathcal{S}_1'(x, z, \text{TAG})\}_{x,z,\text{TAG}}$ and $\{view_{A'}(x, z, \text{TAG})\}_{x,z,\text{TAG}}$. This will be in contradiction to Lemma 5.6. The distinguisher D' has the messages $\{(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)\}_{j \neq i}$ hardwired.[15] Given a joint view $\langle(\sigma_1', \sigma_2'), M'\rangle$ of a left $\langle P_{\tilde{\text{TAG}}}, V_{\tilde{\text{TAG}}}\rangle$ interaction and a $\langle P_{\tilde{\mathsf{tag}}_i}, V_{\tilde{\mathsf{tag}}_i}\rangle$ right interaction, D' augments the view with the right interaction messages $\{(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)\}_{j \neq i}$. The distinguisher D' feeds the augmented view to D and outputs whatever D outputs.

Notice that if $\langle(\sigma_1', \sigma_2'), M'\rangle$ is drawn according to $\{\mathcal{S}_1'(x, z, \text{TAG})\}_{x,z,\text{TAG}}$, then the augmented view is distributed according to $\{\mathsf{SIM}(x, z, \text{TAG})\}_{x,z,\text{TAG}}$. On the other hand, if $\langle(\sigma_1', \sigma_2'), M'\rangle$ is drawn according to $\{view_{A'}(x, z, \text{TAG})\}_{z,x,\text{TAG}}$, then the augmented view is distributed according to $\{view_A(x, z, \text{TAG})\}_{z,x,\text{TAG}}$. Thus D' has exactly the same advantage as D. □

CLAIM 5.17. *Let $x \in L, z \in \{0,1\}^*, \text{TAG} \in \{0,1\}^m$, and let $(view, w)$ denote the output of $\mathcal{S}(x, z, \text{TAG})$ (on input some random tape). Let $\tilde{x}$ be the right-execution statement appearing in view, and let $\tilde{\text{TAG}}$ denote the right-execution tag. Then, if the right execution in view is accepting AND $\text{TAG} \neq \tilde{\text{TAG}}$, then $R_L(\tilde{x}, w) = 1$.*

[15] One could think of D as a family of distinguishers that is indexed by $\{(\tilde{h}^j, \tilde{r}_1^j, \tilde{r}_2^j, \tilde{u}^j)\}_{j \neq i}$ and from which a member is drawn at random.

Proof. Recall that the right interaction in the view output of $\mathcal{S}$ is accepting if and only if the right interaction in the view output of $\mathcal{S}$ is accepting. In addition, the statement proved in the right interaction output by $\mathcal{S}$ is identical to the one proved in the right interaction of $\mathcal{S}'$.

Observe that if $\text{TAG} \neq \tilde{\text{TAG}}$, there must exist $i \in [n]$ for which $(i, \tilde{\text{TAG}}_i) \neq (j, \text{TAG}_j)$ for all $j \in [n]$ (just take the i for which $\tilde{\text{TAG}}_i \neq \text{TAG}_i$). Recall that by construction of the protocol $\langle P_{\tilde{\text{TAG}}}, V_{\tilde{\text{TAG}}} \rangle$, for every $i \in [n]$, the value tag_i is defined as (i, TAG_i). Thus, whenever $\text{TAG} \neq \tilde{\text{TAG}}$ there exists a $\tilde{\mathsf{tag}}_i = (i, \tilde{\text{TAG}}_i)$ that is different from $\mathsf{tag}_j = (j, \text{TAG}_j)$ for all $j \in [n]$. In particular, the tag used by A' in the right interaction will satisfy $\mathsf{tag}_j \neq \tilde{\mathsf{tag}}_i$ for all $j \in [n]$. By Lemma 5.10, we then have that if the right interaction in view output by $\mathcal{S}$ is accepting (and hence also in $\mathcal{S}'$), then $\mathcal{S}'$ will output a witness for $\tilde{x}$. The proof is complete, since $\mathcal{S}$ outputs whatever witness $\mathcal{S}'$ outputs. □

This completes the proof of Proposition 5.14. □

6. Nonmalleable commitments. In this section we present two simple constructions of nonmalleable commitments. The approach we follow is different from the approach used in [14]. Instead of viewing nonmalleable commitments as a tool for constructing nonmalleable zero-knowledge protocols, we reverse the roles and use a nonmalleable zero-knowledge protocol (in particular any simulation-extractable protocol will do) in order to construct a nonmalleable commitment scheme. Our approach is also different from the approach taken by [2].

6.1. A statistically binding scheme (NM with respect to commitment). We start by presenting a construction of a statistically binding scheme which is nonmalleable with respect to commitment. Our construction relies on the following two building blocks:

- a family of (possibly malleable) noninteractive statistically binding commitment schemes,
- a simulation-extractable zero-knowledge argument.

The construction is conceptually very simple: The committer commits to a string using the statistically binding commitment scheme and then proves knowledge of the string committed to using a simulation-extractable argument.

We remark that the general idea behind this protocol is not new. The idea of enhancing a commitment scheme with a proof of knowledge protocol was already explored in [14]. However, as pointed out in [14], this approach cannot work with *any* proof of knowledge protocol, as this proof of knowledge protocol itself might be malleable. (As mentioned above, Dolev, Dwork, and Naor therefore rely on a quite different approach to construct nonmalleable commitments [14].) What we show here is that this approach in fact works if the proof of knowledge protocol is simulation extractable.

Let $\{\text{Com}_r\}_{r \in \{0,1\}^*}$ be a family of *noninteractive* statistically binding commitment schemes (e.g., Naor's commitment [31]), and let $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ be a simulation-extractable protocol. Consider the protocol in Figure 14.[16]

We start by sketching why the scheme is nonmalleable. Note that the commit phase of the scheme consists only of a message specifying an NP statement (i.e., the "statement" $c = \text{Com}_r(v; s)$) and an accompanying "proof" of this statement. Thus,

[16] It is interesting to note that the protocol $\langle C, R \rangle$ is statistically binding even though the protocol $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ is "only" computationally sound. At first sight, this is somewhat counterintuitive since the statistical-binding property is typically associated with all-powerful committers.

Protocol $\langle C, R\rangle$
Security Parameter: 1^n.
String to be committed to: $v \in \{0,1\}^n$.
Commit Phase:
$R \to C$: Pick uniformly $r \in \{0,1\}^n$.
$C \to R$: Pick $s \in \{0,1\}^n$ and send $c = \mathrm{Com}_r(v; s)$.
$C \leftrightarrow R$: Let $\mathrm{TAG} = (r, c)$. Prove using $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$ that there exist $v, s \in \{0,1\}^n$ so that $c = \mathrm{Com}_r(v; s)$. Formally, prove the following statement (r, c) with respect to the witness relation:

$$R_L = \{((r, c), (v, s)) | c = \mathrm{Com}_r(v; s)\}.$$

R: Verify that $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$ is accepting.
Reveal Phase:
$C \to R$: Send v, s.
R: Verify that $c = \mathrm{Com}_r(v; s)$.

FIG. 14. *A statistically binding nonmalleable string commitment protocol* $\langle C, R\rangle$.

intuitively, an adversary that is able to successfully maul a commitment must be able to maul both the commitment Com and also the accompanying proof $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$. The former might be easy, as Com might be malleable. However, as $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$ is simulation extractable (and thus nonmalleable), the latter will be impossible.

At first sight, it seems like it would be sufficient to simply assume that $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$ is nonmalleable to conclude nonmalleability of $\langle C, R\rangle$. Note, however, that the definition of a nonmalleable proof considers only a setting where the statement proven by the man-in-the-middle adversary is fixed ahead of the interaction. In our scenario, we instead also require security with respect to an *adaptively* chosen statement (as the statement proven is related to the commitment chosen by the adversary). This gap is addressed in the definition of simulation extractability; here the man-in-the-middle adversary is also allowed to adaptively choose the statements it will attempt to prove.

Interestingly, to formalize the above argument, we will be required to rely on the *statistical indistinguishability* property of the definition of simulation extractability. Intuitively, the reason for this is the following. For any given man-in-the-middle adversary we are required to construct a stand-alone adversary that succeeds in committing to "indistinguishable" values. In proving that this stand-alone adversary succeeds in this task, we will rely on the simulator-extractor for the man-in-the-middle adversary; however, to do this, we need to make sure that the simulator-extractor indeed will commit to indistinguishable values. Note that it is not sufficient that the simulator-extractor simply proves a statement that is indistinguishable from the statement proved by the man-in-the-middle adversary, as the value committed to is not efficiently computable from the statement. However, the value committed to is computable (although not efficiently) from the statement. Thus, by relying on the statistical indistinguishability property of the simulator-extractor, we can also make sure that the value committed by the simulator-extractor is indistinguishable from that committed to by the man-in-the-middle adversary.[17]

THEOREM 6.1 ($nm\mathcal{C}$ with respect to commitment). *Suppose that* $\{\mathrm{Com}_r\}_{r\in\{0,1\}^*}$ *is a family of* noninteractive *statistically binding commitment schemes and that* $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$ *is simulation extractable. Then* $\langle C, R\rangle$ *is a statistically binding nonmal-*

[17]Note that in the case of nonmalleability with respect to opening, this complication does not arise.

leable commitment scheme with respect to commitment.

Proof. We need to prove that the scheme satisfies the following three properties: statistical binding, computational hiding, and nonmalleability with respect to commitment.

Statistical binding. The binding property of the scheme follows directly from the binding property of the underlying commitment scheme Com: to break the binding property of $nm\mathcal{C}_{\text{TAG}}$ requires first breaking the binding of Com. More formally, given any adversary A that breaks the binding property of $nm\mathcal{C}_{\text{TAG}}$, we construct an adversary A', which on input a message r feeds r to A and then internally honestly emulates all the verification messages in the proof part of $nm\mathcal{C}_{\text{TAG}}$. It follows directly that the success probability of A' is the same as that of A.

Computational hiding. The hiding property follows from the hiding property of Com combined with the $\mathcal{ZK}$ property of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$. More formally, recall that the notion of simulation extractability implies $\mathcal{ZK}$ (see Proposition 3.6) and that $\mathcal{ZK}$ implies *strong* witness indistinguishability[18] [18]. Since the scheme Com produces indistinguishable commitments, it thus follows directly by the definition of strong witness indistinguishability that the protocol $\langle C, R\rangle$ also produces indistinguishable commitments.

Nonmalleability. Consider a man-in-the middle adversary A. We assume without loss of generality that A is deterministic (this is without loss of generality since A can obtain its "best" random tape as auxiliary input). We show the existence of a probabilistic polynomial-time stand-alone adversary S and a negligible function $\nu : N \to N$ such that, for every irreflexive polynomial-time computable relation $\mathcal{R} \subseteq \{0,1\}^n \times \{0,1\}^n$, every $v \in \{0,1\}^n$, and every $z \in \{0,1\}^*$, it holds that

$$\Pr\left[\mathsf{mim}^A_{com}(\mathcal{R}, v, z) = 1\right] \;<\; \Pr\left[\mathsf{sta}^S_{com}(\mathcal{R}, v, z) = 1\right]. \;+\; \nu(n). \tag{6.1}$$

Description of the stand-alone adversary. The stand-alone adversary S uses A as a black box and emulates the left and right interactions for A as follows: the left interaction is emulated internally, while the right interaction is forwarded externally. More precisely, S proceeds as follows on input z. S incorporates $A(z)$ and internally emulates the left interactions for A by simply *honestly* committing to the string 0^n; i.e., to emulate the left interaction, S executes the algorithm C on input 0^n. Messages from the right interactions are instead forwarded externally. Note that S is thus a stand-alone adversary that expects to act as a committer for the scheme $\langle C, R\rangle$.

Analysis of the stand-alone adversary. We proceed to showing that (6.1) holds. Suppose, for contradiction, that this is not the case. That is, there exist an irreflexive polynomial-time relation $\mathcal{R}$ and a polynomial $p(n)$ such that, for infinitely many n, there exist strings $v \in \{0,1\}^n, z \in \{0,1\}^*$ such that

$$\Pr\left[\mathsf{mim}^A_{com}(\mathcal{R}, v, z) = 1\right] - \Pr\left[\mathsf{sta}^S_{com}(\mathcal{R}, v, z) = 1\right] \geq \frac{1}{p(n)}.$$

Fix generic n, v, z for which the above holds. We show how this contradicts the simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$. On a high level, our proof consists of the following steps:

[18] Intuitively, the notion of strong witness indistinguishability requires that proofs of indistinguishable statements are indistinguishable. This is, in fact, exactly the property we need in order to prove that the commitment scheme is computationally hiding.

1. We first note that, since the commit phase of $\langle C, R\rangle$ "essentially" consists only of a statement (r, c) (i.e., the commitment) and a proof of the "validity" of (r, c), A can be interpreted as a simulation-extractability man-in-the-middle adversary A' for $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$.
2. It follows from the simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ that there exists a combined simulator-extractor $\mathcal{S}$ for A' that outputs a view that is statistically close to that of A' while at the same time outputting a witness to all accepting right proofs which use a different right tag $\tilde{\text{TAG}}$ than the tag TAG of the left interaction, i.e., if $\tilde{\text{TAG}} \neq \text{TAG}$.
3. Since the view output by the simulator-extractor $\mathcal{S}$ is *statistically* close to the view of A' in the real interaction, it also follows that the *value* committed to in that view is statistically close to value committed to by A'. (Note that computational indistinguishability would not have been enough to argue the indistinguishability of these values, since they are not efficiently computable from the view.)
4. It also follows that the simulator-extractor $\mathcal{S}$ will also output the witness to each *accepting* right execution such that $\tilde{\text{TAG}} \neq \text{TAG}$. We conclude that $\mathcal{S}$ additionally outputs the value *committed to* in the right execution (except possibly when the value committed to in the right interaction is the same as that committed to in the left).
5. We finally note that if $\mathcal{R}$ (which is irreflexive) "distinguishes" between the value committed to by A and by S, then $\mathcal{R}$ also "distinguishes" the second output of $\mathcal{S}$ (which consists of the committed values) when run on input a commitment (using Com) to v and the second output of $\mathcal{S}$ when run on input a commitment to 0. But, this contradicts the hiding property of Com.

We proceed to a formal proof. One particular complication that arises with the above proof sketch is that in the construction of $\langle C, R\rangle$ we are relying on a family of commitment schemes $\{\text{Com}_r\}_{r\in\{0,1\}^*}$ and not a single noninteractive commitment scheme. Thus, strictly speaking, A is not a man-in-the-middle adversary for the interactive proof $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$. However, the only difference is that A additionally expects to receive a message $\tilde{r}$ on the right and also to send a message r on the left. To get around this problem we rely on the *nonuniform* computational hiding property of $\langle C, R\rangle$.

Note that, since in both experiments mim and sta the right execution is identically generated, there must exist some *fixed* message $\tilde{r}$ such that, conditioned on the event that the first message sent in the right execution is $\tilde{r}$, it holds that the success probability in $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$ is $\frac{1}{p(n)}$ higher than in $\mathsf{sta}^S_{com}(\mathcal{R}, v, z)$. In fact, by the statistical-binding property of Com, it follows that there must exist some message $\tilde{r}$ such that $\text{Com}_{\tilde{r}}$ is *perfectly binding*, and, additionally, conditioned on the event that the first message sent in the right execution is $\tilde{r}$, it holds that the success probability in $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$ is $\frac{1}{2p(n)}$ higher than in $\mathsf{sta}^S_{com}(\mathcal{R}, v, z)$.

Given this message $\tilde{r}$, we must now consider two cases:

1. A (which by assumption is deterministic) sends its first message r in the left interaction directly after receiving $\tilde{r}$ (see Figure 15).
2. Or A first send its first message $\tilde{c}$ in the right interaction.

We first show that, in the second case, A can be used to break the nonuniform hiding property of $\langle C, R\rangle$. Intuitively, this follows from the fact that the value committed to by A on the right is "essentially" determined by the message $\tilde{c}$ sent by A before it has received a single message on the left; it is not "fully" determined by $\tilde{c}$,

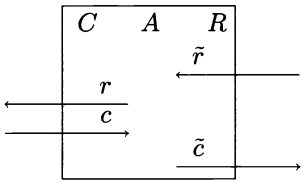


FIG. 15. *An interleaved scheduling of commitments.*

as A can decide whether it fails in the interactive proof and thus potentially change the value determined by $\tilde{c}$ to $\perp$. However, in this case, A can be used to violate the hiding property of $\langle C, R\rangle$ (as the probability of failing the proof must depend on the value it receives a commitment to).

More formally, let $v_{\tilde{c}}$ denote the value committed to in $\tilde{c}$ (using Com). It then holds that the value committed to by A in its right interaction (of $\langle C, R\rangle$) will be $v_{\tilde{c}}$ if A succeeds in the proof following the message $\tilde{c}$ and $\perp$ otherwise. By our assumption that the success probability in $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$ is $\frac{1}{2p(n)}$ higher than in $\mathsf{sta}^S_{com}(\mathcal{R}, v, z)$, conditioned on the event that the first message sent in the right execution is $\tilde{r}$, it thus holds that A "aborts" the proof in the left interaction with different probability in experiments $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$ and $\mathsf{sta}^S_{com}(\mathcal{R}, v, z)$, conditioned on the first message in the right interaction being $\tilde{r}$. However, since the only difference in those experiments is that A receives a commitment to v in mim and a commitment to 0^n in sta, we conclude that A contradicts the (nonuniform) computational-hiding property of Com. Formally, we construct a *nonuniform* distinguisher D for the commitment scheme $\langle C, R\rangle$: D incorporates A, z, r, v, and $v_{\tilde{c}}$, emulates the right execution for A by honestly running the verification procedure of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$, and forwards messages in the left execution externally. D finally outputs $\mathcal{R}(v, v_{\tilde{c}})$ if the proof was accepting and 0 otherwise. The claim follows from the fact that D perfectly emulates $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$ when receiving a commitment to v and perfectly emulates $\mathsf{sta}^S_{com}(\mathcal{R}, v, z)$ when receiving a commitment to 0^n.

We proceed to considering the first (and harder) case depicted in Figure 15, i.e., when A sends its first left message r directly after receiving the message $\tilde{r}$. In this case, we instead directly use A to contradict the hiding property of Com. Towards this goal, we proceed in three steps:

1. We first define a simulation-extractability adversary A'.
2. We next show that A' can be used to violate the nonmalleability property of $\langle C, R\rangle$.
3. In the final step, we show how to use the simulator-extractor $\mathcal{S}$ for A' to violate the hiding property of Com.

Step 1: *Defining a simulation-extractability adversary* A'. We define a simulation-extractability adversary A' for $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$. On input $x, \text{TAG}, z' = (z, \tilde{r})$, A' internally incorporates $A(z)$ and emulates the left and right interactions for A as follows.

1. A' starts by feeding A the message $\tilde{r}$ as part of its right execution. All remaining messages in the right execution are forwarded externally.
2. All messages in A's left interaction are forwarded externally as part of A''s left interaction, except for the *first* message r.

Step 2: *Show that* A' *violates nonmalleability of* $\langle C, R\rangle$. Towards the goal of showing that A violates nonmalleability of $\langle C, R\rangle$, we define the hybrid experiment $\mathsf{hyb}_1(v')$ (relying on the definition of $v, z, r, \tilde{r}$):

1. Let s be a uniform random string, and let $c = \mathrm{Com}_r(v'; s)$.
2. Let $x = (r, c), \text{TAG} = (r, c), z' = (z, \tilde{r})$. Emulate an execution for $A'(x, \text{TAG}, z')$ by honestly providing a proof of x (using tag TAG and the witness (v', s)) as part of its left interaction and honestly verifying the right interaction.
3. Given the view of A' in the above emulation, reconstruct the view *view* of A in the emulation by A'. If the commitment produced by A in the right execution in *view* is valid, let $\tilde{v}$ denote the value committed to; recall that, by our assumption on $\tilde{r}$, this value is uniquely defined (although it is not efficiently computable). If, on the other hand, the commitment produced by A is invalid, let $\tilde{v} = \perp$.
4. Finally, if $\tilde{v} = \perp$, output 0. Otherwise output $\mathcal{R}(v, \tilde{v})$.

Note that $\mathsf{hyb}_1(v)$ is not efficiently samplable, since the third step is not efficient. However, except for that step, every other operation in hyb_1 is efficient. (This will be useful to us at a later stage.)

We have the following claim.

CLAIM 6.2.

$$\Pr\left[\mathsf{hyb}_1(v) = 1\right] - \Pr\left[\mathsf{hyb}_1(0^n) = 1\right] \geq \frac{1}{2p(n)}.$$

Proof. Note that, by the construction of A' and hyb_1, the following hold directly:

1. The view of A in $\mathsf{hyb}_1(v)$ is identically distributed to the view of A in $\mathsf{mim}^A_{com}(\mathcal{R}, v, z)$, conditioned on the event that the first message in the right execution is $\tilde{r}$.
2. The view of A in $\mathsf{hyb}_1(0^n)$ is identically distributed to the view of A in $\mathsf{sta}^A_{com}(\mathcal{R}, v, z)$, conditioned on the event that the first message in the right execution is $\tilde{r}$.

Since the output of the experiments $\mathsf{hyb}, \mathsf{mim}, \mathsf{sta}$ is determined by applying the same fixed function (involving $\mathcal{R}$ and v) to the view of A in those experiments, the claim follows. □

Step 3: *Show that the simulator for A' violates the hiding property of* Com. We next use the simulator-extractor S' for A' to construct an *efficiently computable* experiment that is statistically close to hyb_1.

Towards this goal, we first consider the following experiment hyb_2, which still is not efficient. $\mathsf{hyb}_2(v')$ proceeds just as $\mathsf{hyb}_1(v')$ except that, instead of emulating the left and right interactions for A', hyb_2 runs the combined simulator extractor $\mathcal{S}$ for A' to generate the view of A'.

CLAIM 6.3. *There exists a negligible function $\nu'(n)$ such that, for any string $v' \in \{0,1\}^n$,*

$$|\Pr\left[\mathsf{hyb}_1(v') = 1\right] - \Pr\left[\mathsf{hyb}_2(v') = 1\right]| \leq \nu'(n).$$

Proof. It follows directly from the statistical indistinguishability property of $\mathcal{S}$ that the view of A generated in hyb_1 is statistically close to the view of A generated in hyb_2. The claim is concluded by (again) observing that the success of both hyb_1 and hyb_2 is defined by applying the same (deterministic) function to the view of A. □

Remark 6.4. Note that the proof of Claim 6.3 inherently relies on the *statistical* indistinguishability property of $\mathcal{S}$. Indeed, if the simulation had only been computationally indistinguishable, we would not have been able to argue indistinguishability

of the outputs of hyb_1 and hyb_2. This follows from the fact that the success in experiments hyb_1 and hyb_2 (which depends on the *actual* committed values in the view of A) is not efficiently computable from the view alone.

We next define the final experiment $\mathsf{hyb}_3(v')$ that proceeds just as $\mathsf{hyb}_2(v')$ with the following modification:

- Instead of letting $\tilde{v}$ be set to the actual value committed to in the view *view* of A, $\tilde{v}$ is computed as follows. Recall that the combined-simulator extractor $\mathcal{S}$ outputs both a view and a witness to each accepting right interaction. If the right execution in *view* is accepting[19] and if $\tilde{\text{TAG}} \neq \text{TAG}$, where $\tilde{\text{TAG}}$ is the tag used in the right interaction in *view*, simply set $\tilde{v}$ to be consistent with the witness output by $\mathcal{S}$ (i.e., if $\mathcal{S}$ outputs the witness (v', s'), let $\tilde{v} = v'$). Otherwise (i.e., if $\tilde{\text{TAG}} = \text{TAG}$, or if the right execution was rejecting), let $\tilde{v} = \perp$.

Note that, in contrast to hyb_2, hyb_3 is efficiently computable. Furthermore, the following claim holds.

CLAIM 6.5. *For any string* $v' \in \{0,1\}^n$,

$$\Pr\left[\mathsf{hyb}_2(v') = 1\right] = \Pr\left[\mathsf{hyb}_3(v') = 1\right].$$

Proof. Recall that the view of A in hyb_2 and hyb_3 is identical; the only difference in the experiments is how the final output is computed. It holds by the definition of the simulator-extractor $\mathcal{S}$ that $\mathcal{S}$ *always* outputs the witness to the statement proved by A' if the right interaction is accepting and if $\tilde{\text{TAG}} \neq \text{TAG}$. Thus, whenever *view* contains an accepting right-execution proof such that $\tilde{\text{TAG}} \neq \text{TAG}$, it follows by our assumption that $\text{Com}_{\tilde{r}}$ is perfectly binding and that the output of hyb_2 and hyb_3 is identical. Furthermore, in case the right-execution proof is rejecting, it holds that $\tilde{v} = \perp$ in both hyb_2 and hyb_3, which again means the output in both experiments is identical. Finally, consider the case when the right execution is accepting but $\tilde{\text{TAG}} = \text{TAG}$. By definition, it holds that hyb_3 outputs 0. Now recall that $\text{TAG} = (r, c)$ and $\tilde{\text{TAG}} = (\tilde{r}, \tilde{c})$; in other words, if $\tilde{\text{TAG}} = \text{TAG}$, it means that A fully copied the initial commitment using Com_r. Since $\mathcal{R}$ is irreflexive and $\text{Com}_{\tilde{r}}$ is perfectly binding, it follows that also hyb_2 outputs 0. We conclude that the outputs of hyb_2 and hyb_3 are identically distributed. ∎

By combining the above claims we obtain that there exists some polynomial $p'(n)$ such that

$$\Pr\left[\mathsf{hyb}_3(v) = 1\right] - \Pr\left[\mathsf{hyb}_3(0^n) = 1\right] \geq \frac{1}{p'(n)}.$$

However, since hyb_3 is efficiently samplable, we conclude that this contradicts the (nonuniform) hiding property of Com_r.

More formally, define an additional hybrid experiment $\mathsf{hyb}_4(c')$ that proceeds as follows on input a commitment c' using Com_r: hyb_4 performs the same operations as hyb_3, except that, instead of generating the commitment c, it simply sets $c = c'$. It follows directly from the construction of hyb_4 that $\mathsf{hyb}_4(c')$ is identically distributed to $\mathsf{hyb}_3(0^n)$ when c' is a (random) commitment to 0^n (using Com_r) and is identically distributed to $\mathsf{hyb}_3(v)$ when c' is a commitment to v. We conclude that hyb_4 distinguishes commitments (using Com_r) to 0^n and v. ∎

[19] Note that, since *view* is a *joint* view of A and the honest receiver R in the right execution, one can efficiently determine whether R indeed accepted the commitment.

Remark 6.6 (black-box vs. nonblack-box simulation). Note that the stand-alone committer S constructed in the above proof uses only black-box access to the adversary A, even if the simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ has been proven using a nonblack-box simulation. Thus, in essence, the simulation of our nonmalleable commitment is *always* black box. However, the analysis showing the correctness of the simulator relies on nonblack-box techniques, whenever the simulator-extractor for $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ is proven using nonblack-box techniques.

Since families of noninteractive statistically binding commitment schemes can be based on collision-resistant hash functions (in fact one-way functions are enough [31, 26]) we get the following corollary.

COROLLARY 6.7 (statistical-binding nonmalleable commitment). *Suppose that there exists a family of collision-resistant hash functions. Then there exists a constant-round statistically binding commitment scheme that is nonmalleable with respect to commitment.*

6.2. A statistically hiding scheme (NM with respect to opening). We proceed to the construction of a statistically hiding commitment scheme $\langle \mathbf{C}, \mathbf{R} \rangle$ which is nonmalleable with respect to opening. Our construction relies on a quite straightforward combination of a (family) of noninteractive statistically hiding commitments and a simulation-extractable argument.[20] Let $\{\mathbf{Com}_r\}_{r \in \{0,1\}^*}$ be a family of *noninteractive* statistically hiding commitment schemes (e.g., [10]), and let $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ be simulation-extractable protocol. The protocol is depicted in Figure 16.

Protocol $\langle \mathbf{C}, \mathbf{R} \rangle$

Security Parameter: 1^n.

String to be committed to: $v \in \{0,1\}^n$.

Commit Phase:

$R \to C$: Pick uniformly $r \in \{0,1\}^n$.

$C \to R$: Pick $s \in \{0,1\}^n$ and send $c = \mathbf{Com}_r(v; s)$.

Reveal Phase:

$C \to R$: Send v.

$C \leftrightarrow R$: Let $\text{TAG} = (r, c, v)$. Prove using $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ that there exists $s \in \{0,1\}^n$ so that $c = \mathbf{Com}_r(v; s)$. Formally, prove the following statement (r, c, v) with respect to the witness relation:

$$R_L = \{(r, c, v), s | c = \mathbf{Com}_r(v; s)\}.$$

R: Verify that $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ is accepting.

FIG. 16. *A statistically hiding nonmalleable string commitment protocol* $\langle \mathbf{C}, \mathbf{R} \rangle$.

THEOREM 6.8 ($nm\mathcal{C}$ with respect to opening). *Suppose that* $\{\mathbf{Com}_r\}_{r \in \{0,1\}^*}$ *is a family of* noninteractive *commitment schemes and that* $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ *is a simulation-extractable argument with an efficient prover strategy. Then* $\langle \mathbf{C}, \mathbf{R} \rangle$ *is a nonmalleable commitment scheme with respect to opening. If, furthermore,* $\{\mathbf{Com}_r\}_{r \in \{0,1\}^*}$ *is statistically hiding, then* $\langle \mathbf{C}, \mathbf{R} \rangle$ *is so as well.*

Proof. We need to prove that the scheme satisfies the following three properties: computational binding, (statistical) hiding, and nonmalleability with respect to opening.

[20] Note that, whereas our construction of statistically binding commitments required that the simulation-extractable argument provides a simulation that is statistically close, here we are content with a computationally indistinguishable simulation.

We start by proving the hiding and nonmalleability properties and then return to the proof of the binding property.

(Statistical) hiding. The hiding property follows directly from the hiding property of **Com**. Note that if **Com** is statistically hiding, then $\langle \mathbf{C}, \mathbf{R} \rangle$ is also statistically hiding.

Nonmalleability. We show that for every probabilistic polynomial-time man-in-the-middle adversary A, there exist a probabilistic *expected* polynomial-time stand-alone adversary S and a negligible function $\nu : N \to N$ such that, for every irreflexive polynomial-time computable relation $\mathcal{R} \subseteq \{0,1\}^n \times \{0,1\}^n$, every $v \in \{0,1\}^n$, and every $z \in \{0,1\}^*$, it holds that

$$\Pr\left[\mathsf{mim}^A_{open}(\mathcal{R}, v, z) = 1\right] \; < \; \Pr\left[\mathsf{sta}^S_{open}(\mathcal{R}, v, z) = 1\right] \; + \; \nu(n).$$

We remark that the stand-alone adversary S constructed here will be conceptually quite different from the one constructed in the proof of Theorem 6.1.

Description of the stand-alone adversary. We proceed to describing the stand-alone adversary S. On a high level, S internally incorporates A and emulates the commit phase of the left execution for adversary A by honestly committing to 0^n while externally forwarding messages in the right execution. Once A has completed the commit phase, S interprets the residual adversary (after the completed commit phase) as a man-in-the middle adversary A' for $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$. It then executes the simulator-extractor $\mathcal{S}$ for A' to obtain a witness to the statement proved in the right execution by A' (and thus A). Using this witness S can then complete the decommit phase of the external execution. (Here we rely on the fact that $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ has an efficient prover strategy.)

More formally, the stand-alone adversary S proceeds as follows on input z:

1. S internally incorporates $A(z)$.
2. During the commit phase S proceeds as follows:
   (a) S internally emulates the left interaction for A by honestly committing to 0^n.
   (b) Messages from the right execution are forwarded externally.
3. Once the commit phase has finished S receives the value v. Let $(r, c), (\tilde{r}, \tilde{c})$ denote the left- and right-execution transcripts of A (recall that the left execution has been internally emulated, while the right execution has been externally forwarded).
4. Construct a man-in-the-middle adversary A' for $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$. Informally, A' will simply consist of the residual machine resulting after the above-executed commit phase. More formally, $A'(x, \text{TAG}, z')$ proceeds as follows:
   (a) Parse z' as $(\tilde{r}, \tilde{c}, z)$.
   (b) Parse x as (r, c, v).
   (c) Internally emulate the commit phase $(r, c), (\tilde{r}, \tilde{c})$ for $A(z)$ (i.e., feed A the message $\tilde{r}$ as part of its right execution and c as part of its left execution).
   (d) Once the commit phase has finished, feed v to A.
   (e) Externally forward all the remaining messages during the reveal phase.
5. Let $\mathcal{S}$ denote the simulator-extractor for A'.
6. Let $x = (r, c, v)$, $\text{TAG} = (r, c, v)$, and $z' = (\tilde{r}, \tilde{c}, z)$.
7. Run $\mathcal{S}$ on input (x, TAG, z') to obtain the view *view* and the witness $\tilde{w}$.
8. Finally, if the statement proved in the right execution of *view* is $\tilde{x} = (\tilde{r}, \tilde{c}, \tilde{v})$ (where $\tilde{v}$ is an arbitrary string), the right-execution proof is accepting and

uses a tag $\tilde{\text{TAG}}$ such that $\tilde{\text{TAG}} \neq \text{TAG}$, and $\tilde{w}$ contains a valid witness for $\tilde{x}$, run the honest prover strategy P_{TAG} on input $\tilde{x}$ and the witness $\tilde{w}$. (Otherwise, simply abort.)

Analysis of the stand-alone adversary. Towards the goal of showing (6.2), we define a hybrid stand-alone adversary $\hat{S}$ that also receives v as auxiliary input. $\hat{S}$ proceeds exactly as S, but, instead of feeding A a commitment to 0^n in the commit phase, $\hat{S}$ instead feeds A a commitment to v.

Since both the experiment sta_{open} and the experiment $\hat{S}$ are efficiently computable, the following claim follows directly from the hiding property of **Com**.

CLAIM 6.9. *There exists some negligible function ν' such that*

$$\left| \Pr\left[\mathsf{sta}^{S}_{open}(\mathcal{R}, v, z) = 1\right] - \Pr\left[\mathsf{sta}^{\hat{S}}_{open}(\mathcal{R}, v, z) = 1\right]\right| \leq \nu'(k).$$

We proceed to showing the following claim, which together with Claim 6.9 concludes (6.2).

CLAIM 6.10. *There exists some negligible function ν'' such that*

$$\left| \Pr\left[\mathsf{mim}^{A}_{open}(\mathcal{R}, v, z) = 1\right] - \left[\mathsf{sta}^{\hat{S}}_{open}(\mathcal{R}, v, z) = 1\right]\right| \leq \nu''(k).$$

Proof. Towards the goal of showing this claim we introduce an additional hybrid experiment $\mathsf{hyb}(\mathcal{R}, v, z)$ which proceeds as follows: Emulate $\mathsf{sta}^{\tilde{S}}_{open}(\mathcal{R}, v, z)$, but, instead of defining $\tilde{v}$ as the value (successfully) decommitted to by S, define $\tilde{v}$ as the value (successfully) decommitted to in the view *view* output by simulator-extractor $\mathcal{S}$ (in the execution by $\hat{S}$). We start by noting that it follows directly from the indistinguishability property of the simulator-extractor $\mathcal{S}$ that the following quantity is negligible:[21]

$$\left| \Pr\left[\mathsf{mim}^{A}_{open}(\mathcal{R}, v, z) = 1\right] - \Pr\left[\mathsf{hyb}(\mathcal{R}, v, z) = 1\right]\right|.$$

To conclude the claim, we show that

$$\Pr\left[\mathsf{hyb}(\mathcal{R}, v, z) = 1\right] = \left[\mathsf{sta}^{\hat{S}}_{open}(\mathcal{R}, v, z) = 1\right].$$

Note that the only difference between experiments $\mathsf{hyb}(\mathcal{R}, v, z)$ and $\mathsf{sta}^{\hat{S}}_{open}(\mathcal{R}, v, z)$ is that, in hyb, the value $\tilde{v}$ is taken from the view output by $\mathcal{S}$, whereas in $\mathsf{sta}^{\hat{S}}_{open}$ it is defined as the value successfully decommitted to by $\hat{S}$. Also recall that $\hat{S}$ "attempts" to decommit to the value $\tilde{v}$ successfully decommitted to in the output by $\mathcal{S}$; $\hat{S}$ is successful in this task whenever $\mathcal{S}$ also is able to extract a witness to the right-execution proof. Note that by the simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ it follows that $\mathcal{S}$ *always* outputs a valid witness if the right execution in *view* is accepting, as long as the tag $\tilde{\text{TAG}}$ of the right execution is different from TAG. Thus, in case $\tilde{\text{TAG}} \neq \text{TAG}$, we conclude by the perfect completeness of $\langle P_{\text{TAG}}, V_{\text{TAG}}\rangle$ that the output of experiments sta and hyb is defined in exactly the same way. In case $\tilde{\text{TAG}} = \text{TAG}$, sta will output 0 (as $\hat{S}$ will not even attempt to decommit). However, in this case, it holds that $\tilde{v} = v$ (since $\text{TAG} = (r, c, v)$ and $\tilde{\text{TAG}} = (\tilde{r}, \tilde{c}, \tilde{v})$); this means that hyb will also output 0 (since $\mathcal{R}$ is irreflexive). The claim follows. □

[21] We remark that here it is sufficient that the simulator-extractor outputs a view that is merely computationally indistinguishable from the view in a "real" execution.

We now return to the binding property.

Computational binding. The binding properties of the scheme intuitively follow from the binding property of the underlying commitment scheme **Com** and the "proof of knowledge" property implicitly guaranteed by the simulation-extractability property of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$. A formal proof proceeds along the lines of the proof of nonmalleability (but is simpler.) More precisely, assume for contradiction that there exists some adversary A that is able to violate the binding property of $\langle \mathbf{C}, \mathbf{R} \rangle$. We show how to construct a machine $\hat{A}$ that violates the binding property of **Com**. $\hat{A}$ starts by running A, letting it complete the commit phase by externally forwarding its messages. Once A has completed the commit phase, $\hat{A}$ interprets the residual adversary (after the completed commit phase) as a man-in-the middle adversary A' (that ignores all left-execution messages) for $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$; a formal description of this adversary is essentially identical to the one described in the proof of nonmalleability. $\hat{A}$ then executes the simulator-extractor $\mathcal{S}$ for A' to obtain a witness to the statement proved in the right execution by A' (and thus A). Using this witness $\hat{A}$ can then complete the decommit phase of the external execution of **Com**. It follows directly by the indistinguishability property of the simulator-extractor $\mathcal{S}$ that $\hat{A}$ violates the binding property of **Com** with essentially the same probability as A violates the binding property of $\langle \mathbf{C}, \mathbf{R} \rangle$.

This completes the proof of Theorem 6.8. □

Remark 6.11 (black-box vs. nonblack-box simulation). Note that the stand-alone adversary S constructed in the proof of Theorem 6.8 is very different from the stand-alone adversary constructed in the proof of Theorem 6.1. In particular S constructed above in fact runs the simulator-extractor $\mathcal{S}$ (whereas in the proof of Theorem 6.1 the simulator extractor is simply used in the analysis). As a consequence (in contrast to the simulator constructed in 6.1), the stand-alone adversary S constructed above makes use of the man-in-the middle adversary in a nonblack-box way if relying on a simulation-extractable argument with a nonblack-box simulator.

Since families of noninteractive statistically hiding commitments can be based on collision-resistant hash functions [33, 10] we obtain the following corollary.

COROLLARY 6.12 (statistically hiding nonmalleable commitment). *Suppose that there exists a family of collision-resistant hash functions. Then there exists a constant-round statistically hiding commitment scheme which is nonmalleable with respect to opening.*

Appendix. Missing proofs.

Proposition 4.2 (argument of knowledge). Let $\langle P_{sWI}, V_{sWI} \rangle$ and $\langle P_{UA}, V_{UA} \rangle$ be the protocols used in the construction of $\langle P_{sUA}, V_{sUA} \rangle$. Suppose that $\{\mathcal{H}_n\}_n$ is collision resistant for $T(n)$-sized circuits, that **Com** is statistically hiding, that $\langle P_{sWI}, V_{sWI} \rangle$ is a statistical witness indistinguishable argument of knowledge, and that $\langle P_{UA}, V_{UA} \rangle$ is a universal argument. Then, for any $\text{TAG} \in \{0,1\}^n$, $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ is an interactive argument of knowledge.

Completeness of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ follows from the completeness property of $\langle P_{sUA}, V_{sUA} \rangle$. Specifically, an honest prover P, who possesses a witness w for $x \in L$, can always make the verifier accept by using w as the witness in the n parallel executions of $\langle P_{sUA}, V_{sUA} \rangle$. To demonstrate the argument of knowledge property of $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$, it will be sufficient to prove that $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}} \rangle$ is an argument of knowledge. This is because the prescribed verifier in $\langle P_{\text{TAG}}, V_{\text{TAG}} \rangle$ will accept the proof only if *all* runs of

$\langle P_{\mathsf{tag}_i}, V_{\mathsf{tag}_i}\rangle$ are accepting. [22]

LEMMA A.1. *Suppose that $\{\mathcal{H}_n\}_n$ is collision resistant for $T(n)$-sized circuits, that* **Com** *is statistically hiding, that $\langle P_{sWI}, V_{sWI}\rangle$ is a statistical witness-indistinguishable argument of knowledge, and that $\langle P_{UA}, V_{UA}\rangle$ is a universal argument. Then, for any* $\mathsf{tag} \in [2n]$, *$\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$ is an argument of knowledge.*

Proof. We show the existence of an extractor machine E for protocol $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$.

Description of the extractor machine. E proceeds as follows given oracle access to a malicious prover P^*. E, using black-box access to P^*, internally emulates the role of the honest verifier V_{tag} for P^* *until* P^* provides an accepting proof (i.e., if P^* fails, E restarts P^* and attempts a new emulation of V_{tag}). Let σ_s denote the messages received by P^*, in the successful emulation by E, up until protocol $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}}\rangle$ is reached; let P^*_{WI} denote the residual prover $P^*(\sigma_s)$.

The extractor E next applies the witness extractor E_{WI} for $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}}\rangle$ on P^*_{WI}. If $E_{\mathsf{WI}}^{P^*_{\mathsf{WI}}}$ outputs a witness w, such that $R_L(x, w) = 1$, E outputs the same witness w; otherwise it outputs `fail`. (The reason E constructs P^*_{WI} as above is to ensure that P^*_{WI} convinces V_{WI} with nonzero probability; otherwise E_{WI} is not guaranteed to extract a witness.)

Analysis of the extractor. Let P^* be a nonuniform PPT that convinces the honest verifier V_{tag} of the validity of a statement $x \in \{0,1\}^n$ with probability $\epsilon(n)$. We assume without loss of generality that P^* is deterministic. We need to show the following two properties:

1. The probability that P^* succeeds in convincing V_{tag}, but E does not output a valid witness to x, is negligible.
2. The expected number of steps taken by E is bounded by $\frac{\mathrm{poly}(n)}{\epsilon(n)}$.

We start by noting that since E perfectly emulates the role of the honest verifier V_{tag}, E requires in expectation $\frac{\mathrm{poly}(n)}{\epsilon(n)}$ steps before invoking the extractor E_{WI}. Additionally, by the proof-of-knowledge property of $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}}\rangle$ it follows that the expected running time of E_{WI} is $\frac{\mathrm{poly}(n)}{\epsilon(n)}$. To see this, let the random variable T denote the running time of E_{WI} in the execution by E, and let $T(\sigma)$ denote the running time of E_{WI} given that E chooses the prefix σ. We abuse of notation and let σ denote the event that P^* receives the prefix σ in an interaction with V_{tag}; also let accept denote the event that P^* produces a convincing proof. We have

$$E[T] = \sum_{\sigma} E[T(\sigma)]\Pr(\sigma|\mathsf{accept}) = \sum_{\sigma} \frac{\mathrm{poly}(n)}{\Pr[\mathsf{accept}|\sigma]}\Pr(\sigma|\mathsf{accept})$$
$$= \mathrm{poly}(n)\sum_{\sigma}\frac{\Pr[\sigma]}{\Pr[\mathsf{accept}]} = \frac{\mathrm{poly}(n)}{\Pr[\mathsf{accept}]} = \frac{\mathrm{poly}(n)}{\epsilon(n)},$$

where the second equality follows from the definition of a proof of knowledge. We conclude by the linearity of expectations that the expected running time of E is $\frac{\mathrm{poly}(n)}{\epsilon(n)}$ and thus the second of the above properties holds.

We turn to show that also the first property holds. Assume that there exists some polynomial $p(n)$ such that, for infinitely many n, $\epsilon(n) \geq \frac{1}{p(n)}$ but E^{P^*} fails to output a valid witness with probability $\frac{1}{p(n)}$. Note that E^{P^*} can fail for two reasons:

[22] One could turn any cheating prover for $\langle P_{\mathrm{TAG}}, V_{\mathrm{TAG}}\rangle$ into a cheating prover for $\langle P_{\mathsf{tag}_i}, V_{\mathsf{tag}_i}\rangle$ by internally emulating the role of the verifier V_{tag_j} for $j \neq i$ and forwarding the messages from P_{tag_i} to an external V_{tag_i}.

1. Either the extraction by E_{WI} fails, or
2. E_{WI} outputs $\langle \beta, \delta, s_1, s_2 \rangle$ so that
   - $\hat{\beta} = \mathbf{Com}(\beta; s_1)$,
   - $\hat{\delta} = \mathbf{Com}(\delta; s_2)$,
   - $(\alpha, \beta, \gamma, \delta)$ is an accepting transcript for $\langle P_{\mathsf{UA}}, V_{\mathsf{UA}} \rangle$.

If the second event occurs, we say that E_{WI} outputs a *false witness*. Consider an alternative extractor E' which proceeds just as E except that if in the *first* emulation of V_{tag}, P^* fails in producing a convincing proof, E' directly aborts. Note that the only difference between E and E' is that E continues sampling executions until P^* produces a convincing proof, whereas E' samples only once. Since by our assumption, $\epsilon(n) \geq \frac{1}{p(n)}$, it follows that there exists some polynomial $p'(n)$ such that in the execution by E', E_{WI} either fails or outputs a false witness with probability $\frac{1}{p'(n)}$. It directly follows from the proof-of-knowledge property of $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$ that E_{WI} fails only with negligible probability. We show below that E_{WI} outputs a false witness also with negligible probability; this is a contradiction.

PROPOSITION A.2. *In the execution of E'^{P^*}, E_{WI} outputs a false witness with negligible probability.*

Proof. Towards proving the proposition we start by showing the following lemma.

LEMMA A.3. *Let P^*_{sUA} be a nonuniform polynomial-time machine such that $E_{\mathsf{WI}}^{P^*_{\mathsf{sUA}}(\alpha,\gamma)}$ outputs a false witness to the statement $\bar{x} = (x, \langle h, c_1, c_2, r_1, r_2 \rangle)$ with probability $\epsilon(n) = \frac{1}{\mathrm{poly}(n)}$ given uniformly chosen verifier messages α, γ. Then there exists a* strict *polynomial-time machine* $\mathsf{extract}$ *such that, with probability* $\mathrm{poly}(\epsilon(n))$, $\mathsf{extract}(P^*_{\mathsf{sUA}}, \bar{x})$ *outputs*

- *an index $i \in \{1, 2\}$,*
- *strings y, s, z such that $z = h(\Pi)$, $r_i = \Pi(y, s)$, and $c_i = \mathbf{Com}(z; s)$,*
- *a polynomial-time machine M such that $M(j)$ outputs the jth bit of Π (M is called the "implicit" representation of Π).*

Proof. The proof of the lemma proceeds in the following two steps.

1. Using an argument by Barak and Goldreich [3], we use P^*_{sUA} and E_{WI} to construct a prover P^*_{UA} for the UARG $\langle P_{\mathsf{UA}}, V_{\mathsf{UA}} \rangle$ that succeeds with probability $\mathrm{poly}(\epsilon(n))$.
2. Due to the weak proof of knowledge property of *UARG* we then obtain an index $i \in \{1, 2\}$, strings y, s, a hash $z = h(\Pi)$ so that $r_i = \Pi(y, s)$ and $c_i = \mathbf{C}(z; s)$. We furthermore obtain an "implicit" representation of the program Π.

Step 1. *Constructing P^*_{UA}.* P^*_{UA} proceeds as follows.

- P^*_{UA} starts by receiving a message α from the honest verifier V_{UA}.
- P^*_{UA} incorporates P^*_{sUA} and internally forwards the message α to P^*_{sUA}, resulting in a residual prover $P^*_{\mathsf{sUA}}(\alpha)$.
- P^*_{UA} then internally emulates the role of the honest verifier for P^*_{sUA} until protocol $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$ is reached (i.e., P^*_{UA} uniformly choses a random message $\bar{\gamma}$ that it forwards to P^*_{sUA}, resulting in a residual prover $P^*_{\mathsf{sUA}}(\alpha, \bar{\gamma})$). Thereafter, P^*_{UA} honestly emulates the verifier V_{WI} for $P^*_{\mathsf{sUA}}(\alpha, \bar{\gamma})$. If $P^*_{\mathsf{sUA}}(\alpha, \bar{\gamma})$ succeeds in providing an accepting proof, P^*_{UA} invokes the knowledge extractor E_{WI} on the prover $P^*_{\mathsf{sUA}}(\alpha, \bar{\gamma})$.
- In the event that $P^*_{\mathsf{sUA}}(\alpha, \bar{\gamma})$ does not produce an accepting proof, or if E_{WI} does not output an accepting tuple $\langle \beta, \delta, s_1, s_2 \rangle$, P^*_{UA} halts. Otherwise, it externally forwards the message β to V_{UA} and receives as response γ.

- P^*_{UA} now rewinds P^*_{sUA} until the point where it awaits the message γ and internally forwards γ to P^*_{sUA}, resulting in a residual prover $P^*_{\mathsf{sUA}}(\alpha, \gamma)$. As before P^*_{UA} first honestly verifies the WI proof that $P^*_{\mathsf{UA}}(\alpha, \gamma)$ gives and in the case this proof is accepting applies the extractor E_{WI} to $P^*_{\mathsf{sUA}}(\alpha, \gamma)$.
- If E_{WI} outputs an accepting tuple $\langle \beta', \delta', s'_1, s'_2 \rangle$, such that $\beta' = \beta$, P^*_{UA} forwards δ to V_{UA}, and otherwise it halts.

Since $\langle \alpha, \beta', \gamma, \delta' \rangle$ is an accepting transcript of $\langle P_{\mathsf{UA}}, V_{\mathsf{UA}} \rangle$, it follows that unless P^*_{UA} halts the execution, it succeeds in convincing the verifier V_{UA}.

We show that P^*_{UA} finishes the execution with probability $\mathrm{poly}(\epsilon(n))$. Using the same argument as Barak and Goldreich [3] (of counting "good" verifier messages, i.e., messages that will let the prover succeed with "high" probability; see Claim 4.2.1 in [3]), it can be shown that, with probability $\mathrm{poly}(\epsilon(n))$, P^*_{UA} reaches the case where the extractor outputs $\langle \beta', \delta', s'_1, s'_2 \rangle$. Thus it remains only to show that conditioned on this event, $\beta' \neq \beta$ occurs with polynomial probability. In fact, the event that $\beta' = \beta$ can occur only with negligible probability, or else we would contradict the binding property of **Com** (since $\tilde{\beta} = \mathbf{Com}(\beta, s_1) = \mathbf{Com}(\beta', s'_1)$). We thus conclude that P^*_{UA} succeeds in convincing V_{UA} with probability $\mathrm{poly}(\epsilon(n))$.

Furthermore, since the extractor E_{WI} is applied only when P^*_{UA} provides an accepting proof, it follows from the definition of a proof of knowledge that the *expected* running time of P^*_{UA} is a polynomial, say $g(n)$. Finally, if we truncate the execution of P^*_{UA} after $2g(n)$ steps, we get by the Markov inequality that (the truncated) P^*_{UA} still convinces produces convincing proofs with probability $\mathrm{poly}(\epsilon)$.

Step 2. *Extracting the "false" witness.* By the weak proof of knowledge property of $\langle P_{\mathsf{UA}}, V_{\mathsf{UA}} \rangle$ there exists a strict PPT machine E_{UA} such that $E_{\mathsf{UA}}^{P^*_{\mathsf{UA}}}$ outputs an "implicit" representation of a "witness" to the statement $\bar{x} = (x, \langle h, c_1, c_2, r_1, r_2 \rangle)$ proved by P^*_{UA}. Since the values i, y, s, z have fixed polynomial length, they can all be extracted in polynomial time. Note, however, that since there is not a (polynomial) bound on the length of the program Π, we can extract only an implicit representation of Π. The concludes the proof of the lemma. □

Armed with Lemma A.3, we now turn to show that E_{WI} outputs a false witness with negligible probability in the execution of E'^{P^*}. Suppose for contradiction that there exists a polynomial $p(n)$ such that, for infinitely many n's, E_{WI} outputs a false witness, with probability at least $\epsilon(n) = \frac{1}{p(n)}$. We construct a $T(n)^{O(1)}$-sized circuit family, $\{C_n\}_n$, that finds collisions for $\{\mathcal{H}_n\}_n$ with probability $\mathrm{poly}(\epsilon(n))$.

More specifically, the following hold:

- On input $h \stackrel{\mathrm{R}}{\leftarrow} \mathcal{H}_n$, the circuit C_n incorporates P^* and internally emulates the honest verifier V for P^* until the protocol $\langle P_{\mathsf{sUA}}, V_{\mathsf{sUA}} \rangle$ is reached (i.e., C_n internally sends randomly chosen messages h, r_1, r_2 to P^*, resulting in a residual prover $P^*(h, r_1, r_2)$).
- C_n then invokes the knowledge extractor $\mathsf{extract}$, guaranteed by Lemma A.3, on $P^*(h, r_1, r_2)$, extracting values i, y, s, z and an implicit representation of Π, given by a machine M.
- If $\mathsf{extract}$ fails, C_n outputs `fail`; otherwise it rewinds P^* until the point where it expects to receive the message r_i and then continues the emulation of the honest verifier from this point (using new random coins).
- Once again, when P^* reaches $\langle P_{\mathsf{sUA}}, V_{\mathsf{sUA}} \rangle$, C_n invokes $\mathsf{extract}$ on the residual prover, extracting values i', y', s', z' and an implicit representation of Π', given by a machine M'.
- If the extraction fails or if $i' \neq i$, C_n outputs `fail`.

It remains to analyze the success probability of C_n. We start by noting that $y = y'$ occurs only with negligible probability. This follows from the computational binding property of **Com**. Since the probability that E_{WI} outputs a false witness is $\epsilon(n)$ it must hold that for a fraction $\epsilon(n)/2$ of the verifier messages before protocol $\langle P_{\mathsf{sUA}}, V_{\mathsf{sUA}}\rangle$, E_{WI} outputs a false witness with probability $\epsilon(n)/2$ when given oracle access to P^* having been fed messages in this set of "good" messages. Due to the correctness of $\mathsf{extract}$ it holds that when C_n selects verifier messages from this "good" set, the probability that extraction succeeds (in outputting a false witness) on P^* is

$$\epsilon' = \mathrm{poly}(\epsilon).$$

Thus, given that C_n picks random verifier messages, it holds that the probability that $\mathsf{extract}$ succeeds is at least

$$\epsilon'' = \frac{\epsilon}{2}\epsilon' = \mathrm{poly}(\epsilon).$$

Thus, there exists an index $\sigma \in \{1, 2\}$ such that the extraction outputs the index $i = \sigma$ with probability $\epsilon''' = \epsilon''/2$. Again, for a fraction $\epsilon'''/2$ of verifier messages before slot σ (when $\sigma = 1$, there is only one message, namely h, while when $\sigma = 2$, the messages are h, r_1), the residual prover ($P^*(h)$ when $\sigma = 1$, or $P^*(h, r_1)$ when $\sigma = 2$) succeeds in convincing the verifier with probability $\epsilon'''/2$. We conclude that with probability

$$\epsilon'''/2 \cdot (\epsilon'''/2)^2 = \mathrm{poly}(\epsilon)$$

C_n obtains an implicit representation of programs Π, Π' such that $\exists y, y' \in \{0,1\}^{(|r_i|-n)}$ for which $\Pi(y) = r_i$, $\Pi'(y') = r'_i$, and $h(\Pi) = h(\Pi')$. Using a simple counting argument it follows that with probability $(1 - 2^{-n})$ (over the choices of r_i, r'_i), $\Pi \neq \Pi'$.[23] Thus, by *fully* extracting the programs (from the implicit representation) C_n finds a collision with probability

$$\mathrm{poly}(\epsilon) \cdot (1 - 2^{-n}) = \mathrm{poly}(\epsilon).$$

Note that the time required for extracting these programs is upper bounded by $T(n)^{O(1)}$. Thus, any poly-time prover P^* that can make V accept $x \notin L$ with nonnegligible probability can be used in order to obtain collisions for $h \stackrel{\mathrm{R}}{\leftarrow} \mathcal{H}_n$ in time $T(n)^{O(1)}$ (note that here we additionally rely on the fact that $\mathsf{extract}$ is a strict polynomial-time machine), in contradiction to the collision resistance of $\{\mathcal{H}_n\}_n$ against $T(n)$-sized circuits.[24] This concludes the proof of the proposition. □

This completes the proof of Lemma A.3. □

Basing the construction on "standard" collision-resistant hash functions. Although the above analysis (for the proof of knowledge property of $\langle P_{\mathsf{tag}}, V_{\mathsf{tag}}\rangle$) relies on the assumption that $\{\mathcal{H}_k\}_k$ is a family of hash functions that is collision resistant against $T(k)$-sized circuits, we note that, by using the method of Barak and Goldreich [3], this assumption can be weakened to the (more) standard assumption of collision resistance against polynomial-sized circuits. The main idea in their approach is to replace the arbitrary hashing in Slots 1 and 2 with the following two-step hashing procedure:

[23] Fix Π, Π', y, r_i. Then, with probability 2^{-n} over the choice of r'_i, there exist a $y' \in \{0,1\}^{(|r'_i|-n)}$ so that $\Pi'(y') = r'_i$.

[24] We mention that by slightly modifying the protocol, following the approach by Barak and Goldreich [3], one can instead obtain a polynomial-sized circuit C_n finding a collisions for $\mathcal{H}_k$. More details follow after the proof.

- Apply a "good"[25] error-correcting code ECC to the input string.
- Use tree hashing [29, 8] to the encoded string.

This method has the advantage that, in order to find a collision for the hash function in the "proof of knowledge" proof, the full description of programs Π, Π' is not needed. Instead it is sufficient to look at a randomly chosen position in the description (which can be computed in polynomial time from the implicit representation of Π, Π'). The analysis here relies on the fact that two different codewords differ in random position i with a (positive) constant probability.

Acknowledgments. We are grateful to Johan Håstad and Moni Naor for many helpful conversations and great advice. Thanks to Boaz Barak for useful clarifications of his works. The second author would also like to thank Marc Fischlin, Rosario Gennaro, Yehuda Lindell, and Tal Rabin for insightful discussions regarding nonmalleable commitments. Thanks to Oded Goldreich for useful feedback on an earlier version of this work and to the anonymous referees for their thoughtful comments. Finally, thanks to Huijia Lin for pointing out a subtlety in the proof of Proposition 4.2.

REFERENCES

[1] B. Barak, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 106–115.

[2] B. Barak, *Constant-round coin-tossing or realizing the shared random string model*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, 2002, pp. 345–355.

[3] B. Barak and O. Goldreich, *Universal arguments and their applications*, in Proceedings of the 17th IEEE Conference on Computational Complexity, 2002, pp. 194–203.

[4] M. Bellare and O. Goldreich, *On defining proofs of knowledge*, in Advances in Cryptology—CRYPTO'92, Lecture Notes in Comput. Sci. 740, Springer, Berlin, 1993, pp. 390–420.

[5] M. Blum, *Coin flipping by telephone*, in Advances in Cryptology—CRYPTO 1981, Springer, Berlin, pp. 11-15.

[6] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp. 156–189.

[7] R. Canetti and M. Fischlin, *Universally composable commitments*, in Advances in Cryptology—CRYPTO 2001, Lecture Notes in Comput. Sci. 2139, Springer, Berlin, pp. 19–40.

[8] I. Damgård, *A design principle for hash functions*, in Advances in Cryptology—CRYPTO 1989, Springer, Berlin, pp. 416–427.

[9] I. Damgård and J. Groth, *Non-interactive and reusable nonmalleable commitment schemes*, in Proceedings of the 35th ACM Symposium on Theory of Computing, 2003, pp. 426–437.

[10] I. Damgård, T. Pedersen, and B. Pfitzmann, *On the existence of statistically hiding bit commitment schemes and fail-stop signatures*, J. Cryptology, 10 (1997), pp. 163–194.

[11] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai, *Robust non-interactive zero knowledge*, in Advances in Cryptology—CRYPTO, Springer, Berlin, 2001, pp. 566–598.

[12] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith, *Efficient and non-interactive nonmalleable commitment*, in Advances in Cryptology—EUROCRYPT 2001, Springer, Berlin, pp. 40–59.

[13] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky, *Non-interactive and nonmalleable commitment*, in Proceedings of the 30th ACM Symposium on Theory of Computing, 1998, pp. 141–150.

[14] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.

[15] U. Feige, D. Lapidot, and A. Shamir, *Multiple noninteractive zero knowledge proofs under general assumptions*, SIAM J. Comput., 29 (1999), pp. 1–28.

[25] By "good" here we mean an error-correcting code correcting a constant fraction of error.

[16] U. Feige and A. Shamir, *Witness indistinguishability and witness hiding protocols*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 416–426.
[17] M. Fischlin and R. Fischlin, *Efficient nonmalleable commitment schemes*, in Advances in Cryptology—CRYPTO 2000, Lecture Notes in Comput. Sci. 1880, Springer, Berlin, 2000, pp. 413–431.
[18] O. Goldreich, *Foundation of Cryptography—Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.
[19] O. Goldreich and A. Kahan, *How to construct constant-round zero-knowledge proof systems for NP*, J. Cryptology, 9 (1996), pp. 167–189.
[20] O. Goldreich and Y. Lindell, *Session-key generation using human passwords only*, in Advances in Cryptology—CRYPTO 2001, Springer, Berlin, pp. 408–432.
[21] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. Assoc. Comput. Mach., 38 (1991), pp. 691–729.
[22] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game—a completeness theorem for protocols with honest majority*, in Proceedings of the 19th ACM Symposium on Theory of Computing, 1987, pp. 218–229.
[23] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems*, J. Cryptology, 7 (1994), pp. 1–32.
[24] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
[25] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.
[26] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
[27] J. Kilian, *A note on efficient zero-knowledge proofs and arguments*, in Proceedings of the 24th ACM Symposium on Theory of Computing, 1992, pp. 723–732.
[28] P. D. MacKenzie, M. K. Reiter, and K. Yang, *Alternatives to nonmalleability: Definitions, constructions, and applications*, in Proceedings of the 1st Theory of Cryptology Conference, 2004, pp. 171–190.
[29] R. C. Merkle, *A certified digital signature*, in Advances in Cryptology—CRYPTO 1989, Springer, Berlin, pp. 218–238.
[30] S. Micali, *Computationally sound proofs*, SIAM J. Comput., 30 (2000), pp. 1253–1298.
[31] M. Naor, *Bit commitment using pseudorandomness*, J. Cryptology, 4 (1991), pp. 151–158.
[32] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, *Perfect zero-knowledge arguments for NP using any one-way permutation*, J. Cryptology, 11 (1998), pp. 87–108.
[33] M. Naor and M. Yung, *Universal one-way hash functions and their cryptographic applications*, in Proceedings of the 21st ACM Symposium on Theory of Computing, 1989, pp. 33–43.
[34] M. Nguyen and S. Vadhan, *Simpler session-key generation from short random passwords*, in Proceedings of the 1st Theory of Cryptology Conference, 2004, pp. 428–445.
[35] R. Pass *Bounded-concurrent secure multi-party computation with a dishonest majority*, in Proceedings of the 36th ACM Symposium on Theory of Computing, 2004, pp. 232–241.
[36] R. Pass and A. Rosen, *Bounded-concurrent secure two-party computation in a constant number of rounds*, in Proceedings of the 34th Symposium on Foundations of Computer Science, 2003, pp. 404–413.
[37] R. Pass and A. Rosen, *Concurrent nonmalleable commitments*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, 2005, pp. 563–572.
[38] A. Sahai, *Nonmalleable non-interactive zero knowledge and adaptive chosen-ciphertext security*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 543–553.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 753–766

TENSOR NORMS AND THE CLASSICAL COMMUNICATION COMPLEXITY OF NONLOCAL QUANTUM MEASUREMENT*

YAOYUN SHI† AND YUFAN ZHU†

Abstract. We initiate the study of quantifying nonlocality of a bipartite measurement by the minimum amount of classical communication required to simulate the measurement. We derive general upper bounds in terms of some *tensor norms* of the measurement operator. As applications, we show that (a) if the amount of communication is a constant, then quantum and classical communication protocols with an unlimited amount of shared entanglement or shared randomness compute the same class of functions; and (b) it requires only a constant amount of communication to classically generate an approximation of the output distribution resulting from local measurements on an entangled quantum state, as long as the number of measurement outcomes is a constant.

Key words. quantum entanglement, classical simulation, communication complexity, tensor norms, Bell inequality

AMS subject classifications. 68Q10, 46M05, 47A80

DOI. 10.1137/050644768

1. Introduction and summary of results.

1.1. Background. Although Einstein himself made significant contributions to the development of quantum mechanics, he famously questioned the "completeness" of the theory with a "paradox" that he formulated with Podolsky and Rosen [17]. Following Bohm [7], the essence of the paradox is that two "quantum coins," possessed by two parties called Alice and Bob, may correlate in a state

$$\frac{1}{\sqrt{2}}\left(|\texttt{Head}\rangle_A|\texttt{Tail}\rangle_B - |\texttt{Tail}\rangle_A|\texttt{Head}\rangle_B\right).$$

If each party measures his or her coin, with 1/2 probability, one of the two outcomes is observed. However, once a measurement is made by one party, say, Alice, then Bob would always observe the opposite outcome with certainty. A unique property of the state is that no matter what physical property of the coins is measured—whether it be their positions or their velocities—Bob's outcome is always opposite to that of Alice's *with certainty*. It appeared to Einstein that this consequence could not be fully explained by quantum mechanics, since the uncertainty principle implies that not all pairs of properties can be determined with certainty.

The Einstein–Podolsky–Rosen (EPR) paradox does not show that quantum mechanics is inconsistent. In the far-reaching paper [3], Bell formulated a set of inequalities, now referred to as *Bell inequalities*, that must be satisfied by the correlations produced by any so-called *hidden variable model* of classical physics but nevertheless would be violated by some quantum correlations. The violation has been confirmed by several experiments (see, e.g., [40]). The EPR paradox instead reveals the essential

*Received by the editors November 10, 2005; accepted for publication (in revised form) November 5, 2007; published electronically May 28, 2008. A preliminary version of this paper appeared as part of an article in *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, 2005, pp. 460–467.

http://www.siam.org/journals/sicomp/38-3/64476.html

†Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 (shiyy@eecs.umich.edu, yufanzhu@eecs.umich.edu).

property of quantum states—quantum entanglement—that underlies the many counterintuitive properties and powerful applications of quantum information. A seminal example is the *quantum teleportation* protocol [4], which makes use of one EPR state to transmit a quantum bit by sending two classical bits. Another example is the construction of *unconditionally secure* quantum key distribution protocols [5, 18, 28, 30].

Given its importance, quantum entanglement has been the subject of numerous studies (see, e.g., the books [33, 34]). The focus has been on understanding the inherent quantitative trade-offs among various resources involved in the creation and conversion of entangled states. As entanglement is the result of nonlocal quantum interactions, understanding various aspects of the nonlocality of quantum operations is also of fundamental importance. Quantifying nonlocality of quantum operators is precisely the purpose of this paper.

1.2. Main result. A natural nonlocality measure of a quantum operation is its *generating capacity*, which is the maximum entanglement increase that it can create (see, e.g., [6]). Another approach, from more of a computational point of view, is to measure nonlocality by the amount of resources, such as the time for evolving elementary Hamiltonians or the number of elementary gates, required to simulate the operator (see, e.g., [11, 12]).

In this paper, we take a different approach and investigate a nonlocality measure in the framework of communication complexity. Our work is not the first to apply communication complexity to the study of entanglement. There are studies, which we will review shortly, on the classical communication complexity of simulating quantum correlations. Nevertheless, our emphasis is on quantum operators. Although we focus on measurement operators, our approach can be extended to the most general quantum operations.

Consider the following quantum process. Alice and Bob share a bipartite state $|E\rangle_{AB}$. They apply local operations R_A and R_B to their systems, before a final measurement Q is applied to the joint system, producing a distribution $\mu = \mu(Q, |E\rangle, R_A, R_B)$ of measurement outcomes.

Imagine now that Alice and Bob can process only classical information. However, they both have a classical description of Q and $|E\rangle$. In addition, Alice has a classical description of R_A, and Bob has that of R_B. They hope to simulate the quantum process by producing a random output whose distribution is close to μ. Since Alice does not know R_B and Bob does not know R_A, the simulation requires communication. We allow Alice and Bob to share an unlimited supply of random bits. We define the *classical communication complexity* of Q, denoted by $\mathrm{Com}(Q)$, to be the minimum number of bits that need to be exchanged by a simulating protocol that works for all $|E\rangle$, R_A, and R_B.

Intuitively, $\mathrm{Com}(Q)$ reflects how nonlocal Q is. Consider, for example, the extremal case that Q is the tensor product of two local operations. If there is no quantum correlation in the initial state, it is clear that Alice and Bob could simulate the quantum process without interaction. We shall see that even if the initial state is entangled, they need only exchange a constant number of bits.

On the other hand, $\mathrm{Com}(Q)$ would be much larger for highly nonlocal Q. Let $n \geq 1$ be an integer. Consider the following operator:

$$\mathrm{IP}_n \stackrel{\mathrm{def}}{=} \sum_{\substack{x,y\in\{0,1\}^n \\ x\cdot y=1}} |x\rangle\langle x| \otimes |y\rangle\langle y|. \tag{1}$$

When R_A creates a state $|x\rangle$, $x \in \{0,1\}^n$, and R_B creates $|y\rangle$, $y \in \{0,1\}^n$, then $(IP)_n$ determines if $x \cdot y = 1$. This is the so-called "inner product" function, which is well studied in the communication complexity literature. It is well known that any classical communication protocol for solving the inner product requires $\Omega(n)$ bits of communication. In fact, Cleve et al. [14] proved that $\Omega(n)$ quantum bits are necessary, too. Thus $\mathrm{Com}(\mathrm{IP}) = \Omega(n)$. We do not know if this bound for $\mathrm{Com}(\mathrm{IP})$ is tight.

The goal of this paper is to give a general estimation of $\mathrm{Com}(Q)$ and its applications. It is not immediately clear if $\mathrm{Com}(Q)$ is finite for all Q acting on a finite space, since the dimension of the initial state $|E\rangle$ could be arbitrarily large. Our main result is to derive a general upper bound on $\mathrm{Com}(Q)$ in terms of a certain operator norm $\|Q\|_\diamond$ on Q, which is bounded from above by a polynomial in the dimension of Q.

THEOREM 1.1 (informally). *For any bipartite measurement Q, $\mathrm{Com}(Q) = O(\|Q\|_\diamond^2)$. In particular, if K is the dimension of the space that Q acts on, $\mathrm{Com}(Q) = O(K^2)$.*

The diamond norm $\|\cdot\|_\diamond$ was originally defined on superoperators and has been a powerful tool in the study of quantum interactive proof systems [22] and quantum circuits on mixed states [1]. We use a natural mapping from bipartite operators to superoperators to define the diamond norm on the former based on the diamond norm on the latter.

The approach in proving Theorem 1.1 can be extended to obtain general upper bounds on $\mathrm{Com}(Q)$ in terms of other operators norms. Those norms belong to so-called *tensor norms*, i.e., norms $\|\cdot\|_\alpha$ that satisfy $\|P\|_\alpha \le \|A\| \cdot \|B\|$ whenever $P = A \otimes B$. Tensor norms have been studied for decades and have yielded a great deal of rich concepts and deep results (see, e.g., [16]). In recent years, they have been applied to quantum information theory to characterize and quantify the nonlocality of quantum states [37, 38]. The tensor norms that appear in our upper bounds capture the nonlocality of bipartite operators in their own ways and may have further applications.

1.3. Applications on quantum communication complexity. After obtaining general upper bounds on $\mathrm{Com}(Q)$, we show that they in turn have useful applications on quantum communication complexity. Recall that in the setting of communication complexity [42, 43], Alice and Bob wish to compute a function $f(x, y)$, of which x is known only to Alice and y is known only to Bob. The communication complexity of f is the minimum amount of information that Alice and Bob need to exchange in order to compute f correctly for any input. Communication complexity has been a major research field (see, e.g., the book [27]), with a wide range of applications.

A concrete application of our result is on the advantage of sharing entanglement in quantum protocols, a question that has puzzled many researchers [13, 9, 24, 31]. It is known that sharing entanglement could give a constant additive advantage [13, 9] or save a half of the communication [14]. However, little is known on the limit of the advantage. This is in sharp contrast with the classical case of sharing randomness, where we know that it can only save at most a logarithmic additive term [32]. If there is a quantum protocol that exchanges q qubits with m qubits of prior entanglement, then the best classical simulation we know (see, e.g., [25]) is $\exp(\Omega(q + m))$. This is embarrassingly large, especially when $q \ll m$. Using our upper bound on the classical communication complexity of nonlocal operators, we prove the following result. Recall that in the *simultaneous message passing (SMP)* model with shared randomness, the two parties holding the inputs share an arbitrarily long random string and send a

single message to a third party, who is required to determine the outcome correctly with high probability.

THEOREM 1.2. *If a two-party quantum protocol uses q qubits of communication and m qubits of share entanglement, then it can be simulated by a classical protocol using* $\exp(O(q))$ *bits with shared randomness. The simulation does* not *depend on m. Furthermore, it can be carried out in the SMP model with shared randomness.*

Notice that the exponential dependence on q cannot be improved because of the existence of an exponential gap between the quantum and classical communication complexities for partial functions [35]. As a consequence of the above theorem, we have the following corollary.

COROLLARY 1.3. *If a communication complexity problem has a constant cost quantum communication protocol with shared entanglement, it also has a constant cost classical SMP protocol with shared randomness.*

It is interesting to contrast the above with the following recent result by Yao [44].

THEOREM 1.4 (see [44]). *If a communication complexity problem of input size n has a constant cost classical SMP protocol with shared randomness, then it has an* $O(\log n)$ *cost quantum SMP protocol* without *shared entanglement.*

Combining this result with ours, we have the next corollary.

COROLLARY 1.5. *If a communication complexity problem of input size n has a constant cost two-party quantum protocol with shared entanglement, it has an* $O(\log n)$ *cost quantum SMP protocol without shared entanglement.*

1.4. Applications on simulating quantum correlations. Yet another application of our result is in the classical simulation of quantum correlations. Suppose that two parties, Alice and Bob, are given an entangled quantum state $|E\rangle$. Alice then applies a local measurement Q_A to her system and Bob applies Q_B to his. The result is a correlated joint distribution $\mu(|E\rangle, Q_A, Q_B)$ on both measurement outcomes. If Alice and Bob do not communicate, the resulting distribution may violate the Bell inequalities, and hence may be impossible to generate by the hidden variable model of classical physics [3].

A natural question following from the above work of Bell is how much communication is required for Alice and Bob to simulate the quantum correlation. Most works in this direction focus on the exact simulation and on measurements applied to a constant number of qubits [41, 2, 15, 39, 8, 29]. We study the *approximate* simulation of quantum correlations, where the measurement outcomes take a constant number of possible values while the dimension of the shared entangled state may be arbitrarily large.

THEOREM 1.6 (informally). *Let $\delta \in (0, 2)$, and let $|E\rangle$ be a bipartite state. Then there exists a randomized communication protocol that exchanges an* $O\left(\log \frac{1}{\delta}/\delta^2\right)$ *number of bits, and for each pair of local measurements $(R_A,\ R_B)$, the protocol outputs a distribution that is within δ deviation in statistical distance from $\mu(|E\rangle, R_A, R_B)$.*

1.5. Organization. The rest of the paper is organized as follows. We start with the description of a general framework for classical simulation of quantum protocols. The cost parameter of this framework is then optimized in the next section, giving the main theorem. In the section that follows we give applications of the theorem. Finally, we conclude with several open problems.

2. A simulation framework. Our classical simulation of quantum protocols falls into the following framework. Let p be the acceptance probability (i.e., the probability of outputting 1) of a given quantum protocol (which arises either from a

communication task or from a bipartite measurement). We use $p = \langle\psi_A|\psi_B\rangle$ to denote two vectors $|\psi_A\rangle$ and $|\psi_B\rangle$ that can be prepared by Alice and Bob, respectively. Note that the lengths of the two vectors may be very large in general. Indeed, the shorter their lengths are, the better the simulation is.

More precisely, if for some number C, $\||\psi_A\rangle\| \le C$ and $\||\psi_B\rangle\| \le C$, then the following simulation uses $O(C^4)$ bits. Alice and Bob send Charlie $\||\psi_A\rangle\|$ and $\||\psi_B\rangle\|$, respectively, up to $O(1/C)$ precision. This requires $O(\log C)$ bits. They then proceed to estimate $\cos\theta$ for the angle θ between $|\psi_A\rangle$ and $|\psi_B\rangle$ up to a precision of $O(1/C^2)$. The protocol in Kremer, Nisan, and Ron [26], which is based on the following observation of Goemans and Williamson [20], gives a protocol that accomplishes the latter task using $O(C^4)$ bits.

Assume for simplicity that all vectors are real (the complex number case can be easily reduced to the real case). If $|\psi\rangle$ is a random unit vector in the same space of $|\phi_A\rangle$ and $|\phi_B\rangle$, then

$$\text{Prob}\left[\text{sign}(\langle\psi|\psi_A\rangle) \neq \text{sign}(\langle\psi|\psi_B\rangle)\right] \;=\; \theta/\pi. \tag{2}$$

Hence, in order to estimate $\cos\theta$ with error term δ', it suffices to estimate θ/π to some error term $O(\delta')$ using the above equality checking of signs. This can be achieved by a simple SMP protocol that repeats the following: Alice and Bob interpret the shared randomness as a unit vector $|\psi\rangle$; they then send $\text{sign}(\langle\psi|\psi_A\rangle)$ and $\text{sign}(\langle\psi|\psi_B\rangle)$, respectively, to Charlie. By a simple application of the Chernoff bound, an $O\left(\log\frac{1}{\epsilon}/\delta'^2\right)$ number of repetitions is sufficient to give an estimation of p with δ' accuracy, and the protocol may fail with $\le \epsilon$ probability. With $\delta' = O(\delta/C^2)$, this is $O\left(C^4\log\frac{1}{\epsilon}/\delta^2\right)$ bits.

We note that, using a similar approach, Toner and Bacon [41] gave a protocol that produces a random ±1 variable whose expectation is precisely $\cos\theta$. Their protocol is not asymptotically more efficient than the above. We summarize the above simulation result as follows.

THEOREM 2.1 (see [26, 20]). *Suppose that the acceptance probability p of a quantum protocol can be expressed as $p = \langle\psi_A|\psi_B\rangle$, where $|\psi_A\rangle$ and $|\psi_B\rangle$ are vectors that can be prepared by each party individually and $\||\psi_A\rangle\|,\ \||\psi_B\rangle\| \le C$ for some $C > 0$. Then there is a randomized SMP protocol with shared randomness in which* (a) *Alice and Bob send $O\left(C^4\log\frac{1}{\epsilon}/\delta^2\right)$ bits to Charlie, and* (b) *Charlie outputs a number p' such that $|p' - p| \le \delta$ with probability $\ge 1 - \epsilon$.*

3. The main theorem. In this section, we formally define the classical communication complexity and the diamond norm of bipartite quantum operators, and derive an upper bound on the former in terms of the latter. We shall focus on the case when the measurement gives two outcomes and the dimensions of the two systems are the same. Our results can be extended trivially to more general cases.

We use script letters, $\mathcal{N}$, $\mathcal{M}$, $\mathcal{F}$, ..., to denote Hilbert spaces and use $\mathbf{L}(\mathcal{N})$ to denote the space of operators on $\mathcal{N}$. The identity operator on $\mathcal{N}$ is denoted by $I_{\mathcal{N}}$, and the identity superoperator on $\mathbf{L}(\mathcal{N})$ is denoted by $\mathbf{I}_{\mathcal{N}}$. Recall that a *positive-operator-valued measurement (POVM)* on a Hilbert space $\mathcal{H}$ is a set of positive semidefinite operators $\{Q_1, Q_2, \ldots, Q_m\}$ on $\mathcal{H}$ such that $\sum_{i=1}^m Q_i = I_{\mathcal{H}}$. Each Q_i is called a *measurement element* and corresponds to the measurement outcome i. We may refer to a semidefinite operator Q, $0 \le Q \le 1$, as a *measurement element* of the implicit binary POVM $\{Q, I - Q\}$. In this paper, *physically realizable operator* means a completely positive superoperator that does not increase the trace. For more details on the basic notions of quantum information processing, see the textbooks [33, 34].

3.1. Classical simulation of bipartite measurement. In this subsection we define the central concept of this paper, which is the classical communication complexity of quantum measurement.

DEFINITION 3.1. *Let $\delta, \epsilon \in (0, 1/2)$, and let $Q \in \mathbf{L}(\mathcal{N}_A \otimes \mathcal{N}_B)$ be a measurement element. The* classical communication complexity *of Q with precision δ and success probability $1-\epsilon$, denoted by $\mathrm{Com}_{\delta,\epsilon}(Q)$, is the minimum integer k such that there is a randomized communication protocol between two parties Alice and Bob with shared randomness that uses k bits and satisfies the following conditions for any bipartite state $|E\rangle \in \mathbf{L}(\mathcal{M}_A \otimes \mathcal{M}_B)$ and physically realizable operators $R_A : \mathbf{L}(\mathcal{M}_A) \to \mathbf{L}(\mathcal{N}_A)$ and $R_B : \mathbf{L}(\mathcal{M}_B) \to \mathbf{L}(\mathcal{N}_B)$:*

1. *Alice's input is a classical description of $|E\rangle$ and R_A; similarly, Bob's input is a classical description of $|E\rangle$ and R_B.*
2. *The protocol outputs a real number p such that*

$$|p - \mathrm{tr}(Q(R_A \otimes R_B)(|E\rangle\langle E|))| \le \delta$$

with probability at least $1-\epsilon$. The probability is over the shared randomness.

3.2. The diamond norm on bipartite operators. Let $\mathcal{N}$ be a Hilbert space and $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{N})$ be a superoperator. The *diamond norm* on superoperators is defined as [23]

$$\|T\|_\diamond \stackrel{\mathrm{def}}{=} \inf\{\|A\|\|B\| : \mathrm{tr}_{\mathcal{F}}(A \cdot B^\dagger) = T,\ A,\ B \in \mathbf{L}(\mathcal{N}, \mathcal{N} \otimes \mathcal{F})\}.$$

For our application, the following alternative characterization of the diamond norm is more convenient.

LEMMA 3.2 (see, e.g., [23]). *For any superoperator T,*

$$\|T\|_\diamond = \inf\left\{ \left\|\sum_t A_t^\dagger A_t\right\|^{1/2} \cdot \left\|\sum_t B_t^\dagger B_t\right\|^{1/2} : A_t,\ B_t \in \mathbf{L}(\mathcal{N}),\ T = \sum_t A_t \cdot B_t^\dagger \right\}.$$

Let $\mathcal{N}_A$, $\mathcal{N}_B$, and $\mathcal{N}$ be Hilbert spaces of the same dimension. We fix an isomorphism between any two of them. For an operator in one space, we use the same notation for its images and preimages, under the isomorphisms, in the other spaces.

Let $Q \in \mathbf{L}(\mathcal{N}_A \otimes \mathcal{N}_B)$ be a bipartite operator, and let $Q = \sum_t A_t \otimes B_t^\dagger$ for some $A_t \in \mathbf{L}(\mathcal{N}_A)$ and $B_t \in \mathbf{L}(\mathcal{N}_B)$. Define a mapping $\mathcal{T}$ from bipartite operators on $\mathcal{N}_A \otimes \mathcal{N}_B$ to superoperators $\mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{N})$ by mapping $Q \mapsto \mathcal{T}(Q) \stackrel{\mathrm{def}}{=} \sum_t A_t \cdot B_t^\dagger$. By direct inspection, the mapping is independent of the choice of the decomposition of Q and is in fact an isomorphism.

DEFINITION 3.3. *Let $Q \in \mathbf{L}(\mathcal{N}_A \otimes \mathcal{N}_B)$ be a bipartite operator. The* diamond norm *of Q, denoted by $\|Q\|_\diamond$, is $\|Q\|_\diamond \stackrel{\mathrm{def}}{=} \|\mathcal{T}(Q)\|_\diamond$.*

By Lemma 3.2, for any Q,

$$\|Q\|_\diamond = \inf\left\{ \left\|\sum_t A_t^\dagger A_t\right\|^{1/2} \cdot \left\|\sum_t B_t^\dagger B_t\right\|^{1/2} : \right.$$
$$\left. A_t \in \mathbf{L}(\mathcal{N}_A),\ B_t \in \mathbf{L}(\mathcal{N}_B),\ Q = \sum_t A_t \otimes B_t^\dagger \right\}.$$

Note that if a superoperator $T = A \cdot B$ for some $A, B \in \mathbf{L}(\mathcal{N})$, $\|T\|_\diamond = \|A\| \cdot \|B\|$. Therefore the diamond norm on bipartite operators is a tensor norm.

LEMMA 3.4. *For all* $A, B \in \mathbf{L}((N))$, $\|A \otimes B\|_\diamond = \|A\| \cdot \|B\|$.

A nice property of the superoperator diamond norm is that it is "stable"; i.e., it remains unchanged when tensored with the identity operator on an additional space [23].

LEMMA 3.5. *Let* $\mathcal{N}$, $\mathcal{M}$, *and* $\mathcal{F}$ *be Hilbert spaces, and let* $T : \mathbf{L}(\mathcal{N}) \to \mathbf{L}(\mathcal{M})$ *be a superoperator. Then* $\|\mathbf{I}_\mathcal{F} \otimes T\|_\diamond = \|T\|_\diamond$.

This stability property carries over to our diamond norm and is important for our applications. Let $\mathcal{F}_A$ and $\mathcal{F}_B$ be Hilbert spaces of the same dimension, and let $Q \in \mathbf{L}(\mathcal{N}_A \otimes \mathcal{N}_B)$. Denote by $Q_{\mathcal{F}_A,\mathcal{F}_B}$ the bipartite operator $Q \otimes I_{\mathcal{F}_A \otimes \mathcal{F}_B}$, where the two subsystems are $\mathcal{N}_A \otimes \mathcal{F}_A$ and $\mathcal{N}_B \otimes \mathcal{F}_B$.

LEMMA 3.6. *For any* Q, $\|Q_{\mathcal{F}_A,\mathcal{F}_B}\|_\diamond = \|Q\|_\diamond$.

If Q is a measurement element of a POVM acting on a Hilbert space of dimension K, then we have the following upper bound on $\|Q\|_\diamond$.

PROPOSITION 3.7. *Let* Q *be a bipartite measurement element acting on* $\mathcal{H}_A \otimes \mathcal{H}_B$. *Then* $\|Q\|_\diamond \leq \dim(\mathcal{H}_A \otimes \mathcal{H}_B)$.

Proof. Let $K_{\min} = \min\{\dim(\mathcal{H}_A), \dim(\mathcal{H}_B)\}$, and let $K = \dim(\mathcal{H}_A \otimes \mathcal{H}_B)$. Fix a pure state $|u\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$ and denote its Schmidt decomposition as

$$|u\rangle = \sum_{i=1}^{K_{\min}} \sqrt{p_i}|i\rangle_A|i\rangle_B,$$

where $p_i \geq 0$, $\sum p_i = 1$, and $\{|i\rangle_A : 1 \leq i \leq \dim(\mathcal{H}_A)\}$, $\{|i\rangle_B : 1 \leq i \leq \dim(\mathcal{H}_B)\}$ are orthonormal bases for the two subsystems, respectively. Then

$$|u\rangle\left\langle u| = \sum_{i,j=1}^{K_{\min}} \sqrt{p_i p_j}|i\right\rangle_A \langle j|_A \otimes |i\rangle_B\langle j|_B.$$

Let $A_{i,j} = \sqrt{p_i}|i\rangle_A\langle j|_A$ and $B_{i,j} = \sqrt{p_j}|j\rangle_B\langle i|_B$. We have

$$\left\| \sum_{i,j=1}^{K_{\min}} A_{i,j}^\dagger A_{i,j} \right\| = \left\| \sum_{i,j} p_i \langle i|_A |i\rangle_A |j\rangle_A \langle j|_A \right\| = 1.$$

Similarly, $\| \sum_{i,j} B_{i,j}^\dagger B_{i,j} \| = 1$. Thus,

$$\||u\rangle\langle u|\|_\diamond = \|\mathcal{T}(|u\rangle\langle u|)\|_\diamond \leq \sqrt{\left\| \sum_{i,j} A_{i,j}^\dagger A_{i,j} \right\|} \sqrt{\sum_{i,j} B_{i,j}^\dagger B_{i,j}} \leq 1.$$

Since Q a is measurement element, $Q = \sum_{i=1}^K c_i |u_i\rangle\langle u_i|$ for some $c_i \in [0,1]$ and an orthonormal basis $\{u_i : 1 \leq i \leq K\}$. Thus $\|Q\|_\diamond \leq \sum_{i=1}^K c_i \||u_i\rangle\langle u_i|\|_\diamond \leq K$. ∎

This bound is not far from being optimal for IP_n, in which case $K = 2^{2n}$. To prove a lower bound on $\|\mathrm{IP}_n\|_\diamond$, we use a fundamental characterization of the diamond norm (see [23, Theorem 11.1]).

THEOREM 3.8 (see [23]). *Let* $T : L(\mathcal{N}) \to L(\mathcal{N})$ *be a superoperator and* $\mathcal{G}$ *be a space of the same dimension as* $\mathcal{N}$. *Then*

$$\|T\|_\diamond = \sup_{\rho \in L(\mathcal{N} \otimes \mathcal{G}), \rho \neq 0} \frac{\|(T \otimes \mathbf{I}_\mathcal{G})(\rho)\|_{\mathrm{tr}}}{\|\rho\|_{\mathrm{tr}}}. \tag{3}$$

PROPOSITION 3.9. *For the* IP_n *operator defined in* (1), $\|\mathrm{IP}_n\|_\diamond \geq 2^{n/2-1} - 1/2$.

Proof. By definition,

$$\mathcal{T}(\mathrm{IP}_n) = \sum_{\substack{x,y\in\{0,1\}^n \\ x\cdot y=1}} |x\rangle\langle x| \cdot |y\rangle\langle y|.$$

Setting $\rho = \sum_{x,y} |x\rangle\langle y| \otimes I_{\mathcal{G}}$ in (3), we have

$$\|\mathcal{T}(\mathrm{IP}_n)\|_\diamond \;\geq\; \frac{1}{2^n} \left\| \sum_{\substack{x,y\in\{0,1\}^n \\ x\cdot y=1}} |x\rangle\langle y| \right\|_{\mathrm{tr}}.$$

Let $A = \sum_{x,y\in\{0,1\}^n,\ x\cdot y=1} |x\rangle\langle y|$, J be the all-one matrix, and let H be the Hadamard matrix (i.e., $\sum_{x,y\in\{0,1\}^n}(-1)^{x\cdot y}|x\rangle\langle y|$). Then $A = \frac{1}{2}(J - H)$. The largest eigenvalue of J is 2^n and all other eigenvalues are 0; thus $\|J\|_{\mathrm{tr}} = 2^n$. All of the eigenvalues of H are $\pm 2^{n/2}$; thus $\|H\|_{\mathrm{tr}} = 2^{3n/2}$. Therefore,

$$\|A\|_{\mathrm{tr}} \geq \frac{1}{2}(\|H\|_{\mathrm{tr}} - \|J\|_{\mathrm{tr}}) = \frac{1}{2}(2^{3n/2} - 2^n).$$

Thus $\|\mathrm{IP}_n\|_\diamond \geq 2^{n/2-1} - 1/2$. □

We conclude this subsection by noting that our diamond norm on bipartite operators appears natural in connection with the following matrix analogy of the Cauchy–Schwarz inequality.

THEOREM 3.10 (see Jocić [21]). *For any operators* A_t *and* B_t,

$$\left\| \sum_t A_t \otimes B_t^\dagger \right\| \leq \left\| \sum_t A_t^\dagger A_t \right\|^{1/2} \cdot \left\| \sum_t B_t^\dagger B_t \right\|^{1/2}. \tag{4}$$

Inequality (4) may actually be proved by the same approach that we use to prove Theorem 3.11 below.

3.3. Upperbounding Com(Q) by the diamond norm. We now use the diamond norm to derive an upper bound on $\mathrm{Com}_{\delta,\epsilon}(Q)$. Recall that if $\mathcal{M}$ and $\mathcal{N}$ are two Hilbert spaces, an *isometric embedding* $U : \mathcal{M} \to \mathcal{N}$ is a linear map that satisfies $U^\dagger U = I_{\mathcal{M}}$.

THEOREM 3.11. *For any bipartite measurement element* Q,

$$\mathrm{Com}_{\delta,\epsilon}(Q) = O\left(\|Q\|_\diamond^2 \cdot \log \frac{1}{\epsilon} \Big/ \delta^2 \right). \tag{5}$$

In particular, $\mathrm{Com}_{\delta,\epsilon}(Q) = O(K^2 \log \frac{1}{\epsilon}/\delta^2)$, *where* K *is the dimension of the space on which* Q *acts. Furthermore, this upper bound can be achieved by an SMP protocol with shared randomness.*

Proof. Fix a bipartite state $|E\rangle \in \mathcal{M}_A \otimes \mathcal{M}_B$ and physically realizable operators $R_A : \mathbf{L}(\mathcal{M}_A) \to \mathbf{L}(\mathcal{N}_A)$, $R_B : \mathbf{L}(\mathcal{M}_B) \to \mathbf{L}(\mathcal{N}_B)$. We first consider the case when $R_A = U \cdot U^\dagger$ for an operator $U : \mathcal{M}_A \to \mathcal{N}_A$ with $\|U\| \leq$, and similarly, $R_B = V \cdot V^\dagger$

for $V : \mathcal{M}_B \to \mathcal{N}_B$ with $\|V\| \le 1$. Without loss of generality, we assume that Alice and Bob have agreed on a Schmidt decomposition,

$$|E\rangle = \sum_{i=1}^{\min\{\dim(\mathcal{N}_A),\dim(\mathcal{N}_B)\}} \sqrt{p_i}|i\rangle_A \otimes |i\rangle_B,$$

for some $p_i \ge 0$, $\sum_i p_i = 1$, for an orthonormal basis $\{|i\rangle_A\}_i$ of $\mathcal{N}_A$, and for an orthonormal basis $\{|i\rangle_B\}_i$ of $\mathcal{N}_B$. Denote by $|i_A\rangle \stackrel{\text{def}}{=} U|i\rangle$, and $|i_B\rangle \stackrel{\text{def}}{=} V|i\rangle$. Then Q is applied to $|\bar{E}\rangle \stackrel{\text{def}}{=} (U \otimes V)|E\rangle = \sum_i \sqrt{p_i}|i_A\rangle \otimes |i_B\rangle$.

Suppose $\|Q\|_\diamond$ is achieved under the decomposition $Q = \sum_t A_t \otimes B_t^\dagger$ with which, if $Q_A \stackrel{\text{def}}{=} \sum_t A_t^\dagger A_t$ and $Q_B \stackrel{\text{def}}{=} \sum_t B_t^\dagger B_t$, we have $\|Q_A\| = \|Q_B\| = \|Q\|_\diamond$. With those definitions, we have

$$p = \langle \bar{E}|Q|\bar{E}\rangle = \sum_{i,j,t} \sqrt{p_i p_j}\ \langle i_A|A_t|j_A\rangle \cdot \langle i_B|B_t^\dagger|j_B\rangle.$$

Define two vectors

$$|\psi_A\rangle \stackrel{\text{def}}{=} \sum_{i,j,t} \sqrt{p_j}\ \langle j_A|A_t^\dagger|i_A\rangle\ |i,j,t\rangle, \tag{6}$$

$$|\psi_B\rangle \stackrel{\text{def}}{=} \sum_{i,j,t} \sqrt{p_i}\ \langle i_B|B_t^\dagger|j_B\rangle\ |i,j,t\rangle. \tag{7}$$

Then $p = \langle \psi_A|\psi_B\rangle$. With $\rho_A \stackrel{\text{def}}{=} \sum_j p_j |j_A\rangle\langle j_A|$,

$$\langle \psi_A|\psi_A\rangle = \sum_{i,j,t} p_j |\langle j_A|A_t^\dagger|i_A\rangle|^2 \le \operatorname{tr}(\rho_A Q_A) \le \|Q_A\| = \|Q\|_\diamond.$$

Similarly, $\langle \psi_B|\psi_B\rangle \le \|Q_B\| = \|Q\|_\diamond$. Therefore, by Theorem 2.1, there is an SMP protocol that outputs an δ-approximation of p with probability $\ge 1 - \epsilon$ using $O\left(\|Q\|_\diamond^2 \cdot \log \frac{1}{\epsilon}/\delta^2\right)$ bits. This bound is $O(K^2 \log \frac{1}{\epsilon}/\delta^2)$ as $\|Q\|_\diamond \le K$ by Proposition 3.7.

In general, $R_A = \operatorname{Trace}_{\mathcal{F}_A}(U \cdot U^\dagger)$ for an operator $U_A : \mathcal{M}_A \to \mathcal{N}_A \otimes \mathcal{F}_A$ with $\|U\| \le 1$, and similarly, $R_B = \operatorname{Trace}_{\mathcal{F}_B}(V \cdot V^\dagger)$ for $V : \mathcal{M}_B \to \mathcal{N}_B \otimes \mathcal{F}_B$. Then $p = \operatorname{tr}(Q \otimes I_{\mathcal{F}_A \otimes \mathcal{F}_B}(U \otimes V)|E\rangle\langle E|(U^\dagger \otimes V^\dagger))$. Thus this case reduces to the first case, since $\|Q_{\mathcal{F}_A,\mathcal{F}_B}\|_\diamond = \|Q\|_\diamond$, by Lemma 3.6. □

Remark 3.12. One may improve the above upper bound on $\operatorname{Com}_{\delta,\epsilon}(Q)$ by a more carefully chosen $|\psi_A\rangle$ and $|\psi_B\rangle$ in (6) and (7). More specifically, let $\alpha \in [0,1]$, define

$$|\psi_A^\alpha\rangle = \sum_{i,j,t} \sqrt{p_i^\alpha p_j^{1-\alpha}}\ \langle j_A|A_t^\dagger|i_A\rangle\ |i,j,t\rangle$$

and

$$|\psi_B^\alpha\rangle = \sum_{i,j,t} \sqrt{p_i^{1-\alpha} p_j^\alpha}\ \langle i_B|B_t^\dagger|j_B\rangle\ |i,j,t\rangle.$$

One can verify that minimizing $\||\psi_A\rangle\| \cdot \||\psi_B\rangle\|$ over all decompositions of Q gives rise to a tensor norm; we do not know if is stable under tensoring with identity

superoperators. Although we have not found any useful application of an $\alpha \neq 0$, we cannot rule out the possibility that a carefully chosen α may give a better bound.

Remark 3.13. In the case that $|E\rangle$ is not entangled, the approach in Theorem 3.11 can be used to derive a systematic classical simulation. More specifically, in this context we would like to estimate $p = \langle\phi_A|\otimes\langle\phi_B|Q|\phi_A\rangle\otimes|\phi_B\rangle$ for a state $|\phi_A\rangle$ known only to Alice and a state $|\phi_B\rangle$ known only to Bob. For a decomposition of $Q = \sum_t A_t \otimes B_t^\dagger$, we define

$$|\psi_A\rangle = \sum_t \langle\phi_A|A_t^\dagger|\phi_A\rangle|t\rangle \quad \text{and} \quad |\psi_B\rangle = \sum_t \langle\phi_B|B_t^\dagger|\phi_B\rangle|t\rangle.$$

Then $p = \langle\psi_A|\psi_B\rangle$. It can be verified that

$$\|Q\|_\otimes \stackrel{\text{def}}{=} \inf\left\{\|\psi_A\| \cdot \|\psi_B\| : Q = \sum_t A_t \otimes B_t^\dagger\right\}$$

is a tensor norm and $\|Q\|_\otimes \leq \|Q\|_\diamond$. This approach gives a constant cost simulation of the elegant quantum fingerprint protocol of Buhrman et al. [10] for testing equality of two input strings.

4. Applications. We now apply Theorem 3.11 to derive classical upper bounds on quantum communication complexity.

4.1. Quantum SMP with shared entanglement. If the quantum protocol is in the SMP model with shared entanglement, we immediately have the following.

COROLLARY 4.1 (of Theorem 3.11). *If, in a quantum SMP protocol, Charlie applies a measurement P, then the protocol can be simulated by a classical SMP protocol with shared coins and using $O(\|P\|_\diamond^2)$ bits.*

4.2. Two-party interactive quantum communication with shared entanglement. Now consider the general two-party interactive quantum communication. We need the following lemma by Razborov [36] and Yao [43].

LEMMA 4.2 (see [36, 43]). *Let $\mathcal{P}$ be a two-party interactive quantum communication protocol that uses q qubits. Let $\mathcal{H}_A$ and $\mathcal{H}_B$ be the state spaces of Alice and Bob, respectively. For an input (x, y), denote by $|\Phi_{x,y}\rangle_{AB}$ the joint state of Alice and Bob before the protocol starts. Then there exist linear operators $A_h \in \mathbf{L}(\mathcal{H}_A)$ and $B_h \in \mathbf{L}(\mathcal{H}_B)$ for each $h \in \{0,1\}^{q-1}$ such that*

(a) *$\|A_h\| \leq 1$ and $\|B_h\| \leq 1$ for all $h \in \{0,1\}^{q-1}$;*

(b) *the acceptance probability of $\mathcal{P}$ on input x and y is $\|P|\Phi_{x,y}\rangle\|^2$, where*

$$P \stackrel{\text{def}}{=} \sum_{h\in\{0,1\}^{q-1}} A_h \otimes B_h.$$

We are now ready to prove Theorem 1.2.

Proof of Theorem 1.2. Let $|E\rangle \in \mathcal{N}_A \otimes \mathcal{N}_B$ be the shared entanglement. For an n-bit binary string x, denote by $U_x : \mathcal{N}_A \to \mathcal{H}_A$ the isometric embedding that maps $|\phi\rangle \mapsto |\phi\rangle\otimes|x\rangle\otimes|0\cdots0\rangle$. The isometric embedding $V_y : \mathcal{N}_B \to \mathcal{H}_B$ is defined similarly. Then $|\Phi_{x,y}\rangle = (U_x\otimes V_y)|E\rangle$. Let P, A_h, and B_h be the quantum protocol and the linear operators described in Lemma 4.2. To simulate the quantum protocol is to simulate the measurement element $P^\dagger P$ on $|E\rangle$, $U_x \cdot U_x^\dagger$, and $V_y \cdot V_y^\dagger$. By Theorem 3.11, the acceptance probability can be estimated with $O(\|P^\dagger P\|_\diamond^2)$ bits of communication in

the SMP model with shared randomness. Since

$$\|P^\dagger P\|_\diamond \le \sum_{h,h'} \| \left((A_{h'})^\dagger A_h\right) \otimes \left((B_{h'})^\dagger B_h\right) \|_\diamond = \sum_{h,h'} \|A_h\|\|A_{h'}\|\|B_h\|\|B_{h'}\| \le 2^{2(q-1)},$$

the number of bits communicated in the protocol is $\exp(O(q))$. ∎

Corollary 1.3 follows trivially from the above by setting q to be a constant. Corollary 1.5 follows immediately from Theorem 1.4 and Corollary 1.3.

4.3. Simulating quantum correlations. We shall define precisely what we mean by simulating quantum correlations.

DEFINITION 4.3. *A* quantum measurement game *is a tuple* $G = (|E\rangle_{AB}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{V}_A, \mathcal{V}_B)$*, where* $|E\rangle_{AB}$ *is a bipartite quantum state,* $\mathcal{P}_A$ *(*$\mathcal{P}_B$*) is a set of POVM measurements on the system A (B), and the union of their measurement outcomes is* $\mathcal{V}_A$ *(*$\mathcal{V}_B$*).*

We denote by $\{P_A^v : v \in \mathcal{V}_A\}$ the measurement elements for $P_A \in \mathcal{P}_A$ (and similarly for $P_B \in \mathcal{P}_B$), where P_A^v is the measurement element yielding outcome v. For $P_A \in \mathcal{P}_A$ and $P_B \in \mathcal{P}_B$, denote by $\omega_G(P_A, P_B)$ the distribution on $\mathcal{V}_A \times \mathcal{V}_B$ when $P_A \otimes P_B$ is applied to $|E\rangle$. Recall that the statistical distance between two distributions $\pi = (p_1, \ldots, p_n)$ and $\tilde{\pi} = (\tilde{p}_1, \ldots, \tilde{p}_n)$ is $\|\pi - \tilde{\pi}\|_1 \stackrel{\text{def}}{=} \sum_i |p_i - \tilde{p}_i|$.

DEFINITION 4.4. *Let* $\delta \in (0, 2)$*. A* classical simulation *of a quantum measurement game* $G = (|E\rangle_{AB}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{V}_A, \mathcal{V}_B)$ *with statistical distance* δ *is a randomized communication protocol with shared randomness between two parties Alice and Bob such that*

(a) *Alice's input is a classical description of an element* $P_A \in \mathcal{P}_A$*, and Bob's input is a classical description of an element* $P_B \in \mathcal{P}_B$*;*
(b) *at the end of the protocol, Alice (resp., Bob) outputs some* $v \in \mathcal{V}_A$ *(*$v' \in \mathcal{V}_B$*), resulting in a distribution* $\tilde{\omega}(P_A, P_B)$ *on* $\mathcal{V}_A \times \mathcal{V}_B$*;*
(c) *for all* $(P_A, P_B) \in \mathcal{P}_A \times \mathcal{P}_B$*,* $\|\tilde{\omega}(P_A, P_B) - \omega(P_A, P_B)\|_1 \le \delta$*.*

We are now able to rigorously state Theorem 1.6.

THEOREM 4.5. *Let* $G = (|E\rangle_{AB}, \mathcal{P}_A, \mathcal{P}_B, \mathcal{V}_A, \mathcal{V}_B)$ *be a quantum measurement game. Let* $m = |\mathcal{V}_A| \cdot |\mathcal{V}_B|$*, and let* $\delta \in (0, 2)$*. There is a classical simulation of* G *with* δ *statistical distance that exchanges an* $O(\frac{m^3}{\delta^2} \cdot \log \frac{m}{\delta})$ *number of bits. In particular, the simulation cost is* $O(\log \frac{1}{\delta}/\delta^2)$ *if* $m = O(1)$*.*

Proof. Let $\epsilon = \delta/4$. Fix a pair of measurements $(P_A, P_B) \in \mathcal{P}_A \times \mathcal{P}_B$. In the classical simulation protocol, for each $(v, v') \in \mathcal{V}_A \times \mathcal{V}_B$, Alice and Bob first compute the probability of outputting (v, v') to be within $\frac{\delta}{2m}$ deviation with probability at least $1 - \epsilon/m$. Then with probability $\ge 1 - \epsilon$, all those computed probabilities are within $\frac{\delta}{2m}$ deviation to the correct values. They then output a random (v, v') according to the probabilities computed. Then

$$\|\tilde{\omega}(P_A, P_B) - \omega(P_A, P_B)\|_1 \le \epsilon \cdot 2 + (1 - \epsilon) \cdot \frac{\delta}{2} \le \delta.$$

Now fix a pair of possible outcome (v, v'). The estimation of $\omega_G(P_A, P_B)$ becomes the simulation of the identity operator, with the initial state being $|E\rangle$ and the local physically realizable operators being $\sqrt{P_A^v} \cdot \sqrt{P_A^v}$ and $\sqrt{P_B^{v'}} \cdot \sqrt{P_B^{v'}}$ for some $P_A \in \mathcal{P}_A$ and $P_B \in \mathcal{P}_B$.

Note that the identity operator has diamond norm 1 (this follows from Definition 3.3 and Theorem 3.8). By Theorem 3.11, the probability of observing outcome (v, v') can be calculated with precision $\frac{\delta}{2m}$ and with probability at least $1 - \epsilon/m$

by a classical protocol using $O(\frac{m^2}{\delta^2}\log(\frac{m}{\epsilon}))$ bits. Thus the overall simulation cost is $O(\frac{m^3}{\delta^2}\log(\frac{m}{\delta}))$ bits, which is $O(\log\frac{1}{\delta}/\delta^2)$ when $m = O(1)$. □

5. Conclusion and open problems. We introduce an alternative measure of nonlocality of bipartite quantum measurement, which is the minimum amount of classical communication required to simulate the quantum measurement. We give an upper bound of this nonlocality measure by constructing in terms of a tensor norm. Variants of our upper bound protocol also lead to variants of the main upper bound in terms of other tensor norms. We then apply our upper bound to the classical simulation of quantum communication protocols and the approximation of quantum correlations by local hidden variable models augmented with classical communication. In particular, we show that quantum and classical communication protocols with unlimited shared entanglement or randomness compute the same set of functions if the amount of communication is a constant. We also show that local measurement of an entangled state can be simulated by a local hidden variable model with a constant amount of communication, as long as the number of measurement outcomes is a constant.

Our study is only the first step toward understanding the classical communication complexity of bipartite measurement. An obvious open problem is to prove or disprove that the bound in Theorem 3.11 is tight. Another basic problem is to prove a strong lower bound (exponential in the number of qubits) on $\mathrm{Com}(Q)$ for some Q.

It would be interesting to relate $\mathrm{Com}(Q)$ to other measures of nonlocality, such as entanglement capacity and the minimum number of elementary gates required, or the amount of time for evolving an elementary Hamiltonian, in order to approximate Q. It is conceivable that the comparison of those measures may lead to a unique and representative measure of operator nonlocality.

Gavinsky [19] made recent progress on the question of the usefulness of quantum entanglement. He showed that entanglement is responsible for the exponential quantum saving in performing some communication task in a restricted communication model. Whether or not entanglement could result in exponential savings for the two-party interactive communication model and for the computation of functions remains open. Can Theorem 1.2 be strengthened so that the amount of entanglement need not exceed a linear size of the messages?

The cost of our protocol for simulating quantum correlations depends linearly on the number of measurement outcomes. Is this dependence necessary or can one dramatically reduce it?

Finally, it appears a very promising direction to us to explore further the connections of tensor norms and nonlocality of quantum states and operations.

Acknowledgments. We are indebted to Wei Huang, Amnon Ta-Shma, and the anonymous reviewers for their valuable suggestions on improving the presentation of this paper.

REFERENCES

[1] D. Aharonov, A. Kitaev, and N. Nisan, *Quantum circuits with mixed states*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1998, pp. 20–30.

[2] D. Bacon and B. F. Toner, *Bell inequalities with auxiliary communication*, Phys. Rev. Lett., 90 (2003), 157904.

[3] J. S. Bell, *On the Einstein–Podolsky–Rosen paradox*, Physics, 1 (1964), pp. 195–200.

[4] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, *Teleporting an unknown quantum state via dual classical and Einstein–Podolsky–Rosen channels*, Phys. Rev. Lett., 70 (1993), pp. 1895–1899.

[5] C. H. Bennett and G. Brassard, *Quantum cryptography: Public key distribution and coin tossing*, in Proceedings of the IEEE international Conference on Computers, Systems and Signal Processing (Bangalore, India), IEEE Press, New York, 1984, pp. 175–179.

[6] C. H. Bennett, A. W. Harrow, D. W. Leung, and J. A. Smolin, *On the capacities of bipartite Hamiltonians and unitary gates*, IEEE Trans. Inform. Theory, 49 (2003), pp. 1895–1911.

[7] D. Bohm, *The Paradox of Einstein, Rosen, and Podolsky*, Quantum Theory and Measurement, Prentice–Hall, Englewood Cliffs, NJ, 1951, pp. 611–623.

[8] G. Brassard, R. Cleve, and A. Tapp, *Cost of exactly simulating quantum entanglement with classical communication*, Phys. Rev. Lett., 83 (1999), pp. 1874–1877.

[9] H. Buhrman, R. Cleve, and W. van Dam, *Quantum entanglement and communication complexity*, SIAM J. Comput., 30 (2001), pp. 1829–1841.

[10] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, *Quantum fingerprinting*, Phys. Rev. Lett., 87 (2001), 167902.

[11] A. M. Childs, H. L. Haselgrove, and M. A. Nielsen, *Lower bounds on the complexity of simulating quantum gates*, Phys. Rev. A, 68 (2003), pp. 52311–52316.

[12] A. M. Childs, D. W. Leung, F. Verstraete, and G. Vidal, *Asymptotic entanglement capacity of the Ising and anisotropic Heisenberg interactions*, Quantum Inf. Comput., 3 (2003), pp. 97–105.

[13] R. Cleve and H. Buhrman, *Substituting quantum entanglement for communication*, Phys. Rev. A, 56 (1997), pp. 1201–1204.

[14] R. Cleve, W. van Dam, M. Nielsen, and A. Tapp, *Quantum entanglement and the communication complexity of the inner product function*, in Quantum Computing and Quantum Communications, Lecture Notes in Comput. Sci. 1509, Springer, Berlin, 1999, pp. 61–74.

[15] J. A. Csirik, *Cost of exactly simulating a bell pair using classical communication*, Phys. Rev. A, 66 (2002), 014302.

[16] A. Defant and K. Floret, *Tensor Norms and Operator Ideals*, North–Holland Math. Stud. 176, North–Holland., Amsterdam, 1993.

[17] A. Einstein, B. Podolsky, and N. Rosen, *Can quantum-mechanical description of reality be considered complete?*, Phys. Rev., 47 (1935), pp. 777–780.

[18] A. K. Ekert, *Quantum cryptography based on Bell's theorem*, Phys. Rev. Lett., 67 (1991), pp. 661–663.

[19] D. Gavinsky, *On the role of shared entanglement*, Quantum Inform. Comput., to appear. Available online at http://www.arxiv.org/ebs/quant-ph/0604052.

[20] M. X. Goemans and D. P. Williamson, .879-*approximation algorithms for MAX CUT and MAX* 2*SAT*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1994, pp. 422–431.

[21] D. R. Jocić, *The Cauchy–Schwarz norm inequality for elementary operators in Schatten ideals*, J. London Math. Soc., 60 (1999), pp. 925–934.

[22] A. Kitaev and J. Watrous, *Parallelization, amplification, and exponential time simulation of quantum interactive proof systems*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2000, pp. 608–617.

[23] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classical and Quantum Computation*, Grad. Stud. Math. 47, AMS, Providence, RI, 2002. Translated by Lester J. Senechal from the 1999 Russian original.

[24] H. Klauck, *Lower bounds for quantum communication complexity*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), B. Werner, ed., IEEE Press, Piscataway, NJ, 2001, pp. 288–297.

[25] I. Kremer, *Quantum Communication*, Master's thesis, Computer Science Department, Hebrew University, Jerusalem, Israel, 1995.

[26] I. Kremer, N. Nisan, and D. Ron, *On randomized one-round communication complexity*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1995, pp. 596–605.

[27] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[28] H.-K. Lo and H. F. Chau, *Unconditional security of quantum key distribution over arbitrarily long distances*, Science, 283 (1999), pp. 2050–2056.

[29] T. Maudlin, *Bell's inequality, information transmission and prism Models*, in the Proceedings of PSA, Vol. 1 D. Hull, M. Forbes, and K. Okruhlik, eds., Philosophy of Science Association, Bloomsburg, PA, 1992, pp. 404–417.
[30] D. Mayers, *Unconditional security in quantum cryptography*, J. ACM, 48 (2001), pp. 351–406.
[31] A. Nayak and J. Salzman, *On communication over an entanglement-assisted quantum channel*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2002, pp. 698–704.
[32] I. Newman, *Private vs. common random bits in communication complexity*, Inform. Process. Lett., 39 (1991), pp. 67–71.
[33] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[34] J. Preskill, *Quantum Information and Computation*, Lecture Notes for course Physics 229, Caltech, Pasadena, CA, 2007; available online at http://www.theory.caltech.edu/~preskill/ph229.
[35] R. Raz, *Exponential separation of quantum and classical communication complexity*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1999, pp. 358–367.
[36] A. A. Razborov, *Quantum communication complexity of symmetric predicates*, Izv. Ross. Akad. Nauk Ser. Mat. 67 (2003), pp. 145–159 (in Russian). English translation available at http://genesis.mi.ras.ru/∼razborov/qcc_eng.ps.
[37] O. Rudolph, *A separability criterion for density operators*, J. Phys. A., 33 (2000), pp. 3951–3955.
[38] O. Rudolp, *Further results on the cross norm criterion for separability*, Quantum Inform. Process., 4 (2005), pp. 219–239.
[39] M. Steiner, *Towards quantifying non-local information transfer: Finite-bit non-locality*, Phys. Lett. A, 270 (2000), pp. 239–244.
[40] W. Tittel, J. Brendel, H. Zbinden, and N. Gisin, *Violation of Bell inequalities by photons more than* 10 *km apart*, Phys. Rev. Lett., 81 (1998), pp. 3563–3566.
[41] B. Toner and D. Bacon, *Communication cost of simulating Bell correlations*, Phys Rev Lett., 91 (2003), 187904.
[42] A. C.-C. Yao, *Some complexity questions related to distributive computing*, in Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1979, pp. 209–213.
[43] A. C.-C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Press, Piscataway, NJ, 1993, pp. 352–361.
[44] A. C.-C. Yao, *On the power of quantum fingerprinting*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2003, pp. 77–81.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 767–801

I/O-EFFICIENT PLANAR SEPARATORS*

ANIL MAHESHWARI† AND NORBERT ZEH‡

Abstract. We present I/O-efficient algorithms for computing optimal separator partitions of planar graphs. Our main result shows that, given a planar graph G with N vertices and an integer $r > 0$, a vertex separator of size $O(N/\sqrt{r})$ that partitions G into $O(N/r)$ subgraphs of size at most r and boundary size $O(\sqrt{r})$ can be computed in $O(\text{sort}(N))$ I/Os. This bound holds provided that $M \geq 56r \log^2 B$. Together with an I/O-efficient planar embedding algorithm presented in [N. Zeh, *I/O-Efficient Algorithms for Shortest Path Related Problems*, Ph.D. thesis, School of Computer Science, Carleton University, Ottawa, ON, Canada, 2002], this result is the basis for I/O-efficient solutions to many other fundamental problems on planar graphs, including breadth-first search and shortest paths [L. Arge, G. S. Brodal, and L. Toma, *J. Algorithms*, 53 (2004), pp. 186–206; L. Arge, L. Toma, and N. Zeh, *I/O-efficient algorithms for planar digraphs*, in Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, 2003, pp. 85–93], depth-first search [L. Arge et al., *J. Graph Algorithms Appl.*, 7 (2003), pp. 105–129; L. Arge and N. Zeh, *I/O-efficient strong connectivity and depth-first search for directed planar graphs*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 2003, pp. 261–270], strong connectivity [L. Arge and N. Zeh, *I/O-efficient strong connectivity and depth-first search for directed planar graphs*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 2003, pp. 261–270], and topological sorting [L. Arge and L. Toma, *Simplified external memory algorithms for planar DAGs*, in Proceedings of the 9th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 3111, Springer-Verlag, Berlin, New York, 2004, pp. 493–503; L. Arge, L. Toma, and N. Zeh, *I/O-efficient algorithms for planar digraphs*, in Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, 2003, pp. 85–93]. Our second result shows that, given I/O-efficient solutions to these problems, a general separator algorithm for graphs with costs and weights on their vertices [L. Aleksandrov et al., *Partitioning planar graphs with costs and weights*, in Proceedings of the 4th Workshop on Algorithm Engineering and Experiments, Lecture Notes in Comput. Sci. 2409, Springer-Verlag, Berlin, New York, 2002, pp. 98–107] can be made I/O-efficient. Many classical separator theorems are special cases of this result. In particular, our I/O-efficient version allows the computation of a separator as produced by our first separator algorithm, but without placing any constraints on r in relation to the memory size.

Key words. I/O-efficient algorithms, memory hierarchies, graph algorithms, planar graphs, graph separators

AMS subject classifications. 49M27, 68Q25, 90C06, 90C35

DOI. 10.1137/S0097539705446925

1. Introduction.

I/O-efficient graph algorithms have received considerable attention because massive graphs arise naturally in many applications. Recent Web crawls, for example, produced graphs of on the order of 200 million nodes and 2 billion edges. Recent work in Web modeling uses depth-first search (DFS), breadth-first search (BFS), and the computation of shortest paths and connected components as primitive operations for investigating the structure of the Web [12]. Massive graphs

*Received by the editors February 4, 2005; accepted for publication (in revised form) January 11, 2008; published electronically May 28, 2008. An extended abstract of this paper appeared in the *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco), ACM, New York, SIAM, Philadelphia, 2002, pp. 382–389. This research was supported by NSERC and NCE GEOIDE.

http://www.siam.org/journals/sicomp/38-3/44692.html

†School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada (maheshwa@scs.carleton.ca).

‡Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada (nzeh@cs.dal.ca).

are also often manipulated in geographic information systems (GIS), where many fundamental problems can be formulated as basic graph problems. Yet another example of a massive graph is AT&T's phone call graph [13]. When working with such large data sets, the transfer of data between internal and external memory, and not the internal-memory computation, is often the bottleneck. Thus, I/O-efficient algorithms can lead to considerable run time improvements.

Planar graphs are a natural abstraction of many real-world problems. For example, the graphs arising in GIS are often planar or "almost planar." On the theoretical side, planar graphs are among the fundamental combinatorial structures used in algorithmic graph theory. Planar separators have played the key role in designing divide-and-conquer algorithms for planar graphs. The classical separator theorem for planar graphs by Lipton and Tarjan [27], coupled with linear-time planar embedding algorithms [10, 18, 24, 26], has led to phenomenal developments in algorithmic graph theory. Numerous research results that followed describe efficient algorithms for computing a variety of separators of other sparse graphs and discuss applications of separators such as lower bounds on the size of Boolean circuits, approximation algorithms for NP-complete problems, nested dissection of sparse systems of linear equations, load balancing in parallel numerical simulations based on the finite element method, partitioning triangular irregular networks in the field of GIS, and encoding graphs.

In external memory, planar separators have been the key to obtaining I/O-efficient algorithms for a variety of problems on embedded planar graphs, including BFS and single-source shortest paths [5, 8], DFS [6, 9], strong connectivity [9], and topological sorting [8, 7].

Existing internal-memory algorithms for computing planar separators (see, e.g., [2, 3, 27]) are not I/O-efficient because they use BFS to gather structural information about the graph, and BFS and DFS are among those fundamental graph problems for which truly I/O-efficient solutions on general graphs are still lacking. The existing I/O-efficient BFS-algorithms for planar graphs [5, 8] are separator-based; hence, using them in a separator algorithm creates a circular dependency. In this paper, we present a new algorithm that applies BFS only to a compressed version of the given graph and combines this with graph contraction techniques to obtain an optimal separator partition in an I/O-efficient manner. An added benefit of our algorithm is that it does not rely on a planar embedding of the given graph.

1.1. Model of computation and previous work. The algorithms in this paper are designed and analyzed in the I/O-model of Aggarwal and Vitter [1]. This model quite accurately captures the characteristics of current hard drives and yet is simple enough to allow the I/O-complexity of complex algorithms to be analyzed. In the I/O-model, the computer is equipped with two levels of memory. The *internal memory* (or *memory* for short) is of limited size, capable of holding M data items. The disk-based *external memory* is of conceptually unlimited size and is divided into blocks of B consecutive data items. All computation has to be performed on data in internal memory. The transfer of data between internal and external memory happens by means of *I/O-operations* (or *I/Os* for short), each of which transfers one block of data to or from the disk. The complexity of an algorithm is the number of I/O-operations it performs.

For surveys of results obtained in the I/O-model and its extensions, we refer the reader to [28, 32]. The following results are relevant to the work presented in this paper.

It has been shown in [1] that sorting and permuting an array of N data items

take $\mathrm{sort}(N) = \Theta((N/B)\log_{M/B}(N/B))$ and $\mathrm{perm}(N) = \Theta(\min\{N, \mathrm{sort}(N)\})$ I/Os, respectively; scanning an array of size N takes $\Theta(N/B)$ I/Os.

In internal memory, that is, in the RAM model, the problem of computing planar separators is well studied. Lipton and Tarjan [27] were the first to show that every planar graph with N vertices has a $\frac{2}{3}$-separator of size $\mathrm{O}\big(\sqrt{N}\big)$, that is, a vertex set of size $\mathrm{O}\big(\sqrt{N}\big)$ whose removal partitions G into two subgraphs containing at most $2N/3$ vertices each. They also presented a linear-time algorithm to compute such a separator. In [21], it has been shown that every graph of genus g has a $\frac{2}{3}$-separator of size $\mathrm{O}\big(\sqrt{gN}\big)$ and that such a separator can be computed in linear time. In [2], a linear-time algorithm for computing a t-separator of size $\mathrm{O}\big(\sqrt{(g+1/t)N}\big)$ for an embedded graph of genus g was given; for $0 < t < 1$, a *t-separator* is a vertex set whose removal partitions G into subgraphs of size at most tN. Other results deal with computing small simple-cycle separators of planar graphs [29], edge separators of planar graphs [15], separators of planar graphs with negative or multiple vertex weights [17], and separators of low cost [3, 16].

In [25], the first I/O-efficient algorithm for computing planar separators was presented. This algorithm is an external-memory version of Lipton and Tarjan's algorithm and computes a $\frac{2}{3}$-separator of size $\mathrm{O}\big(\sqrt{N}\big)$; its I/O-complexity is $\mathrm{O}(\mathrm{sort}(N))$, provided that a planar embedding and a BFS-tree of the graph are given. In [5], an external version of Goodrich's multiway separator algorithm [22] has been developed. This algorithm computes a t-separator of size $\mathrm{O}\big((N/B)\log_{M/B}(N/B) + \sqrt{N/t}\big)$ and takes $\mathrm{O}(\mathrm{sort}(N))$ I/Os, again assuming that an embedding and a BFS-tree are given.

A number of subsequent papers develop a hierarchy of reductions that lead to $\mathrm{O}(\mathrm{sort}(N))$-I/O algorithms for a variety of fundamental problems on embedded planar graphs if an optimal separator decomposition can be obtained in $\mathrm{O}(\mathrm{sort}(N))$ I/Os: In [5, 8], separator-based ideas first pioneered in [19] were used to obtain I/O-efficient shortest-path algorithms for undirected and directed planar graphs. These algorithms can, of course, also be used to compute BFS-trees of planar graphs. In [6], a DFS-algorithm for undirected planar graphs was presented; this algorithm uses BFS in a planar graph derived from the dual and, hence, also depends on planar separators. The directed DFS-algorithm of [9] needs to compute directed spanning trees of the graph; currently, the only I/O-efficient algorithm for this problem is the shortest-path algorithm of [8]. Other I/O-efficient algorithms for planar graphs that rely on separators are the strong connectivity algorithm of [9] and the algorithms of [7, 8] for topologically sorting planar directed acyclic graphs.

1.2. New results. The two main results of our paper are the following:

(i) Given a planar graph G with N vertices and an integer $r > 0$, it takes $\mathrm{O}(\mathrm{sort}(N))$ I/Os to compute a vertex separator S of size $\mathrm{O}\big(N/\sqrt{r}\big)$ whose removal partitions G into $\mathrm{O}(N/r)$ subgraphs of size at most r and boundary size $\mathrm{O}(\sqrt{r})$. The bound on the I/O-complexity of the algorithm holds as long as the internal memory has size at least $56r\log^2 B$.

(ii) Using a bootstrapping approach based on our second algorithm, discussed below, the memory requirements of the algorithm can be reduced from $M = \Omega\big(B^2\log^2 B\big)$ to $M = \Omega\big(B^2\big)$ for the special case when $r = B^2$. This special case is important because $r = B^2$ is the granularity of the partition required by all separator-based I/O-efficient algorithms for planar graphs that have been developed so far.

Thanks to our algorithm, a wide variety of problems, as discussed in the previous section, can be solved in $\mathrm{O}(\mathrm{sort}(N))$ I/Os if $M = \Omega\big(B^2\big)$. In particular, a shortest-path tree of a planar graph can be obtained in this complexity. Using this fact, we

show the following I/O-efficient versions of results from [3]:

(iii) Let $G = (V, E)$ be a planar graph with N vertices, let $w : V \to \mathbb{R}^+$ and $c : V \to \mathbb{R}^+$ be assignments of nonnegative weights and costs to the vertices of G, and let $0 < t < 1$ be an arbitrary constant. Let $w(G) = \sum_{x \in V} w(x)$ and $C(G) = \sum_{x \in V} (c(x))^2$. If the size of the internal memory is $\Omega(B^2)$, it takes $O(\mathrm{sort}(N))$ I/Os to compute a vertex separator S of cost $c(S) \leq 4\sqrt{2 \cdot C(G)/t}$ such that no connected component of $G - S$ has weight exceeding $tw(G)$.

If all vertices have weight and cost equal to 1 and we choose $t = r/N$, we obtain a separator of size $O(N/\sqrt{r})$ that partitions the graph into pieces of size at most r. This matches the result produced by our first algorithm, but without requiring that $M \geq 56r \log^2 B$. More precisely, it allows the computation of arbitrarily coarse partitions, while our first algorithm is restricted to computing partitions into pieces of size $O(M/\log^2 B)$ if an I/O-complexity of $O(\mathrm{sort}(N))$ is desired.

(iv) Let $G = (V, E)$ be a planar graph with N vertices, let $w : V \to \mathbb{R}^+$ be an assignment of nonnegative weights to the vertices of G, let $0 < t < 1$ be an arbitrary constant, and let $w(G) = \sum_{x \in V} w(x)$. If $w(x) \leq tw(G)$ for every vertex $x \in V$, there exists an edge separator $S \subseteq E$ of size $|S| \leq 4\sqrt{2(\sum_{x \in V} (\deg(x))^2)/t}$ such that no connected component of $G - S$ has weight exceeding $tw(G)$. Such a separator can be computed in $O(\mathrm{sort}(N))$ I/Os, provided that $M = \Omega(B^2)$.

The algorithm in result (i) performs $O(N \log N)$ work in internal memory. Result (iii) and, thus, also results (ii) and (iv) rely on an I/O-efficient shortest-path algorithm for planar graphs. Currently, the best such algorithm performing $O(\mathrm{sort}(N))$ I/Os is the algorithm of [5], which performs $O(N \log N + BN)$ work in internal memory. The algorithms in result (ii)–(iv) inherit this computational bound, but their computational bound would decrease to $O(N \log N + T(N))$ with the development of a planar shortest-path algorithm that performs $O(\mathrm{sort}(N))$ I/Os and $T(N) = o(N \log N + BN)$ computation in internal memory.

The presentation of our results is organized as follows. In section 2, we introduce the necessary terminology and notation and discuss some technical results that will be useful in our algorithms. In section 3, we discuss a graph contraction procedure that is used many times in our algorithms. Our algorithm for partitioning an unweighted planar graph is presented in section 4. The partition produced by the algorithm does not have all the properties required by the shortest-path algorithm of [5] or by any of the other separator-based I/O-efficient algorithms for planar graphs [7, 8, 9]. In section 5, we explain how to ensure the required additional properties. In section 6, we discuss our I/O-efficient algorithm for computing separators of planar graphs with costs and weights. In section 7, we explain how to combine the algorithms from sections 4 and 6 to reduce the memory requirements of our unweighted separator algorithm to $M = \Omega(B^2)$. We present concluding remarks in section 8.

2. Preliminaries.

2.1. Graphs and planarity.

We assume that the reader is familiar with standard graph-theoretic terms and notation as defined, for example, in [23, 31]. In this paper, all graphs are simple, that is, do not contain parallel edges or loops, even though the results are easy to generalize to multigraphs. Let $G = (V, E)$ be a graph. For a set $W \subseteq V$ of vertices, we use $G[W]$ to denote the subgraph of G whose vertex set is W and whose edge set consists of all edges in G that have both endpoints in W; we call $G[W]$ the subgraph of G *induced* by vertex set W. For a set of vertices $W \subseteq V$, let $G - W = G[V \setminus W]$. For a vertex $x \in V$, let $G - x = G - \{x\}$. For a set

of edges $F \subseteq E$, let $G - F = (V, E - F)$.

For a vertex $x \in G$, the *(open) neighborhood* $\mathcal{N}(x)$ of x is the set of vertices adjacent to x, that is, $\mathcal{N}(x) = \{y \in V : xy \in E\}$; the *closed neighborhood* of x is $\mathcal{N}[x] = \{x\} \cup \mathcal{N}(x)$. We generalize this to vertex sets and subgraphs by defining $\mathcal{N}[V'] = \bigcup_{x \in V'} \mathcal{N}[x]$, $\mathcal{N}(V') = \mathcal{N}[V'] \setminus V'$, $\mathcal{N}[G'] = G[\mathcal{N}[V']]$, and $\mathcal{N}(G') = G[\mathcal{N}(V')]$, where $G' = (V', E')$. We often call $\mathcal{N}(G')$ the *boundary* of G'. The *degree* $\deg(x)$ of a vertex x is the number of edges incident to x. Since we assume that G is simple, we have $\deg(x) = |\mathcal{N}(x)|$.

A graph G is *planar* if it can be drawn in the plane so that the edges of G do not intersect, except at their endpoints. Such a drawing of G is called a *topological embedding* of G; we denote it by $\mathcal{E}(G)$. Every topological embedding of G defines an order of the edges incident to each vertex $x \in G$ clockwise around x. A representation of these orders for all vertices $x \in G$ is called a *combinatorial embedding* of G, which we denote by $\hat{G}$. Given a topological embedding $\mathcal{E}(G)$ consistent with a combinatorial embedding $\hat{G}$ of G, we call the connected regions of $\mathbb{R}^2 \setminus \mathcal{E}(G)$ the *faces* of $\hat{G}$. The *size* of a face of $\hat{G}$ is the number of edges on its boundary. Let F denote the set of faces of $\hat{G}$. By Euler's formula, $|V| + |F| - |E| = 2$. In particular, $|E| \leq 3|V| - 6$. We define the *size* $|G|$ of a planar graph $G = (V, E)$ to be the number of vertices in G.

For an embedded planar graph $G = (V, E)$ and an edge $e \in E$, the *dual* e^* of e is the edge $f_1 f_2$, where f_1 and f_2 are the two faces of $\hat{G}$ that have edge e on their boundaries. The dual of G is the multigraph $G^* = (F, E^*)$, where $E^* = \{e^* : e \in E\}$.

A *partition* of a set S is a collection $\mathcal{S} = \{S_1, \ldots, S_k\}$ of subsets of S so that every element of S belongs to exactly one set S_i, that is, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and $\bigcup_{i=1}^{k} S_i = S$.

Given a graph $G = (V, E)$ and a partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of the vertex set of G, the *contraction* of $\mathcal{V}$ in G is the graph $G/\mathcal{V} = (\mathcal{V}, E')$, where $E' = \{V_i V_j : \exists\, xy \in E$ such that $x \in V_i$ and $y \in V_j\}$. If the vertices of G have weights, we define $w(V_i) = \sum_{x \in V_i} w(x)$ for all $1 \leq i \leq k$. If the graph $G[V_i]$ is connected for all $1 \leq i \leq k$, we call $G/\mathcal{V}$ an *edge contraction*. If there exist vertex sets $X_1, \ldots, X_k$ such that, for all $1 \leq i \leq k$ and all $x \in V_i$, $\mathcal{N}(x) \setminus V_i = X_i$, that is, if all vertices in V_i have the same neighbors in $V(G) \setminus V_i$, we call $G/\mathcal{V}$ a *vertex bundling*. We will use the following facts. The first one states that edge contractions and vertex bundlings preserve planarity. Intuitively, the second one says that undoing any contraction preserves separators.

FACT 2.1. *If G is planar, then every edge contraction or vertex bundling of G is planar.*

FACT 2.2. *Let $\mathcal{S} \subseteq \mathcal{V}$; $S = \bigcup_{V_i \in \mathcal{S}} V_i$; $V_i, V_j \notin \mathcal{S}$; and $x \in V_i$ and $y \in V_j$. If V_i and V_j belong to different connected components of $(G/\mathcal{V}) - \mathcal{S}$, then x and y belong to different connected components of $G - S$.*

2.2. Graph separators. Given an assignment $w : V \to \mathbb{R}^+$ of weights to the vertices of a graph $G = (V, E)$ and a parameter $0 < t < 1$, we call a set $S \subseteq V$ of vertices a *t-vertex separator* of G if no connected component of $G - S$ has weight greater than $tw(G)$, where $w(G) = \sum_{x \in V} w(x)$ is the weight of G. Similarly, a *t-edge separator* of G is a set $S \subseteq E$ of edges so that no connected component of the graph $G - S$ has weight exceeding $tw(G)$. Since we are mainly interested in vertex separators in this paper, we refer to them simply as *separators*. If G is unweighted, we give every vertex of G weight one and define separators w.r.t. these weights.

In our separator algorithm for unweighted graphs, we apply the following result to a compressed version of the graph we want to partition. This produces a first separator that is then refined during a sequence of refinement steps.

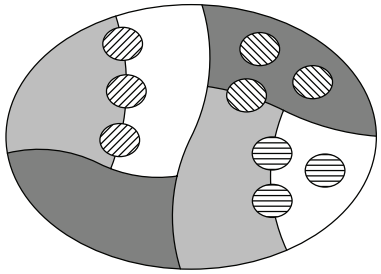


FIG. 2.1. *A regular partition. The solid regions are connected. Each of the hatched regions is disconnected but shares its boundary with at most two solid regions and no hatched region.*

THEOREM 2.1 (Aleksandrov and Djidjev [2]). *Given a planar graph G of size N, an assignment $w : V \to \mathbb{R}^+$ of weights to the vertices of G, and a constant $0 < t < 1$, a t-vertex separator of size at most $4\sqrt{N/t}$ for G can be computed in linear time.*

By grouping the connected components of $G-S$, where G is an unweighted graph and S is a t-separator of G, we obtain a tN-partition of G. More precisely, let $r > 0$ be an integer, let $t = r/N$, and let S be a t-separator of G. Then an *r-partition* of G is a pair $\mathcal{P} = (S, \{G_1, \ldots, G_q\})$ with the following properties:

(i) $G_1, \ldots, G_q$ are vertex-induced subgraphs of G;
(ii) the set $\{V(G_1), \ldots, V(G_q)\}$ is a partition of $V(G) \setminus S$;
(iii) $\mathcal{N}(V(G_i)) \subseteq S$ for all $1 \le i \le q$; and
(iv) $|G_i| \le r$ for all $1 \le i \le q$.

The third condition captures that every subgraph G_i is a collection of connected components of $G - S$; that is, no connected component of $G - S$ has vertices in two such subgraphs.

If G is a planar graph, we call an r-partition $\mathcal{P} = (S, \{G_1, \ldots, G_q\})$ *normal* if $|S| = \mathrm{O}(N/\sqrt{r})$, $q = \mathrm{O}(N/r)$, and $\sum_{i=1}^{q} |\mathcal{N}(G_i)| = \mathrm{O}(N/\sqrt{r})$. A normal r-partition $\mathcal{P} = (S, \{G_1, \ldots, G_q\})$ is *c-proper* for some constant $c > 0$ if $|\mathcal{N}(G_i)| \le c\sqrt{r}$ for all $1 \le i \le q$. If we do not want to specify c, we say that $\mathcal{P}$ is *proper*. A proper r-partition is stronger than a normal r-partition because the latter may contain subgraphs with a large boundary, as long as the total size of all subgraph boundaries is small; in the former, every individual subgraph has to have a small boundary. Finally, we call an r-partition $\mathcal{P} = (S, \{G_1, \ldots, G_q\})$ *regular* if, for every $1 \le i \le q$, one of the following conditions holds:

(i) The graph $\mathcal{N}[G_i]$ is connected, or
(ii) there are at most two indices $1 \le j < k \le q$, $i \notin \{j, k\}$, such that $\mathcal{N}(V(G_i)) \cap \mathcal{N}(V(G_j)) \neq \emptyset$ and $\mathcal{N}(V(G_i)) \cap \mathcal{N}(V(G_k)) \neq \emptyset$. $\mathcal{N}[G_j]$ and $\mathcal{N}[G_k]$ are connected in this case.

This concept is visualized in Figure 2.1, which is reproduced from Frederickson [19], who shows that every planar graph has a proper r-partition and that every planar graph of bounded degree has a regular proper r-partition. The following result states that a proper r-partition of a planar graph can be obtained efficiently.

THEOREM 2.2 (Frederickson [19]). *Given a planar graph G of size N and a normal r-partition $\mathcal{P} = (S, \{G_1, \ldots, G_p\})$ of G, a proper r-partition $\mathcal{P}' = (S', \{G'_1, \ldots, G'_q\})$ of G such that $S \subseteq S'$ can be computed in $\mathrm{O}(N \log N)$ time.*

Frederickson also shows how to obtain a normal r-partition of a planar graph in $\mathrm{O}(N \log N)$ time. Together with Theorem 2.2, this implies that a proper r-partition of a planar graph can be computed in $\mathrm{O}(N \log N)$ time. To prove Theorem 2.2, Fred-

erickson considers each graph G_i in turn and partitions it into subgraphs of boundary size $O(\sqrt{r})$. When partitioning G_i, the algorithm requires knowledge of only $\mathcal{N}[G_i]$. Thus, if $|\mathcal{N}[G_i]| \leq M$ for all $1 \leq i \leq p$, we can implement Frederickson's procedure by loading each graph $\mathcal{N}[G_i]$ into internal memory and partitioning G_i without incurring any further I/Os. Assuming that each graph $\mathcal{N}[G_i]$ is stored in consecutive memory locations, we thus have the following.

COROLLARY 2.3. *Given a planar graph G of size N and a normal r-partition $\mathcal{P} = (S, \{G_1, \ldots, G_p\})$ of G, a proper r-partition $\mathcal{P}' = (S', \{G'_1, \ldots, G'_q\})$ of G such that $S \subseteq S'$ can be computed in $O(N/B)$ I/Os, provided that $|\mathcal{N}[G_i]| \leq M$ for all $1 \leq i \leq p$.*

2.3. Bipartite planar graphs. A graph $G = (V, E)$ is *bipartite* if the vertex set V can be partitioned into two sets U and W such that $x \in U$ and $y \in W$ for every edge $xy \in E$. In this case, we write $G = (U, W, E)$. We use the following two simple results to bound the sizes of certain bipartite planar graphs in different parts of our algorithms.

LEMMA 2.4. *Let $G = (U, W, E)$ be a simple connected bipartite planar graph such that every vertex in W has degree at least three. Then $|W| < 2|U|$.*

Proof. Consider a planar embedding $\hat{G}$ of G. Since G is bipartite, every face of $\hat{G}$ has size at least four and, thus, $|F| \leq |E|/2$. By Euler's formula, $|V| + |F| - |E| = 2$, that is,

$$\begin{aligned} 2 &= |V| + |F| - |E| \\ &\leq |V| - |E|/2, \\ |E| &< 2|V|. \end{aligned}$$

On the other hand, $|E| \geq 3|W|$, so that

$$\begin{aligned} 3|W| &< 2|V| \\ &= 2(|U| + |W|), \\ |W| &< 2|U|. \end{aligned}$$ □

COROLLARY 2.5. *Let $G = (U, W, E)$ be a simple connected bipartite planar graph such that no two vertices $x, y \in W$ of degree at most two have the same open neighborhood. Then $|G| < 7|U|$.*

Proof. We have to prove that $|W| < 6|U|$. To this end, we divide W into three subsets and bound the size of each of these sets. Let W_1 be the set of degree-1 vertices in W, W_2 the set of degree-2 vertices in W, and W_3 the set of vertices of degree at least three in W, that is, $W_3 = W \setminus (W_1 \cup W_2)$.

Since there are no two vertices $x, y \in W$ with $\deg(x) = \deg(y) = 1$ and $\mathcal{N}(x) = \mathcal{N}(y)$, W_1 contains at most $|U|$ vertices, that is, one per vertex in U; see Figure 2.2(a).

To count the vertices in W_3, we consider the bipartite planar graph $G_3 = (U_3, W_3, E_3)$ induced by all edges incident to vertices in W_3; see Figure 2.2(b). By Lemma 2.4 and since $U_3 \subseteq U$, we have $|W_3| < 2|U_3| \leq 2|U|$.

To count the vertices in W_2, consider the graph $H_2 = (U_2, E_2)$, where $U_2 = \mathcal{N}(W_2) \subseteq U$ and $E_2 = \{xz :$ there exists a vertex $y \in W_2$ with $\mathcal{N}(y) = \{x, z\}\}$; see Figure 2.2(d). Since there are no two vertices $x, y \in W_2$ with $\mathcal{N}(x) = \mathcal{N}(y)$, H_2 contains exactly one edge per vertex in W_2, that is, $|E_2| = |W_2|$. Next we argue that H_2 is planar, which, by Euler's formula, implies that $|W_2| = |E_2| < 3|U_2| \leq 3|U|$.

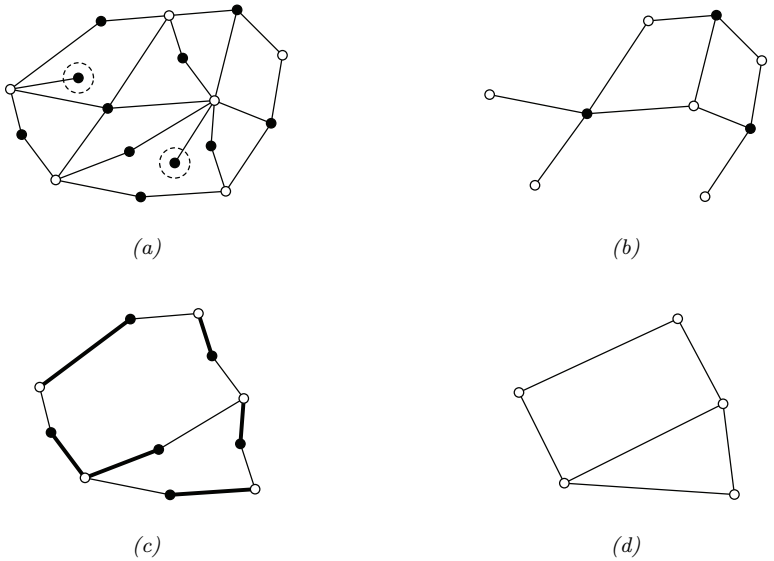


FIG. 2.2. *Illustration of the proof of Corollary* 2.5. (a) *A bipartite planar graph* $G = (U, W, E)$*; the white vertices are in* U*, and the black ones are in* W*. The circled vertices are in* W_1*.* (b) *The subgraph* G_3 *of* G *induced by all edges incident to vertices in* W_3*.* (c) *The subgraph* G_2 *of* G *induced by all edges incident to vertices in* W_2*.* (d) *The graph* H_2 *obtained from* G_2 *by contracting the bold edges in* (c).

To see that H_2 is planar, consider the subgraph G_2 of G induced by all edges incident to vertices in W_2; see Figure 2.2(c). Since G is planar, G_2 is planar. H_2 can be obtained from G_2 by contracting one of the edges incident to each vertex $y \in W_2$. By Fact 2.1, this implies that H_2 is planar.

In summary, we have $|W| = |W_1| + |W_2| + |W_3| < |U| + 3|U| + 2|U| = 6|U|$, that is, $|G| = |U| + |W| < 7|U|$. □

2.4. Primitive operations. Our algorithms make frequent use of a number of primitive operations. In order to avoid repeatedly discussing their implementations, we discuss them here and then refer to this section whenever we make use of such an operation.

Set operations. All elementary operations on two sets A and B—union, intersection, difference, etc.—can be carried out in $O(\mathrm{sort}(N))$ I/Os if the two sets A and B are represented as unordered sequences of elements sort A and B (assuming that every element is represented by a unique integer ID). Then scan the two sorted lists to produce $C = A \odot B$, where $\odot \in \{\cap, \cup, \setminus\}$.

Even though it is not a set operation as such, we want to mention duplicate removal here. Some operations may produce multisets. In order to obtain a proper representation of the set of elements in such a multiset, we need to remove duplicates. This can be done in $O(\mathrm{sort}(N))$ I/Os by sorting and scanning the multiset.

Copying labels. Since pointers are mostly useless in I/O-efficient graph algorithms, graphs are often represented as an (unsorted) list of vertices, each with a unique vertex ID, and as an (unsorted) list of edges, each labeled with the IDs of its two endpoints. Edges do not store any pointers to their endpoints. Thus, if certain labels are assigned to the vertices of the graph, the edges have no knowledge of the labels of their endpoints. However, it takes $O(\mathrm{sort}(N))$ I/Os to label all edges with the labels of

their endpoints. Sort the vertices by their IDs. Designate one endpoint of every edge as being the first endpoint. Then sort the edges by the IDs of their first endpoints. Now scan the sorted vertex and edge sets to label every edge with the label of its first endpoint. To label the edges with the labels of their second endpoints, sort the edges by the IDs of their second endpoints, and repeat the copying process.

Graph contraction. Given a labeling of the vertices of graph G that represents a partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of the vertex set of G, that is, assigns the same label to two vertices if and only if they belong to the same set $V_i \in \mathcal{V}$, the graph $G/\mathcal{V}$ can be computed in $O(\text{sort}(N))$ I/Os as follows: First, label the edges of G with the labels of their endpoints. Now replace every vertex with its label and every edge xy with the edge ab, where a is the label of x and b is the label of y. Finally, remove duplicates from the resulting vertex and edge sets.

In this paper, we choose the label of a set $V_i \in \mathcal{V}$ to be the ID of a *representative* $x \in V_i$. We will then often refer to the vertex V_i in $G/\mathcal{V}$ as the vertex x, taking the point of view that x has survived the contraction and that all other vertices in V_i have been *contracted into* x.

Connected components. Chiang et al. [14] proved that it takes $O(\text{sort}(N))$ I/Os to compute the connected components of an N-vertex planar graph G, that is, to compute a labeling of the vertices of G such that two vertices have the same label if and only if they are connected by a path in G.

3. Uniform graph contraction. In this section, we discuss a general contraction procedure for planar graphs that is used repeatedly in our algorithms. In general, when using graph contraction, one repeatedly contracts edges until some goal is reached (usually a sufficient reduction of the size of the graph). Since it would not be I/O-efficient to contract edges one at a time, I/O-efficient algorithms based on graph contraction usually contract many edges simultaneously. More precisely, these algorithms compute an edge contraction $G/\mathcal{V}$ of G such that $|\mathcal{V}| \leq c|V|$, for some $c < 1$; that is, $G/\mathcal{V}$ has only a constant fraction of the vertices of G. The algorithms of [5, 14, 30] for computing connected components and minimum spanning trees are based on exactly this idea. However, the partition $\mathcal{V}$ used in these algorithms is nonuniform in the sense that some sets $V_i \in \mathcal{V}$ may be large, while others may be small; that is, some vertices in $G/\mathcal{V}$ represent many vertices in G, and others represent only few. As we will see, our separator algorithm for unweighted graphs relies heavily on the compression being uniform, that is, on every set $V_i \in \mathcal{V}$ having constant size. Our goal in this section is to compute a partition $\mathcal{V}$ such that $|\mathcal{V}| \leq c|V|$ and that every set in $\mathcal{V}$ has constant size. In order to achieve this, we compute $G/\mathcal{V}$ in two phases: First, we compute an edge contraction $G_1 = G/\mathcal{V}_1$ and then a vertex bundling $G_2 = G_1/\mathcal{V}_2$.

In section 3.1, we give a high-level description of our uniform contraction procedure and prove a general bound on the size of the compressed graph it produces. In section 3.2, we show how to implement this procedure in $O(\text{sort}(N))$ I/Os on an N-vertex planar graph.

3.1. The high-level procedure. Our separator algorithm requires not only that $\mathcal{V}$ be a partition of V into subsets of constant size, but also that each subset be of bounded weight according to appropriately chosen weights assigned to the vertices in V. In this section, we assume more generally that the input to our contraction procedure consists of a planar graph G, a constant number of real-valued functions $w_1, \ldots, w_k$ assigning weights to the vertices of G, and a set of weight thresholds $u_1, \ldots, u_k$. Initially, every vertex x satisfies $w_j(x) \leq u_j$ for all $1 \leq j \leq k$; we say that vertex x is *within bounds*. Our goal is to compute a contraction $G/\mathcal{V}$ of G such

that each set V_i in $\mathcal{V}$ is within bounds, that is, $w_j(V_i) = \sum_{x \in V_i} w_j(x) \leq u_j$ for all $1 \leq j \leq k$. We say that a vertex or set x that is within bounds is *light* if $w_j(x) \leq u_j/2$ for all $1 \leq j \leq k$; otherwise, we call it *heavy*. The main result of this section is stated in the following theorem.

THEOREM 3.1. *Let G be a planar graph, $w_1, \ldots, w_k$ real-valued functions assigning weights to the vertices of G, and $u_1, \ldots, u_k$ weight thresholds such that every vertex $x \in G$ satisfies $w_j(x) \leq u_j$ for all $1 \leq j \leq k$. Then it takes $\mathrm{O}(\mathrm{sort}(N))$ I/Os to compute a planar graph G_2 such that G_2 is a contraction of G, all vertices of G_2 are within bounds, and more than $|G_2|/7$ vertices in G_2 are heavy.*

The contraction procedure consists of two phases: The *edge contraction phase* computes an edge contraction $G_1 = G/\mathcal{V}_1$ of G such that all vertices of G_1 are within bounds and every edge of G_1 has at least one heavy endpoint. The *bundling phase* computes a vertex bundling $G_2 = G_1/\mathcal{V}_2$ of G_1 such that all vertices of G_2 are within bounds and there are no two light vertices $x, y \in G_2$ of degree at most two that have the same open neighborhood. Graph G_2 is the final graph we return.

In section 3.2, we prove that the two phases of this procedure can be implemented in $\mathrm{O}(\mathrm{sort}(N))$ I/Os. Here we prove that G_2 has the properties claimed in Theorem 3.1.

By the description of the contraction procedure, the vertices of G_2 are explicitly ensured to be within bounds. Graph G_1 is an edge contraction of G, and G_2 is a vertex bundling of G_1. Hence, G_2 is a contraction of G and, by Fact 2.1, planar. We have to prove that at least every seventh vertex in G_2 is heavy.

LEMMA 3.2. *If h is the number of heavy vertices in G_2, then $|G_2| < 7h$.*

Proof. Consider the subgraph G' of G_2 induced by all edges incident to light vertices in G_2. Since G' contains all light vertices of G_2, it suffices to prove that $|G'| < 7h'$, where $h' \leq h$ is the number of heavy vertices in G'. We use Corollary 2.5 to do so.

First, observe that G' is bipartite: By the definition of G', every edge in G' has at least one light endpoint. If there was an edge with two light endpoints in $G' \subseteq G_2$, then G_1 would have to contain an edge with two light endpoints because G_2 is a vertex bundling of G_1, but we ensure explicitly that G_1 contains no such edge.

Observe also that no two light vertices of degree at most two in G_2 have the same neighbors. Hence, the same is true in G', and G' satisfies the conditions of Corollary 2.5 with U being the set of heavy vertices and W being the set of light vertices in G'. Thus, by Corollary 2.5, $|G'| < 7h'$. ∎

3.2. I/O-efficient implementation. In this section, we discuss how to implement the two phases of the contraction procedure in $\mathrm{O}(\mathrm{sort}(N))$ I/Os.

3.2.1. Edge contraction phase. To implement the edge contraction phase, we compute a set F of edges and define $\mathcal{V}_1$ to be the partition of V corresponding to the connected components of (V, F). We use $\mathcal{V}_F$ to denote this partition. We choose the edges in F so that the graph (V, F) is a forest, which makes the computation of connected components and, thus, the computation of the partition $\mathcal{V}_F$, easy [14].

The set F is not necessarily a subset of E. However, whenever we add an edge xy to F, there exists an edge $x'y' \in E$ such that x and x' belong to the same set V_1, and y and y' belong to the same set V_2 in $\mathcal{V}_F$. Hence, if $G[V_1]$ and $G[V_2]$ are connected, so is $G[V_1 \cup V_2]$, and the addition of edge xy to F maintains that $G/\mathcal{V}_F$ is an edge contraction of G.

We compute the edge set F iteratively, starting with $F = \emptyset$. Each iteration computes a set F' of edges such that $G/\mathcal{V}_{F \cup F'}$ is an edge contraction of G and each of its vertices is within bounds. The edges in F' are then added to F. This iterative

process stops as soon as every edge in $G/\mathcal{V}_F$ has at least one heavy endpoint. At this point, we define $G_1 = G/\mathcal{V}_F$.

To compute the set F' efficiently, each iteration consists of three steps: The first step extracts the subgraph H_1 of $G/\mathcal{V}_F$ induced by all edges in $G/\mathcal{V}_F$ whose endpoints are both light. We call these edges *contractible* and call H_1 the *contractible subgraph* of $G/\mathcal{V}_F$. The second step computes a maximal matching F_1' of H_1 and contracts the edges in F_1', producing the graph $H_2 = H_1/\mathcal{V}_{F_1'}$. We call a vertex in H_2 *matched* if it represents the two endpoints of an edge in F_1'; otherwise, the vertex represents only one vertex in H_1, and we call it *unmatched*. Note that all neighbors of an unmatched vertex are matched because F_1' is maximal. The third step adds edges between matched and unmatched vertices to a set F_2' so that every nonsingleton set in $\mathcal{V}_{F_2'}$ contains exactly one matched vertex. To determine which edges to add to F_2', we inspect each unmatched vertex x in turn. If there is a matched neighbor y of x that is contained in a light set V_y in $\mathcal{V}_{F_2'}$, we add the edge xy to F_2', thereby adding x to V_y. After this third step, we define $F' = F_1' \cup F_2'$ and add the edges in F' to F. This finishes the iteration.

Note that testing the loop condition—whether or not $G/\mathcal{V}_F$ contains two adjacent light vertices—is easy because, by the definition of H_1, this is the case if and only if H_1 is nonempty. It is also unnecessary to compute $G/\mathcal{V}_F$ from scratch after each iteration: Each iteration is interested only in the contractible subgraph of $G/\mathcal{V}_F$. If H_1 and H_1' are the contractible subgraphs of $G/\mathcal{V}_F$ and $G/\mathcal{V}_{F\cup F'}$, respectively, it is easy to see that $H_1' \subseteq H_1/\mathcal{V}_{F'} = H_2/\mathcal{V}_{F_2'}$. Thus, each iteration has to compute only $H_2 = H_1/\mathcal{V}_{F_1'}$ and $H = H_2/\mathcal{V}_{F_2'}$, and the next iteration can extract its contractible subgraph from H. This leads to the contraction procedure shown in Algorithm 1.

Before providing the implementation details and analyzing the I/O-complexity of this procedure, we show that it correctly implements the edge contraction phase of our uniform graph contraction procedure.

LEMMA 3.3. *Let G_1 be the graph produced by procedure* CONTRACTEDGES. *Then G_1 is an edge contraction of G, every vertex of G_1 is within bounds, and every edge of G_1 has at least one heavy endpoint.*

Proof. We prove by induction on the number of iterations that the graph $G/\mathcal{V}_F$ is an edge contraction of G and that all its vertices are within bounds. Since we exit with $G_1 = G/\mathcal{V}_F$ only when every edge in $G/\mathcal{V}_F$ has a heavy endpoint, this proves the lemma.

Before the first iteration, $G/\mathcal{V}_F = G$ is a trivial edge contraction of G because $F = \emptyset$. Moreover, all vertices of G are assumed to be within bounds.

So assume that, before the current iteration, $G/\mathcal{V}_F$ is an edge contraction of G and all its vertices are within bounds. We want to prove that the same is true for $G/\mathcal{V}_{F\cup F'}$.

First, we prove that $G/\mathcal{V}_{F\cup F'}$ is an edge contraction of G. Since $G/\mathcal{V}_F$ is an edge contraction of G, every set in $\mathcal{V}_F$ induces a connected subgraph of G. Every edge xy added to F_1' is an edge of $H_1 \subseteq G/\mathcal{V}_F$. Hence, there exist two sets $V_1, V_2 \in \mathcal{V}_F$ and an edge $x'y' \in E(G)$ such that $x, x' \in V_1$ and $y, y' \in V_2$. By adding edge xy to F_1', the sets V_1 and V_2 are merged into the set $V_1 \cup V_2$ in $\mathcal{V}_{F\cup F_1'}$. Since $G[V_1]$ and $G[V_2]$ are connected, the existence of edge $x'y' \in E(G)$ implies that $G[V_1 \cup V_2]$ is connected, and $G/\mathcal{V}_{F\cup F_1'}$ remains an edge contraction of G after adding edge xy to F_1'.

As for Step 3, every edge added to F_2' is an edge of $H_2 = H_1/\mathcal{V}_{F_1'} \subseteq G/\mathcal{V}_{F\cup F_1'}$. Since we have just argued that $G/\mathcal{V}_{F\cup F_1'}$ is an edge contraction of G, the same argument as in the previous paragraph implies that $G/\mathcal{V}_{F\cup F_1'\cup F_2'}$ is an edge contraction of G.

Algorithm 1 The edge contraction phase.

Procedure ContractEdges

Input: A weighted graph $G = (V, E)$ and a set of weight thresholds based on which every vertex in G is classified as light or heavy.

Output: An edge contraction $G_1 = G/\mathcal{V}_F$ of G.

1: $H \leftarrow G$
2: $F \leftarrow \emptyset$
3: **while** H is not empty **do**
4: Step 1: Extract the contractible subgraph
$H_1 \leftarrow$ the contractible subgraph of H
5: Step 2: Contract a maximal matching
Compute a maximal matching F_1' of H_1
$H_2 \leftarrow H_1/\mathcal{V}_{F_1'}$
6: Step 3: Contract edges incident to unmatched vertices
$F_2' \leftarrow \emptyset$
Let $\mathcal{V}_{F_2'}$ be the partition of $V(H_2)$ defined by F_2'.
for every unmatched vertex $x \in H_2$ **do**
if x has a (matched) neighbor y contained in a light set $V_y \in \mathcal{V}_{F_2'}$ **then**
Add edge xy to F_2' and increase the weights of V_y by the corresponding weights of x.
end if
end for
$F \leftarrow F \cup F_1' \cup F_2'$
$H \leftarrow H_2/\mathcal{V}_{F_2'}$
7: **end while**
8: $G_1 \leftarrow G/\mathcal{V}_F$

To see that all vertices of $G/\mathcal{V}_{F\cup F'}$ are within bounds, we first observe that all vertices of $G/\mathcal{V}_{F\cup F_1'}$ are within bounds. Indeed, each vertex $x \in G/\mathcal{V}_{F\cup F_1'}$ represents one or two vertices in $G/\mathcal{V}_F$. In the former case, x is obviously within bounds. In the latter case, the two vertices represented by x both belong to H_1 and are thus light; hence, x is within bounds in this case as well.

In the third step, when adding an edge xy to F_2', x is an unmatched vertex of H_2, that is, represents a single vertex in H_1 and is thus light. The set V_y containing y is light because otherwise we would not add edge xy to F_2'. Thus, after adding xy to F_2', and thereby x to V_y, the vertex in $G/\mathcal{V}_{F\cup F_1'\cup F_2'}$ representing V_y remains within bounds. Arguing inductively over all edges added to F_2' in Step 3, we obtain that all vertices in $G/\mathcal{V}_{F\cup F_1'\cup F_2'}$ are within bounds. This finishes the proof of the lemma. □

Procedure ContractEdges is fairly easy to implement in O(sort(N)) I/Os. To prove this, we show how to implement every iteration of the while-loop in lines 3–7 in O(sort($|H|$)) I/Os, show that $|H|$ decreases by a factor of at least two from one iteration to the next, and that the rest of the algorithm takes O(sort(N)) I/Os. We start by analyzing the I/O-complexity of one iteration of the while-loop.

Extracting the contractible subgraph. The contractible subgraph of H can be extracted in O(sort($|H|$)) I/Os: First, we label all edges with the weights of their endpoints, and then we scan the edge set to discard all edges that have at least one heavy

endpoint. The result is the edge list of the contractible subgraph. Now we scan this edge list and create a list of the edges' endpoints. To produce the vertex list of H_1, we remove duplicates from the resulting list. As discussed in section 2.4, all steps of this construction take $O(\text{sort}(|H|))$ I/Os.

Computing and contracting a matching. Zeh [33] presents an algorithm that computes a maximal matching of a graph $G = (V, E)$ in $O(\text{sort}(N))$ I/Os, where $N = |V| + |E|$. Since H_1 is planar, we have $N = O(|H_1|)$ when applying this procedure to H_1. Hence, the matching can be computed in $O(\text{sort}(|H_1|))$ I/Os. Once the matching is given, its edges can be contracted in $O(\text{sort}(|H_1|))$ I/Os by using the contraction procedure from section 2.4.

Contracting edges between matched and unmatched vertices. To contract edges between matched and unmatched vertices, that is, to implement Step 3 of the iteration, we start by extracting all edges in H_2 that are incident to unmatched vertices. This is easily done in $O(\text{sort}(|H_2|)) = O(\text{sort}(|H_1|))$ I/Os by labeling all edges with the statuses of their endpoints and discarding all edges that have two matched endpoints. (Recall that every edge in H_2 has at least one matched endpoint.)

For the resulting bipartite graph H', we compute the following information: Let V_m be the set of matched vertices, and let V_u be the set of unmatched vertices in H'. We arbitrarily number the vertices in V_m as $y_1, \ldots, y_r$ and the vertices in V_u as $x_1, \ldots, x_s$. We sort the edges in H' by their unmatched endpoints as primary keys and by their matched endpoints as secondary keys. For every edge $x_i y_j$, we store the unmatched endpoint $x_{i'}$ of the next edge incident to y_j; that is, if y_j is adjacent to vertices $x_{i_1}, \ldots, x_{i_t}$ with $i_1 < \cdots < i_t$, then, for $1 \le h < t$, edge $x_{i_h} y_j$ stores the ID of vertex $x_{i_{h+1}}$. Edge $x_{i_t} y_j$ stores **nil** to indicate that it is the last edge incident to y_j.

This information can easily be computed in $O(\text{sort}(|H'|))$ I/Os: We sort the edges by their matched endpoints as primary keys and by their unmatched endpoints as secondary keys. Then we scan the sorted edge list to compute, for every edge $x_i y_j$, the endpoint of the next edge incident to y_j. (If the next edge in the sorted sequence is $x_{i'} y_j$, then this endpoint is $x_{i'}$. If the next edge is $x_{i'} y_{j'}$ with $j' \neq j$, then edge $x_i y_j$ stores **nil**.) Now we sort the edges by their unmatched endpoints as primary keys and by their matched endpoints as secondary keys.

Given that graph H' has been prepared in this manner, we find the edges in F_2' as follows: We scan the sorted edge list, which is equivalent to inspecting the unmatched vertices in sorted order and scanning their adjacency lists. For every vertex x_i, as soon as we find an edge $x_i y_j$ such that the set V_{y_j} is light, we add edge $x_i y_j$ to F_2' and increase each weight $w_h(V_{y_j})$ by the corresponding weight $w_h(x_i)$ of x_i. The remaining edges in x_i's adjacency list are then ignored. If all neighbors of x_i are contained in heavy sets, no edge incident to x_i is added to F_2'.

In order to implement this procedure correctly, we need a mechanism to inform every edge incident to a vertex y_j about the current weights of the set V_{y_j} at the time when this edge is inspected. We use a priority queue Q to do this. Initially, we have $w_h(V_{y_j}) = w_h(y_j)$ for all h, because $F_2' = \emptyset$ and, hence, $V_{y_j} = \{y_j\}$. For every matched vertex y_j, we insert the initial weights of V_{y_j} into Q, with priority equal to the ID of the first edge incident to y_j. For every edge $x_i y_j$ inspected during the scan, we perform a Delete-Min operation on Q to retrieve the current weights of V_{y_j}. If edge $x_i y_j$ is added to F_2', the weights of V_{y_j} are updated; otherwise, they remain unchanged. Then, if edge $x_i y_j$ stores a vertex $x_{i'} \neq$ **nil** as the next unmatched vertex incident to y_j, we insert the current weights of V_{y_j} into Q, but now with priority $x_{i'} y_j$.

The whole procedure requires one scan of the sorted edge list of H', and two

priority queue operations per edge, once the edges of H' are stored in the right order and store the appropriate successor pointers as defined above. If we use a buffer tree [4] to implement the priority queue, every priority queue operation takes $O((1/B)\log_{M/B}(r/B)) = O((1/B)\log_{M/B}(|H'|/B))$ I/Os amortized. Thus, the construction of F_2' takes $O(\mathrm{sort}(|H'|)) = O(\mathrm{sort}(|H_1|))$ I/Os.

Total cost of the loop. The previous three paragraphs establish that each iteration of the while-loop takes $O(\mathrm{sort}(|H|))$ I/Os. Also observe that $|H| = N$ before the first iteration. To prove that the cost of the whole loop is $O(\mathrm{sort}(N))$ I/Os, it is therefore sufficient to show that $|H|$ decreases by a factor of at least two from one iteration to the next.

LEMMA 3.4. *The loop in lines* 3–7 *of procedure* CONTRACTEDGES *takes* $O(\mathrm{sort}(N))$ *I/Os.*

Proof. Assume that there are k iterations. For $1 \le i \le k$, let $H^{(i)}$ and $H_1^{(i)}$ be snapshots of H and H_1 at the beginning of the ith iteration and after Step 1 of the ith iteration, respectively. Then the total cost of Step 1 over all iterations is $O\bigl(\sum_{i=1}^{k}\mathrm{sort}\bigl(\bigl|H^{(i)}\bigr|\bigr)\bigr)$, and the cost of Steps 2 and 3 over all iterations is $O\bigl(\sum_{i=1}^{k}\mathrm{sort}\bigl(\bigl|H_1^{(i)}\bigr|\bigr)\bigr)$. It is easy to see that $\bigl|H^{(i+1)}\bigr| \le \bigl|H_1^{(i)}\bigr|$ for all $1 \le i < k$. Hence, the total cost of all iterations is $O\bigl(\mathrm{sort}\bigl(\bigl|H^{(1)}\bigr|\bigr) + \sum_{i=1}^{k}\mathrm{sort}\bigl(\bigl|H_1^{(i)}\bigr|\bigr)\bigr)$, which is $O(\mathrm{sort}(N) + \sum_{i=1}^{k}\mathrm{sort}(|H_1^{(i)}|))$ because $H^{(1)} = G$. Next we prove that, for $1 \le i < k$, $\bigl|H_1^{(i+1)}\bigr| \le \bigl|H_1^{(i)}\bigr|/2$. Hence, $\sum_{i=1}^{k}\bigl|H_1^{(i)}\bigr| \le 2\bigl|H_1^{(1)}\bigr| \le 2N$, and the total cost of the loop is $O(\mathrm{sort}(N))$ I/Os.

Consider the ith iteration. After Step 2, there are at most $|H_1^{(i)}|/2$ matched vertices in H_2 because each represents two vertices in $H_1^{(i)}$. To bound the number of vertices in $H_1^{(i+1)}$ by $|H_1^{(i)}|/2$, we argue that every vertex in $H_1^{(i+1)}$ represents a set V' in $\mathcal{V}_{F_2'}$ that contains a matched vertex. In particular, we argue that every vertex in $H^{(i+1)} = H_2/\mathcal{V}_{F_2'}$ that represents a singleton set in $\mathcal{V}_{F_2'}$ containing only an unmatched vertex has only heavy neighbors and, thus, does not belong to the contractible subgraph of $H^{(i+1)}$. To see that this is true, observe that we inspect every unmatched vertex x in H_2 to check whether it has a matched neighbor y whose containing set $V_y \in \mathcal{V}_{F_2'}$ is light; if so, we add the edge xy to F_2', thereby adding x to V_y. Thus, if x remains in a singleton set, all its neighbors in $H^{(i+1)}$ are heavy. This finishes the proof. □

I/O-complexity of the edge contraction phase. To complete the analysis, we have to consider the costs of lines 1, 2, and 8 of the algorithm. Lines 1 and 2 are easy to implement in $O(N/B)$ I/Os. Line 8 requires computing the connected components of the graph $G_F = (V, F)$. This can be done in $O(\mathrm{sort}(N))$ I/Os (see section 2.4). The connected components algorithm identifies components by labeling every vertex in a connected component of G_F with a representative. As argued in section 2.4, the graph $G/\mathcal{V}_F$ can then be computed in $O(\mathrm{sort}(N))$ I/Os. Together with Lemmas 3.3 and 3.4, this proves the following.

LEMMA 3.5. *The edge contraction phase of the uniform graph contraction procedure takes* $O(\mathrm{sort}(N))$ *I/Os.*

3.2.2. Bundling phase. The bundling phase assumes that no two light vertices in the input graph G_1 are adjacent. By Lemma 3.3, this is true for the graph produced by the edge contraction phase. We first extract all light vertices of degree at most two and represent each such vertex x as a triple $(x, y_1, \mathbf{nil})$ or (x, y_1, y_2), depending on whether it has one or two (heavy) neighbors. We sort these triples by their last two

components, thereby storing all vertices with the same neighbors consecutively. Then we scan this sorted list; that is, we scan each group C of vertices with the same neighbors. For each such group, we form subgroups, starting with the first vertex in C. For every subgroup, we record its total weights $W_1, \ldots, W_k$, which are the sums of the corresponding weights of the vertices in the group. For every inspected vertex x, we add it to the current group if $W_h \leq u_h/2$ for all $1 \leq h \leq k$. Otherwise, vertex x starts a new group. Groups are represented by labeling every vertex in a group with the ID of the first vertex in the group. The final grouping represents the partition $\mathcal{V}_2$ of $V(G_1)$, and we compute the graph $G_2 = G_1/\mathcal{V}_2$ using the graph contraction procedure from section 2.4.

LEMMA 3.6. *Given an N-vertex planar graph G_1 that has no two adjacent light vertices and all of whose vertices are within bounds, the bundling phase takes* $\mathrm{O}(\mathrm{sort}(N))$ *I/Os and produces a vertex bundling $G_2 = G_1/\mathcal{V}_2$ of G_1 that contains no two light vertices of degree at most two that have the same open neighborhood. All vertices in G_2 are within bounds.*

Proof. The I/O-bound of the procedure is obvious because we sort and scan the vertex and edge sets of G_1 a constant number of times and then apply the contraction procedure from section 2.4. It is also obvious that $G_1/\mathcal{V}_2$ is a vertex bundling because we add two vertices to the same set in $\mathcal{V}_2$ only if they have the same neighbors.

Next assume that G_2 contains two light vertices x and y of degree at most two and such that $\mathcal{N}(x) = \mathcal{N}(y)$. Then x and y are the representatives of two sets V_1 and V_2 in $\mathcal{V}_2$ that are subsets of the set C of all vertices with neighborhood $\mathcal{N}(x)$, and both sets are light. Assume w.l.o.g. that V_1 is formed before V_2, and let z be the first vertex in C that is not in V_1. Since V_1 is light, z would have been added to V_1, a contradiction.

Now assume that G_2 contains a vertex x that is out of bounds. Since all vertices of G_1 are within bounds, x must represent some set V_1 formed by collecting vertices that belong to a set C of light vertices with the same neighbors. Let y be the last vertex in C that is added to V_1. Since we add y to V_1, V_1 is light before the addition of y. Since y is light, this implies that V_1 remains within bounds after adding y to it, a contradiction. ∎

Lemmas 3.5 and 3.6 together establish the I/O-complexity of the uniform graph contraction procedure claimed in Theorem 3.1 and, thus, finish the proof of Theorem 3.1.

4. An algorithm for partitioning unweighted graphs. In this section, we present our main result: an I/O-efficient algorithm for computing a proper r-partition of an unweighted planar graph G. In particular, we prove the following.

THEOREM 4.1. *Given a planar graph G and an integer $r > 0$, a proper r-partition of G can be computed in* $\mathrm{O}(\mathrm{sort}(N))$ *I/Os, provided that $M \geq 56r \log^2 B$.*

Our algorithm (Algorithm 2) consists of three steps. The first two compute a separator S_0 of size $\mathrm{O}(N/\sqrt{r})$ that defines an $\left(r \log^2 B\right)$-partition of G. The last step then refines this partition to an r-partition by adding at most $\mathrm{O}(N/\sqrt{r})$ more vertices to the separator. The reason for not computing an r-partition immediately in the first two steps is that Step 2 consists of $\log B$ iterations, each of which adds $\mathrm{O}(N/\sqrt{r'})$ vertices to the separator if an r'-partition is desired. By choosing $r' = r \log^2 B$, we ensure that the total number of separator vertices chosen in the first two steps is $\mathrm{O}(N/\sqrt{r})$, and the refinement in Step 3 increases this number by only a constant factor.

Algorithm 2 Computing a separator for an unweighted planar graph.

Procedure SEPARATOR

Input: A planar graph $G = (V, E)$ and an integer $r > 0$.
Output: A proper r-partition $\mathcal{P} = (S, \mathcal{R})$ of G.

1: $\ell \leftarrow \lfloor \log B \rfloor - 1$

2: Step 1: Compute the graph hierarchy

$G_0 \leftarrow G$
for $i = 1, \ldots, \ell$ **do**
Compute a graph G_i that satisfies properties (GH3)–(GH6) from G_{i-1}.
end for

3: Step 2: Compute the separator S_0

Apply Theorem 2.1 to compute a separator $S_\ell \subseteq V(G_\ell)$ of size $O(|G_\ell|/(\sqrt{r} \log B))$ and whose removal partitions G_ℓ into connected components of size at most $r \log^2 B$.
for $i = \ell - 1, \ldots, 0$ **do**
Compute a separator S_i for G_i. Separator S_i consists of two sets S_i' and S_i''. S_i' is the set of vertices represented by the vertices in S_{i+1}. S_i'' is the set of separator vertices introduced in order to partition the connected components of $G_i - S_i'$ into subgraphs of size at most $r \log^2 B$.
end for

4: Step 3: Compute the final partition

Compute the partition $\mathcal{P} = (S, \mathcal{R})$ by dividing the connected components of $G - S_0$ into $O(N/r)$ subgraphs of size at most r and boundary size $O(\sqrt{r})$ and adding the required separator vertices to S_0.

Step 3 is easy to implement I/O-efficiently: Given the assumption that $M \geq 56r \log^2 B$, every connected component of $G - S_0$ fits in internal memory. Hence, we can compute the desired r-partition by loading each component of $G - S_0$ into memory and applying Theorem 2.1 to it without incurring any further I/Os.

In Steps 1 and 2, we apply the same idea iteratively. In Step 1, we construct a hierarchy of $\ell = \lfloor \log B \rfloor - 1$ planar graphs $G_0, \ldots, G_\ell$, where $G_0 = G$ and $|G_\ell| = O(N/B)$. We obtain each graph G_i from the previous graph G_{i-1} using the uniform graph contraction procedure from section 3. Given the reduced size of G_ℓ, we can compute an $\big(r \log^2 B\big)$-partition of G_ℓ in $O(N/B)$ I/Os using Theorem 2.1. Let S_ℓ be the set of separator vertices used in this partition. In Step 2, we undo the contraction steps that produced graphs $G_1, \ldots, G_\ell$ from G_0, one graph at a time. In each iteration, we derive a separator S_i for G_i from the separator S_{i+1} computed for G_{i+1} in the previous iteration. We do this as follows: Since G_{i+1} is obtained from G_i using the uniform graph contraction procedure, every vertex in G_{i+1} represents a set of vertices in G_i. Let S_i' be the set of vertices in G_i represented by the vertices in S_{i+1}. In order to obtain an $\big(r \log^2 B\big)$-partition of G_i, we partition every component of $G_i - S_i'$ into subgraphs of size at most $r \log^2 B$ and add the separator vertices used in this partition to a set S_i''. The separator S_i is the union of sets S_i' and S_i''. S_0 is the separator of $G_0 = G$ obtained at the end of this process.

In order to carry out Step 2 efficiently, and in order to obtain a small separator S_0 at the end of Step 2, the graph hierarchy $G_0, \ldots, G_\ell$ computed in Step 1 has to satisfy a number of properties.

First, we want every graph G_{i+1} to have roughly half as many vertices as G_i. This guarantees that the cost of Steps 1 and 2 is dominated by the cost of the computation on $G_0 = G$ and, since $\ell = \lfloor \log B \rfloor - 1$, that G_ℓ has size $O(N/B)$.

Second, during the construction of the separator S_i from S'_i, we would like to use an internal-memory algorithm to partition each connected component of $G_i - S'_i$; that is, we want each such component to fit in memory. Since S_{i+1} defines an $\left(r \log^2 B\right)$-partition of G_{i+1} and $M \geq 56r \log^2 B$, every connected component of $G_i - S'_i$ fits in memory if every vertex in G_{i+1} represents at most 56 vertices in G_i.

Finally, observe that, once we add a vertex x in G_i to S_i, all vertices in G represented by x belong to S_0. Thus, to guarantee that S_0 is small, we have to ensure that no vertex in G_i represents too many vertices of G.

These conditions are formalized in the following properties which we require the graph hierarchy $G_0, \ldots, G_\ell$ to have:

(GH1) $\ell = \lfloor \log B \rfloor - 1$,
(GH2) $G = G_0$,
(GH3) For $i = 0, \ldots, \ell$, G_i is planar,
(GH4) For $i = 0, \ldots, \ell$, $|G_i| \leq 7N/2^{i-1}$,
(GH5) For $i = 1, \ldots, \ell$, every vertex in G_i represents at most 56 vertices in G_{i-1}, and
(GH6) For $i = 0, \ldots, \ell$, every vertex in G_i represents at most 2^{i+1} vertices in G.

In section 4.1, we show how to compute a graph hierarchy with these properties in $O(\mathrm{sort}(N))$ I/Os. In section 4.2, we show that the desired separator S_0 of G can be computed from this graph hierarchy in $O(\mathrm{sort}(N))$ I/Os. In section 4.3, we provide the details of Step 3 and show that this step can be carried out in the same I/O bound as Steps 1 and 2, thereby establishing the complexity of Algorithm 2 claimed in Theorem 4.1.

4.1. The graph hierarchy. The first step of our algorithm is the computation of a hierarchy of graphs $G_0, \ldots, G_\ell$ that satisfy properties (GH1)–(GH6). We start by setting $G_0 = G$. Then we compute each graph G_i from the previous graph G_{i-1} by using the uniform graph contraction procedure from section 3. To ensure properties (GH5) and (GH6), we assign a *weight* $w(x)$ and a *size* $s(x)$ to every vertex in G_{i-1}; the former is equal to the number of vertices in G represented by x, and the latter is equal to the number of vertices in G_{i-1} represented by x, that is, equal to 1. We then pass a weight threshold $u_w = 2^{i+1}$ and a size threshold $u_s = 56$ to the contraction procedure.

Note that the assignment of weights and sizes to the vertices of G_{i-1} before the construction of G_i is easily accomplished. Initially, we set $w(x) = s(x) = 1$ for every vertex x in $G_0 = G$. Subsequently, before computing G_i from G_{i-1}, the size of every vertex in G_{i-1} can be reset to 1 in a single scan over the vertex set of G_{i-1}; as a result of the construction of G_{i-1} from G_{i-2}, every vertex in G_{i-1} already stores its correct weight.

LEMMA 4.2. *A graph hierarchy $G_0, \ldots, G_q$ with properties* (GH1)–(GH6) *can be constructed in* $O(\mathrm{sort}(N))$ *I/Os.*

Proof. First, we prove that the graph hierarchy computed by the procedure we have just described has the desired properties. Properties (GH1) and (GH2) are trivially satisfied. Property (GH3) is easy to prove by induction: For $i = 0$, $G_0 = G$ and is thus planar. For $i > 0$, the planarity of G_i follows from the planarity of G_{i-1} because, by Theorem 3.1, the uniform contraction procedure preserves planarity.

Properties (GH5) and (GH6) are also easy to show by induction. In particular,

$w(x) = 1 \leq 2$ for every vertex $x \in G_0$, and property (GH5) holds vacuously in this case. When constructing G_i from G_{i-1}, every vertex in G_{i-1} is within bounds because, by the induction hypothesis, it has weight at most 2^i and size 1 (after resetting its size to 1). Hence, by Theorem 3.1, every vertex in G_i is within bounds and, thus, has weight at most 2^{i+1} and size at most 56.

It remains to show property (GH4). For graph G_0, property (GH4) holds because $|G_0| = |G| = N \leq 7N/2^{-1}$. To prove the claim for graphs $G_1, \ldots, G_\ell$, let h_i be the number of heavy vertices in G_i. By Theorem 3.1, each graph G_i has size less than $7h_i$. To prove property (GH4), it is therefore sufficient to prove that $h_i \leq N/2^{i-1}$.

We prove this claim by induction. We partition the heavy vertices into two categories. A heavy vertex of type I has weight exceeding 2^i. A type-II vertex has weight at most 2^i and size greater than 28. Graph G_i contains less than $N/2^i$ type-I vertices and less than $|G_{i-1}|/28$ type-II vertices; that is, $h_i < N/2^i + |G_{i-1}|/28$.

For $i = 1$, we obtain $h_1 < N/2 + N/28 < N/2^0$. For $i > 1$, we obtain

$$h_i < \frac{N}{2^i} + \frac{|G_{i-1}|}{28} \tag{1}$$

$$\leq \frac{N}{2^i} + \frac{h_{i-1}}{4} \tag{2}$$

$$\leq \frac{N}{2^i} + \frac{N}{2^i} \tag{3}$$

$$= \frac{N}{2^{i-1}}. \tag{4}$$

Equation (2) follows from (1) because $|G_{i-1}| \leq 7h_{i-1}$, as argued above. Equation (3) follows from (2) by the induction hypothesis.

To bound the I/O-complexity, we recall that, by Theorem 3.1, the construction of graph G_i from graph G_{i-1} takes $\mathrm{O}(\mathrm{sort}(|G_{i-1}|))$ I/Os. By property (GH4), we have $\sum_{i=0}^{\ell} |G_i| = \mathrm{O}(N)$. Thus, the total I/O-complexity is $\sum_{i=1}^{\ell} \mathrm{O}(\mathrm{sort}(|G_{i-1}|)) = \mathrm{O}(\mathrm{sort}(N))$. □

4.2. The separator hierarchy. In Step 2 of Algorithm 2, we use the graph hierarchy computed in the first step to construct a relatively coarse partition of G. In particular, we compute a separator S_0 of size $\mathrm{O}(N/\sqrt{r})$ whose removal partitions G into connected components of size at most $r \log^2 B$.

First, we compute a partition of G_ℓ into subgraphs of size at most $r \log^2 B$. To do so, we use an arbitrary linear-time planar embedding algorithm (see, e.g., [11]) to compute a planar embedding of G_ℓ, and then we apply Theorem 2.1 to compute the desired partition. Let $S_\ell = S''_\ell$ be the computed separator.

In the loop in Step 2, we apply the following iterative strategy to compute separators $S_{\ell-1}, \ldots, S_0$ for graphs $G_{\ell-1}, \ldots, G_0$: Given the separator S_{i+1} computed for graph G_{i+1} in the previous iteration, we construct the set S'_i of vertices in G_i represented by the vertices in S_{i+1}. Then we apply Theorem 2.1 to each connected component of $G_i - S'_i$ whose size exceeds $r \log^2 B$, in order to partition it into subgraphs of size at most $r \log^2 B$. Let S''_i be the set of separator vertices introduced by partitioning the connected components of $G_i - S'_i$ in this manner. Then $S_i = S'_i \cup S''_i$.

LEMMA 4.3. *The separator S_0 of G computed in Step 2 of Algorithm 2 has size $\mathrm{O}(N/\sqrt{r})$. The connected components of $G - S_0$ have size at most $r \log^2 B$.*

Proof. The bound on the size of the connected components of $G - S_0$ is explicitly guaranteed by the construction. We bound the size of S_0 as follows: For every vertex

$x \in G_i$, let $R_0(x)$ be the set of vertices in G represented by x. From our computation of S_0 it follows that, for every vertex $y \in S_0$, there exists a unique set S_i'' and a unique vertex $x \in S_i''$ such that $y \in R_0(x)$. Hence, we have

$$|S_0| = \sum_{i=0}^{\ell} \sum_{x \in S_i''} |R_0(x)|.$$

By property (GH6), we have $|R_0(x)| = w(x) \leq 2^{i+1}$ for all $x \in S_i''$. Hence,

$$|S_0| \leq \sum_{i=0}^{\ell} 2^{i+1} |S_i''|.$$

Since we compute S_i'' by applying Theorem 2.1 to disjoint subgraphs of G_i, partitioning each into pieces of size at most $r \log^2 B$, we have $|S_i''| \leq 4|G_i|/(\sqrt{r} \log B)$. By property (GH4), this implies that $|S_i''| \leq 28N/(2^{i-1}\sqrt{r} \log B)$. Thus,

$$|S_0| \leq \sum_{i=0}^{\ell} 2^{i+1} \frac{28N}{2^{i-1}\sqrt{r} \log B} = \sum_{i=0}^{\ell} \frac{112N}{\sqrt{r} \log B} = \frac{112N}{\sqrt{r}}. \qquad \square$$

LEMMA 4.4. *Step* 2 *of Algorithm* 2 *takes* $O(\mathrm{sort}(N))$ *I/Os to compute the separator* S_0*, provided that* $M \geq 56r \log^2 B$.

Proof. The computation of the separator S_ℓ takes $O(N/B)$ I/Os by Theorem 2.1 and because graph G_ℓ has size at most $7N/2^{\ell-1} = O(N/B)$. Since the sizes of graphs $G_0, \ldots, G_\ell$ are geometrically decreasing, it suffices to show that the separator S_i can be constructed from S_{i+1} in $O(\mathrm{sort}(|G_i|))$ I/Os. This implies then that Step 2 takes $O(\mathrm{sort}(|G_0|)) = O(\mathrm{sort}(N))$ I/Os.

Since the uniform graph contraction procedure labels every vertex x in G_i with the vertex in G_{i+1} representing x, we can compute the separator S_i' in $O(\mathrm{sort}(|G_i|))$ I/Os: First, we sort the vertices in S_{i+1} by their IDs, and then we sort the vertices in G_i by their representatives in G_{i+1}. Then we scan the two lists and mark all those vertices in G_i as being in S_i' whose representatives in G_{i+1} belong to S_{i+1}. Now it takes $O(\mathrm{sort}(|G_i|))$ I/Os to compute the connected components of $G_i - S_i'$ (see section 2.4).

Since every connected component of $G_{i+1} - S_{i+1}$ has size at most $r \log^2 B$, it follows from property (GH5) and Fact 2.2 that every connected component of $G_i - S_i'$ has size at most $56r \log^2 B \leq M$; that is, each such component fits in memory. Thus, we can load each connected component of $G_i - S_i'$ whose size exceeds $r \log^2 B$ into memory and apply Theorem 2.1 to partition it into connected components of size at most $r \log^2 B$. As the computation of Theorem 2.1 is carried out in internal memory, partitioning the connected components of $G_i - S_i'$ into subgraphs of size at most $r \log^2 B$ takes $O(|G_i|/B)$ I/Os. Thus, the computation of S_i takes $O(\mathrm{sort}(|G_i|))$ I/Os, and the total I/O-bound follows. $\square$

4.3. Computing the final partition. In order to obtain the final partition in Step 3 of Algorithm 2, we have to partition the connected components of $G - S_0$ into subgraphs of size at most r. We also have to merge subgraphs to reduce their number to $O(N/r)$, while maintaining the bounds on their size and boundary size. We do this as follows: First, we use Theorem 2.1 to partition each connected component of $G - S_0$ into pieces of size at most r. This adds $O(N/\sqrt{r})$ vertices to the separator and, thus, increases the separator size by only a constant factor. The resulting partition may

contain more than $O(N/r)$ subgraphs, and the total boundary size of its subgraphs may exceed $O(N/\sqrt{r})$. We group the subgraphs in the current partition to reduce their number to $O(N/r)$ and their total boundary size to $O(N/\sqrt{r})$. Finally, we partition each subgraph in the resulting partition into subgraphs of boundary size $O(\sqrt{r})$. This is similar to Frederickson's approach [19] and, as argued below, increases both the number of subgraphs in the partition and the total boundary size by only a constant factor.

Since every connected component of $G - S_0$ has size at most $r \log^2 B \leq M$, the partition of $G - S_0$ into connected components of size at most r can be computed by loading each connected component of $G - S_0$ into internal memory and applying Theorem 2.1 to it. Let $S' \supseteq S_0$ be the separator produced by this step. Section 4.3.1 discusses how to obtain a normal r-partition $\mathcal{P}' = (S', \{G'_1, \ldots, G'_r\})$ of G from S'. Section 4.3.2 then refines this partition to make it proper.

4.3.1. Grouping components. Intuitively, we compute the partition $\mathcal{P}'$ in two phases: The first phase groups connected components of boundary size at most two with other components that have the same boundary, while ensuring that none of the resulting subgraphs has size greater than r. This phase reduces the total boundary size of all subgraphs to $O(N/\sqrt{r})$. The second phase merges subgraphs that share boundary vertices until no two subgraphs sharing boundary vertices can be merged without producing a subgraph of size greater than r. This reduces the number of subgraphs to $O(N/r)$. Note that merging subgraphs in this manner cannot increase the total boundary size; that is, the total boundary size of all subgraphs in the partition remains $O(N/\sqrt{r})$, and the resulting partition $\mathcal{P}'$ is normal.

To determine which connected components to group in these two phases, we use an auxiliary graph H_0 whose vertices represent separator vertices and connected components of $G - S'$. Both phases operate on H_0, grouping component vertices rather than actual components. After the two phases have been applied to H_0, we obtain $\mathcal{P}'$ by merging the components in the partition that correspond to merged component vertices in H_0.

Graph H_0 contains all vertices in S' and one vertex per connected component of $G - S'$. There is an edge between two separator vertices in H_0 if such an edge exists in G. There is an edge between a separator vertex x and a component vertex representing a component G' of $G - S'$ if $x \in \mathcal{N}(G')$. Finally, every vertex in H_0 has a weight equal to the number of vertices in G it represents.

Graph H_0 is easily constructed from G and S' in $O(\mathrm{sort}(N))$ I/Os: First, we compute the connected components of $G - S'$, thereby labeling every vertex in $G - S'$ with the ID of the component it belongs to; we also label every vertex in S' with its own ID. Then we apply the contraction procedure from section 2.4 to G.

Reducing the total boundary size. Merging components of boundary size at most two that have the same boundary is equivalent to merging component vertices in H_0 that have the same neighbors and degree at most two. The latter is easily achieved using the uniform graph contraction procedure (in fact, only the bundling phase is sufficient). For the purpose of applying this procedure, we change the weight of every separator vertex to r, leave the weights of all component vertices unchanged, and set the weight threshold to r. Then the edge contraction phase does nothing because every edge of H_0 has at least one endpoint that is a separator vertex, that is, is heavy. The bundling phase merges light component vertices that have the same neighbors and degree at most two. By Theorem 3.1, the application of the uniform contraction procedure takes $O(\mathrm{sort}(|H_0|)) = O(\mathrm{sort}(N))$ I/Os. The next lemma proves that this

produces a graph H_1 from H_0 whose component vertices represent subgraphs of $G-S'$ of size at most r each and sufficiently small total boundary size. Note that the bound on the size of H_1 stated in the lemma implies the claimed bound on the boundary size of the corresponding partition of G because the total boundary size is equal to the number of edges between component vertices and separator vertices in H_1. Since H_1 is planar and has size $O(N/\sqrt{r})$, there are only $O(N/\sqrt{r})$ edges in H_1.

LEMMA 4.5. *Applying the uniform contraction procedure to H_0 produces a planar graph H_1 of size $O(N/\sqrt{r})$. Every vertex in H_1 has weight at most r.*

Proof. Since H_0 is an edge contraction of G, Fact 2.1 implies that H_0 is planar. By Theorem 3.1, this implies that H_1 is planar. Before applying the contraction procedure to H_0, all vertices are within bounds, that is, have weight at most r. Hence, by Theorem 3.1, every vertex in H_1 has weight at most r. Finally, to bound the size of H_1, observe that H_1 contains only $O(N/\sqrt{r})$ heavy vertices: $O(N/\sqrt{r})$ separator vertices and $O(N/r)$ heavy component vertices; the latter is true because the total weight of all component vertices in H_1 is at most N. By Theorem 3.1, this implies that H_1 has $O(N/\sqrt{r})$ vertices. ∎

Reducing the number of subgraphs. To reduce the size of H_1 to $O(N/r)$ by further merging vertices, we first reset the weight of every separator vertex to 1 and then apply the uniform contraction procedure to H_1, again with weight threshold r. Since $|H_1| \leq N$, this takes $O(\mathrm{sort}(N))$ I/Os by Theorem 3.1. Since a vertex is heavy if its weight exceeds $r/2$, and the total weight of all vertices in the resulting graph H_2 is N, there are at most $2N/r$ heavy vertices in H_2. By Theorem 3.1, this implies that the total size of H_2 is $O(N/r)$. Moreover, since no vertex in H_1 has weight exceeding r, Theorem 3.1 implies that no vertex in H_2 has weight exceeding r. Thus, we have the following.

LEMMA 4.6. *Applying the uniform contraction procedure to H_1 produces a planar graph H_2 of size $O(N/r)$. Every vertex in H_2 has weight at most r.*

The final grouping. Every component vertex in H_2 now represents a subgraph in the partition $\mathcal{P}'$. We finish the computation of $\mathcal{P}'$ by labeling every nonseparator vertex with the ID of the subgraph it belongs to. This is easily achieved by sorting and scanning the vertex sets of G, H_0, H_1, and H_2 a constant number of times. Indeed, every vertex in $G - S'$ is initially labeled with the connected component of $G - S'$ that contains it, that is, with its representative in H_0. Similarly, every vertex in H_0 is labeled with its representative in H_1, and every vertex in H_1 is labeled with its representative in H_2. Thus, sorting and scanning suffices to label every vertex in H_1, and subsequently every vertex in H_0 and G, with its representative in H_2. This labeling represents the subgraphs in $\mathcal{P}'$.

LEMMA 4.7. *Given the separator S_0, a normal r-partition $\mathcal{P}'$ of G can be computed in $O(\mathrm{sort}(N))$ I/Os.*

Proof. The I/O-bound of computing $\mathcal{P}'$ from S_0 follows from the above discussion of the different steps required to obtain $\mathcal{P}'$ from G and S_0.

There are only $O(N/r)$ subgraphs in $\mathcal{P}'$ because there are only $O(N/r)$ component vertices in H_2 and each of them defines one subgraph in $\mathcal{P}'$. Every subgraph in the partition has size equal to the weight of its representative in H_2; by Lemma 4.6, no vertex in H_2 has weight exceeding r. Finally, by Lemma 4.5, there are only $O(N/\sqrt{r})$ edges in H_1. Each such edge represents an adjacency between a separator vertex and a subgraph in the partition $\mathcal{P}''$ of G represented by H_1. Thus, $\mathcal{P}''$ has total boundary size $O(N/\sqrt{r})$. Since partition $\mathcal{P}'$ is obtained by merging subgraphs in $\mathcal{P}''$, the total boundary size of the subgraphs in $\mathcal{P}'$ cannot be greater than the total boundary size of the subgraphs in $\mathcal{P}''$. ∎

4.3.2. Ensuring small boundary size. In order to obtain a proper r-partition from the normal r-partition $\mathcal{P}'$, we have to reduce the boundary size of each individual subgraph to $O(\sqrt{r})$ by further partitioning each subgraph in $\mathcal{P}'$ whose boundary size exceeds this bound. In order to do so, we apply Corollary 2.3. This, however, requires that each graph $\mathcal{N}[G'_i]$ fit in memory. While $|G'_i| \leq r \leq M$, the graph $\mathcal{N}[G'_i]$ may be big and may not fit in memory.

We solve this problem by first augmenting the separator so that its size increases by only a constant factor, and every graph G''_i in the resulting partition satisfies $|\mathcal{N}[G''_i]| = O(r) \leq M$. This then allows us to apply Corollary 2.3 to obtain the final partition.

To augment the separator, we consider each graph G'_i in the partition $\mathcal{P}'$ in turn. Let $\tilde{G}_i$ be the graph obtained from $\mathcal{N}[G'_i]$ as follows: First, we remove all edges between vertices in $\mathcal{N}(G'_i)$. Then we merge all vertices in $\mathcal{N}(G'_i)$ whose resulting degree is at most 2 that have the same set of neighbors (which all belong to G'_i). For every vertex in $\tilde{G}_i$ that represents more than one vertex in $\mathcal{N}(G'_i)$, we add its neighbors in G'_i to a set S''_i. Then we define $\mathcal{P}'' = (S'', \{G''_1, \ldots, G''_p\})$, where $G''_i = G[V(G'_i) \setminus S''_i]$ and $S'' = S' \cup \bigcup_{i=1}^{p} S''_i$; that is, in $\mathcal{P}''$, the vertices in S''_i are removed from the graph G'_i and are added to the separator.

LEMMA 4.8. *Let $\mathcal{P}'' = (S'', \{G''_1, \ldots, G''_p\})$ be the partition obtained from $\mathcal{P}'$ using the above transformation. Then $\mathcal{P}''$ is normal, and every graph $\mathcal{N}[G''_i]$, $1 \leq i \leq p$, has size $O(r)$.*

Proof. Let $\mathcal{P}' = (S', \{G'_1, \ldots, G'_p\})$. Then $|S'| = O(N/\sqrt{r})$ and $\sum_{i=1}^{p} |\mathcal{N}(G_i)| = O(N/\sqrt{r})$ because $\mathcal{P}'$ is normal. Now consider subgraphs G'_i and G''_i, and let $\mathcal{N}_{\text{sh}}(G'_i) = \mathcal{N}(G'_i) \cap \mathcal{N}(G''_i)$ and $\mathcal{N}_{\text{dis}}(G'_i) = \mathcal{N}(G'_i) \setminus \mathcal{N}(G''_i)$.

We start by proving that $|S''_i| \leq |\mathcal{N}_{\text{dis}}(G'_i)|$. Consider a vertex $x \in S''_i$. This vertex belongs to S''_i because there exists a vertex $y \in \tilde{G}_i$ that represents a set V_y of $h \geq 2$ vertices in $\mathcal{N}(G'_i)$ that are adjacent to x. We charge each such vertex $1/h \leq 1/2$ for the addition of x to S''_i. Since each vertex in V_y is adjacent to at most two vertices in G'_i, each vertex in V_y is charged for at most two vertices, and the charge to each vertex is at most 1. Thus, the total number of charged vertices is an upper bound on $|S''_i|$. Note, however, that all neighbors of a charged vertex that belong to G'_i are added to S''_i. Hence, every charged vertex belongs to $\mathcal{N}_{\text{dis}}(G'_i)$ and $|S''_i| \leq |\mathcal{N}_{\text{dis}}(G'_i)|$. This immediately implies that partition $\mathcal{P}''$ is normal: The size of S'' is $|S''| = |S'| + \sum_{i=1}^{p} |S''_i| \leq |S'| + \sum_{i=1}^{p} |\mathcal{N}_{\text{dis}}(G'_i)| \leq |S'| + \sum_{i=1}^{p} |\mathcal{N}(G'_i)| = O(N/\sqrt{r})$. For each graph G''_i, we have $|\mathcal{N}(G''_i)| \leq |\mathcal{N}_{\text{sh}}(G'_i)| + |S''_i| \leq |\mathcal{N}_{\text{sh}}(G'_i)| + |\mathcal{N}_{\text{dis}}(G'_i)| = |\mathcal{N}(G'_i)|$. Therefore, $\sum_{i=1}^{p} |\mathcal{N}(G''_i)| \leq \sum_{i=1}^{p} |\mathcal{N}(G'_i)| = O(N/\sqrt{r})$.

To bound the size of each graph $\mathcal{N}[G''_i]$, we partition its vertices into two groups: those in G'_i and those in $\mathcal{N}_{\text{sh}}(G'_i)$. Since $|G'_i| \leq r$, $\mathcal{N}[G''_i]$ can contain at most r vertices that belong to G'_i. Next, observe that there are no two vertices x and y of degree at most two in $\mathcal{N}_{\text{sh}}(G'_i)$ such that $\mathcal{N}(x) = \mathcal{N}(y)$: if there were two such vertices, their neighbors in G'_i would have been added to S''_i. Thus, the subgraph of G induced by the edges between vertices in $\mathcal{N}_{\text{sh}}(G'_i)$ and vertices in G'_i satisfies the conditions of Corollary 2.5, and the number of vertices in $\mathcal{N}_{\text{sh}}(G'_i)$ is less than $6|G'_i| \leq 6r$. The size of $\mathcal{N}[G''_i]$ is therefore at most $7r$. ∎

The computation of partition $\mathcal{P}''$ takes $O(\text{sort}(N))$ I/Os. The computation of graph $\tilde{G}_i$, for every graph $\mathcal{N}[G'_i]$, is easily carried out using the uniform graph contraction procedure after assigning weight 1 to every separator vertex and weight $2N$ to every vertex in G'_i; the weight threshold is $2N$. The construction of set S''_i now requires scanning the vertex set of $\tilde{G}_i$ and adding the neighbors of all separator vertices

of weight at least two and degree at most two to S_i''. Thus, the computation for each graph $\mathcal{N}[G_i']$ takes $O(\text{sort}(|\mathcal{N}[G_i']|))$ I/Os, and the total cost is $\sum_{i=1}^{p} O(\text{sort}(|\mathcal{N}[G_i']|))= O(\text{sort}(N))$.

To obtain the final partition, we apply Theorem 2.2 to every subgraph G_i'' in partition $\mathcal{P}''$ whose boundary size exceeds $c\sqrt{r}$, for an appropriate constant $c > 0$. By Corollary 2.3, this can be done in $O(N/B)$ I/Os and increases the size of the separator and the number of graphs in the partition by only a constant factor; that is, this final step produces a proper r-partition of G.

Since we have shown that all three steps of Algorithm 2 can be carried out in $O(\text{sort}(N))$ I/Os and that the final partition we obtain is proper, we have thus shown Theorem 4.1.

5. Regular partitions. Our result from the previous section provides an algorithm for computing proper r-partitions of planar graphs, as long as r is small; but the computed partitions are not necessarily regular. In general, without a bound on the degrees of the vertices in the graph, a regular proper r-partition may not exist. For planar graphs of degree three, however, regular proper r-partitions do exist [19], and the algorithms of [5, 7, 8, 9] rely on the existence of such partitions. In this section, we show how to modify the partition produced by Algorithm 2 to obtain a regular proper r-partition for a planar graph of degree three.

Given such a graph G, we use Algorithm 2 to compute a proper r-partition $\mathcal{P} = (S, \{G_1, \ldots, G_p\})$ of G. We are, however, interested only in the separator S and discard the grouping of the connected components $G - S$ into subgraphs. Next we regroup the connected components of $G - S$ to obtain the desired regular proper r-partition $\mathcal{P}' = (S, \{G_1', \ldots, G_q'\})$. This grouping is again similar to [19].

We use an auxiliary graph H to compute the desired grouping. Graph H contains one vertex per connected component of $G - S$. There is an edge between two vertices in H if the two corresponding components of $G - S$ share a boundary vertex. Since every vertex in G has degree at most three, graph H is planar. We give two weights $s(x)$ and $b(x)$ to each vertex x in H: $s(x)$ is the size, that is, the number of vertices in the connected component represented by x; $b(x)$ is the size of the component's boundary. Note that $s(x) \leq r$ and $b(x) \leq c\sqrt{r}$, for some $c > 0$, because $\mathcal{P}$ is a proper r-partition.

Now we apply the uniform graph contraction procedure to H, with thresholds $u_s = r$ and $u_b = c\sqrt{r}$. This produces a graph H', each of whose vertices represents a set of vertices in H and, thus, a set of connected components of $G - S$. The graphs $G_1', \ldots, G_q'$ in partition $\mathcal{P}'$ are defined as the graphs represented by the vertices in H'. By the arguments in section 4.3, this partition of $G - S$ can be computed in $O(\text{sort}(N))$ I/Os.

THEOREM 5.1. *Given an N-vertex planar graph G none of whose vertices has degree greater than three, and an integer $r > 0$, a regular proper r-partition of G can be computed in* $O(\text{sort}(N))$ *I/Os, provided that* $M \geq 56r \log^2 B$.

Proof. The I/O-complexity of the procedure follows immediately from Theorem 4.1 and our discussion above. Next, we argue that the produced partition $\mathcal{P}'$ is proper. Since the partition $\mathcal{P}$ produced by Algorithm 2 is proper, we have $|S| = O(N/\sqrt{r})$. This implies that the total boundary size of the connected components of $G - S$ cannot exceed $O(N/\sqrt{r})$ because every vertex in G has degree at most three, and, thus, every vertex in S is adjacent to at most three connected components of $G - S$. Therefore, there are only $O(N/r)$ heavy vertices in H': at most $2N/r$ vertices of size greater than $r/2$ and $O(N/r)$ vertices of boundary size greater than $c\sqrt{r}/2$.

By Theorem 3.1, this implies that $|H'| = O(N/r)$, that is, that partition $\mathcal{P}'$ has $O(N/r)$ subgraphs. Since every vertex x in H has weights $s(x) \leq r$ and $b(x) \leq c\sqrt{r}$, Theorem 3.1 implies that the same is true for every vertex in H'. Thus, no subgraph in partition $\mathcal{P}'$ has size exceeding r or boundary size exceeding $c\sqrt{r}$, and partition $\mathcal{P}'$ is proper.

In order to prove that $\mathcal{P}'$ is regular, we analyze the two phases of the computation of H' from H. The edge contraction phase of the uniform contraction procedure merges only vertices that are adjacent. In G, this corresponds to merging connected components of $G - S$ that share boundary vertices. Thus, for every graph G_i'' in the resulting partition, the graph $\mathcal{N}[G_i'']$ is connected. The bundling phase merges only vertices of degree at most two that have the same neighbors. Moreover, these neighbors are heavy, that is, are not merged with any other vertices during the bundling phase. Thus, the merging of vertices during the bundling phase corresponds to producing merged subgraphs that are potentially disconnected but share boundary vertices with at most two other subgraphs G_j' and G_k', which satisfy that $\mathcal{N}[G_j']$ and $\mathcal{N}[G_k']$ are connected. □

6. Low-cost separators and edge separators. In this section, we show that the results from sections 4 and 5 can be used to obtain an I/O-efficient version of the following theorem by Aleksandrov et al. [3].

THEOREM 6.1 (Aleksandrov et al. [3]). *Given a planar graph $G = (V, E)$, a cost function $c : V \to \mathbb{R}^+$, a weight function $w : V \to \mathbb{R}^+$, and a real number $0 < t < 1$, there exists a t-vertex separator S of cost $c(S) \leq 4\sqrt{2C(G)/t}$ for G, where $c(S) = \sum_{x \in S} c(x)$ and $C(G) = \sum_{x \in V} (c(x))^2$. Such a separator can be computed in linear time.*

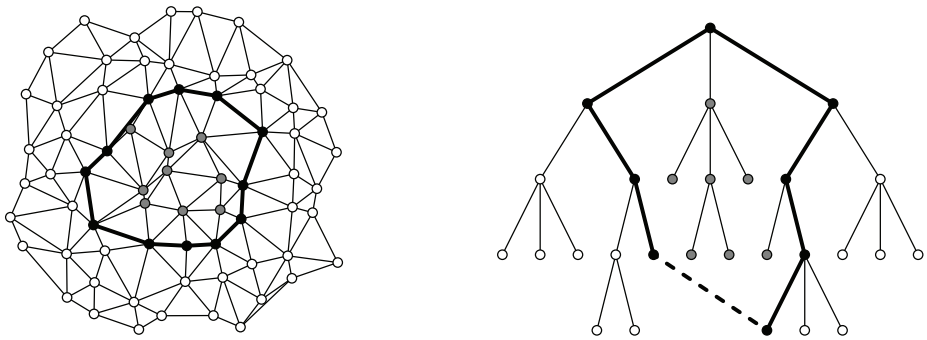
In this theorem, the sizes of the subgraphs in the computed partition are measured in terms of the total *weight* assigned to their vertices by a weight function w. Similarly, the size of the separator S is measured in terms of the total *cost* assigned to the vertices in S by a cost function c. The former is a fairly standard notion already considered in the classical paper by Lipton and Tarjan [27]. The latter is a more recent concept that allows a number of separator theorems to be seen as special cases of Theorem 6.1. Theorem 2.1, for example, can be obtained from Theorem 6.1 by choosing $c(x) = 1$ for all $x \in V$, and Aleksandrov et al. have shown how to obtain a generalization of the edge separator theorem of [15] from Theorem 6.1 (see also Theorem 6.3 in section 6.3). The main result of this section is as follows.

THEOREM 6.2. *Given a planar graph $G = (V, E)$, a cost function $c : V \to \mathbb{R}^+$, and a weight function $w : V \to \mathbb{R}^+$, a separator S as in Theorem* 6.1 *can be computed in* $O(\mathrm{sort}(N))$ *I/Os, provided that* $M = \Omega\big(B^2 \log^2 B\big)$.

Theorem 6.2 is more general than Theorem 4.1 because it takes vertex costs and weights into account; moreover, even in the unweighted case, Theorem 4.1 requires that $r = tN = O\big(M/\log^2 B\big)$, while Theorem 6.2 places no such restriction on r.

Our exposition of the algorithm that proves Theorem 6.2 is organized as follows. In section 6.1, we review the algorithm by Aleksandrov et al., as it forms the basis for our I/O-efficient version. In section 6.2, we provide I/O-efficient implementations of the three main steps of this algorithm, thereby obtaining an I/O-efficient version of the algorithm. This proves Theorem 6.2. In section 6.3, we briefly argue that this also leads to an I/O-efficient version of the edge separator algorithm of [3].

We assume throughout sections 6.1 and 6.2 that the given graph is triangulated; since any planar graph can be triangulated in $O(\mathrm{sort}(N))$ I/Os [25], and a separator



(a) (b)

FIG. 6.1. (a) *The bold cycle separates the white vertices on its outside from the grey ones on its inside.* (b) *The dashed nontree edge defines the bold fundamental cycle.*

of the resulting triangulation is also a separator of the original graph, this is not a restriction.

6.1. The algorithm of Aleksandrov et al. The algorithm of [3] can be seen as a nontrivial extension of Lipton and Tarjan's algorithm [27]. The first observation is that every simple cycle C in G separates the vertices inside C from those outside C in the given embedding of G; see Figure 6.1(a). The central goal then is to compute a collection of cycles that partition G into regions of the desired weight. These cycles are chosen from the set of *fundamental cycles* w.r.t. a spanning tree T of G, where a fundamental cycle $F(e)$ consists of an edge $e \in E(G) \setminus E(T)$ and the path in T connecting the two endpoints of e; see Figure 6.1(b). We refer to an edge $e \in E(G) \setminus E(T)$ as a *nontree edge*, while every edge in T is a *tree edge*.

In order to obtain a separator of low cost in this manner, it is necessary to bound the number of fundamental cycles comprising the separator, as well as the total cost of the vertices on each fundamental cycle. As we will see below, the former is easy, as the number of required cycles is inversely proportional to t. To ensure that each fundamental cycle is of low cost, Aleksandrov et al. compute T as a shortest-path tree w.r.t. appropriate edge weights that guarantee that the depth of every vertex x in T (that is, the weighted distance of x from the root of T) is equal to the total cost of the vertices on the path from the root of T to x; the cost of any fundamental cycle is then at most twice the *radius* of T, where the radius of the tree is the maximum depth of any of its vertices. By itself, this does not yet guarantee low cost of each fundamental cycle because T may have a large radius. To fix this, Aleksandrov et al. first find a set of vertices of low total cost whose removal partitions T into subgraphs $G_0, \ldots, G_p$ of low depth, where the *depth* of a graph G_i is the maximal difference between the depths (in T) of any two vertices in G_i; see Figure 6.2(a). We call these graphs $G_0, \ldots, G_p$ *layers*. They then triangulate each layer and obtain a spanning tree for the resulting triangulation whose radius is bounded by the depth of the layer. Hence, each fundamental cycle w.r.t. this spanning tree has low cost, and the layer can be partitioned using fundamental cycles; see Figure 6.2(b).

In summary, the algorithm therefore consists of two phases. The first phase computes the shortest-path tree T and partitions G into shallow layers by removing an appropriate set of vertices of low total cost. The second phase partitions each layer by

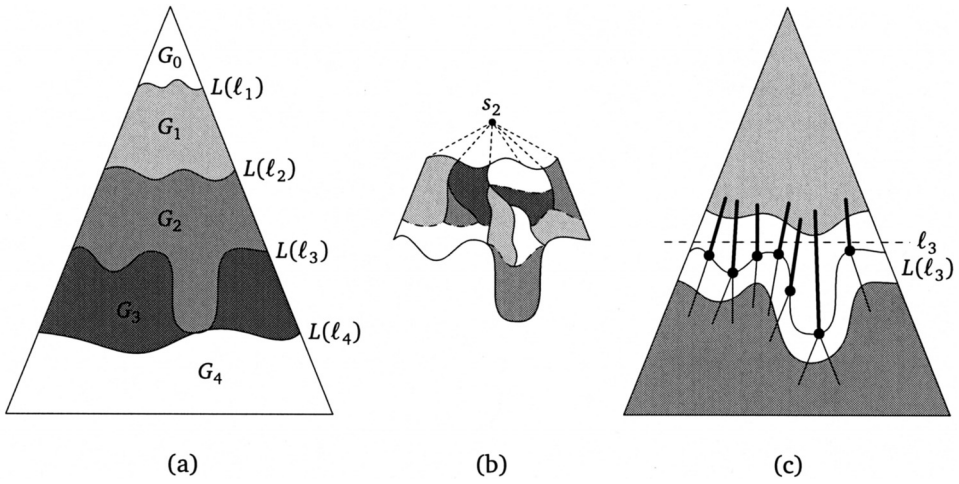


FIG. 6.2. (a) *The partition of G into layers $G_0, \ldots, G_4$ using levels $L(\ell_1), \ldots, L(\ell_4)$.* (b) *The partition of G_2 into subgraphs of weight at most $tw(G)$ using fundamental cycles. The dotted edges are the ones introduced to connect all vertices that were adjacent to vertices in $L(\ell_2)$ to the new root vertex s_2. The dashed edges are the nontree edges defining the fundamental cycles that are used to partition G_2.* (c) *The definition of level $L(\ell_3)$. The bold edges are all edges in T spanning depth ℓ_3. Their bottom endpoints are in $L(\ell_3)$. The sets $V^-(\ell_3)$ and $V^+(\ell_3)$ are shown in light and dark grey, respectively. Note that there are no edges between $V^-(\ell_3)$ and $V^+(\ell_3)$.*

removing the vertices belonging to a small set of fundamental cycles. The separator consists of all vertices removed in these two phases.

Next, we provide more details on the three parts of the algorithm: the computation of the shortest-path tree T, the partitioning of G into shallow layers, and the partitioning of each layer using fundamental cycles.

6.1.1. The shortest-path tree. To compute the spanning tree T used in the separator algorithm, recall that we assume that G is triangulated. The algorithm starts by choosing one face f, adding a new vertex s of cost and weight zero inside this face, and connecting s to the three vertices on the boundary of f.

Next, every edge xy of G is replaced with two directed edges xy and yx, and each directed edge xy is assigned a weight $w'(xy) = c(y)$; that is, the weight of every edge is equal to the cost of its head. Tree T is the shortest-path tree obtained by computing single-source shortest paths from s w.r.t. edge weights w'. It is easy to see that the distance of a vertex x from s in T equals the cost of x's ancestors in T, including x itself. For every vertex $x \in G$, let its *depth* be $d(x) = \text{dist}_T(s, x)$, and let the *radius* of T be $r(T) = \max\{d(x) : x \in V\}$.

6.1.2. Cutting G into layers. The set of vertices used to partition G into shallow layers is the union of a set of *levels* $L(\ell_1), \ldots, L(\ell_p)$, where each level $L(\ell_i)$ is the set of bottom endpoints of all edges spanning depth ℓ_i: $L(\ell_i) = \{y : e = xy \in T \text{ and } d(x) < \ell_i \leq d(y)\}$. See Figure 6.2(c) for an illustration.

Every level $L(\ell_i)$ is a separator of G whose removal partitions the vertex set V of G into two sets $V^-(\ell_i) = \{x \in V : d(x) < \ell_i\}$ and $V^+(\ell_i) = \{x \in V : d(x) \geq \ell_i \text{ and } x \notin L(\ell_i)\}$ so that no vertex in $V^-(\ell_i)$ is adjacent to a vertex in $V^+(\ell_i)$. The set of levels $L(\ell_1), \ldots, L(\ell_p)$ then partitions G into $p+1$ subgraphs $G_0, \ldots, G_p$, where $G_0 = G[V^-(\ell_1)]$, $G_p = G[V^+(\ell_p)]$, and, for $0 < i < p$, $G_i = G[V^+(\ell_i) \cap V^-(\ell_{i+1})]$.

Graphs $G_0, \ldots, G_p$ are the layers we want to compute.

There is obviously a trade-off between the cost of levels $L(\ell_1), \ldots, L(\ell_p)$ and the cost of partitioning the layers $G_0, \ldots, G_p$ using fundamental cycles. By choosing more levels $\ell_1, \ldots, \ell_p$, the layers can be made more shallow, thereby reducing the cost of the fundamental cycles used to partition them. This, however, increases the total cost of levels $L(\ell_1), \ldots, L(\ell_p)$.

As we will see, this trade-off is balanced by partitioning G into $p+1 = \lfloor r(T)/h \rfloor + 1$ layers of depth at most $2h$, where $h = \sqrt{tC(G)/8}$. This is achieved by choosing the value ℓ_i defining each level $L(\ell_i)$ from the interval $((i-1)h, ih]$ so that the level $L(\ell_i)$ has minimal cost among all levels $L(\ell)$ with $(i-1)h < \ell \leq ih$. Indeed, this ensures that two consecutive values ℓ_i and ℓ_{i+1} differ by at most $2h$; that is, every layer has depth at most $2h$. As shown by Aleksandrov et al., it also ensures that the cost of the union S_1 of levels $L(\ell_1), \ldots, L(\ell_p)$ is $c(S_1) \leq C(G)/h = 2\sqrt{2C(G)/t}$.

6.1.3. Partitioning the layers. The removal of levels $L(\ell_1), \ldots, L(\ell_p)$ partitions G into layers $G_0, \ldots, G_p$. If a layer G_i has weight at most $tw(G)$, it does not have to be partitioned further. In general, however, G_i may have a weight exceeding $tw(G)$ and needs to be partitioned into subgraphs of weight at most $tw(G)$. This is done by augmenting G_i to obtain a triangulation that has a spanning tree T_i of diameter not exceeding the depth of G_i; this augmented version of G_i is then partitioned using fundamental cycles w.r.t. T_i.

The augmentation of G_0 involves simply triangulating it. For $i > 0$, graph G_i is augmented in two steps: First, a vertex s_i of weight and cost zero is added to G_i and connected to all vertices in G_i that, in G, are adjacent to vertices in $L(\ell_i)$. The resulting graph is then triangulated, and the edges of G_i are assigned weights as in section 6.1.1. Tree T_i is now chosen to be a shortest-path tree of G_i with root s_i, where $s_0 = s$. See Figure 6.2(b) for an illustration.

The approach for finding the fundamental cycles used to partition G_i into subgraphs of weight at most $tw(G)$ can be explained as follows: Let T_i^* be the dual of T_i. This tree is obtained from the dual G_i^* of G_i by removing all those edges that are dual to edges in T_i. Thus, every edge e^* in T_i^* corresponds to a nontree edge e of G_i and, hence, represents a fundamental cycle $F(e)$ in G_i. If the vertex corresponding to the outer face of G_i is chosen as the root of T_i^*, the descendant vertices and edges of e^* in T_i^* represent the region enclosed by $F(e)$.

The goal now is to assign weights $w^*(e^*)$ to the edges of T_i^* so that the total weight of the edges in T_i^* equals the total weight of the vertices in G_i, and the total weight of the descendant edges of an edge e^* in T_i^* is an upper bound on the weight of the vertices in G_i enclosed by $F(e)$. Given such an assignment of edge weights, it suffices to partition T_i^* into a small number of subtrees of weight at most $tw(G)$ by removing a set of edges from T_i^*; the vertices on the corresponding fundamental cycles then form a separator partitioning G_i into subgraphs of weight at most $tw(G)$.

The weight function w^* is obtained by charging the weight of every vertex x in G_i to some edge e^* of T_i^*: If x has at least one incident nontree edge, edge e^* is chosen to be the dual of one of these edges. Otherwise, e^* is chosen to be the dual of a nontree edge e both of whose endpoints are neighbors of x in T_i. It is easy to see that such an edge always exists.

The weight function is easily seen to have the two properties above: Since every vertex of G_i is charged to exactly one edge of T_i^*, the total weight of the edges in T_i^* equals the total weight of the vertices in G_i. A vertex x in the region enclosed by a fundamental cycle $F(e)$ must have been charged to an edge e_1^* in T_i^* such that e_1 is

also contained in the region enclosed by $F(e)$. Thus, e_1^* is a descendant edge of e^* in T_i^*, and the weight of the descendant edges of e^* is an upper bound on the weight of the vertices in G_i enclosed by $F(e)$. See [2] for a more rigorous argument.

In order to partition T_i^* into subtrees of weight at most $tw(G)$ by removing a set of edges X_i, a leaf of T_i^* is chosen as the root of T_i^*, and the edges of T_i^* are then inspected from the bottom up. For every edge e^*, if the total weight of all its descendant edges, including e^* itself, exceeds $tw(G)/2$, the subtree below e^* is pruned from T_i^* by adding e^* to the edge separator X_i. The edges in the pruned subtree are then no longer counted when determining the total weight of the descendant edges of any ancestor of e^*.

Since the vertices in the dual of a planar triangulation have degree at most three and the root of T_i^* has degree one, T_i^* is a binary tree. Thus, the above procedure ensures that each subtree in the produced partition has weight at most $tw(G)$ and, hence, that the fundamental cycles in the set $\mathcal{F}(X_i) = \{F(e) : e^* \in X_i\}$ partition G_i into subgraphs of weight at most $tw(G)$. To bound the number of fundamental cycles in $\mathcal{F}(X_i)$, observe that every edge e^* in X_i has descendant edges of total weight at least $tw(G)/2$ and that every edge of T_i^* is counted as a descendant edge of at most one edge in X_i. Hence $|\mathcal{F}(X_i)| = |X_i| \leq \frac{2w^*(T_i^*)}{tw(G)} = \frac{2w(G_i)}{tw(G)}$. The total number of fundamental cycles used to partition the layers $G_0, \ldots, G_p$ is therefore at most $\frac{2w(G)}{tw(G)} = 2/t$. Since each layer has depth at most $2h$, the cost of the vertices on one fundamental cycle is at most $4h$ and, hence, the total cost of all fundamental cycles in $\mathcal{F}(X_0) \cup \cdots \cup \mathcal{F}(X_p)$ is at most $8h/t = 2\sqrt{2C(G)/t}$.

Let S_2 be the set of vertices on the fundamental cycles in $\mathcal{F}(X_0) \cup \cdots \cup \mathcal{F}(X_p)$. The final separator $S = S_1 \cup S_2$ partitions G into subgraphs of weight at most $tw(G)$ and has cost $c(S_1) + c(S_2) \leq 4\sqrt{2C(G)/t}$.

Aleksandrov et al. [3] showed how to implement this procedure in linear time. In the next section, we show how to carry out the three steps of the algorithm in O(sort(N)) I/Os.

6.2. An I/O-efficient algorithm.

6.2.1. Computing T. In order to compute the shortest-path tree T, we need to compute a planar embedding of G, triangulate G, add a new vertex of cost and weight 0 inside one of its faces, assign weights $w'(e)$ to the edges of G, and finally, compute single-source shortest paths w.r.t. these edge weights.

A planar embedding of G can be computed in O(sort(N)) I/Os [33]; an embedded planar graph can be triangulated in the same I/O-bound [25]. Next, we extract a description of the faces of the triangulation as lists of vertices, each containing the boundary vertices of one face sorted clockwise around that face; this can also be done in O(sort(N)) I/Os. We add a vertex s to G and traverse the vertex list representing one of the faces of G to add edges between s and the vertices on the boundary of this face to G. Now it takes O(sort(N)) I/Os to label every edge of G with the costs of its endpoints, replace every edge of G with its corresponding directed edges, and assign weights as defined in section 6.1.1 to these edges (see section 2.4). The shortest-path tree T can now be computed in O(sort(N)) I/Os using the shortest-path algorithm of [8].

This procedure for computing T is where we depend on Theorems 4.1 and 5.1. The embedding algorithm of [33] relies on a proper B^2-partition of G; the shortest-path algorithm of [8] requires a regular proper B^2-partition.

6.2.2. Cutting T into layers. To compute the levels $L(\ell_1), \ldots, L(\ell_p)$ used to partition G into layers $G_0, \ldots, G_p$, we first need to compute the values $\ell_1, \ldots, \ell_p$ and then extract the vertices belonging to $S_1 = L(\ell_1) \cup \cdots \cup L(\ell_p)$.

To compute values $\ell_1, \ldots, \ell_p$, we label both endpoints of every edge in T with their costs and their distances from s. Then we sort the edges of T by the distances of their tails from s and scan the sorted edge list to simulate a sweep from $\ell = -\infty$ to $\ell = +\infty$. During this sweep, we maintain the cost $c(L(\ell))$ of the current level $L(\ell)$ and keep track of the value i such that $(i-1)h < \ell \leq ih$. We also maintain the minimum cost $c_{\min}(i)$ of all levels $L(\ell')$ with $(i-1)h < \ell' \leq ih$ we have seen so far, as well as the level ℓ_i such that $(i-1)h < \ell_i \leq ih$ and $c(L(\ell_i)) = c_{\min}(i)$. When $\ell = d(x_j)$, for some vertex x_j, we perform the following operations: First, we test whether $d(x_{j+1}) > ih$. If so, we have finished processing all levels $L(\ell')$ with $(i-1)h < \ell' \leq ih$, so we report ℓ_i, increase i by one, and initialize $c_{\min}(i) = +\infty$. Then we decrease $c(L(\ell))$ by $c(x_j)$ and increase $c(L(\ell))$ by the total cost of the heads of all edges having x_j as their tail. This produces $c(L(d(x_{j+1})))$. If $c(L(d(x_{j+1}))) < c_{\min}(i)$, we set $c_{\min}(i) = c(L(d(x_{j+1})))$ and $\ell_i = d(x_{j+1})$.

Given values $\ell_1, \ldots, \ell_p$ and the edge set of T as sorted above, the set $S_1 = L(\ell_1) \cup \cdots \cup L(\ell_p)$ can be extracted as follows: We scan the list of values $\ell_1, \ldots, \ell_p$ and the edge set of T, again to simulate a sweep from $\ell = -\infty$ to $\ell = +\infty$. During the sweep we maintain the index i of the next level ℓ_i to be passed by the sweep; initially, $i = 1$. When the sweep passes the tail of an edge xy, its tail is at a depth less than ℓ_i. Thus, we add its head y to $L(\ell_i)$ if $d(y) \geq \ell_i$. When the sweep passes level ℓ_i, we increase i by one and, thus, start constructing the next level $L(\ell_{i+1})$. Since this computation of set S_1 requires sorting and scanning the edge set of T a constant number of times, its I/O-complexity is $\mathrm{O}(\mathrm{sort}(N))$.

6.2.3. Partitioning the layers. The final step of the algorithm extracts graphs $G_0, \ldots, G_p$, computes shortest-path trees $T_0, \ldots, T_p$ for these graphs, and partitions each graph G_i, $0 \leq i \leq p$, into subgraphs of weight at most $tw(G)$ using fundamental cycles w.r.t. T_i.

Computing the layers. To compute graphs $G_0, \ldots, G_p$, we first compute the set $V - S_1$ and sort the vertices in $V - S_1$ by their distances from s. We scan the vertices in $V - S_1$ and the values $-\infty = \ell_0, \ldots, \ell_{p+1} = r(T)$ to partition $V - S_1$ into sets $V_0, \ldots, V_p$, where $V_i = \{x \in V - S_1 : \ell_i < d(x) \leq \ell_{i+1}\}$. For $1 \leq i \leq p$, we add a new vertex s_i to V_i. This produces the vertex sets of graphs $G_0, \ldots, G_p$.

Next we partition E into sets $E_0, \ldots, E_p$, $E_1^-, \ldots, E_p^-$, and E^+ such that every edge in E_i has both endpoints in V_i; every edge xy in E_i^- has one endpoint, say y, in V_i, and the other endpoint, x, satisfies $x \in S_1$ and $d(x) < d(y)$; and set E^+ contains the remaining edges. This partition is easily computed in $\mathrm{O}(\mathrm{sort}(N))$ I/Os: We label every edge with the membership of its endpoints in $V_0, \ldots, V_p$ or S_1 and with their distances from s. Every edge can then determine its membership in one of the sets based on its local information, and we can sort E to obtain the desired partition. Graph G_i is now defined as $G_i = (V_i, E_i \cup E_i')$, where $E_i' = \{s_i y, y s_i : xy \in E_i^- \text{ and } d(x) < d(y)\}$. Finally, we triangulate G_i using the algorithm of [25].

This procedure requires sorting and scanning the vertex and edge sets of G a constant number of times. In addition, we invoke the $\mathrm{O}(\mathrm{sort}(N))$-I/O triangulation algorithm of [25] on graphs $G_0, \ldots, G_p$, whose total size is $\mathrm{O}(N)$. Hence, the construction of graphs $G_0, \ldots, G_p$ takes $\mathrm{O}(\mathrm{sort}(N))$ I/Os.

Computing shortest-path trees and their duals. Each shortest-path tree T_i can be computed in $\mathrm{O}(\mathrm{sort}(|G_i|))$ I/Os using the procedure described in section 6.2.1.

To construct T_i^*, we compute the dual $G_i^* = (F_i, E_i^*)$ of G_i, which can be done in $O(\text{sort}(|G_i|))$ I/Os [25]. Then we remove all edges dual to edges in T_i from E_i^*. This takes another $O(\text{sort}(|G_i|))$ I/Os (see section 2.4). Thus, in total, the construction of trees $T_0, \ldots, T_p$ and $T_0^*, \ldots, T_p^*$ takes $O(\text{sort}(N))$ I/Os.

Computing dual edge weights. Before computing an edge separator of T_i^* and the corresponding set of fundamental cycles, we have to assign weights $w^*(e^*)$ as defined in section 6.1.3 to the edges of T_i^*. We do this in two phases. First, we partition V_i into two sets V_i' and V_i'' such that every vertex in V_i' has an incident nontree edge, while all edges incident to a vertex in V_i'' are tree edges. While doing this, we also identify a nontree edge e incident to each vertex $x \in V_i'$ and add $w(x)$ to $w^*(e^*)$. In the second phase, we find a nontree edge e for every vertex $x \in V_i''$ such that both endpoints of e are neighbors of x in T_i; we add $w(x)$ to $w^*(e^*)$. The details follow.

To implement the first phase, we create a list Y_i of nontree edges of G_i. More precisely, Y_i contains directed edges xy and yx for every nontree edge xy of G_i. We sort the edges in Y_i by their tails and sort the vertices in V_i by their IDs. Now a single scan of lists V_i and Y_i suffices to identify all vertices x in V_i such that Y_i contains at least one edge with tail x. These are the vertices in V_i'; all other vertices belong to V_i''. During this scan, we also extract, for every vertex $x \in V_i'$, the first edge e with tail x from Y_i and add a pair $(e^*, w(x))$ to a list W. This list will be used after the second phase to compute the weights of the edges in T_i^*.

To implement the second phase, we observe that, for every vertex $x \in V_i''$ and every nontree edge yz such that y and z are both neighbors of x in T_i, one endpoint of yz, say y, must be a child of x in T_i, and the other, z, must be x's parent in T_i or another child of x. This implies in particular that, for every nontree edge yz, there exists at most one vertex $x \in V_i''$ such that y and z are both neighbors of x in T_i. Our goal, therefore, is to partition the nontree edges of G_i into sets $E(x)$ such that both endpoints of each edge in $E(x)$ are neighbors of x in T_i; for every vertex $x \in V_i''$, we then choose one edge from $E(x)$ and add the pair $(e^*, w(x))$ to W. To obtain the partition into sets $E(x)$, we first label every vertex $x \in T_i$ with its grandparent in T_i: We create a second copy P_i of V_i, sort the vertices in P_i by their IDs, and sort the vertices in V_i by the IDs of their parents. This ensures that the vertices in V_i are stored in the same order as their parents in P_i. Since each vertex in P_i also stores the ID of its parent, a single scan of V_i and P_i now suffices to label every vertex in V_i with the ID of its grandparent. Now we label every nontree edge of G_i with the parents and grandparents of its endpoints. A nontree edge yz belongs to $E(x)$ if and only if x is the parent of both y and z or, w.l.o.g., x is the parent of y and z is the grandparent of y. This can now be tested based on the local information stored with edge yz. If edge yz satisfies this condition, we label it as belonging to $E(x)$. We sort the nontree edges by their membership in sets $E(x)$ and scan V_i'' and the sorted edge list to add a pair $(e^*, w(x))$ to W for every vertex $x \in V_i''$ and the first edge $e \in E(x)$.

To finish the computation of the weights of the edges in T_i^*, we sort the edges of T_i^* by their IDs and the pairs in W by their first components. A single scan of these two sorted lists now suffices to add $w(x)$ to $w^*(e^*)$, for every pair $(e^*, w(x)) \in W$.

Since this procedure sorts and scans lists of size $O(|G_i|)$ a constant number of times, the assignment of weights to the edges of T_i^* takes $O(\text{sort}(|G_i|))$ I/Os. The total cost for all graphs $G_0, \ldots, G_p$ is therefore $O(\text{sort}(N))$.

Computing the edge separator and fundamental cycles. To obtain the edge separator X_i of T_i^*, we root T_i^* in an arbitrary leaf, compute a preorder numbering of T_i^* w.r.t. the chosen root, and direct all edges in T_i^* from children to parents. This

can be done using the Euler tour technique and list ranking [14]. The construction of the edge separator X_i as described in section 6.1.3 can now be implemented using the time-forward processing technique of [14]. Given the edge separator X_i of T_i^* produced by this procedure, we mark the endpoints of all edges $e \in G_i$ such that $e^* \in X_i$ and then process T_i from the bottom up (using time-forward processing again) to identify all vertices of G_i that belong to the fundamental cycles defined by the edges in X_i. We add these vertices to S_2.

This procedure applies the Euler tour technique, list ranking, and time-forward processing to T_i and T_i^*, both of which have size $O(|G_i|)$. Hence, this takes $O(\text{sort}(|G_i|))$ I/Os. Apart from this, we sort and scan lists of size $O(|G_i|)$. Thus, partitioning each graph G_i takes $O(\text{sort}(|G_i|))$ I/Os, and the computation of the whole separator S_2 takes $O(\text{sort}(N))$ I/Os.

6.2.4. Final remarks. Since we have shown that the computation of both S_1 and S_2 takes $O(\text{sort}(N))$ I/Os, Theorem 6.2 is proved. Since the algorithm relies on the separator algorithm of section 4 and the shortest-path algorithm of [8], it inherits their memory requirements. In particular, the latter requires a proper $\Theta\big(B^2\big)$-partition as part of the input and uses $\Theta\big(B^2\big)$ main memory to carry out its computation. The algorithm from section 4 can be used to produce the desired partition in $O(\text{sort}(N))$ I/Os, provided that $M = \Omega\big(B^2 \log^2 B\big)$.

As a final comment, note that the shortest-path algorithm of [8] relies on a *regular* proper $\Theta\big(B^2\big)$-partition, which is guaranteed to exist only if the graph has bounded degree. The triangulations in which we need to compute shortest paths may not satisfy this constraint, but given an embedding, each planar graph G can be transformed into a planar graph G' of size $O(|G|)$ such that every vertex in G' has degree at most three. This is done by replacing every vertex x of degree $\deg(x) > 3$ with a cycle of $\deg(x)$ vertices and making every edge incident to x incident to a different vertex on this cycle. This takes $O(\text{sort}(|G|))$ I/Os. Moreover, if the edges in each cycle replacing a high-degree vertex are given weight 0, this transformation preserves the distances between vertices. Thus, the algorithm of [8] can be used to compute shortest paths in G and in the layers $G_0, \ldots, G_p$.

6.3. Edge separators. The final result of this section is an I/O-efficient edge separator algorithm. Aleksandrov et al. [3] showed that Theorem 6.1 can also be used to compute optimal edge separators of planar graphs as follows: Define the cost of each vertex to be equal to its degree. Then compute a vertex separator S of cost at most $4\sqrt{2(\sum_{x \in V}(\deg(x))^2)/t}$ and add all edges incident to a vertex in S to the edge separator.

It is easy to verify that the computation of the vertex costs and the extraction of the edge separator from the computed vertex separator can be carried out in $O(\text{sort}(N))$ I/Os. Hence, the following result is an immediate consequence of Theorem 6.2.

THEOREM 6.3. *Let $G = (V, E)$ be a planar graph, let $0 < t < 1$ be a real number, and let $w : V \to \mathbb{R}^+$ be a weight function so that $w(x) \leq tw(G)$ for all $x \in V$. Then there exists a set S of at most $4\sqrt{2(\sum_{v \in V}(\deg(v))^2)/t}$ edges so that no connected component of $G - S$ has weight exceeding $tw(G)$. Such an edge separator S can be found in $O(\text{sort}(N))$ I/Os, provided that $M = \Omega\big(B^2 \log^2 B\big)$.*

7. Improving the memory requirements. In this final section of the paper, we show how to reduce the memory requirements of our algorithm from section 4 to $M \geq \max\big(196B^2, 7r\big)$. The resulting algorithm also produces a separator significantly

smaller than the one produced by the algorithm in section 4. However, these two improvements come at the expense of increasing the internal-memory computation of the algorithm from $O(N \log N)$ to $O(N \log N + NB)$.

Recall the reason why $M \geq 56r \log^2 B$ is required for the algorithm in section 4: If we choose to compute an r'-partition of each graph G_i in the graph hierarchy, then the separator vertices introduced at each level correspond to $O\big(N/\sqrt{r'}\big)$ vertices in G; no better upper bound is known. Since there are $\lfloor \log B \rfloor$ levels in the hierarchy, and we want a separator of size $O(N/\sqrt{r})$, we have to ensure that $O\big(\big(N/\sqrt{r'}\big) \log B\big) = O(N/\sqrt{r})$, which we achieve by choosing $r' = r \log^2 B$. This now forces us to use $56r \log^2 B$ main memory because, as argued in section 4, every piece of $G_i - S'_i$ has size at most $56r' = 56r \log^2 B$, and we need to load each such piece into memory to partition it into smaller pieces.

So the central problem is that, if we were to choose $r' = r$, then every level in the hierarchy adds $O(N/\sqrt{r})$ vertices to the final separator of G, that is, we would obtain a separator that is too big by a factor of $\log B$. Next we explain how to avoid this problem by using a recursive bootstrapping approach.

The centerpiece of the algorithm is the separator algorithm from section 6, but now using vertex costs and weights equal to 1. This algorithm takes $O(\mathrm{sort}(N))$ I/Os using only $\Theta(B)$ main memory if we ignore the costs and memory requirements of computing an embedding of G, computing the shortest-path tree T of G, and computing the shortest-path trees $T_0, \ldots, T_p$ for layers $G_0, \ldots, G_p$. Given appropriate separator decompositions, the computation of the embedding and the shortest-path computations take $O(\mathrm{sort}(N))$ I/Os and require $\Theta\big(B^2\big)$ main memory [5, 33]. Our strategy is to obtain these separator decompositions by recursive application of our algorithm.

Embedding G. To compute a planar embedding of G, we require a proper $\Theta\big(B^2\big)$-partition $\mathcal{P} = (S, \{G_1, \ldots, G_q\})$ of G. We obtain this partition as follows: First, we apply the uniform graph contraction procedure to G and recursively compute a proper B^2-partition $\tilde{\mathcal{P}} = \big(\tilde{S}, \big\{\tilde{G}_1, \ldots, \tilde{G}_q\big\}\big)$ of the resulting graph $\tilde{G}$. Then we choose S to be the set of vertices in G represented by the vertices in $\tilde{S}$, and each graph G_i in $\mathcal{P}$ to be the subgraph of G represented by the vertices in $\tilde{G}_i$. To bound the number of vertices in G represented by each vertex in $\tilde{G}$, we assign weight 1 to every vertex in G and provide a weight threshold u, to be specified later, to the uniform graph contraction procedure. This guarantees that $|S| \leq u\big|\tilde{S}\big| = O(N/B)$, $|G_i| \leq u\big|\tilde{G}_i\big| = O\big(B^2\big)$, and $|\mathcal{N}(G_i)| = O(B)$ for all $1 \leq i \leq q$. It also ensures that $\sum_{i=1}^{q} |\mathcal{N}(G_i)| \leq u \sum_{i=1}^{q} \big|\mathcal{N}\big(\tilde{G}_i\big)\big| = O(N/B)$. Thus, $\mathcal{P}$ is a proper $\Theta\big(B^2\big)$-partition of G. An embedding of G can now be obtained from $\mathcal{P}$ in $O(\mathrm{sort}(N))$ I/Os [33]. In total, the cost of computing a planar embedding of G is $O(\mathrm{sort}(N))$ I/Os plus the cost of the recursive call on $\tilde{G}$.

Computing $T, T_0, \ldots, T_p$. Given a planar embedding of G, we use the procedure from section 6.2.4 to transform G into a planar graph G' of degree at most three and so that the distances between vertices in G are the same as the distances between their representatives in G'. This takes $O(\mathrm{sort}(N))$ I/Os. Now we apply the procedure from the previous paragraph to obtain a proper $\Theta\big(B^2\big)$-partition of G' and then use the procedures from sections 4.3 and 5 to augment this partition to obtain a regular proper B^2-partition of G'. The construction of G' takes $O(\mathrm{sort}(N))$ I/Os and, as discussed in the previous paragraph, the cost of computing a proper $\Theta(B^2)$-partition of G' is $O(\mathrm{sort}(|G'|))$ plus the cost of the recursive call on a compressed version $\tilde{G}'$ of G'. As discussed in sections 4.3 and 5, augmenting the computed partition to a regular proper

$\Theta(B^2)$-partition also takes $\mathrm{O}(\mathrm{sort}(|G'|))$ I/Os. Given such a partition, we use the single-source shortest-path algorithm of [8] to obtain a shortest-path tree T' of G' in $\mathrm{O}(\mathrm{sort}(|G'|))$ I/Os, which is easily transformed into a shortest-path tree T of G in the same number of I/Os. Thus, the total cost of computing T is $\mathrm{O}(\mathrm{sort}(N) + \mathrm{sort}(|G'|))$ plus the cost of the recursive call on $\tilde{G}'$.

The shortest-path trees $T_0, \ldots, T_p$ are obtained by applying the same procedure to graphs $G_0, \ldots, G_p$. We denote the degree-3 graphs obtained from $G_0, \ldots, G_p$ by $G'_0, \ldots, G'_p$ and the compressed versions of $G'_0, \ldots, G'_p$ for which we recursively compute proper B^2-partitions by $\tilde{G}'_0, \ldots, G'_p$. Using this notation, our discussion above implies that the computation of each tree T_i takes $\mathrm{O}(\mathrm{sort}(|G_i|) + \mathrm{sort}(|G'_i|))$ I/Os plus the cost of a recursive call on $\tilde{G}'_i$.

Analysis. Summing up the costs of the individual steps of our algorithm, the cost of computing a proper B^2-partition of G is

$$T(N) = \mathrm{O}(\mathrm{sort}(N)) + \mathrm{O}\big(\mathrm{sort}\big(\big|G'\big|\big)\big) + T\big(\big|\tilde{G}\big|\big) + T\big(\big|\tilde{G}'\big|\big) + \sum_{i=0}^{q} \Big(\mathrm{sort}(|G_i|) + \mathrm{sort}\big(\big|G'_i\big|\big) + T\big(\big|\tilde{G}'_i\big|\big)\Big).$$

The first $\mathrm{O}(\mathrm{sort}(N))$ term includes the cost of computing the separator once the trees $T, T_0, \ldots, T_p$ have been computed.

To bound this recurrence by $\mathrm{O}(\mathrm{sort}(N))$, we first bound the sizes of the different graphs involved in the computation as follows: Graph G has N vertices. Graph G' has at most $6N$ vertices, at most two per edge in G. Graphs $G_0, \ldots, G_p$ contain at most N vertices, as they are vertex-disjoint subgraphs of G. Thus, graphs $G'_0, \ldots, G'_p$ also contain at most $6N$ vertices. The total size of graphs $G, G', G_0, \ldots, G_p, G'_0, \ldots, G'_p$ is therefore $\mathrm{O}(N)$, which implies that the sort$(\cdot)$-terms in the above recurrence sum to $\mathrm{O}(\mathrm{sort}(N))$, simplifying the recurrence to

$$T(N) = \mathrm{O}(\mathrm{sort}(N)) + T\big(\big|\tilde{G}\big|\big) + T\big(\big|\tilde{G}'\big|\big) + \sum_{i=0}^{p} T\big(\big|\tilde{G}'_i\big|\big).$$

Now, if we choose $u = 196$, graph $\tilde{G}$ contains at most $|G|/98$ heavy vertices and, by Theorem 3.1, has size at most $|G|/14$. Similarly, $\big|\tilde{G}'\big| \le |G'|/14$ and, for $0 \le i \le p$, $\big|\tilde{G}'_i\big| \le |G'_i|/14$. Thus, we have

$$\begin{aligned}\big|\tilde{G}\big| + \big|\tilde{G}'\big| + \sum_{i=0}^{p}\big|\tilde{G}'_i\big| &\le \frac{|G|}{14} + \frac{|G'|}{14} + \sum_{i=0}^{p}\frac{|G'_i|}{14} \\ &\le \frac{N}{14} + \frac{6N}{14} + \frac{6N}{14} \\ &= \frac{13N}{14},\end{aligned}$$

and the recurrence solves to $T(N) = \mathrm{O}(\mathrm{sort}(N))$. The memory requirements of the embedding and shortest-path algorithms are at most $uB^2 = 196B^2$, as we provide them with uB^2-partitions. This proves that we can compute a t-vertex separator and, thus, a t-edge separator, for any $0 < t < 1$, in $\mathrm{O}(\mathrm{sort}(N))$ I/Os, provided that $M \ge 196B^2$. In order to obtain a (regular) proper r-partition from an (r/N)-separator, we still have to be able to load subgraphs of size at most $7r$ into internal memory. This leads to the following result.

THEOREM 7.1. *The partitions in Theorems* 4.1 *and* 5.1 *can be computed in* $O(\mathrm{sort}(N))$ *I/Os, provided that* $M \geq \max\bigl(196B^2, 7r\bigr)$. *The separators in Theorems* 6.2 *and* 6.3 *can be computed in* $O(\mathrm{sort}(N))$ *I/Os, provided that* $M \geq 196B^2$.

The memory requirements in Theorem 7.1 can be reduced further to $M \geq \max\bigl(cB^2, 7r\bigr)$ and $M \geq cB^2$, for an arbitrarily small constant $c > 0$. Indeed, $196B^2$ memory is required because we recursively compute B^2-partitions of the compressed graphs, which correspond to $\bigl(196B^2\bigr)$-partitions in the uncompressed graphs. Instead, we can compute $\bigl(cB^2/196\bigr)$-partitions of the compressed graphs, thereby obtaining $\bigl(cB^2\bigr)$-partitions of the uncompressed graphs. This affects the sizes of the produced separators—and, thus, the performance of the embedding and shortest-path algorithms—by only a constant factor but reduces the memory requirements.

8. Conclusions. In this paper, we have demonstrated that different types of separator decompositions of planar graphs can be computed I/O-efficiently. Using these partitions, a wide variety of fundamental problems on planar graphs can be solved I/O-efficiently.

A number of open questions remain, however. The constant factors in our algorithms—in terms of the size of the produced separator, the memory requirements, and the efficiency—are big. In order for the algorithms to be of practical value, these constant factors have to be reduced. Furthermore, even though the individual steps of the algorithm are fairly simple, the algorithm consists of too many of them. This makes the algorithm tedious to implement and impacts the efficiency of the algorithm; for example, only a small number of sorting steps is affordable in practice. From a practical point of view, it would therefore be desirable to have a simpler—even possibly a theoretically suboptimal—algorithm for computing separators I/O-efficiently. On the theoretical side, the most important open questions are whether separators can be computed in $O(\mathrm{sort}(N))$ I/Os using $o\bigl(B^2\bigr)$ main memory and whether they can be computed in $O(\mathrm{sort}(N))$ I/Os cache-obliviously. See [20] for a discussion of cache-obliviousness.

Acknowledgments. We would like to thank Richard Cole and the anonymous referees for helpful comments on how to improve the presentation of the results in this paper.

REFERENCES

[1] A. AGGARWAL AND J. S. VITTER, *The input/output complexity of sorting and related problems*, Comm. ACM, (1988), pp. 1116–1127.

[2] L. ALEKSANDROV AND H. DJIDJEV, *Linear algorithms for partitioning embedded graphs of bounded genus*, SIAM J. Discrete Math., 9 (1996), pp. 129–150.

[3] L. ALEKSANDROV, H. DJIDJEV, H. GUO, AND A. MAHESHWARI, *Partitioning planar graphs with costs and weights*, in Proceedings of the 4th Workshop on Algorithm Engineering and Experiments, Lecture Notes in Comput. Sci. 2409, Springer-Verlag, Berlin, New York, 2002, pp. 98–107.

[4] L. ARGE, *The buffer tree: A technique for designing batched external data structures*, Algorithmica, 37 (2003), pp. 1–24.

[5] L. ARGE, G. S. BRODAL, AND L. TOMA, *On external-memory MST, SSSP and multi-way planar graph separation*, J. Algorithms, 53 (2004), pp. 186–206.

[6] L. ARGE, U. MEYER, L. TOMA, AND N. ZEH, *On external-memory planar depth first search*, J. Graph Algorithms Appl., 7 (2003), pp. 105–129.

[7] L. ARGE AND L. TOMA, *Simplified external memory algorithms for planar DAGs*, in Proceedings of the 9th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 3111, Springer-Verlag, Berlin, New York, 2004, pp. 493–503.

[8] L. Arge, L. Toma, and N. Zeh, *I/O-efficient algorithms for planar digraphs*, in Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, 2003, pp. 85–93.
[9] L. Arge and N. Zeh, *I/O-efficient strong connectivity and depth-first search for directed planar graphs*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 2003, pp. 261–270.
[10] K. Booth and G. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.
[11] J. Boyer and W. Myrvold, *Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 140–146.
[12] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, *Graph structure in the web*, Computer Networks, 33 (2000), pp. 309–320.
[13] A. L. Buchsbaum and J. R. Westbrook, *Maintaining hierarchical graph views*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 566–575.
[14] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, *External-memory graph algorithms*, in Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1995, pp. 139–149.
[15] K. Diks, H. N. Djidjev, O. Sykora, and I. Vrto, *Edge separators of planar and outerplanar graphs with applications*, J. Algorithms, 14 (1993), pp. 258–279.
[16] H. N. Djidjev, *Partitioning graphs with costs and weights on vertices: Algorithms and applications*, Algorithmica, 28 (2000), pp. 51–75.
[17] H. N. Djidjev and J. R. Gilbert, *Separators in graphs with negative and multiple vertex weights*, Algorithmica, 23 (1999), pp. 57–71.
[18] S. Even and R. E. Tarjan, *Computing an st-numbering*, Theoret. Comput. Sci., 2 (1976), pp. 339–344.
[19] G. N. Frederickson, *Fast algorithms for shortest paths in planar graphs, with applications*, SIAM J. Comput., 16 (1987), pp. 1004–1022.
[20] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, *Cache-oblivious algorithms*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 1999, pp. 285–297.
[21] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan, *A separator theorem for graphs of bounded genus*, J. Algorithms, 5 (1984), pp. 391–407.
[22] M. T. Goodrich, *Planar separators and parallel polygon triangulation*, J. Comput. System Sci., 51 (1995), pp. 374–389.
[23] F. Harary, *Graph Theory*, Addison-Wesley, New York, 1969.
[24] J. Hopcroft and R. E. Tarjan, *Efficient planarity testing*, J. ACM, 21 (1974), pp. 549–568.
[25] D. Hutchinson, A. Maheshwari, and N. Zeh, *An external memory data structure for shortest path queries*, Discrete Appl. Math., 126 (2003), pp. 55–82.
[26] A. Lempel, S. Even, and I. Cederbaum, *An algorithm for planarity testing of graphs*, in Theory of Graphs: International Symposium (Rome, 1966), Gordon and Breach, New York, 1967, pp. 215–232.
[27] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.
[28] U. Meyer, P. Sanders, and J. Sibeyn, eds., *Algorithms for Memory Hierarchies: Advanced Lectures*, Lecture Notes in Comput. Sci. 2625, Springer-Verlag, Berlin, New York, 2003.
[29] G. L. Miller, *Finding small simple cycle separators for 2-connected planar graphs*, J. Comput. System Sci., 32 (1986), pp. 265–279.
[30] K. Munagala and A. Ranade, *I/O-complexity of graph algorithms*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 687–694.
[31] W. T. Tutte, *Graph Theory*, Cambridge University Press, Cambridge, UK, 2001.
[32] J. S. Vitter, *External memory algorithms and data structures: Dealing with massive data*, ACM Comput. Surveys, 33 (2001), pp. 209–271.
[33] N. Zeh, *I/O-Efficient Algorithms for Shortest Path Related Problems*, Ph.D. thesis, School of Computer Science, Carleton University, Ottawa, ON, Canada, 2002.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 802–824

APPROXIMATE SHORTEST PATHS IN ANISOTROPIC REGIONS*

SIU-WING CHENG†, HYEON-SUK NA‡, ANTOINE VIGNERON§, AND YAJUN WANG†

Abstract. Our goal is to find an approximate shortest path for a point robot moving in a planar subdivision with n vertices. Let $\rho \geqslant 1$ be a real number. Distances in each face of this subdivision are measured by a convex distance function whose unit disk is contained in a concentric unit Euclidean disk and contains a concentric Euclidean disk with radius $1/\rho$. Different convex distance functions may be used for different faces, and obstacles are allowed. These convex distance functions may be asymmetric. For any $\varepsilon \in (0,1)$ and for any two points v_s and v_d, we give an algorithm that finds a path from v_s to v_d whose cost is at most $(1+\varepsilon)$ times the optimal. Our algorithm runs in $O(\frac{\rho^2 \log \rho}{\varepsilon^2} n^3 \log(\frac{\rho n}{\varepsilon}))$ time. This bound does not depend on any other parameters; in particular it does not depend on the minimum angle in the subdivision. We give applications to two special cases that have been considered before: the weighted region problem and motion planning in the presence of uniform flows. For the weighted region problem with weights in $[1,\rho] \cup \{\infty\}$, the time bound of our algorithm improves to $O(\frac{\rho \log \rho}{\varepsilon} n^3 \log(\frac{\rho n}{\varepsilon}))$.

Key words. computational geometry, approximation algorithm, shortest path, weighted region, convex distance function

AMS subject classifications. 68U05, 68W25

DOI. 10.1137/06067777X

1. Introduction.

The problem of computing a shortest path between two points arises naturally in geographic information systems, VLSI design, logistics, and motion planning. Another area of application for shortest path algorithms is computer graphics, where the geometric properties of shortest paths along a surface can be exploited for mesh cutting and editing. (See the article by V. Surazhsky et al. [20].) The path lies in a geometric environment in all of these applications. This environment is usually represented by a polygonal (or polyhedral) subdivision. Different metrics may be used in different regions of the subdivision in order to model friction, wind, steepness, or any other mechanical constraint.

Due to these applications and to the variety of possible geometric environments and metrics, algorithms for geometric shortest path problems have been extensively studied. We mention the most relevant work here and refer the interested reader to the survey by Mitchell [12] for more details.

In the weighted region problem [13], a point robot moves within a planar subdivision $\mathcal{T}$, each face f of $\mathcal{T}$ being associated with a weight $w_f > 0$. The cost of a path within a face f is the length of this path multiplied by w_f. Assume that the faces of the subdivision $\mathcal{T}$ are all triangular. (We can make this assumption as any planar subdivision can be triangulated by introducing a linear number of edges.) We denote

*Received by the editors December 15, 2006; accepted for publication (in revised form) January 25, 2008; published electronically May 28, 2008.
http://www.siam.org/journals/sicomp/38-3/67777.html

†Department of Computer Science and Engineering, HKUST, Hong Kong (scheng@cse.ust.hk, yalding@cse.ust.hk).

‡School of Computing, Soongsil University, Seoul, Korea (hsnaa@computing.ssu.ac.kr). This author's research was supported by the KRF funded by the Korean Government (R04-2004-000-10004-0).

§Département de Mathématiques et Informatique Appliquées, INRA, UR341, Domaine de Vilvert, 78352 Jouy-en-Josas cedex, France (antoine.vigneron@jouy.inra.fr). This author's research was partially supported by a Marie Curie international reintegration grant.

by n the number of vertices of $\mathcal{T}$. The first approximation scheme for the weighted region problem was given by Mitchell and Papadimitriou [13]. It runs in $O(n^8 L)$ time, where L represents the maximum number of bits of the input numbers (such as the integer coordinates of the vertices of the subdivision, and the weights). This algorithm is a continuous version of Dijkstra's algorithm for finding shortest paths in a graph. Other algorithms for the weighted region problem discretize the search space by placing Steiner points and by finding a shortest path in a graph whose nodes are Steiner points or input vertices and whose edges are line segments. In particular, Aleksandrov, Maheshwari, and Sack [2] and Sun and Reif [19] gave algorithms that have linear dependency in n. However, their time bounds depend on the minimum angle in $\mathcal{T}$ (and the weights, too, in the algorithm by Aleksandrov, Maheshwari, and Sack [2]).

The main limitation of the weighted region model is that it models only situations where the metrics are isotropic. It cannot account for the effect of wind, current, or any other force field that favors some directions of travel. A more general model was introduced by Reif and Sun for motion planning in the presence of uniform flows [16]. In each face f of $\mathcal{T}$, the velocity of the robot is the sum of a flow $\vec{v}_f$ and a control velocity chosen by the robot. It allows one to model friction and a uniform flow within each region. Reif and Sun [16] showed that this problem is PSPACE hard in three dimensions and gave an FPTAS for the two-dimensional (2D) case. (Some geometric parameters were treated as constant, such as the minimum angle in $\mathcal{T}$.)

As pointed out by Aleksandrov, Maheshwari, and Sack [2], one challenge is to remove the dependence on parameters other than n and ε in the running time. It is also desirable to handle more general types of metrics in order to model a larger class of problems that arise in applications. (See the article by Sellen [17] on the direction-weighted problem, where the cost is proportional to a continuous function of the direction.) We make progress in both aspects in this paper.

1.1. Our results. We consider a generalization of Reif and Sun's model for motion planning in the presence of uniform flows [16]. A point robot moves within a planar subdivision from a source point v_s to a target point v_d. The planar subdivision $\mathcal{T}$ may have holes in order to model obstacles. The distance within each face f of the subdivision is measured according to a possibly asymmetric convex distance function [4]. (See section 2.2 for a definition of convex distance functions.) Different convex distance functions may be used for different faces. The cost of a path is measured according to these distance functions. (See section 2 for the case of a polygonal path and section 6 for the case of a rectifiable path.) Let B_f denote the unit "disk" of the distance function of a face f. We assume that B_f is contained in a concentric unit Euclidean disk and that B_f contains a concentric Euclidean disk with radius $1/\rho$. In other words, there exists $\rho \geqslant 1$ such that the speed of the robot in any direction and within any face of $\mathcal{T}$ is in the interval $[1/\rho, 1]$. The weighted region problem and the problem of path planning in the presence of uniform flows are special cases in our model.

We assume that the distance between two points under any of the convex distance functions can be computed in $O(1)$ time. Our model of computation is the standard real-RAM model [15] in which the operations $(+, -, \times, /)$ can be performed in constant time.

Our main results include an algorithm that computes in time $O(\frac{\rho^2 \log \rho}{\varepsilon^2} n^3 \log(\frac{\rho n}{\varepsilon}))$ a polygonal path whose cost is at most $(1 + \varepsilon)$ times the optimal. This is the first algorithm that can handle general convex distance functions. For the weighted region problem, the time bound improves to $O(\frac{\rho \log \rho}{\varepsilon} n^3 \log(\frac{n\rho}{\varepsilon}))$. Our time bounds have the

nice feature that they do not depend on the geometry of $\mathcal{T}$. It is not obvious that an optimal path exists in our model. Indeed, we give an example in which no polygonal path is optimal. Nevertheless, using the theory of length spaces [3, 10], we can prove that there exists an optimal *rectifiable* path. (A rectifiable path is a path with finite Euclidean length.) Furthermore, we show that there exists a $(1+\varepsilon)$-approximate shortest path that is polygonal and has $O(\rho n^2/\varepsilon)$ links. This is instrumental to establishing the correctness and the complexity of our algorithm.

Our approach is the following. We define a k-link path to be a polygonal path with at most k edges, each edge being contained in a face of $\mathcal{T}$ and having its endpoints on the boundary of this face. (Our definition is different from the one given by Daescu et al. [5].) We first give approximation algorithms that return a polygonal path with cost at most $(1+\varepsilon)$ times the cost of any k-link path (see section 3). In the weighted region problem, Mitchell and Papadimitriou [13] showed that there exists a shortest path that is a k-link path with $k = \Theta(n^2)$, so we apply our algorithm with $k = \Theta(n^2)$ and obtain the result stated above. In the general case, we show that for any polygonal path P, there exists a $(21\rho n^2/\varepsilon)$-link path with cost at most $(1+\varepsilon)$ times the cost of P (see sections 4 and 5). Thus, by choosing $k = 21\rho n^2/\varepsilon$, our algorithm returns a path with cost at most $(1+\varepsilon)$ times the cost of any polygonal path. Then, in section 6, we prove that there exists an optimal rectifiable path and show that there exists a polygonal path with cost arbitrarily close to the optimal. It follows that our algorithm returns a $(1+\varepsilon)$-approximate shortest path within the class of rectifiable paths.

1.2. Comparison with previous work. A direct comparison cannot be made with any previous result because our algorithm is the first that can handle general convex distance functions. Anisotropic shortest path problems have been studied on terrains [11, 18] in a special case that models some common mechanical constraints on a mobile robot. This model cannot account for the presence of flows (even uniform flows), so it is not more general than ours. On the other hand, our algorithm applies to planar subdivisions, not terrains. The problems of navigating through weighted regions and in the presence of uniform flows are special cases in our model. So we compare them with the previous results for these problems.

Weighted region problem. In this case, ρ is equal to the ratio of the maximum weight to the minimum weight. Other than the dependence on n and ε, there is a factor of $\Omega(1/\theta_{\min})$ in the worst-case running times of the algorithms by Aleksandrov, Maheshwari, and Sack [1, 2] and Sun and Reif [19], where $\theta_{\min}$ is the minimum angle in $\mathcal{T}$. The worst-case running time of the algorithms by Aleksandrov, Maheshwari, and Sack [1, 2] depend on ρ, too. Our running time is independent of the geometry of $\mathcal{T}$, although it has a higher dependence on n, ε, and ρ. The algorithm of Mitchell and Papadimitriou [13] has a running time of $O(n^8 \log \frac{nN\rho}{\varepsilon})$, where the input vertices have integer coordinates in $[0, N]$. In comparison, our algorithm works in the standard real RAM model, and our algorithm has a smaller dependence on n but a higher dependence on ε and ρ.

Movement in the presence of uniform flows. Assume that there is a uniform flow of velocity $\vec{v}_f$ in each face f of $\mathcal{T}$. The robot can apply a control velocity $\vec{v}_r$ in any direction such that $\|\vec{v}_r\|$ is at most some constant c_f. For each face f, we define a convex distance function whose unit "disk" B_f is a Euclidean disk with radius c_f and centered at $O + \vec{v}_f$, where O is the origin. Thus, B_f is contained in a disk with radius $c_f + \|\vec{v}_f\|$ and centered at O, and B_f contains a disk with radius $c_f - \|\vec{v}_f\|$ and centered at O. Let $v_{\min} = \min\{c_f - \|\vec{v}_f\| : f \in \mathcal{T}\}$. Let $v_{\max} = \max\{c_f + \|\vec{v}_f\| : f \in \mathcal{T}\}$. Assuming that $v_{\min} > 0$, we can model the problem with $\rho = v_{\max}/v_{\min}$ and find an

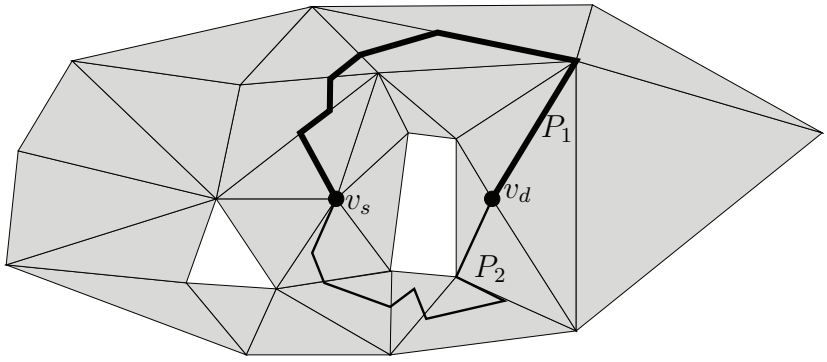


FIG. 1. *The underlying space $|\mathcal{T}|$ is shaded. The path P_1 is a 7-link path, and thus it is $\mathcal{T}$-respecting. The path P_2 is a $\mathcal{T}$-respecting path with 8 links, but it is not an 8-link path.*

approximate shortest path in time $O(\frac{\rho^2 \log \rho}{\varepsilon^2} n^3 \log(\frac{\rho n}{\varepsilon}))$. An algorithm by Reif and Sun runs in $O\left(\frac{nC_{\text{skew}}}{\varepsilon}(\log \frac{C_{\text{skew}}}{\varepsilon})(\log \frac{C_{\text{skew}}}{\varepsilon} + \log n)\right)$ time [16], where C_{skew} is defined as follows. Let $\lambda = \max\{c_f/c_{f'} : \text{adjacent faces } f \text{ and } f'\}$. Let $\rho_{\min} = \min\{c_f/\|\vec{v}_f\| : f \in \mathcal{T}\}$. Let $\theta_{\min}$ be the minimum angle in $\mathcal{T}$. Then $C_{\text{skew}} = \Theta(\frac{\lambda(\rho_{\min}+1)}{\theta_{\min}(\rho_{\min}-1)})$. Reif and Sun's algorithm requires $\rho_{\min} > 1$, which is equivalent to our condition of $v_{\min} > 0$. Our running time does not depend on $\theta_{\min}$, but Reif and Sun's running time has a better dependence on n, ε, and ρ.

2. Notation and preliminaries.

2.1. Environment. We model the environment $\mathcal{T}$ in which the point robot can move by a *simplicial complex* [6]: it is a collection of triangles such that any two triangles can intersect only along a common edge or vertex or not at all. A *face* of $\mathcal{T}$ is a triangle in $\mathcal{T}$. We consider faces to be closed subsets of $\mathbb{R}^2$. A *vertex* (resp., an *edge*) of $\mathcal{T}$ is a vertex (resp., an edge) of some face of $\mathcal{T}$. (We do not allow dangling edges or vertices; that is, all edges and vertices must be edges or vertices of a face of $\mathcal{T}$.) The *underlying space* $|\mathcal{T}| \subset \mathbb{R}^2$ of $\mathcal{T}$ is the union of its faces (see Figure 1). We assume that $|\mathcal{T}|$ is connected, but we allow $|\mathcal{T}|$ to have holes. The point robot is free to move inside $|\mathcal{T}|$, but it cannot move outside $|\mathcal{T}|$. We use n to denote the number of vertices in $\mathcal{T}$, and we assume that n is finite.

For any two points $p, q \in \mathbb{R}^2$, we denote by $\overline{pq}$ the closed, oriented line segment from p to q. In particular, when $p \neq q$, we have $\overline{pq} \neq \overline{qp}$. We denote by $\|pq\|$ the Euclidean distance between p and q. For any two points $p, q \in |\mathcal{T}|$, the *geodesic distance* between p and q is the Euclidean length of the shortest polyline in $|\mathcal{T}|$ with endpoints p and q. We use $\|pq\|_{\mathcal{T}}$ to denote this geodesic distance.

For each face f of $\mathcal{T}$, we denote by $\text{int}(f)$ (resp., $\text{bd}(f)$) the interior (resp., boundary) of f according to the usual topology of $\mathbb{R}^2$. For a segment $\overline{pq}$, we use $\text{int}(\overline{pq})$ to denote the open line segment from p to q. When X is a polyline with endpoints p, q, we use $\text{int}(X)$ to denote $X \setminus \{p, q\}$. A *chord* of a face f of $\mathcal{T}$ is an oriented line segment $\overline{pq}$ such that $\text{int}(\overline{pq}) \subset \text{int}(f)$ and $\{p, q\} \subset \text{bd}(f)$. If $\overline{pq} \subset \text{bd}(f)$, we say that $\overline{pq}$ is a *boundary segment*.

2.2. Convex distance functions. Each face f of $\mathcal{T}$ is associated with a compact convex set $B_f \subset \mathbb{R}^2$ that contains the origin O in its interior. The *convex distance*

function associated with f is defined by

$$\forall\, x, y \in f,\ d_f(x,y) = \min\{\lambda \in [0,+\infty) : y \in x + \lambda B_f\}.$$

This type of distance function has been studied before [4, 7] in the context of Voronoi diagrams. A convex distance function is not necessarily a metric as it may be asymmetric. Still, d_f satisfies the triangle inequality, and the shortest path from p to q is the oriented line segment $\overline{pq}$.

If $\mathrm{int}(\overline{pq}) \subset \mathrm{int}(f)$ for some face f, the cost of $\overline{pq}$ is defined as $\mathrm{cost}(\overline{pq}) = d_f(p,q)$. If $\overline{pq}$ is contained in an edge e of $\mathcal{T}$ that is adjacent to exactly one face f, we also define $\mathrm{cost}(\overline{pq})$ to be $d_f(p,q)$. On the other hand, if e is adjacent to two faces f_1 and f_2, we define $\mathrm{cost}(\overline{pq})$ to be $\min(d_{f_1}(p,q), d_{f_2}(p,q))$.

We assume that there exists $\rho \geqslant 1$ such that, for any face f, the set B_f contains a Euclidean disk with radius $1/\rho$ centered at the origin, and B_f is contained in the unit Euclidean disk centered at the origin. Intuitively, it means that the speed allowed in any direction is always in the interval $[1/\rho, 1]$. It implies that, for any face f and for any points $p, q \in f$, we have $\|pq\| \leqslant \mathrm{cost}(\overline{pq}) \leqslant \rho\,\|pq\|$. Another useful consequence is that for any two chords $\overline{ps}$ and $\overline{qr}$ of the same face f, we have

$$\begin{aligned} \mathrm{cost}(\overline{ps}) = d_f(p,s) &\leqslant d_f(p,q) + d_f(q,r) + d_f(r,s) \\ &= d_f(p,q) + \mathrm{cost}(\overline{qr}) + d_f(r,s) \\ &\leqslant \rho\,\|pq\| + \mathrm{cost}(\overline{qr}) + \rho\,\|rs\|. \end{aligned} \tag{1}$$

A similar derivation shows that the above inequality also holds when $\overline{ps}$ and $\overline{qr}$ are boundary segments contained in the same edge of $\mathcal{T}$. (If the edge containing $\overline{ps}$ and $\overline{qr}$ is incident to two faces f_1 and f_2, one needs to replace $d_f(x,y)$ by $\min(d_{f_1}(x,y), d_{f_2}(x,y))$ for any x and y in the above derivation.)

2.3. Polygonal paths. We consider paths from a point v_s to a point v_d in $|\mathcal{T}|$. Without loss of generality, we can assume that v_s and v_d are vertices of $\mathcal{T}$. If not, we can force v_s and v_d to be vertices by splitting the triangle(s) containing them into smaller triangles. (These smaller triangles inherit their convex distance functions from the triangles containing them.)

A *polygonal path* is a polyline in $|\mathcal{T}|$ with endpoints v_s and v_d. A *link* is an edge of a polygonal path, and a *node* is a vertex of a polygonal path—we use this terminology to avoid confusion with edges and vertices of $\mathcal{T}$. We identify a polygonal path with its sequence of nodes. Thus if a polygonal path P has the node sequence $(v_s = p_0, p_1, \ldots, p_m = v_d)$, we write $P = (p_0, p_1, \ldots, p_m)$. We do not require the nodes of P to be distinct. The *length* of P is defined as $\mathrm{length}(P) = \sum_{i=1}^{m} \|p_{i-1}p_i\|$.

A *$\mathcal{T}$-respecting path* is a polygonal path P such that each link of P is contained in a face of $\mathcal{T}$ (see Figure 1). The cost of a $\mathcal{T}$-respecting path P is simply the sum of the costs of its links when P is traversed from v_s to v_d. Therefore, we have $\mathrm{cost}(P) = \sum_{i=1}^{m} \mathrm{cost}(\overline{p_{i-1}p_i})$. It implies that $\mathrm{cost}(P)/\rho \leqslant \mathrm{length}(P) \leqslant \mathrm{cost}(P)$. We also define $\mathrm{cost}(P)$ when P is an arbitrary polygonal path. We first obtain a $\mathcal{T}$-respecting path P' by introducing new nodes at the intersections between the links of P and the edges of $\mathcal{T}$. We then define $\mathrm{cost}(P) = \mathrm{cost}(P')$.

For any integer k, a *k-link path* is a polygonal path with at most k links, and whose links are either chords or boundary segments (see Figure 1). In particular, a k-link path is $\mathcal{T}$-respecting, and none of its nodes lies in the interior of a face. Our definition is different from previous work [5] on k-link paths, where the path is not required to be $\mathcal{T}$-respecting. By our definitions, a $\mathcal{T}$-respecting path can have a node in the interior of a face of $\mathcal{T}$, but a k-link path cannot.

3. Approximation algorithms. We present algorithms for approximating a shortest k-link path for some given $\varepsilon \in (0,1)$ and k. The cost of the output polygonal path is less than $(1+\varepsilon)$ times the cost of any k-link path, but the output path is allowed to have more than k links. After showing the existence of an approximate shortest path with few links, we can use these algorithms to find $(1+\varepsilon)$-approximate shortest paths.

We first present a simple algorithm in section 3.1, which discretizes the environment with a graph and then computes a shortest path in the graph. Unlike the previous discretization schemes [1, 2], our discretization makes use of the global geodesic distance $\|v_s v_d\|_{\mathcal{T}}$. In section 3.2, we show another simple idea to space out the graph vertices that are far away from v_s and v_d. It results in a reduction of the graph size and hence an improvement of the running time. We show that it is possible to improve the running time further by working with the graph vertices alone (without computing the graph edges), by employing the algorithm BUSHWHACK [19].

3.1. A simple algorithm. In this section we present a simple algorithm that computes an approximate k-link shortest path. It is based on two ideas. First, we observe that any $\frac{4}{3}$-approximate shortest path lies in a region delimited by an ellipse with diameter $\frac{4}{3}\rho\|v_s v_d\|_{\mathcal{T}}$. Second, we discretize the problem by placing Steiner points uniformly along the portion of each edge of $\mathcal{T}$ that lies inside this ellipse. With an appropriate spacing of the Steiner points, we show that there is an approximate shortest path whose nodes are all Steiner points or vertices of $\mathcal{T}$. We find one such path by computing a shortest path in a graph whose nodes are the Steiner points and the vertices of $\mathcal{T}$, using Dijkstra's algorithm.

Let $k \geqslant 2n-4$ denote an integer. Because $\mathcal{T}$ is a planar graph with n vertices, it has at most $2n-4$ faces [14]. It implies that there exists a k-link path from v_s to v_d. Therefore, for any $\varepsilon > 0$, there exists a k-link path P_k^{ε} such that

$$\operatorname{cost}(P_k^{\varepsilon}) \leqslant \left(1+\frac{\varepsilon}{3}\right) \inf\{\operatorname{cost}(P_k) : P_k \text{ is a } k\text{-link path}\}.$$

This path has the following property.

LEMMA 1. *If $k \geqslant 2n-4$ and $\varepsilon \in (0,1)$, then $\operatorname{cost}(P_k^{\varepsilon}) \leqslant \frac{4\rho}{3}\|v_s v_d\|_{\mathcal{T}}$.*

Proof. Let $G = (g_0, g_1, \ldots, g_m)$ be a $\mathcal{T}$-respecting path with length $\|v_s v_d\|_{\mathcal{T}}$ and such that m is minimum. The path G cannot have two links inside the same face of $\mathcal{T}$: if there were two such links, say $\overline{g_{i-1}g_i}$ and $\overline{g_j g_{j+1}}$ with $i \leqslant j$, we could remove from G the nodes $(g_i, \ldots, g_j)$. It would yield a polygonal path with at most the same length but fewer nodes. As we noted above, $\mathcal{T}$ has at most $2n-4$ faces. Therefore, G is a $(2n-4)$-link path and hence a k-link path. We conclude that $\operatorname{cost}(P_k^{\varepsilon}) \leqslant \left(1+\frac{\varepsilon}{3}\right)\operatorname{cost}(G) \leqslant \frac{4}{3}\operatorname{cost}(G) = \frac{4}{3}\sum_{i=1}^{m}\operatorname{cost}(\overline{g_{i-1}g_i}) \leqslant \frac{4}{3}\sum_{i=1}^{m}\rho\|g_{i-1}g_i\| \leqslant \frac{4\rho}{3}\operatorname{length}(G) = \frac{4\rho}{3}\|v_s v_d\|_{\mathcal{T}}$. ∎

Let $\mathbb{E}$ denote the following elliptic region:

$$\mathbb{E} = \left\{ x \in \mathbb{R}^2 : \|v_s x\| + \|v_d x\| \leqslant \frac{4\rho}{3}\|v_s v_d\|_{\mathcal{T}} \right\}.$$

Since $\operatorname{length}(P_k^{\varepsilon}) \leqslant \operatorname{cost}(P_k^{\varepsilon})$, we know by Lemma 1 that $P_k^{\varepsilon} \subset \mathbb{E}$. For each edge e of $\mathcal{T}$, we place a maximal set of equally spaced points on $\operatorname{int}(e \cap \mathbb{E})$ (see Figure 2). The spacing is $\delta = \frac{\varepsilon}{6\rho k}\|v_s v_d\|_{\mathcal{T}}$. The following lemma shows that our Steiner points give an accurate discretization.

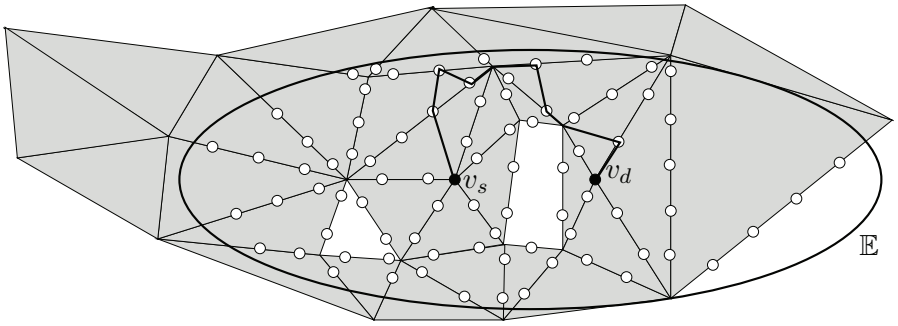


FIG. 2. *The white points are the Steiner points. The polygonal path is a 9-link path whose nodes are all Steiner points or vertices of $\mathcal{T}$.*

LEMMA 2. *If $k \geqslant 2n-4$ and $\varepsilon \in (0,1)$, there exists a k-link path S_k such that $\operatorname{cost}(S_k) \leqslant (1+\varepsilon)\operatorname{cost}(P_k)$ for any k-link path P_k, and all the nodes of S_k are Steiner points or vertices of $\mathcal{T}$.*

Proof. Let $(p_0, p_1, \ldots, p_m)$ be the node sequence of P_k^ε. For any $i \in [0,m]$, we associate a point s_i to p_i as follows. If p_i is a vertex of $\mathcal{T}$, we set $s_i = p_i$. Otherwise, since P_k^ε is a k-link path, p_i lies in the interior of some edge e of $\mathcal{T}$. Thus, there is a Steiner point $x \in \operatorname{int}(e)$ such that $\|xp_i\| \leqslant \delta$. We set $s_i = x$. We denote $S_k = (s_0, s_1, \ldots, s_m)$. (It is possible that $s_{i-1} = s_i$ for some i, in which case the link $\overline{s_{i-1}s_i}$ degenerates to a point.) By construction, for any $i \in [1,m]$, the links $\overline{p_{i-1}p_i}$ and $\overline{s_{i-1}s_i}$ either are chords of the same face of $\mathcal{T}$ or are contained in the same edge of $\mathcal{T}$. Therefore, S_k is a k-link path, and inequality (1) implies that

$$\operatorname{cost}(S_k) \leqslant \sum_{i=1}^{m} \rho\,\|s_{i-1}p_{i-1}\| + \operatorname{cost}(\overline{p_{i-1}p_i}) + \rho\,\|s_ip_i\| \tag{2}$$

$$\leqslant \operatorname{cost}(P_k^\varepsilon) + 2\rho m\delta \tag{3}$$

$$= \operatorname{cost}(P_k^\varepsilon) + \frac{m\varepsilon}{3k}\,\|v_sv_d\|_{\mathcal{T}}. \tag{4}$$

By definition, P_k^ε is a k-link path. So $m \leqslant k$, and thus $\operatorname{cost}(S_k) \leqslant \operatorname{cost}(P_k^\varepsilon) + \frac{\varepsilon}{3}\,\|v_sv_d\|_{\mathcal{T}}$. Since $\|v_sv_d\|_{\mathcal{T}} \leqslant \operatorname{length}(P_k^\varepsilon) \leqslant \operatorname{cost}(P_k^\varepsilon)$, we have $\operatorname{cost}(S_k) \leqslant (1+\varepsilon/3)\operatorname{cost}(P_k^\varepsilon)$. Thus, for any k-link path P_k, $\operatorname{cost}(S_k) \leqslant (1+\varepsilon/3)^2\operatorname{cost}(P_k) \leqslant (1+\varepsilon)\operatorname{cost}(P_k)$. ∎

The *Steiner graph* is the directed weighted graph defined as follows. Its nodes are the Steiner points and the vertices of $\mathcal{T}$. There is a directed edge (p,q) between any two nodes p and q that lie on the boundary of the same face of $\mathcal{T}$. The edge (p,q) is assigned the weight $\operatorname{cost}(\overline{pq})$.

Here is the pseudocode of our approximation algorithm.

Approximate$(\mathcal{T}, k, \varepsilon)$

1. Compute $\|v_sv_d\|_{\mathcal{T}}$.
2. Compute the Steiner graph.
3. Compute a weighted shortest path S in the Steiner graph.
4. Output S.

In the analysis of this algorithm, we use the standard real-RAM model [15] and assume that, for any points p and q in the same face of $\mathcal{T}$, we can compute $\operatorname{cost}(\overline{pq})$ in $O(1)$ time. We obtain the following result.

LEMMA 3. *If $k \geqslant 2n-4$ and $\varepsilon \in (0,1)$, then* Approximate$(\mathcal{T}, k, \varepsilon)$ *computes a polygonal path S such that* $\operatorname{cost}(S) \leqslant (1+\varepsilon)\operatorname{cost}(P_k)$ *for any k-link path P_k. The algorithm can be implemented to run in $O\left(nk^2\rho^4/\varepsilon^2\right)$ time.*

Proof. Correctness follows from Lemma 2. The geodesic distance computation in line 1 can be performed in $O(n^2) = O(nk)$ time [12]. The diameter of $\mathbb{E}$ is $\frac{4}{3}\rho\,\|v_s v_d\|_{\mathcal{T}}$, so each edge e of $\mathcal{T}$ contains $O(k\rho^2/\varepsilon)$ Steiner points. Therefore, the Steiner graph has $O(nk\rho^2/\varepsilon)$ nodes and $O(nk^2\rho^4/\varepsilon^2)$ edges. Using a Fibonacci heap, Dijkstra's algorithm can be implemented to compute single-source shortest paths in $O(|E| + |V|\log|V|)$ time for a graph with $|V|$ nodes and $|E|$ edges [8]. We use this method to implement step 4 and, thus, we obtain the desired running time. ∎

3.2. A sparser Steiner graph. We can speed up the above algorithm by reducing the number of Steiner points. The idea is to place Steiner points more sparsely on portions of edges that are far from v_s and v_d. So we introduce a family of elliptic regions. For $0 \leqslant i \leqslant \lceil \log\rho \rceil$, each region is defined as

$$\mathbb{E}_i = \left\{ x \in \mathbb{R}^2 : \|v_s x\| + \|v_d x\| \leqslant \frac{4\rho}{2^i\,3}\,\|v_s v_d\|_{\mathcal{T}} \right\}.$$

For convenience, we take $\mathbb{E}_{\lceil \log\rho \rceil + 1}$ to denote the empty set.

We construct a set of Steiner points as follows. For each edge e of $\mathcal{T}$ and for any integer i such that $0 \leqslant i \leqslant \lceil \log\rho \rceil$, we insert the intersection points between e and the boundary of $\mathbb{E}_i$, and then we place a maximal set of points on $\operatorname{int}(e \cap (\mathbb{E}_i \setminus \mathbb{E}_{i+1}))$ with uniform spacing $\delta_i = \frac{\varepsilon}{2^{i+1}\,6k}\,\|v_s v_d\|_{\mathcal{T}}$. After obtaining the new set of Steiner points, the Steiner graph is constructed as before. Observe that now, the number of Steiner graph nodes per edge is $O((k\rho\log\rho)/\varepsilon)$ instead of $O(k\rho^2/\varepsilon)$. Therefore, the size of the Steiner graph is $O((nk^2\rho^2\log^2\rho)/\varepsilon^2)$, and the running time improves to $O((nk^2\rho^2\log^2\rho)/\varepsilon^2)$.

It remains to prove that this new algorithm is correct. Let j denote the largest integer such that $P_k^\varepsilon \subset E_j$. (There is such an integer because $P_k^\varepsilon \subset \mathbb{E}_0 = \mathbb{E}$ by Lemma 1.) For every edge e of $\mathcal{T}$, the distance between two consecutive Steiner points along $e \cap E_j$ is at most δ_j. Thus, using the derivation in inequalities (2) and (3) in the proof of Lemma 2, we can show that $\operatorname{cost}(S_k) \leqslant \operatorname{cost}(P_k^\varepsilon) + \rho\varepsilon\,\|v_s v_d\|_{\mathcal{T}}/(2^{j+1}\,3)$. As P_k^ε is not contained in $\mathbb{E}_{j+1}$, we have

$$\operatorname{length}(P_k^\varepsilon) \geqslant 4\rho\,\|v_s v_d\|_{\mathcal{T}}/(2^{j+1}\,3)$$

and thus $\operatorname{cost}(S_k) \leqslant (1+\varepsilon/4)\operatorname{cost}(P_k^\varepsilon)$. Recall that $\operatorname{cost}(P_k^\varepsilon) \leqslant (1+\varepsilon/3)\operatorname{cost}(P_k)$ for any k-link path P_k. It follows that $\operatorname{cost}(S_k) \leqslant (1+\varepsilon)\operatorname{cost}(P_k)$. Therefore, we have proved the following result.

LEMMA 4. *If $k \geqslant 2n-4$ and $\varepsilon \in (0,1)$, we can compute in $O((nk^2\rho^2\log^2\rho)/\varepsilon^2)$ time a polygonal path S such that* $\operatorname{cost}(S) \leqslant (1+\varepsilon)\operatorname{cost}(P_k)$ *for any k-link path P_k.*

3.3. Further improvement. We can improve the running time further if we can avoid computing explicitly the edges of the Steiner graph. This means that, instead of using Dijkstra's algorithm, we need to use an algorithm that does not require these edges to produce a shortest path. BUSHWHACK, an algorithm by Sun and Reif [19], does exactly this, provided that the cost function is *pseudo-Euclidean*.

The cost function is pseudo-Euclidean if it meets the following two conditions. First, it obeys the triangle inequality in the interior of each face (in other words, a shortest path in the interior of a face is a line segment). Second, let p be a point inside a face f and let e be an edge of f such that $p \notin e$. Let $\varphi_{p,e} : e \to \mathbb{R}$ be the

function defined by $\varphi_{p,e}(x) = \mathrm{cost}(\overline{px})$. For any p and e, this function is required to have the following property: we can compute in $O(1)$ time a partition of e into $O(1)$ subsegments such that $\varphi_{p,e}$ is monotone along each such subsegment.

The convexity of the distance functions in $\mathcal{T}$ implies that $\varphi_{p,e}$ has only one local extremum, which is a global minimum. (In degenerate cases, when B_f is not strictly convex, this minimum can be achieved over a closed segment and not just at a single point.) So if we assume that this minimum can be computed in $O(1)$ time, our metric is pseudo-Euclidean.

When there are m Steiner points per edge, the BUSHWHACK algorithm runs in time $O(mn\log(mn))$ instead of $O(m^2n + mn\log(mn))$ for Dijkstra's algorithm. The algorithm presented in section 3.2 uses $m = O((k\rho\log\rho)/\varepsilon)$ Steiner points per edge. Thus, we obtain the following improved algorithm.

THEOREM 5. *If $k \geqslant 2n-4$ and $\varepsilon \in (0,1)$, we can compute in time*

$$O\left(\frac{nk\rho\log\rho}{\varepsilon}\log\left(\frac{k\rho}{\varepsilon}\right)\right)$$

a polygonal path S such that $\mathrm{cost}(S) \leqslant (1+\varepsilon)\,\mathrm{cost}(P_k)$ for any k-link path P_k.

3.4. Applications. In the weighted region problem, each face f is associated with a weight $w_f \in [1,\rho]$ (assuming that the minimum weight is scaled to 1). According to our terminology, for each face f, the set B_f is the Euclidean disk centered at the origin with radius $1/w_f$. Obstacles (holes in $|\mathcal{T}|$) are still allowed; in other words, we allow weights to be in $[1,\rho]\cup\{+\infty\}$. Mitchell and Papadimitriou [12, 13] proved that, in the weighted region problem, the shortest path is an $O(n^2)$-link path. Substituting this bound for k into Theorem 5 yields the following result.

COROLLARY 6. *Consider the weighted region problem in a planar subdivision with n vertices and with weights in $[1,\rho]\cup\{+\infty\}$. For any $\varepsilon \in (0,1)$, we can compute a $(1+\varepsilon)$-approximate shortest path in time $O(\frac{\rho\log\rho}{\varepsilon}n^3\log(\frac{n\rho}{\varepsilon}))$.*

To realize the $(1+\varepsilon)$-approximation bound in Corollary 6, one would need to extract from [13] the constant c_0 hidden in the $O(n^2)$ bound on the number of links. Nevertheless, the proof of Lemma 2 reveals that even if k is set to be c_1n^2 for some constant $c_1 < c_0$, the approximation ratio is still $1+O(\varepsilon)$.

In the anisotropic setting where each face f is associated with a convex distance function d_f, no bound on the number of links in the shortest path was known before. In Theorem 16 in section 5 and Corollary 23 in section 6, for any $\varepsilon \in (0,1)$, we prove a bound of $21\rho n^2/\varepsilon$ on the number of links in a polygonal path whose cost is at most $(1+\varepsilon)$ times the optimal. Substituting this bound for k into Theorem 5 yields the following result.

COROLLARY 7. *Consider the anisotropic shortest path problem in a planar subdivision with n vertices. For any $\varepsilon \in (0,1)$, we can compute a $(1+\varepsilon)$-approximate shortest path in time $O(\frac{\rho^2\log\rho}{\varepsilon^2}n^3\log(\frac{\rho n}{\varepsilon}))$.*

As we mentioned in the introduction (section 1.2), the shortest path problem in the presence of uniform flows can be solved using Corollary 7.

COROLLARY 8. *Consider the shortest path problem in the presence of uniform flows as defined in section 1.2. Assume that $v_{\min} > 0$. For any $\varepsilon \in (0,1)$, we can compute a $(1+\varepsilon)$-approximate shortest path in time $O(\frac{\rho^2\log\rho}{\varepsilon^2}n^3\log(\frac{\rho n}{\varepsilon}))$, where $\rho = v_{\max}/v_{\min}$.*

4. Cost-preserving path transformations. In this section we show how to transform a $\mathcal{T}$-respecting path into another $\mathcal{T}$-respecting path that has at most the same cost and is free of some undesirable features. This result is used in section 5 to show that any polygonal path can be converted into a polygonal path with $O(\rho n^2/\varepsilon)$ links such that the cost increases by a factor of at most $1+\varepsilon$.

4.1. Preliminaries. Let $P = (v_s = p_0, \ldots, p_m = v_d)$ be a polygonal path. We allow $p_i = p_{i+1}$; in this case, we call $\overline{p_i p_{i+1}} = \{p_i\}$ a *degenerate link*. The path P is *self-intersecting* if

- there exist $i < j$ such that $\overline{p_{i-1}p_i} \cap \overline{p_j p_{j+1}} \neq \emptyset$, or
- there exists i such that $p_{i-1} \in \overline{p_i p_{i+1}}$ or $p_{i+1} \in \overline{p_{i-1}p_i}$.

The path P is *simple* if it is not self-intersecting. The points in which P self-intersects are not necessarily nodes of P.

To describe the cost-preserving transformations, we give a classification of nodes in P. The vertices of $\mathcal{T}$ that are nodes of P are called the nodes of P *inherited* from $\mathcal{T}$. The endpoints of degenerate links are called *degenerate nodes*. The other nodes are classified as follows (Figure 3 shows some examples).

- *Transversal node*: $p_i \in \mathrm{int}(e)$ for some edge e, and $\overline{p_{i-1}p_i} \cap \mathrm{int}(f) \neq \emptyset$ and $\overline{p_i p_{i+1}} \cap \mathrm{int}(g) \neq \emptyset$, where f and g are the two faces incident to e.
- *Critical node of entry*: $p_i \in \mathrm{int}(e)$ and $\mathrm{int}(\overline{p_i p_{i+1}}) \cap e \neq \emptyset$ for some edge e, and $\overline{p_{i-1}p_i} \cap \mathrm{int}(f) \neq \emptyset$ for a face f incident to e.
- *Critical node of exit*: $p_i \in \mathrm{int}(e)$ and $\mathrm{int}(\overline{p_{i-1}p_i}) \cap e \neq \emptyset$ for some edge e, and $\overline{p_i p_{i+1}} \cap \mathrm{int}(f) \neq \emptyset$ for a face f incident to e.
- *Reflective node*: $p_i \in \mathrm{int}(e)$ for some edge e, and $\overline{p_{i-1}p_i} \cap \mathrm{int}(f) \neq \emptyset$ and $\overline{p_i p_{i+1}} \cap \mathrm{int}(f) \neq \emptyset$ for a face f incident to e.
- *Interior node*: $p_i \in \mathrm{int}(f)$ for some face f.
- *Linear node*: $p_i \in \mathrm{int}(e)$, $\mathrm{int}(\overline{p_{i-1}p_i}) \cap e \neq \emptyset$, and $\mathrm{int}(\overline{p_i p_{i+1}}) \cap e \neq \emptyset$ for some edge e.

In the definition of a linear, interior, or reflective node p_i, we allow $\overline{p_{i-1}p_i}$ and $\overline{p_i p_{i+1}}$ to overlap. We call a polygonal path *nonredundant* if it does not contain any degenerate, reflective, interior, or linear node. Some intermediate results of the cost-preserving transformations may be self-intersecting paths, so our classification of nodes does not assume simplicity. Our classification does not assume paths to be $\mathcal{T}$-respecting either.

Our cost-preserving transformations manipulate polygonal paths by modifying subpaths and concatenating polygonal paths. Recall that, for convenience, we use the same notation for a polygonal path and its sequence of nodes. For any integers i, j such that $0 \leqslant i \leqslant j \leqslant m$, we use $P[i,j]$ to denote the subpath $(p_i, \ldots, p_j)$. Given two subpaths $A = (a_1, \ldots, a_{m'})$ and $B = (b_1, \ldots, b_{m''})$, we denote by $A \cdot B = (a_1, \ldots, a_{m'}, b_1, \ldots, b_{m''})$ the concatenation of A and B.

4.2. Splitting a path. Let $P = (p_0, \ldots, p_m)$ be a polygonal path. The procedure Split converts P into a $\mathcal{T}$-respecting path with the same cost. This procedure operates as follows. For each vertex v that lies in the interior of a link of P, it splits this link at v and makes v a node of P. For each crossing point x between an edge of $\mathcal{T}$ and a link of P, it splits the link at point x and introduces the node x.

Split(Polygonal path P)

1. If there exist a vertex v of $\mathcal{T}$ and a link $\overline{p_i p_{i+1}}$ of P such that $v \in \mathrm{int}(\overline{p_i p_{i+1}})$, then return Split$(P[0,i] \cdot (v) \cdot P[i+1,m])$.
2. If there exist an edge e of $\mathcal{T}$, a link $\overline{p_i p_{i+1}}$ of P, and a point x such that $x = \mathrm{int}(e) \cap \mathrm{int}(\overline{p_i p_{i+1}})$, then return Split$(P[0,i] \cdot (x) \cdot P[i+1,m])$.
3. Return P.

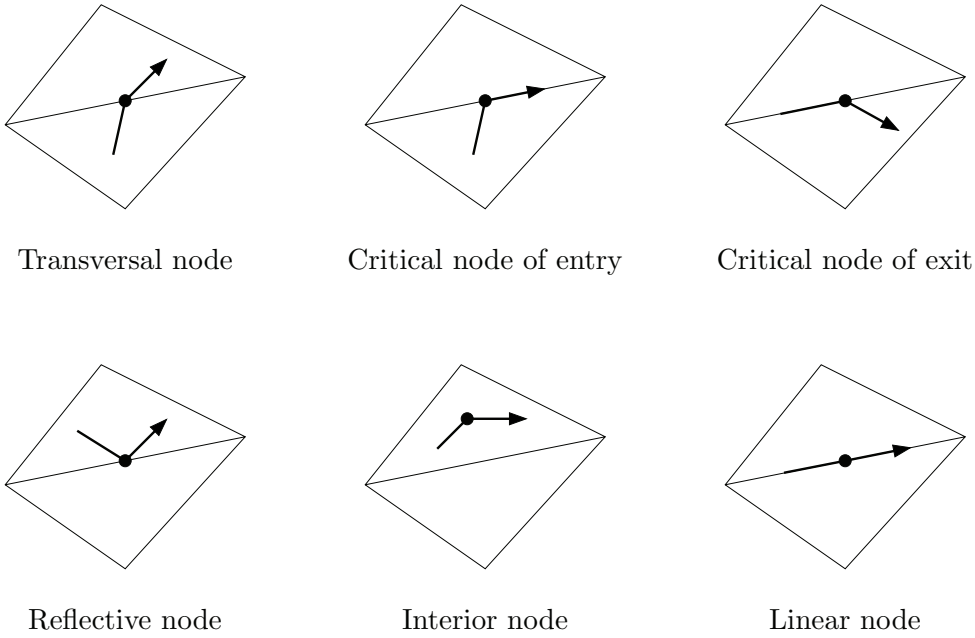


FIG. 3. *Different types of nodes of P other than degenerate nodes and nodes inherited from $\mathcal{T}$.*

The following lemma describes the effect of Split.

LEMMA 9. *For any polygonal path P, the path P' returned by* Split(P) *is $\mathcal{T}$-respecting, and* $\text{cost}(P') = \text{cost}(P)$.

4.3. Reducing a path. Let $P = (p_0, \ldots, p_m)$ be a $\mathcal{T}$-respecting path. The procedure Reduce eliminates some locally nonoptimal features in P, including degenerate, reflective, interior, and linear nodes, as well as self-intersections.

Reduce($\mathcal{T}$-respecting path $P = (p_0, \ldots, p_m)$)

1. If there exists i such that $p_i = p_{i+1}$,
   then return Reduce$(P[0, i-1] \cdot P[i+1, m])$.
2. If a node p_i is a reflective, interior, or linear node,
   then return Reduce$(P[0, i-1] \cdot P[i+1, m])$.
3. If P self-intersects, then we are in one of the following two cases:
   (a) There exist $i < j$ and $x \in |\mathcal{T}|$ such that $x \in \overline{p_{i-1}p_i} \cap \overline{p_j p_{j+1}}$.
   Then return Reduce$(P[0, i-1] \cdot (x) \cdot P[j+1, m])$.
   (b) There exists i such that $p_{i-1} \in \overline{p_i p_{i+1}}$ or $p_{i+1} \in \overline{p_{i-1}p_i}$.
   Then return Reduce$(P[0, i-1] \cdot P[i+1, m])$.
4. Return P.

It is clear that the number of nodes in P decreases after one application of step 1, 2, or 3. It implies that Reduce terminates.

Recall that the input path is assumed to be $\mathcal{T}$-respecting. Clearly, this condition continues to hold after applying step 1, 2, or 3 any number of times. In addition, when Reduce terminates, the output path P' returned by Reduce(P) is simple and nonredundant.

LEMMA 10. *Let P be a $\mathcal{T}$-respecting path. The path P' returned by* Reduce(P) *is simple, nonredundant, and $\mathcal{T}$-respecting. If $P' \neq P$, then P' has fewer links than P, and* $\text{cost}(P') \leqslant \text{cost}(P)$.

4.4. Sliding subpaths. We call (i, j) a *critical pair* if p_i is a critical node of exit, p_j is a critical node of entry, $i < j$, and all of the nodes in the interior of the subpath $P[i, j]$ are transversal nodes (see Figure 4). We introduce a procedure Slide to remove a critical pair in P if there is one. The same technique of sliding subpaths was used by Mitchell and Papadimitriou [13]. Because Reduce will be applied before applying Slide, we assume that P is a simple, nonredundant, $\mathcal{T}$-respecting path.

Slide(simple, nonredundant, $\mathcal{T}$-respecting path P)

1. Look for a critical pair (i, j) in P.
2. If no critical pair is found, return P.
3. Slide $P[i, j]$ in a direction such that $\text{cost}(P)$ does not increase, until one of the following situations occurs:
   (a) $p_{i-1} = p_i$. Then return $P[0, i-1] \cdot P[i+1, m]$.
   (b) $p_j = p_{j+1}$. Then return $P[0, j-1] \cdot P[j+1, m]$.
   (c) P self-intersects or $P[i, j]$ hits a vertex of $\mathcal{T}$. Then return P.

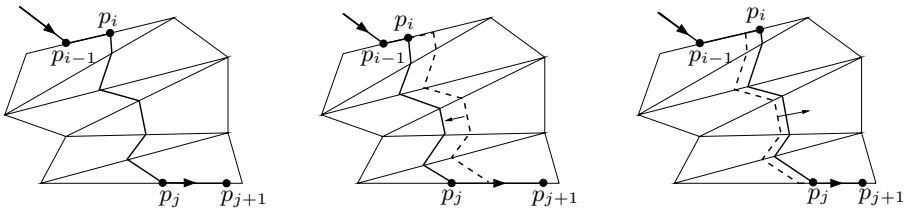


FIG. 4. *A critical pair (i, j). The subpath $P[i, j]$ can slide to the left or to the right.*

We explain the sliding operation in step 3 in more detail. Refer to Figure 4. Sliding $P[i, j]$ means shifting p_i, all of the transversal nodes on $P[i, j]$, and p_j along the corresponding edges in such a way that the links in $P[i, j]$ remain parallel to their original positions. The links $\overline{p_{i-1}p_i}$, $\overline{p_jp_{j+1}}$, and those in $P[i, j]$ may lengthen or shrink. Let Δ be the signed distance of the sliding of p_i from its original position. We take Δ to be positive (resp., negative) if p_i slides to the right (resp., left). Take a sliding link $\overline{p_ap_{a+1}}$. As the slope of $\overline{p_ap_{a+1}}$ is kept constant, the cost of $\overline{p_ap_{a+1}}$ is an affine function of Δ. Clearly, the costs of $\overline{p_{i-1}p_i}$ and $\overline{p_jp_{j+1}}$ are also affine functions of Δ. Thus, the total change in $\text{cost}(P)$ is an affine function of Δ. Since the value of an affine function is minimized at the boundary of its domain, we conclude that there is a direction in which we can slide $P[i, j]$ without increasing $\text{cost}(P)$. The stopping criteria in step 3 exhausts all of the possibilities in which (i, j) no longer satisfies the criteria of being a critical pair or P ceases to be simple. The following lemma describes the effect of Slide.

LEMMA 11. *Let P be a simple, nonredundant, $\mathcal{T}$-respecting path. The path P' returned by* Slide(P) *satisfies the following properties:*

(i) *P' is a $\mathcal{T}$-respecting path.*
(ii) $\text{cost}(P') \leqslant \text{cost}(P)$*, and P' has no more links than P.*
(iii) *If $P' \neq P$ and P' has as many links as P, then P' is self-intersecting or P' has more nodes inherited from $\mathcal{T}$ than P does.*

4.5. Combining the transformations. We define a procedure Simplify that makes use of the previous procedures. The input is a polygonal path P.

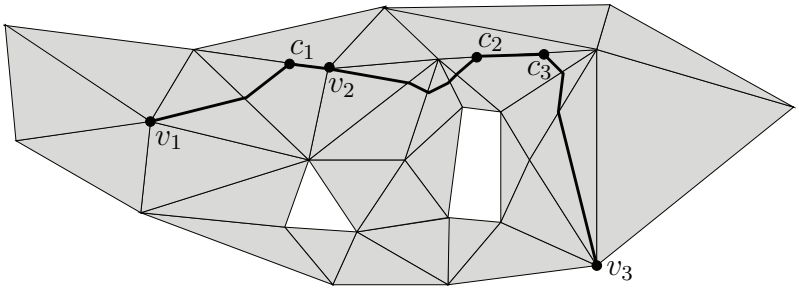


FIG. 5. *An example of a simplified path Q_0. There are two critical nodes c_2 and c_3 between vertices v_2 and v_3.*

Simplify(Polygonal path P)

1. $Q_1 :=$ Split(P).
2. $Q_2 :=$ Reduce(Q_1).
3. $Q_0 :=$ Slide(Q_2). If $Q_0 \neq Q_2$, set $Q_1 := Q_0$ and go back to step 2.
4. Return Q_0.

LEMMA 12. *Let P be a polygonal path. The path Q_0 returned by* Simplify(P) *has the following properties:*

(i) $\text{cost}(Q_0) \leqslant \text{cost}(P)$.

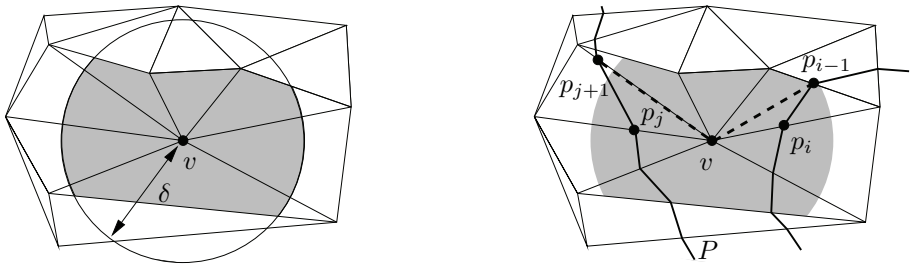
(ii) *Q_0 is a simple, nonredundant, $\mathcal{T}$-respecting path.*

(iii) *Q_0 has no critical pair and has at most $2n$ critical vertices of entry or exit.*

Proof. We first show the termination of Simplify. By Lemmas 10 and 11, the number of links in the path never increases at steps 2 and 3. If the number of links stays the same in step 3, then Lemma 11 says that Q_0 is self-intersecting or it has more nodes inherited from $\mathcal{T}$ than Q_2 does. If Q_0 is self-intersecting, the number of links in the path decreases when Reduce is called in the next step. So if steps 2 and 3 iterate without decreasing the number of links in the path, it means that the path remains simple and the number of nodes inherited from $\mathcal{T}$ increases from iteration to iteration. But this can happen at most $n-2$ times before all vertices of $\mathcal{T}$ lie on the path. Therefore, the number of links in the path must decrease after at most $n-1$ iterations of steps 2 and 3. Clearly, the number of links cannot decrease below one, so Simplify must terminate.

Let $(q_0, \ldots, q_m)$ be the node sequence of Q_0. Property (i) follows from the fact that Split, Reduce, and Slide do not increase the cost. Property (ii) follows from Lemma 10. The termination of Simplify implies that no critical pair is detected by Slide. So if we walk along Q_0 from v_s to v_d and encounter a critical node of exit q_i, we will see a node $q_{i'}$ inherited from $\mathcal{T}$ before encountering any critical node of entry or exit. We charge q_i to $q_{i'}$. The node $q_{i'}$ can only be charged once, so there are at most n critical nodes of exit in Q_0. By a symmetric argument (following Q_0 backward from v_d to v_s), we can show that there are at most n critical nodes of entry. Hence, the total number of critical nodes is at most $2n$. ∎

In fact, the proof of Lemma 12 shows even more. Between any two consecutive nodes of Q_0 inherited from $\mathcal{T}$, there is at most one critical node of entry and one critical node of exit. In case these two critical nodes are present, they lie on the same edge e, and the path enters and exits e from the same side (see Figure 5). All of the other nodes (between these two consecutive inherited nodes) are transversal nodes.



(a) (b)

FIG. 6. (a) *The δ-neighborhood of v is shaded.* (b) *The effect of* Shortcut*(P, δ, v). The subpath $P[i-1, j+1]$ is replaced by the dashed subpath (p_{i-1}, v, p_{j+1}).*

5. Path complexity. In this section we show that any polygonal path P can be approximated by a $(21\rho n^2/\varepsilon)$-link path with cost at most $(1+\varepsilon)\,\mathrm{cost}(P)$. This result, combined with the algorithms for approximating k-link shortest paths that we presented in section 3, allows us to compute a polygonal path with cost at most $(1+\varepsilon)\,\mathrm{cost}(P)$ in time $O(\frac{\rho^2 \log \rho}{\varepsilon^2} n^3 \log(\frac{\rho n}{\varepsilon}))$.

5.1. Shortcut near vertices. We introduce a procedure Shortcut that removes nodes in the vicinity of vertices of $\mathcal{T}$. Shortcut takes as input a polygonal path P, a real number $\delta > 0$, and a vertex v of $\mathcal{T}$. Shortcut will be invoked repeatedly, starting with the output of Simplify. The path may no longer be simple or nonredundant after several applications of Shortcut.

Let v be a vertex of $\mathcal{T}$. The *δ-neighborhood* of v is the intersection of the δ-radius Euclidean disk centered at v with the union of the faces that contain v (see Figure 6(a)). Shortcut(P, δ, v) checks if P has a node in the interior of the δ-neighborhood of v. If so, let p_i and p_j be the first and last nodes of P, respectively, in the interior of the δ-neighborhood of v—here the interior is taken in the topological sense. Then Shortcut replaces $P[i-1, j+1]$ with the subpath (p_{i-1}, v, p_{j+1}) (see Figure 6(b)). The nodes p_{i-1} and p_{j+1} lie on the boundary of a face incident to v, so the new path is still inside $|\mathcal{T}|$. The links $\overline{p_{i-1}v}$ and $\overline{vp_{j+1}}$ introduced by Shortcut may make the path redundant or self-intersecting. Here is the pseudocode of Shortcut.

Shortcut(Polygonal path P, $\delta > 0$, vertex v)

1. If no node of P is in the interior of the δ-neighborhood of v, return P.
2. Let p_i and p_j be the first and last nodes along P, respectively, that lie in the interior of the δ-neighborhood of v. Let p_0 and p_m be the first and last nodes of P.
3. If $v = p_0$, return $(v) \cdot P[j+1, m]$.
4. If $v = p_m$, return $P[0, i-1] \cdot (v)$.
5. Return $P[0, i-1] \cdot (v) \cdot P[j+1, m]$.

The following lemma gives a bound on the cost of the path obtained by applying Shortcut.

LEMMA 13. *Let P be a $\mathcal{T}$-respecting path. Then the path returned by* Shortcut(P, δ, v) *has cost less than* $\mathrm{cost}(P) + 2\rho\delta$.

Proof. Either $\overline{p_{i-1}p_i}$ is a chord of a face incident to v or it is contained in an edge incident to v. In any case, as $\|p_i v\| < \delta$, we have $\mathrm{cost}(\overline{p_{i-1}v}) < \mathrm{cost}(\overline{p_{i-1}p_i}) + \rho\delta$.

Similarly, we can prove that $\text{cost}(\overline{vp_{j+1}}) < \text{cost}(\overline{p_jp_{j+1}}) + \rho\delta$. ∎

Remark. Reif and Sun [19] also used a disk neighborhood. Their disk radius is related to the local geometry around the vertex v, and the disk is completely contained in the union of the faces incident to v. In our construction, the disk radius will be set to be proportional to the global path cost. (See the procedure Convert in section 5.2.) As shown in Figure 6, our disk may not be contained in the union of faces incident to v.

5.2. Path conversion. The pseudocode below describes a procedure Convert that transforms a polygonal path P into another polygonal path R. We show that R has at most $21\rho n^2/\varepsilon$ links and its cost is at most $(1+\varepsilon)\,\text{cost}(P)$. We denote by $\{v_1, v_2, \ldots, v_n\}$ the vertices of $\mathcal{T}$ in the following pseudocode.

Convert(Polygonal path P, $\varepsilon \in (0,1)$)
1. $Q_0 :=$ Simplify(P).
2. $\delta := \varepsilon\,\text{cost}(P)/(2\rho n)$.
3. For $i = 1$ to n, $Q_i :=$ Shortcut(Q_{i-1}, δ, v_i).
4. $R := Q_n$.
5. Return R.

The procedure Convert has the following properties.

LEMMA 14. *Let $R = (r_1, \ldots, r_\ell)$ be the path returned by* Convert(P, ε).

(i) $\text{cost}(R) \leqslant (1+\varepsilon)\ \text{cost}(P)$.
(ii) *R is a $\mathcal{T}$-respecting path with distinct nodes.*
(iii) *If $\{r_i, r_{i+1}\}$ contains no vertex of $\mathcal{T}$, then $\overline{r_ir_{i+1}}$ is a link of Q_0.*
(iv) *If v is a vertex of $\mathcal{T}$ and r_i lies in the interior of an edge incident to v, then $\|r_iv\| \geqslant \delta$.*

Proof. By Lemma 13, the path cost increases by at most $2\rho\delta = \varepsilon\,\text{cost}(P)/n$ for each call to Shortcut. By Lemma 12, $\text{cost}(Q_0) \leqslant \text{cost}(P)$; thus $\text{cost}(R) \leqslant \text{cost}(Q_0) + \varepsilon\ \text{cost}(P) \leqslant (1+\varepsilon)\ \text{cost}(P)$. By Lemma 12 again, Q_0 is a $\mathcal{T}$-respecting path with distinct nodes. Each call to Shortcut preserves these two properties. This proves (ii). Properties (iii) and (iv) follow from the working of Shortcut. ∎

We prove a technical lemma which essentially says that, for any edge e, the transversal nodes in the output of Convert(P, ε) that lie on e are sparse.

LEMMA 15. *Let $R = (r_1, \ldots, r_\ell)$ be the path returned by* Convert(P, ε). *Consider an edge e of $\mathcal{T}$ and a node $r_i \in \text{int}(e)$. Suppose that the following conditions hold:*

- *There exists a node $r_j \in \text{int}(e)$ with $j > i+1$. If there are several such nodes we choose j to be the minimum.*
- *All nodes in $\text{int}(R[i,j])$ are transversal nodes.*

Then $\text{cost}(R[i,j]) > \delta$.

Proof. We denote by ℓ_e the support line of e, and we denote by x the first intersection point between $R[i,j]$ and ℓ_e. If $x \neq r_j$, then $x \notin \text{int}(e)$ as R is $\mathcal{T}$-respecting (see Figure 7(a)). By Lemma 14(iv), we know that $\|r_ix\| > \delta$ and thus $\text{cost}(R[i,j]) \geqslant \|r_ix\| > \delta$.

We now assume that $x = r_j$, and thus $\text{int}(R[i,j])$ lies entirely on one side of ℓ_e. We denote by R_i the region delimited by $R[i,j]$ and $\overline{r_ir_j}$ (see Figure 7(b)). We first prove that there exists an edge e' of $\mathcal{T}$ with one endpoint v' inside R_i. Let E denote the set of edges of $\mathcal{T}$ that contain a node of $R[i,j]$. Pick an edge of E. If this edge has an endpoint inside R_i, we are done. Otherwise, this edge crosses $\text{int}(R[i,j])$ transversally, so it separates a portion of R_i away from e. By recursively applying the argument on this portion of R_i, we must find an edge $e' \in E$ that has an endpoint inside R_i.

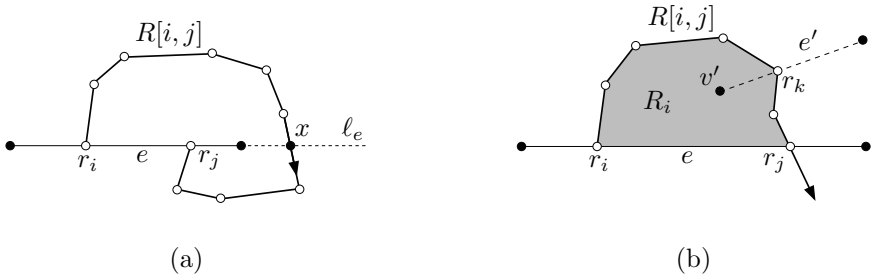


FIG. 7. *In* (a)*, the subpath* $R[i,j]$ *intersects* ℓ_e *outside* e *at point* x*. In* (b)*, the node* v' *is in the interior of the region* R_i *delimited by* $R[i,j]$ *and* $\overline{r_i r_j}$*.*

Let r_k denote a node of $R[i,j]$ that lies on e'. Because no vertex of $\mathcal{T}$ lies on $R[i,j]$, Lemma 14(iv) implies that $\|r_k v'\| \geqslant \delta$. It follows that $\text{length}(R[i,j]) > \|r_k v'\| \geqslant \delta$. Hence, $\text{cost}(R[i,j]) \geqslant \text{length}(R[i,j]) > \delta$. ∎

We are now ready to derive a pseudopolynomial bound on the path complexity.

THEOREM 16. *For any* $\varepsilon \in (0,1)$ *and for any polygonal path* P*, there exists a path* P^ε *with at most* $21\rho n^2/\varepsilon$ *links such that* $\text{cost}(P^\varepsilon) \leqslant (1+\varepsilon)\,\text{cost}(P)$*. In addition,* P^ε *can be chosen to be simple and nonredundant.*

Proof. Let $Q_0 = \mathsf{Simplify}(P)$, $R = \mathsf{Convert}(P,\varepsilon)$, and $P^\varepsilon = \mathsf{Reduce}(R)$. By Lemmas 10, 12, and 14, we know that

- P^ε is simple, nonredundant, and $\mathcal{T}$-respecting;
- $\text{cost}(P^\varepsilon) \leqslant \text{cost}(R) \leqslant (1+\varepsilon)\ \text{cost}(P)$.

Reduce does not increase the number of links. Hence, it suffices to prove that R has $21\rho n^2/\varepsilon$ nodes. By Lemma 14(ii), R is a $\mathcal{T}$-respecting path with distinct nodes.

Take an edge e of $\mathcal{T}$. We denote by N_e the set of nodes $r_i \in \text{int}(e)$. We show below that $|N_e| \leqslant 4\rho n/\varepsilon + 3n + 4$. We assume that $|N_e| \geqslant 2$; otherwise, we are done. Let r_i and r_j, $i < j$, be two nodes in N_e that are consecutive in the order along R.

Case 1. r_{i-1} is a vertex of $\mathcal{T}$. Because $r_i \in \text{int}(e)$, r_{i-1} must be one of the four vertices of the two triangles incident to e. So there are at most four such r_i's in N_e.

Case 2. There is a vertex v of $\mathcal{T}$ in $R[i,j]$. We charge r_i to v; thus there are at most n such r_i's in N_e.

Case 3. $j = i+1$. As $r_i, r_{i+1} \in \text{int}(e)$, r_i and r_{i+1} are not vertices of $\mathcal{T}$. Neither is r_{i-1} as Case 1 does not apply. By Lemma 14(iii), (r_{i-1}, r_i, r_{i+1}) is a subpath of Q_0. We have $\overline{r_i r_{i+1}} \subset \text{int}(e)$ as $r_i, r_{i+1} \in \text{int}(e)$. It follows that r_i is a critical node in Q_0 because Q_0 is nonredundant and $\mathcal{T}$-respecting.

Case 4. There is no vertex of $\mathcal{T}$ in $R[i,j]$, but there is a critical node r_k in $\text{int}(R[i,j])$. We charge r_i to r_k. Lemma 14(iii) implies that $R[i,j]$ is a subpath of Q_0. Thus r_k is also a critical node in Q_0.

Case 5. There is no vertex of $\mathcal{T}$ in $R[i,j]$ and no critical node in $\text{int}(R[i,j])$. Again, Lemma 14(iii) implies that $R[i,j]$ is a subpath of Q_0. Because Q_0 is $\mathcal{T}$-respecting and nonredundant, all of the nodes in $\text{int}(R[i,j])$ are transversal nodes. By Lemma 15, we have $\text{cost}(R[i,j]) > \delta$. Because $\text{cost}(R) \leqslant (1+\varepsilon)\,\text{cost}(P) < 2\,\text{cost}(P)$ and $\delta = \frac{\varepsilon}{2\rho n}\,\text{cost}(P)$, we get $\text{cost}(R[i,j]) > \frac{\varepsilon}{4\rho n}\,\text{cost}(R)$. So there are at most $4\rho n/\varepsilon$ such r_i's in N_e.

Cases 1, 2, and 5 contribute at most $4\rho n/\varepsilon + n + 4$ nodes to N_e. The nodes in Cases 3 and 4 are critical nodes in Q_0. By Lemma 12(iii), there are at most $2n$

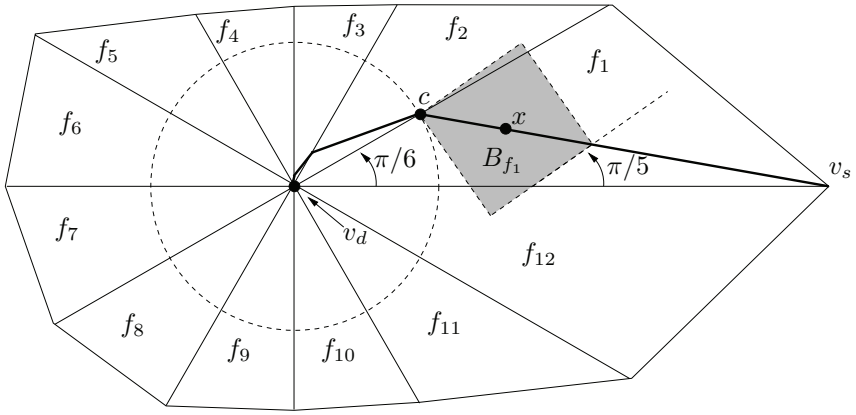


FIG. 8. *B_{f_1} is a square centered at the origin and with edge length $\sqrt{2}$. B_{f_i} is obtained by rotating B_{f_1} by an angle $(i-1)\pi/6$. We know that the shortest path from v_s to v_d is not polygonal. We conjecture that the shortest path from v_s to v_d is an infinite sequence of line segments, forming a spiral around v_d.*

critical nodes in Q_0, so Cases 3 and 4 contribute at most $2n$ nodes to N_e. In all, $|N_e| \leqslant 4\rho n/\varepsilon + 3n + 4$. The number of edges is at most $3n-6$ for a planar graph with n vertices. Summing over all edges of $\mathcal{T}$ and including the vertices of $\mathcal{T}$, the number of nodes in R is at most $n + (3n-6)(4\rho n/\varepsilon + 3n + 4) \leqslant 21\rho n^2/\varepsilon$. □

6. General paths. The statement of Theorem 16 is not entirely satisfactory. For instance, it does not tell us whether there exists a shortest path and if so, whether it is a polygonal path. We give an example in section 6.1 in which the shortest path cannot be polygonal. Nonetheless, we show that there exists a shortest rectifiable path. Intuitively, a path is rectifiable if its length is finite; for example, the class of rectifiable paths includes piecewise C^1 paths. Furthermore, we show that for any $\alpha > 1$, there exists a polygonal path with cost at most α times the optimal. These results imply Corollary 7 in section 3.4.

Throughout this section we do not require the endpoints of a path to be v_s and v_d. Also, we do not require paths to be polygonal unless stated explicitly otherwise.

6.1. Example of a nonpolygonal shortest path. There may not be any (exact) shortest polygonal path. We give an example in Figure 8 in which $\rho = \sqrt{2}$ and there is no shortest polygonal path.

We prove it by contradiction. Consider the points x and c in Figure 8. Assume that the last link of a polygonal path P is $\overline{xv_d}$. Then, if we insert the node c between x and v_d in P (thus replacing the subpath (x, v_d) by (x, c, v_d)), we obtain a polygonal path with strictly smaller cost than P. The same argument shows that the shortest path intersects some edge an infinite number of times.

A similar example can be constructed for any $\rho > 1$ by changing B_{f_i} into a regular polygon with enough edges and by placing a larger number of faces around v_d.

6.2. Rectifiable paths. Intuitively, a path is rectifiable if it has finite length. The length of a path can be defined as the supremum of the length of the polygonal paths inscribed in this path. This definition is a common way of introducing the length of a curve; for instance, see Guggenheimer's book [9]. It allows one to define the length of a curve that is not necessarily piecewise C^1. It can also be proved

that, for a C^1 curve, the length defined in this way coincides with the other usual definition through calculus, where the length is obtained by integrating the norm of the derivative of the curve [9, Theorem 2-3].

We do not require paths to be polygonal, so a path is a continuous function $\pi : [a, b] \to |\mathcal{T}|$ where $a < b \in \mathbb{R}$. We use $\pi[a, b]$ to denote $\{\pi(t) : a \leqslant t \leqslant b\}$ and $\pi(a, b)$ to denote $\{\pi(t) : a < t < b\}$. We say that π is a path from x to y if $\pi(a) = x$ and $\pi(b) = y$.

An $[a, b]$-*sequence* is a finite increasing sequence $\sigma = (t_0, t_1, \ldots, t_m)$ such that $a = t_0 < t_1 < \cdots < t_m = b$. A polygonal path *inscribed in* π is a polyline $\pi_\sigma = (\pi(t_0), \pi(t_1), \ldots, \pi(t_m))$ such that σ is an $[a, b]$-sequence for some $a < b$. When there is no ambiguity, we abuse notation and use π_σ to denote $\bigcup_{i=1}^{m} \overline{\pi(t_{i-1})\pi(t_i)}$.

The *length* of a path $\pi : [a, b] \to |\mathcal{T}|$ is defined as

$$\text{length}(\pi) = \sup\{\, \text{length}(\pi_\sigma) : \sigma \text{ is an } [a, b]\text{-sequence}\,\}.$$

The path π is *rectifiable* if length(π) is finite.

An inscribed polygonal path π_σ is not necessarily contained in $|\mathcal{T}|$. However, there always exists an inscribed polygonal path contained in $|\mathcal{T}|$.

LEMMA 17. *For any path $\pi : [a, b] \to |\mathcal{T}|$, there exists an $[a, b]$-sequence σ such that $\pi_\sigma \subset |\mathcal{T}|$.*

Proof. Let F denote the set of edges e of $\mathcal{T}$ such that $e \cap \pi(a, b) \neq \emptyset$. We prove the lemma by induction on $|F|$. If $F = \emptyset$, then $\pi(a, b)$ lies in a face of $\mathcal{T}$ and we are done. Otherwise, let S denote the union of the faces that contain $\pi(a)$. Note that S is star-shaped around a; i.e., for any $x \in S$, we have $\overline{ax} \subset S$. If $\pi(b) \in S$, we are done (choose $\sigma = (a, b)$). Otherwise, let $c = \max\{t : \pi(t) \in S\}$. Then $\pi(c, b)$ does not intersect any edge e in F such that $e \subset S$. Thus, by our induction hypothesis, there is a $[c, b]$-sequence σ' such that $\pi_{\sigma'} \subset |\mathcal{T}|$. Choosing σ as the concatenation of (a) and σ', we conclude that $\pi_\sigma \subset |\mathcal{T}|$. ∎

6.3. Cost distance. We introduce a new distance function and prove some of its properties. This distance function yields a measure of the cost of a path. We prove that this new cost measure coincides with the path cost defined in section 2 in the case of polygonal paths. We also prove the existence of an optimal rectifiable path under this cost measure, and so the optimal path can be approximated using our algorithmic result in section 3.

Let x, y be two points in $|\mathcal{T}|$. The *cost distance* from x to y, denoted by $\mathrm{d}(x, y)$, is defined as follows:

$$\mathrm{d}(x, y) = \inf\{\text{cost}(P) : P \text{ is a polygonal path from } x \text{ to } y \text{ and } P \subset |\mathcal{T}|\}.$$

The distance function $\mathrm{d}(\cdot, \cdot)$ is well defined, as the cost of a polygonal path is well defined and positive. It has several useful properties, listed in the following lemma. In particular, it is continuous. On the contrary, $\text{cost}(\overline{xy})$ may not be continuous in (x, y), for instance, when we move $\overline{xy}$ from the interior of a face to its boundary.

LEMMA 18. *The cost distance has the following properties.*

(i) *For any $x, y \in |\mathcal{T}|$, $x = y$ if and only if $\mathrm{d}(x, y) = 0$.*
(ii) *For any $x, y, z \in |\mathcal{T}|$, $\mathrm{d}(x, z) \leqslant \mathrm{d}(x, y) + \mathrm{d}(y, z)$.*
(iii) *For any $x, y \in |\mathcal{T}|$, $\|xy\| \leqslant \mathrm{d}(x, y)$.*
(iv) *$\mathrm{d}(\cdot, \cdot)$ is continuous over $|\mathcal{T}|^2$.*
(v) *For any $x, z \in |\mathcal{T}|$, there exists $y \in |\mathcal{T}|$ such that $\mathrm{d}(x, y) = \mathrm{d}(y, z) = \mathrm{d}(x, z)/2$.*

Proof. The correctness of (i), (ii), and (iii) follow from the definition of $\mathrm{d}(\cdot,\cdot)$ and the triangle inequality.

We now prove (iv). Let x_0 and y_0 be two points in $|\mathcal{T}|$. Let D_δ and D'_δ denote the disks centered at x_0 and y_0 with radius δ, respectively. We assume that $\delta > 0$ is sufficiently small so that $D_\delta \cap |\mathcal{T}|$ and $D'_\delta \cap |\mathcal{T}|$ are star-shaped around x_0 and y_0, respectively. So, for any $x \in D_\delta \cap |\mathcal{T}|$ and for any $y \in D'_\delta \cap |\mathcal{T}|$, we have $|\mathrm{d}(x,y) - \mathrm{d}(x_0,y_0)| \leqslant \mathrm{d}(x,x_0) + \mathrm{d}(y_0,y) \leqslant \rho\,\|x_0x\| + \rho\,\|y_0y\|$. It follows that $\lim_{(x,y)\to(x_0,y_0)} \mathrm{d}(x,y) = \mathrm{d}(x_0,y_0)$. Thus $\mathrm{d}(\cdot,\cdot)$ is continuous over $|\mathcal{T}|^2$.

We now prove (v). By definition, for any $n \in \mathbb{N}$, there exists a polygonal path $A_n \subset |\mathcal{T}|$ from x to z such that $\mathrm{cost}(A_n) \leqslant \mathrm{d}(x,z) + 2/n$. There is a point y_n on A_n such that the cost of the subpath from x to y_n is $\mathrm{cost}(A_n)/2$. Then $\mathrm{d}(x,y_n) \leqslant \mathrm{d}(x,z)/2 + 1/n$ and $\mathrm{d}(y_n,z) \leqslant \mathrm{d}(x,z)/2 + 1/n$. Since $|\mathcal{T}|$ is compact, $\{y_n\}$ has a limit point, which we denote by y. By (iv), the cost distance $\mathrm{d}(\cdot,\cdot)$ is continuous; thus $\mathrm{d}(x,y) \leqslant \lim_{n\to\infty} \mathrm{d}(x,z)/2 + 1/n = \mathrm{d}(x,z)/2$. Similarly, $\mathrm{d}(y,z) \leqslant \mathrm{d}(x,z)/2$. We complete the proof using (ii). ∎

Properties (i) and (ii) show that $\mathrm{d}(\cdot,\cdot)$ is a quasi metric: it is similar to a metric, except that it is not symmetric.

6.4. Cost of a rectifiable path. We define a measure of path cost using the distance function $\mathrm{d}(\cdot,\cdot)$. Our definition is similar to the definition of the length of a rectifiable path. The exposition of this section follows the lecture notes by Lang [10] on metric geometry and the book by D. Burago, Y. Burago, and Ivanov [3]. The results in these notes and this book do not apply to our problem directly, because $\mathrm{d}(\cdot,\cdot)$ is not a metric, so we reprove the results we need.

The d-*cost* of a path $\pi : [a,b] \to |\mathcal{T}|$ is defined as

$$C(\pi) = \sup\left\{ \sum_{i=1}^{m} \mathrm{d}(\pi(t_{i-1}),\pi(t_i)) : \sigma = (t_0,\ldots,t_m) \text{ is an } [a,b]\text{-sequence such that } \pi_\sigma \subset |\mathcal{T}| \right\}.$$

By Lemma 17, the above supremum is well defined. We say that π is $\mathcal{T}$-*rectifiable* if $C(\pi)$ is finite.

The above definition of the d-cost works for paths specified as maps. We can extend it to polygonal paths as follows. Let $P = (p_0,\ldots,p_m)$ be a polygonal path. Construct the map $\pi : [0,1] \to |\mathcal{T}|$ such that $\pi(i/m) = p_i$ for any integer $i \in [0,m]$, and π is affine over $[(i-1)/m, i/m]$ for any integer $i \in [1,m]$. Then we define $C(P) = C(\pi)$. The following lemma shows that $C(P)$ coincides with $\mathrm{cost}(P)$ as defined in section 2.

LEMMA 19. *For any polygonal path P, we have $C(P) = \mathrm{cost}(P)$.*

Proof. We associate P with a function $\pi : [0,1] \to |\mathcal{T}|$ as explained earlier. Since $\pi[t,t']$ is a polygonal path from $\pi(t)$ to $\pi(t')$, by the definition of $\mathrm{d}(\cdot,\cdot)$, we know that for any $t < t'$ in $[0,1]$, $\mathrm{d}(\pi(t),\pi(t')) \leqslant \mathrm{cost}(\pi[t,t'])$. It follows that $C(P) \leqslant \mathrm{cost}(P)$.

We now prove that $C(P) \geqslant \mathrm{cost}(P)$. We put a ball with radius δ centered at each node of P, each crossing between P and the edges of $\mathcal{T}$, each crossing between links of P, and each vertex of $\mathcal{T}$ lying on P. We make δ sufficiently small so that the balls are mutually disjoint, each ball intersects only the link(s) of P that contain the ball center, and each ball intersects only the edge(s) and face(s) of $\mathcal{T}$ that contain the ball center.

Assume that there are k links in P. Since $\mathcal{T}$ has $O(n)$ vertices and edges, there are $O(k^2 + kn)$ balls. We denote by $\mathcal{B}$ the set of points in these balls. Clearly, $P \setminus \mathcal{B}$ is a collection of disjoint line segments, each lying in the interior of an edge or a face of $\mathcal{T}$.

Pick a line segment $\overline{xy}$ in $P \setminus \mathcal{B}$. First, assume that $\overline{xy}$ lies in the interior of a face f of $\mathcal{T}$. Let ε be the Euclidean distance between $\overline{xy}$ and $\mathrm{bd}(f)$. Let $[a, b]$ be the subinterval of $[0, 1]$ such that $\pi[a, b] = \overline{xy}$. Consider any $[a, b]$-sequence $(s_0, \ldots, s_\ell)$ such that $\|\overline{\pi(s_{i-1})\pi(s_i)}\| < \varepsilon/\rho$ for any i. So $\mathrm{cost}(\overline{\pi(s_{i-1})\pi(s_i)}) < \varepsilon$ for any i. Consider any polygonal path from $\pi(s_{i-1})$ to $\pi(s_i)$. If the path stays in $\mathrm{int}(f)$, its cost is no less than $\mathrm{cost}(\overline{\pi(s_{i-1})\pi(s_i)})$. If the path reaches $\mathrm{bd}(f)$, then the path cost is at least $\varepsilon > \mathrm{cost}(\overline{\pi(s_{i-1})\pi(s_i)})$. It follows that $\mathrm{d}(\pi(s_{i-1}), \pi(s_i)) = \mathrm{cost}(\overline{\pi(s_{i-1})\pi(s_i)})$ for any i. If $\overline{xy}$ lies in the interior of an edge of $\mathcal{T}$, let F be the union of the face(s) incident to e and let ε be the distance between $\overline{xy}$ and $\mathrm{bd}(F) \setminus \mathrm{int}(e)$. Again, we obtain $\mathrm{d}(\pi(s_{i-1}), \pi(s_i)) = \mathrm{cost}(\overline{\pi(s_{i-1})\pi(s_i)})$ by considering separately polygonal paths that stay in $\mathrm{int}(F) \cup \mathrm{int}(e)$ and polygonal paths that reach $\mathrm{bd}(F) \setminus \mathrm{int}(e)$.

Consider any $[0, 1]$-sequence $\sigma = (t_0, \ldots, t_m)$ such that $\pi_\sigma \subset |\mathcal{T}|$ and for each endpoint x of a segment in $P \setminus \mathcal{B}$, there exists $i \in [0, m]$ such that $\pi(t_i) = x$. This implies that if $[a, b]$ is a subinterval of $[0, 1]$ such that $\pi[a, b] = \overline{xy}$ for a segment $\overline{xy}$ in $P \setminus \mathcal{B}$, then σ contains a $[a, b]$-sequence. Then the result in the previous paragraph implies that $\sum_{i=1}^{m} \mathrm{d}(\pi(t_{i-1}), \pi(t_i)) \geqslant \sum_{\overline{xy}} \mathrm{cost}(\overline{xy})$, where the second sum runs over all segments $\overline{xy}$ in $P \setminus \mathcal{B}$. Since $C(P) = C(\pi)$ is the supremum over all $[0, 1]$-sequences that yields a polygonal path in $|\mathcal{T}|$, we conclude that $C(P) = C(\pi) \geqslant \sum_{\overline{xy}} \mathrm{cost}(\overline{xy})$.

The intersection $P \cap \mathcal{B}$ has a total length of $O(\delta(k^2 + kn))$. Therefore, $\mathrm{cost}(P \cap \mathcal{B}) \leqslant c\rho\delta(k^2 + kn)$ for some constant $c > 0$. By the triangle inequality, $\sum_{\overline{xy}} \mathrm{cost}\,\overline{xy} \geqslant \mathrm{cost}(P) - c\rho\delta(k^2 + kn)$, which tends to $\mathrm{cost}(P)$ as $\delta \to 0$. Hence, $C(P) \geqslant \mathrm{cost}(P)$. ∎

Consider the infimum of $C(\pi)$ over all rectifiable paths π from a point x to another point y in $|\mathcal{T}|$. Our goal is to show that some rectifiable path from x to y achieves this infimum and hence this path is shortest. First, we show that this infimum is equal to $\mathrm{d}(x, y)$.

LEMMA 20. *For any $x, y \in |\mathcal{T}|$,*

$$\mathrm{d}(x, y) = \inf\{C(\pi) : \pi \text{ is a rectifiable path from } x \text{ to } y\}.$$

Proof. By definition, $\mathrm{d}(x, y)$ is the infimum of $\mathrm{cost}(P)$ over all polygonal paths P from x to y in $|\mathcal{T}|$. A polygonal path P is rectifiable and $C(P) = \mathrm{cost}(P)$ by Lemma 19. Therefore, $\mathrm{d}(x, y) \geqslant \inf\{C(\pi) : \pi \text{ is a rectifiable path from } x \text{ to } y\}$. Assume to the contrary that $\mathrm{d}(x, y) > \inf\{C(\pi) : \pi \text{ is a rectifiable path from } x \text{ to } y\}$. Then there exists a rectifiable path $\pi' : [a, b] \to |\mathcal{T}|$ from x to y such that $\mathrm{d}(x, y) > C(\pi')$. It follows from the definition of $C(\pi')$ that there exists an $[a, b]$-sequence $\sigma = (t_0, \ldots, t_m)$ such that

$$\mathrm{d}(x, y) > \sum_{i=1}^{m} \mathrm{d}(\pi'(t_{i-1}), \pi'(t_i)).$$

However, this is impossible by Lemma 18(ii). ∎

Next, we show that $\mathcal{T}$-rectifiability is equivalent to rectifiability. It follows that any rectifiable path has a finite d-cost (in particular, any piecewise C^1 path).

LEMMA 21. *A path π in $|\mathcal{T}|$ is rectifiable if and only if π is $\mathcal{T}$-rectifiable.*

Proof. Suppose that a path $\pi : [a, b] \to |\mathcal{T}|$ is rectifiable. By definition, $C(\pi)$ is equal to the supremum of $\sum_{i=1}^{m} \mathrm{d}(\pi(t_{i-1}), \pi(t_i))$ over all $[a, b]$-sequences $\sigma = (t_0, \ldots, t_m)$

such that $\pi_\sigma \subset |\mathcal{T}|$. By Lemmas 19 and 20, we have

$$\mathrm{d}(\pi(t_{i-1}), \pi(t_i)) \leqslant C\left(\overline{\pi(t_{i-1})\pi(t_i)}\right) = \mathrm{cost}\left(\overline{\pi(t_{i-1})\pi(t_i)}\right).$$

Thus, $C(\pi) \leqslant \sup_\sigma \mathrm{cost}(\pi_\sigma) \leqslant \sup_\sigma \rho\ \mathrm{length}(\pi_\sigma)$. By definition,

$$\mathrm{length}(\pi) \geqslant \mathrm{length}(\pi_\sigma).$$

Therefore, $C(\pi) \leqslant \rho\ \mathrm{length}(\pi)$ which is finite as π is rectifiable. So π is $\mathcal{T}$-rectifiable.

Suppose that a path $\pi : [a, b] \to |\mathcal{T}|$ is $\mathcal{T}$-rectifiable. By definition, $\mathrm{length}(\pi) = \sup_\sigma \mathrm{length}(\pi_\sigma)$, where the supremum is taken over all $[a, b]$-sequences. For each such σ, by applying Lemma 17 to successive numbers in σ, we can find an $[a, b]$-sequence σ' such that σ is a subsequence of σ' (and thus $\mathrm{length}(\pi_\sigma) \leqslant \mathrm{length}(\pi_{\sigma'})$) and $\pi_{\sigma'} \subset |\mathcal{T}|$. Therefore, it is also true that $\mathrm{length}(\pi) = \sup_\sigma \mathrm{length}(\pi_\sigma)$ over all $[a, b]$-sequences σ such that $\pi_\sigma \subset |\mathcal{T}|$. By Lemma 18(iii), we have $\mathrm{length}(\pi_\sigma) \leqslant \sum_{i=1}^{m} \mathrm{d}(\pi(t_{i-1}), \pi(t_i))$ for any $\sigma = (t_0, \ldots, t_m)$. Therefore,

$$\mathrm{length}(\pi) = \sup_\sigma \mathrm{length}(\pi_\sigma) \leqslant \sup_\sigma \left\{ \sum_{i=1}^{m} \mathrm{d}(\pi(t_{i-1}), \pi(t_i)) \right\}$$

over all $[a, b]$-sequences $\sigma = (t_0, \ldots, t_m)$ such that $\pi_\sigma \subset |\mathcal{T}|$. By definition, the right-hand side is equal to $C(\pi)$, which is finite as π is $\mathcal{T}$-rectifiable. So $\mathrm{length}(\pi)$ is finite and π is rectifiable. □

We are ready to show that there exists a shortest rectifiable path from x to y. The proof is analogous to the proof of the midpoint lemma in the lecture notes by Lang [10] and the proof in the book by D. Burago, Y. Burago, and Ivanov [3, Theorem 2.4.16].

THEOREM 22. *For any $x, y \in |\mathcal{T}|$, there exists a rectifiable path π^* from x to y such that*

$$C(\pi^*) = \inf\{C(\pi) : \pi \text{ is a rectifiable path from } x \text{ to } y\}.$$

Proof. We claim that it suffices to prove the existence of a path $\pi^* : [0, 1] \to |\mathcal{T}|$ from x to y such that $C(\pi^*) \leqslant \mathrm{d}(x, y)$. Notice that, if this is true, π^* is $\mathcal{T}$-rectifiable and hence rectifiable by Lemma 21. Then $\mathrm{d}(x, y) \leqslant C(\pi^*)$ by Lemma 20 and so $C(\pi^*) = d(x, y)$. We assume, without loss of generality, that $\mathrm{d}(x, y) = 1$.

Let $U = \{i/2^j : i, j \in \mathbb{N} \text{ and } 0 \leqslant i/2^j \leqslant 1\}$. First, we recursively define π^* over U, starting with $\pi^*(0) = x$ and $\pi^*(1) = y$. By Lemma 18(v), we can choose $\pi^*(1/2)$ such that $\mathrm{d}(\pi^*(0), \pi^*(1/2)) = \mathrm{d}(\pi^*(1/2), \pi^*(1)) = 1/2$. We repeat this process recursively: we choose $\pi^*(3/4)$ such that $\mathrm{d}(\pi^*(1/2), \pi^*(3/4)) = \mathrm{d}(\pi^*(3/4), \pi^*(1)) = 1/4$, and so on. This completes the definition of π^* over U.

With this construction, for any $r < r' \in U$, we have $\mathrm{d}(\pi^*(r), \pi^*(r')) \leqslant r' - r$. Thus, by Lemma 18(iii), we have $\|\pi^*(r)\pi^*(r')\| \leqslant r' - r$. In other words, π^* is Lipschitz over U. As U is dense in $[0, 1]$, we can extend π^* to a Lipschitz (and thus continuous) function over $[0, 1]$, by taking

$$\forall t \in [0, 1], \pi^*(t) = \lim_{\substack{r \in U \\ r \to t}} \pi^*(r).$$

By Lemma 18(iv), we know that $\mathrm{d}(\cdot, \cdot)$ is continuous. Thus, as $\mathrm{d}(\pi^*(r), \pi^*(r')) \leqslant r' - r$ for any $r < r' \in U$, we conclude that $\mathrm{d}(\pi^*(t), \pi^*(t')) \leqslant t' - t$ for any $t < t' \in [0, 1]$.

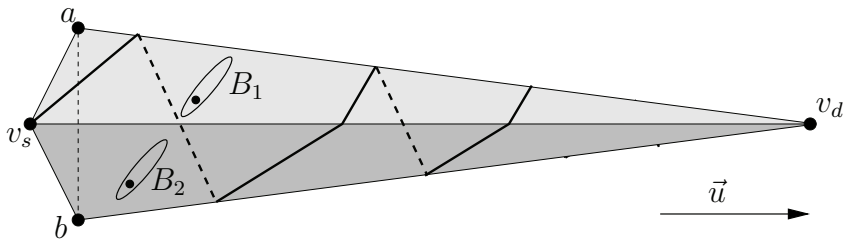


FIG. 9. *The terrain is formed by triangles abv_d, av_sv_d, and bv_sv_d. The ellipses B_1 and B_2 are the unit balls of faces av_sv_d and bv_sv_d, respectively. The vertices a, b, and v_s are fixed, and the convex distance functions are fixed. When v_d goes to infinity in direction $\vec{u}$, the number of times a* 1.01*-approximate shortest path has to turn around the axis $(v_d, \vec{u})$ goes to infinity.*

Finally, by definition, $C(\pi^*) = \sup_\sigma\{\sum_{i=1}^m \mathrm{d}(\pi^*(t_{i-1}), \pi^*(t_i))\}$ over all $[0,1]$-sequences $\sigma = (t_0, \ldots, t_m)$ such that $\pi_\sigma \subset |\mathcal{T}|$. We have proved that

$$\mathrm{d}(\pi^*(t_{i-1}), \pi^*(t_i)) \leqslant t_i - t_{i-1}.$$

So $\sum_{i=1}^m \mathrm{d}(\pi^*(t_{i-1}), \pi^*(t_i)) \leqslant 1$. Hence, $C(\pi^*) \leqslant 1 = \mathrm{d}(x, y)$. □

By the definition of a $\mathcal{T}$-rectifiable path, the following corollary holds.

COROLLARY 23. *For any $\alpha > 1$ and for any $x, y \in |\mathcal{T}|$, there exists a polygonal path S^α from x to y such that $\mathrm{cost}(S^\alpha) = C(S^\alpha) \leqslant \alpha\, C(\pi^*)$, where π^* is a shortest rectifiable path from x to y.*

Theorem 22 and Corollary 23 allow us to apply the results in sections 3 and 5 to approximate the shortest rectifiable path. We summarize our main results in the following.

- Theorem 22 shows that there is a shortest rectifiable path. Furthermore, Corollary 23 shows that, for any $\alpha > 1$, there exists a polygonal path with cost at most α times the optimal.
- Corollary 23 and Theorem 16 imply that, for any $\varepsilon \in (0,1)$, there exists a $(21\rho n^2/\varepsilon)$-link path with cost at most $(1+\varepsilon)$ times the optimal.
- Corollary 23 and Theorem 16 allow us to apply Theorem 5 to approximate shortest paths in anisotropic regions. As stated in Corollary 7, for any $\varepsilon \in (0,1)$, we can compute in time $O(\frac{\rho^2 \log \rho}{\varepsilon^2} n^3 \log(\frac{\rho n}{\varepsilon}))$ a polygonal path S with cost at most $(1+\varepsilon)$ times the optimal.

7. Conclusion. We have given algorithms for shortest path problems in planar subdivisions. A natural question is whether our results can be generalized to higher dimensions. The algorithms for k-link paths that we presented in section 3, as well as the proof of the existence of a shortest rectifiable path (section 6), generalize directly to the case where $\mathcal{T}$ is a 2D simplicial complex properly embedded in $\mathbb{R}^d$ (for any integer d).

However, our bound $O(\rho n^2/\varepsilon)$ on the number of links of an approximate shortest path (Theorem 16) does not generalize to higher dimensions. It does not even generalize to the case of a terrain. See the example shown in Figure 9, where $n = 4$ and ρ is fixed. The number of edges in a 1.01-approximate shortest path goes to infinity when v_d goes to infinity in direction $\vec{u}$.

REFERENCES

[1] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, *Approximation algorithms for geometric shortest path problems*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 286–295.

[2] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, *Determining approximate shortest paths on weighted polyhedral surfaces*, J. ACM, 52 (2005), pp. 25–53.

[3] D. Burago, Y. Burago, and S. Ivanov, *A Course in Metric Geometry*, AMS Providence, RI, 2001.

[4] P. Chew and R. Dyrsdale, *Voronoi diagrams based on convex distance functions*, in Proceedings of the First Annual Symposium on Computational Geometry, ACM, New York, 1985, pp. 235–244.

[5] O. Daescu, J. Mitchell, S. Ntafos, J. Palmer, and C.-K. Yap, *k-link shortest paths in weighted subdivisions*, in Algorithms and Data Structures, Lecture Notes in Comput. Sci. 3608, Springer, Berlin, 2005, pp. 325–337.

[6] H. Edelsbrunner, *Geometry and Topology for Mesh Generation*, Cambridge University Press, Cambridge, UK, 2001.

[7] S. Fortune, *Voronoi diagrams and Delaunay triangulations*, in Computing in Euclidean Geometry, D.-Z. Du and F. Hwang, eds., World Scientific, River Edge, NJ, 1995, pp. 193–233.

[8] M. Fredman and R. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.

[9] H. W. Guggenheimer, *Differential Geometry*, Dover, New York, 1977.

[10] U. Lang, *Length Spaces*, lecture notes, 2006, www.math.ethz.ch/~lang/mg.pdf.

[11] M. Lanthier, A. Maheshwari, and J.-R. Sack, *Shortest anisotropic paths on terrains*, in Proceedings of the 26th International Colloquium on Automata, Languages and Programming, Springer, London, 1999, pp. 524–533.

[12] J. Mitchell, *Geometric shortest paths and network optimization*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 633–701.

[13] J. Mitchell and C. Papadimitriou, *The weighted region problem: Finding shortest paths through a weighted planar subdivision*, J. ACM, 38 (1991), pp. 18–73.

[14] T. Nishizeki, *Planar Graphs: Theory and Algorithms*, North–Holland, Amsterdam, 1988.

[15] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer, New York, 1985.

[16] J. Reif and Z. Sun, *Movement planning in the presence of flows*, Algorithmica, 39 (2004), pp. 127–153.

[17] J. Sellen, *Direction weighted shortest path planning*, in Proceedings of the IEEE International Conference on Robotics and Automation, 1995, pp. 1970–1975.

[18] Z. Sun and J. Reif, *On finding energy-minimizing paths on terrains*, IEEE Trans. Robotics, 21 (2005), pp. 102–114.

[19] Z. Sun and J. Reif, *On finding approximate optimal paths in weighted regions*, J. Algorithms, 58 (2006), pp. 1–32.

[20] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. Gortler, and H. Hoppe, *Fast exact and approximate geodesics on meshes*, ACM Trans. Graph., 24 (2005), pp. 553–560.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 825–840

A PRIMAL-DUAL BICRITERIA DISTRIBUTED ALGORITHM FOR CAPACITATED VERTEX COVER*

F. GRANDONI†, J. KÖNEMANN‡, A. PANCONESI§, AND M. SOZIO¶

Abstract. In this paper we consider the capacitated vertex cover problem, which is the variant of vertex cover where each node is allowed to cover a limited number of edges. We present an efficient, deterministic, distributed approximation algorithm for the problem. Our algorithm computes a $(2+\epsilon)$-approximate solution which violates the capacity constraints by a factor of $(4+\epsilon)$ in a polylogarithmic number of communication rounds. On the other hand, we also show that every efficient distributed approximation algorithm for this problem must violate the capacity constraints. Our result is achieved in two steps. We first develop a 2-approximate, sequential primal-dual algorithm that violates the capacity constraints by a factor of 2. Subsequently, we present a distributed version of this algorithm. We demonstrate that the sequential algorithm has an inherent need for synchronization which forces any naive distributed implementation to use a linear number of communication rounds. The challenge in this step is therefore to achieve a reduction of the communication complexity to a polylogarithmic number of rounds without worsening the approximation guarantee.

Key words. vertex cover, approximation algorithms, distributed algorithms, primal-dual algorithms

AMS subject classifications. 68W15, 68W25, 68W40, 05C70, 05C85

DOI. 10.1137/06065310X

1. Introduction.

The *capacitated vertex cover* problem (capVC) is the variant of vertex cover in which there is a limit on the number of edges that a vertex can cover. A precise formulation of the problem is as follows. We are given an n-vertex undirected graph $G=(V,E)$, nonnegative weights wt_v, and vertex capacities $B_v \geq 1$ for all vertices $v \in V$. A solution to a given capVC instance consists of a subset $C \subseteq V$ and an assignment $\pi : E \to C$ of edges to vertices such that

1. $\pi(e) \in \{u,v\} \cap C \ \forall$ edges $e=(u,v) \in E$, and
2. $|\pi^{-1}(v)| \leq B_v \ \forall\ v \in C$.

The first set of constraints says that every edge must be covered by some vertex in the cover C. The second condition limits the number of edges that can be assigned to any cover vertex v to B_v. The goal is to find a feasible solution that has minimum total weight

$$\mathrm{wt}(C) := \sum_{v \in C} \mathrm{wt}_v.$$

*Received by the editors February 28, 2006; accepted for publication (in revised form) January 28, 2008; published electronically May 28, 2008. A preliminary version of this paper appeared in *Proceedings of the 24th Symposium on Principles of Distributed Computing*, Las Vegas, NV, 2005.
http://www.siam.org/journals/sicomp/38-3/65310.html

†Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, 00133, Roma, Italy (grandoni@disp.uniroma2.it).

‡Department of Combinatorics and Optimization, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada (jochen@uwaterloo.ca). This work was done while being on leave at the Dipartimento di Informatica, Sapienza Università di Roma.

§Dipartimento di Informatica, Sapienza Università di Roma, Via Salaria 113, 00198, Rome, Italy (ale@di.uniroma1.it).

¶Department of Databases and Information Systems, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany (msozio@mpi-inf.mpg.de).

We emphasize the difference between the above *hard-capacity* version of capVC and its *soft-capacity* counterpart (capVC$_s$): in capVC$_s$, each vertex $v \in V$ may be included $x_v \geq 0$ times in a cover. Vertex v then contributes $x_v \mathrm{wt}_v$ to the weight of the cover, and the maximum number of edges that can be assigned to v is $x_v B_v$.

In this paper we present *bicriteria* sequential and distributed approximation algorithms for the (hard) capacitated vertex cover problem. Given a feasible instance of the problem of optimal weight opt, our sequential primal-dual algorithm computes a vertex cover C of weight $\sum_{v \in C} \mathrm{wt}_v \leq 2\mathrm{opt}$, which assigns at most $2B_v$ edges to each cover vertex $v \in C$. Note that, differently from the hard-capacity case, capacities might be violated. However, the amount of the violation is bounded, which is not the case for capVC$_s$. We also remark that, unlike capVC$_s$, every $v \in C$ contributes wt_v to the weight of the cover even when its capacity B_v is exceeded.

The distributed implementation of our method has an additional input parameter $\epsilon > 0$ and computes a cover of weight at most $(2+\epsilon)\mathrm{opt}$ that violates the capacity bound of each cover vertex by a factor of at most $(4+\epsilon)$. In the synchronous, message-passing model of computation, the distributed algorithm takes $O(\log(nW)/\epsilon)$ many rounds, where

$$W = \mathrm{wt}_{max}/\mathrm{wt}_{min}$$

is the ratio of largest to smallest vertex weight in the given instance. This reduces to $O(\log n/\epsilon)$ for the interesting case of unit weights. We remark that our algorithm is deterministic, while typically efficient distributed algorithms for graph problems require randomization (see [11, 20, 22, 23, 26, 27] among others).

Observe that any sublinear distributed algorithm for capVC must violate the capacity constraints. Consider for instance a ring where every vertex has unit capacity. A feasible solution provides a consistent orientation of the ring, something that requires a linear number of communication rounds. Therefore a bicriteria solution is the best one can hope for in a distributed setting. In this paper we show that indeed every efficient distributed approximation algorithm for capVC must violate the capacity constraints by a large additive term.

In our opinion the most interesting aspect of our work is that the distributed algorithm is derived in a systematic fashion from a sequential primal-dual algorithm. To our knowledge, the first result of this kind is the $(2+\epsilon)$-approximate vertex cover algorithm described in [16]. Although described for the PRAM setting, the algorithm can be easily adapted to the distributed case. Our paper takes the primal-dual approach pioneered in [16] one step further, giving a new and considerably more sophisticated example. Chudak, Erlebach, and Panconesi [4] recently showed that the techniques introduced in this paper can be extended to yield efficient distributed primal-dual algorithms for vertex cover with soft capacities and for the facility location problem. The power of the primal-dual method in the design of approximation algorithms is well established. In this paper we provide further evidence to the fact that it is also a valuable tool in the design of distributed algorithms.

Capacity constraints arise naturally in distributed computing and computer networking; e.g., the scatternet-formation problem of ad hoc Bluetooth networks asks for a small dominating set where each vertex in the set dominates at most 7 vertices [6]. More generally, a small dominating set can act as the backbone of the routing infrastructure of an ad hoc network (see [28, 30] and the references therein). Capacities model computational and energy limitations and provide effective means to enforce load distribution among the vertices of the backbone. To the best of our knowledge,

our paper is the first result that considers a capacitated network design problem from the distributed computing point of view. Recently, Moscibroda and Kuhn gave an LP-based, bicriteria distributed solution to the capacitated dominating set problem [17].

Related work. Vertex covers with capacities has received considerable attention in recent years in the sequential setting, while our main motivation is to study it from a distributed point of view. The capacitated vertex-cover problem was first introduced by Guha et al. [12], who presented a simple 4-approximate LP-rounding based algorithm for capVC$_s$. Later on, the authors showed a 2-approximate primal-dual algorithm. Subsequently, Gandhi et al. [10] presented a 2-approximate LP-rounding algorithm for capVC$_s$.

The hard-capacitated vertex-cover problem is significantly harder than its soft-capacitated variant. Chuzhoy and Naor [5] first gave a sophisticated 3-approximate LP-rounding algorithm for the special case of capVC with uniform vertex weights. Finally, in [9], Gandhi et al. presented an LP-rounding-based 2-approximation algorithm for capVC with uniform weights.

In [5], Chuzhoy and Naor also showed that capVC in the presence of nonuniform vertex weights is as hard to approximate as set-cover. Lund and Yannakakis [24] proved that there is no $o(\log n)$-approximation for the latter problem unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$) (see also [8] for a refined result). Based on work by Bellare et al. [3] and Raz and Safra [29], Alon, Moshkovitz, and Safra [1] recently improved upon this result and showed that no $o(\log n)$-approximation for the set-cover problem is possible unless P = NP. Chuzhoy and Naor's work implies that these hardness bounds translate to the capVC problem.

The best known approximation algorithm for the vertex-cover problem without capacity constraints is due to Karakostas [15], who presented a $2 - \Theta(1/\sqrt{\log(n)})$-approximation for the problem. This improves upon earlier $(2 - o(1))$-approximation algorithms due to Halperin [13], Bar-Yehuda and Even [2], and Hochbaum [14]. As mentioned, the same bound is essentially achievable in the distributed setting [16].

Unconditional lower-bounds based on communication constraints, as opposed to unproven complexity theoretic assumptions, have been proved since the early stages [21, 25]. For more recent work, see [18]. Also, Elkin [7] recently established trade-offs between the performance guarantee of a distributed approximation algorithm for the minimum-cost spanning tree problem and the number of communication rounds it needs.

Finally, we mention that LP-duality has been previously used to design distributed algorithms for the dominating set problem [19, 27].

Our contribution. The first result we give is a bicriteria primal-dual approximation algorithm for capVC.

THEOREM 1. *Given a feasible* capVC *instance with capacities* $B_v \geq 1$ *for all* $v \in V$*, there is a polynomial-time primal-dual algorithm that computes a vertex cover* (C, π) *of weight at most* $2opt$ *that assigns at most* $2B_v$ *edges to each vertex* $v \in C$*.*

We remark that if the input instance does not have a feasible solution, then our algorithm either computes a feasible solution for the (capacity) relaxed version of the problem or it terminates with a certificate of infeasibility.

Theorem 1 is a natural step toward proving the main result of this paper.

THEOREM 2. *Given a feasible instance of* capVC *with capacities* $B_v \geq 1$ *for all* $v \in V$*, and let* $\epsilon \in (0, 1]$ *be an input parameter. There is a distributed deterministic algorithm that computes a vertex cover* (C, π) *of weight at most* $(2+\epsilon)$opt *that assigns at most* $(4+\epsilon)B_v$ *edges to each vertex* $v \in C$*. The algorithm needs* $O(\log(nW)/\epsilon)$ *rounds.*

We remark that the message-size of our algorithm is $O(\log n + \log \mathrm{wt}_{max})$.

Note that the running time is strongly polylogarithmic for polynomially large weights only. This includes the important special case of unit weights. Obtaining a strongly polylogarithmic algorithm in general is a challenging open problem.

Similar to the sequential case, if the input instance does not have a feasible solution the algorithm either computes a feasible solution for the relaxed version of the problem, or terminates with a certificate of infeasibility. The latter, however, is necessarily local in nature. That is, some vertices will know that the algorithm has failed, but it requires a linear number of communication rounds to distribute this information across the network in general.

These theorems are complemented by the following lower-bound on the communication complexity of any algorithm for the weighted capVC problem with hard capacities.

THEOREM 3. *Let $B, k \geq 1$ be integer parameters. There is a* capVC *instance with uniform vertex capacities B, for which any distributed approximation algorithm that assigns less than $(1+1/k) \cdot B$ edges to all vertices must take at least k communication rounds.*

This result shows in particular that violating the capacity constraints is necessary and provides a trade-off between violation of capacities and running time.

Organization of this paper. In section 2 we describe a sequential algorithm for the capacitated vertex-cover problem and give a proof of Theorem 1. Section 3 shows how to turn the sequential algorithm into a distributed one. This is done in two steps. First, we show how to convert the sequential algorithm into a distributed one that computes a vertex cover that satisfies the approximation requirement. In this step we assign only a subset of the edges. In the second and final step we assign all of the remaining edges. The proof of Theorem 4 is given in section 4.

2. A sequential primal-dual algorithm. We present a so called *primal-dual* algorithm for the capVC problem. The algorithm and its analysis are based on linear programming duality. In section 3 we therefore introduce a linear programming formulation of the problem together with its dual. Following that we describe our sequential algorithm and conclude this section with an analysis of the presented method.

2.1. A linear programming formulation. The problem can be formulated as an integer program where we introduce a binary indicator variable x_v for each $v \in V$. We let $x_v = 1$ if $v \in C$ and $x_v = 0$ otherwise. For each edge $e = (u, v) \in E$ we introduce two binary variables $y_{e,v}$ and $y_{e,u}$. For $w \in \{u, v\}$ we let $y_{e,w} = 1$ if and only if $\pi(e) = w$. In the following, let $\delta(v)$ be the set of edges incident to vertex $v \in V$ in G:

$$
\begin{aligned}
&\text{(IP)} && \min \quad \sum_{v \in V} \mathrm{wt}_v \cdot x_v && \\
&\text{(1)} && \text{s.t.} \quad y_{e,v} + y_{e,u} \geq 1 && \forall e = (u, v) \in E, \\
& && \quad\ y_{e,w} \leq x_w && \forall e = (u, v) \in E, \\
&\text{(2)} && && \forall w \in \{u, v\} \\
&\text{(3)} && \quad \sum_{e=(v,u) \in \delta(v)} y_{e,v} \leq B_v \cdot x_v && \forall v \in V, \\
&\text{(4)} && \quad\ y_{e,v}, x_v \in \{0, 1\} && \forall e \in E, v \in V.
\end{aligned}
$$

We now let (LP) be the standard LP relaxation obtained from (IP) by replacing the constraints (4) by

$$\begin{aligned} y_{e,v} &\geq 0 && \forall e = (u,v) \in E, \\ 0 \leq x_v &\leq 1 && \forall v \in V. \end{aligned} \tag{5}$$

In the following we use $(i)_v$, $(i)_e$, and $(i)_{e,v}$ to denote constraint (i) for vertex $v \in V$, edge $e \in E$, and pair $(e,v) \in E \times V$, respectively. In the linear-programming dual of (LP) we associate variables $\alpha_e, \beta_{e,w}, \gamma_v$ and ω_v with constraints $(1)_e$, $(2)_{e,w}$, $(3)_v$, and $(5)_v$, respectively. The linear programming (LP) dual of is then

$$\text{(D)} \qquad \max \quad \sum_{e \in E} \alpha_e - \sum_{v \in V} \omega_v$$

$$\text{s.t.} \quad \alpha_e \leq \beta_{e,w} + \gamma_w \qquad \forall e = (u,v) \in E, \ \forall w \in \{u,v\} \tag{6}$$

$$\sum_{e=(u,v) \in E} \beta_{e,v} \leq \mathrm{wt}_v + (\omega_v - B_v \cdot \gamma_v) \qquad \forall v \in V, \tag{7}$$

$$\alpha, \beta, \gamma, \omega \geq 0.$$

2.2. The algorithm. We remark that the following simple LP rounding scheme, similar to that proposed by Guha et al. [12], yields a 2-approximate vertex cover in which each vertex v covers at most $2B_v$ edges: Solve the LP relaxation (LP) and let (x,y) be its optimal solution. The cover set C consists of all vertices v with $x_v \geq 1/2$. For $e = (u,v) \in E$, constraint $(1)_e$ implies that there is $w \in \{u,v\}$ such that $y_{e,w} \geq 1/2$. We assign edge e to vertex w in this case. Clearly, the weight of the vertices in C is at most twice the optimal LP value. Moreover, each vertex $v \in C$ has at most $2B_v$ assigned edges.

We provide an alternate primal-dual algorithm in this section. As we shall see later, this algorithm possesses an efficient distributed implementation.

The high-level idea in primal-dual algorithms is to find a pair of feasible solutions for (D) and (IP). Subsequently, we upper-bound the performance ratio of the algorithm by bounding the multiplicative gap between the objective values of the two solutions. Thus, our goal is to find a primal-dual pair of solutions whose objective functions values are within a small multiplicative constant of each other. Primal-dual algorithms typically construct such a primal-dual pair in an iterative manner: Starting from a trivial feasible dual solution and an infeasible primal one, the algorithm continuously raises the objective value of the dual solution while maintaining its feasibility, and it changes the partial primal solution in order to attain feasibility.

Our primal-dual capVC algorithm starts with the dual feasible solution $\alpha = \beta = \gamma = \omega = 0$ and the infeasible primal solution $x = y = 0$. In order to obtain a feasible vertex cover, we have to a) select a set of cover vertices, and b) assign each edge $e \in E$ to one of its endpoints (which must be in the cover). As is typical in primal-dual approximation algorithms, these decisions are governed by *primal complementary slackness*.

In the following we say that a vertex $v \in V$ is *tight* for a current dual solution $(\alpha, \beta, \gamma, \omega)$ if constraint $(7)_v$ holds with equality. Similarly, a pair $(e,w) \in E \times V$ is tight if constraint $(6)_{e,w}$ is satisfied with equality. Our algorithm will now increase the value of some of the dual variables and, as a consequence, create tight vertices and tight edge-vertex pairs. Tight vertices are candidates for our final cover and we

will eventually choose a subset of these. Each edge e will eventually be assigned to one of its tight endpoints. In particular, edge $e = (u, v) \in E$ will only be assigned to endpoint $w \in \{u, v\}$ if (e, w) is tight. Once an edge is assigned to a vertex, we will remove it from the graph and continue. Similarly, once all edges incident to a certain vertex $v \in V$ have been decided, the vertex is removed from the graph. The algorithm terminates, when the graph is empty and, hence, when all edges have been assigned.

We now describe the algorithm. As customary with primal-dual algorithms, we describe it as a continuous process that can be implemented in polynomial-time by standard techniques. Initially all edges are unassigned. At any given point, the algorithm increases the value of dual variables α_e of all unassigned edges uniformly at the same (unit) rate. Increasing variables α_e for unassigned edges increases the left-hand side of constraints of type (6) and we will have to also increase some of the β and γ variables in order to maintain dual feasibility. We describe the update process for these variables for each vertex $v \in V$ depending on its tightness.

v is nontight. In this case, we increase $\beta_{e,v}$ for all $e \in \delta(v)$ uniformly. Thus, the left- and right-hand side of constraint $(6)_{e,v}$ for all $e \in \delta(v)$ increase at the same rate and feasibility is maintained.

v is tight. If v has at most $2B_v$ incident edges, we add v to the cover, assign all edges in $\delta(v)$ to v, and delete v and the newly assigned edges from G. Otherwise v has more than $2B_v$ incident edges. In this case, we increase ω_v at rate B_v, γ_v at unit rate, and we leave $\beta_{e,v}$ as is for all $e \in \delta(v)$. As a consequence, the left- and right-hand sides of $(7)_v$ remain unchanged, and the left- and right-hand sides of $(6)_{e,v}$ for all $e \in \delta(v)$ change at the same (unit) rate. Feasibility is therefore also maintained in this case.

We emphasize that our algorithm maintains a feasible dual solution for (D) for the original instance. In particular, deleting a vertex v and an edge $e \in \delta(v)$ means that the values of variables $\alpha_e, \beta_{e,v}, \omega_v$, and γ_v are frozen at their current state from this point on in the algorithm. The algorithm terminates when all edges have been assigned.

We demonstrate the algorithm using the example instance in Figure 1(i), where we let $B_u = 2$, $B_v = 3$, and $B_w = \infty$ for all other vertices w. We also choose $\mathrm{wt}_a = 2$, $\mathrm{wt}_u = 5$, $\mathrm{wt}_v = 6$, and all other vertices which have infinite weight. In the following we use V and E to refer to the vertex and edge sets of the given instance.

Running our algorithm for one time unit results in $\alpha_e = 1$ for all $e \in E$ and $\beta_{e,w} = 1$ for all $w \in V$ and for all $e \in \delta(w)$. At this point, constraint $(7)_u$ is tight. As u has $5 > 2 \cdot B_u = 4$ unassigned incident edges, we cannot assign any edge at

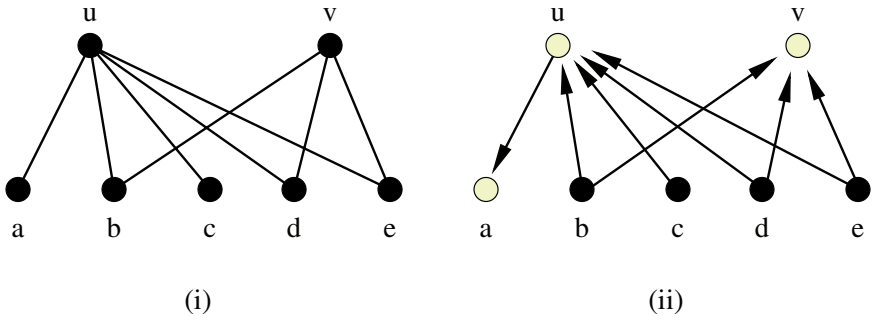


FIG. 1. *An example instance for the primal-dual capVC algorithm.*

this point. Thus, we continue to increase α_e for all edges $e \in E$. Simultaneously, we increase $\beta_{e,w}$ for all $w \in V \setminus \{u\}$ and for all $e \in \delta(w)$ at unit rate, increase ω_u at rate $B_u = 2$, and increase γ_u at unit rate.

After one more time unit, the positive dual variable values are

$$\begin{aligned} \alpha_e &= 2 \quad \forall e \in E, \\ \beta_{e,u} &= 1 \quad \forall e \in \delta(u), \\ \beta_{e,w} &= 2 \quad \forall w \in V \setminus \{u\}, \forall e \in \delta(w), \\ \omega_u &= 2, \\ \gamma_u &= 1. \end{aligned}$$

At this point, vertices a, u, and v are tight. Vertices a and v have one and three incident edges, respectively, and we can hence add both vertices to the cover and delete them and their incident edges from the graph. The number of remaining edges all of which are incident to u is now $4 = 2B_u$. We can assign all of them to u.

Figure 1(ii) shows the computed primal solution; cover vertices are shaded and arc directions indicate edge assignment. We note that the primal solution is feasible for (IP) only if we relax the capacity constraint of vertex u. In fact, it is not hard to see that any feasible solution to (IP) for this instance must have infinite weight.

2.3. Analysis. In this section we present a proof of Theorem 1.

Assume first that the algorithm from section 2.2 does not terminate for a given input instance. It is then not hard to see that the algorithm must reach a point in the execution, where each tight vertex $v \in V$ has a degree of more than $2B_v$ and where each remaining edge is incident to tight vertices on both ends. Using the pigeon-hole principle, it follows that, in any assignment of edges to vertices, there must be at least one tight vertex v that is assigned more than B_v edges. Thus the given input instance is infeasible, and the set of tight vertices together with the set of unassigned edges certifies this fact.

In the following we focus on feasible capVC instances. For such instances our algorithm terminates with a cover $\mathcal{C}$, an assignment $\{y_{e,v}\}_{e \in E, v \in V}$ of edges to vertices in the cover, and a corresponding dual solution $(\alpha, \beta, \omega, \gamma)$. We first show that the dual is feasible for (D).

LEMMA 1. *The dual solution* $(\alpha, \beta, \omega, \gamma)$ *is feasible for* (D).

Proof. We can think of the execution of the algorithm as a process over time: The algorithm starts at time zero and then raises α_e by one for all edges per unit of time. We prove the lemma by induction on (appropriately discretized) time.

Our initial dual solution is clearly feasible. Now consider a later time in the algorithm. Let $\mathcal{O}$ be the set of tight vertices at that time.

For a vertex $v \in V \setminus \mathcal{O}$ and for an edge $e \in \delta(v)$ we raise α_e and $\beta_{e,v}$ simultaneously and hence maintain dual feasibility. For a vertex $v \in \mathcal{O}$ we raise ω_v at a rate of B_v per time unit and raise γ_v at unit rate. For all edges $e \in \delta(v)$ we raise α_e at unit rate as well. It is not hard to see that we maintain dual feasibility this way. ∎

We are ready to prove that our algorithm computes a 2-approximate primal solution.

LEMMA 2. *Our algorithm terminates with a vertex cover* $\mathcal{C}$ *and a corresponding feasible dual solution* $(\alpha, \beta, \gamma, \omega)$ *whenever there exists a feasible solution* (x, y) *for*

(LP). In particular, we must have

$$\sum_{v\in\mathcal{C}} \mathrm{wt}_v \le 2\left(\sum_{e\in E}\alpha_e - \sum_{v\in V}\omega_v\right).$$

Proof. Let $v \in \mathcal{C}$ be a vertex in the computed vertex cover, and let $e \in \delta(v)$ be an edge that is incident to v. Notice that our algorithm always maintains

$$\alpha_e \ge \beta_{e,v} \tag{8}$$

since α_e is raised whenever $\beta_{e,v}$ increases and the rate of increase is the same.

Observe also that γ_v is only increased if the degree $\deg(v)$ of vertex v exceeds $2B_v$. Let $\delta_1(v) \subseteq \delta(v)$ be the set of edges that are incident to v when γ_v is increased for the last time in the algorithm and notice that we must have $|\delta_1(v)| > 2B_v$.

Consider an edge $e \in \delta_1(v)$ and note that γ_v and α_e increase at the same rate after the point of time where v becomes tight. Notice also that the algorithm increases α_e and $\beta_{e,v}$ at the same rate before v becomes tight. Variable $\beta_{e,v}$ is not increased after v becomes tight, and γ_v is not increased before v becomes tight. Therefore, for all $e \in \delta_1(v)$ we must have

$$\alpha_e = \beta_{e,v} + \gamma_v. \tag{9}$$

Since v is tight when the algorithm adds it to the cover and deletes it from the graph, it must also be the case that

$$\sum_{e\in\delta(v)} \beta_{e,v} = \mathrm{wt}_v + \omega_v - B_v\cdot\gamma_v = \mathrm{wt}_v, \tag{10}$$

where the last equality follows from the fact that we raise ω_v at a rate of B_v if and only if we raise γ_v at a rate of 1.

We use $\delta_2(v) = \delta(v) \setminus \delta_1(v)$ and obtain

$$\mathrm{wt}_v \le \sum_{e\in\delta(v)} \beta_{e,v} \le \sum_{e\in\delta_1(v)} (\alpha_e - \gamma_v) + \sum_{e\in\delta_2(v)} \alpha_e \le \sum_{e\in\delta(v)} \alpha_e - 2B_v\gamma_v, \tag{11}$$

where the first inequality uses (10), the second inequality uses (8) and (9), and the last inequality follows from the fact that v is incident to at least $2B_v$ edges whenever γ_v is increased.

Summing (11) over all $v \in \mathcal{C}$ gives

$$\sum_{v\in\mathcal{C}} \mathrm{wt}_v \le \sum_{e\in E} |e\cap\mathcal{C}|\cdot\alpha_e - 2\cdot\sum_{v\in\mathcal{C}} B_v\gamma_v. \tag{12}$$

Now observe that we raise γ_v and ω_v only for tight vertices in our algorithm. Given that the input instance is feasible, the degree of any tight vertex v will eventually drop below $2B_v$. It therefore follows from the algorithm description that any vertex that becomes tight during the execution of the algorithm is eventually included in the vertex cover $\mathcal{C}$. Hence (12) implies

$$\sum_{v\in\mathcal{C}} \mathrm{wt}_v \le \sum_{e\in E} |e\cap\mathcal{C}|\cdot\alpha_e - 2\cdot\sum_{v\in V} B_v\gamma_v.$$

The lemma follows from $B_v\gamma_v = \omega_v$ for all $v \in V$ and from the fact that $|e \cap \mathcal{C}| \le 2$. ∎

Lemmas 1 and 2 complete the proof of Theorem 1.

3. A distributed algorithm. In this section we will describe a distributed primal-dual algorithm for capVC which uses the ideas developed in section 2. As before, the algorithm maintains a pair of (infeasible) primal and (feasible) dual solutions at all times. However, these solutions need to be stored in a distributed fashion: each vertex $v \in V$ stores and manipulates

(a) primal variables x_v and $y_{e,v}$ for all $e \in \delta(v)$, and

(b) dual variables γ_v, ω_v, α_e, and $\beta_{e,v}$ for all $e \in \delta(v)$.

Note that, for every edge $e = (u, v)$, both u and v store a copy of α_e. The algorithm guarantees the consistency of the two copies.

Observe that a naive distributed implementation of the method described in section 2 yields an algorithm that needs a linear number of communication rounds in the worst case. In order to illustrate this, consider a graph G with vertex set $\{v_1, \ldots, v_n, u_1, \ldots, u_{2B}\}$ for some $n, B \geq 1$. Let the edge-set of G be

$$E = \{(v_i, v_{i+1}) : 1 \leq i \leq n-1\} \cup \{(v_i, u_j) : 1 \leq i \leq n, 1 \leq j \leq 2B-1\} \cup \{(v_n, u_{2B})\}$$

and notice that vertex v_1 has a degree of $2B$ while vertices $v_2, \ldots, v_n$ have a degree of $2B + 1$. Let the cost of vertices $v_1, \ldots, v_n$ be 0 and assign a unit cost to all other vertices in G. In the execution of the sequential primal-dual algorithm, all vertices $v_1, \ldots, v_n$ are tight immediately and all other vertices are nontight. Vertex v_1 is the only tight vertex with a degree of at most $2B$. After assigning the $2B$ edges in $\delta(v_1)$ to v_1, the degree of vertex v_2 drops to $2B$. In general, the degree of vertex v_i drops to $2B$ after assigning edges to vertices $v_1, \ldots, v_{i-1}$ for all $1 \leq i \leq n$. Doing this in a distributed fashion takes n communication rounds.

Adapting the algorithm in order to cope with the above synchronization problem is not an easy task. In fact it can be seen that synchronous increase of the duals is at the heart of Lemma 2, where it is used to argue that the dual constraints of type (6) are satisfied with equality at all times.

The distributed algorithm has two main phases.

Vertex-selection. In this phase we compute a vertex cover $\mathcal{C} \subseteq V$ that is $(2 + \epsilon)$-approximate. It is here that we solve the above mentioned synchronization problem. While computing an approximate cover, we also assign part of the edges to the vertices in $\mathcal{C}$. At most $2B_v$ edges are assigned to each $v \in \mathcal{C}$.

Edge-assignment. Here, we assign all of the remaining edges to the vertices in $\mathcal{C}$. This time, at most $(2 + \epsilon)B_v$ edges are assigned to each $v \in \mathcal{C}$.

For ease of presentation we assume from now on that the given capVC instance is feasible.

3.1. Vertex-selection phase. As said, the distributed algorithm mimics the primal-dual algorithm from section 2. Each vertex $v \in V$ stores part of the dual solution and it initially sets $\gamma_v = \omega_v = 0$ and it also lets $\alpha_e = \beta_{e,v} = 0$ for all edges $e \in \delta(v)$.

The distributed algorithm works in rounds. At the beginning of any given round i, we let the *residual weight* wt_v^i of vertex v be the difference between the right- and left-hand sides of $(7)_v$ for the current feasible dual solution. Thus, we initialize wt_v^0 to wt_v for all $v \in V$. A vertex $v \in V$ is either *active* or *inactive* in any given round i. An active vertex v can be in one of two states:

nontight Vertex v is nontight whenever the slack in constraint $(7)_v$ is more than $\theta \cdot \mathrm{wt}_v$ for a parameter $\theta \geq 0$ whose exact value will be determined later.

tight We let the state of vertex v be tight if $\mathrm{wt}_v^i \leq \theta \cdot \mathrm{wt}_v$ and if v has

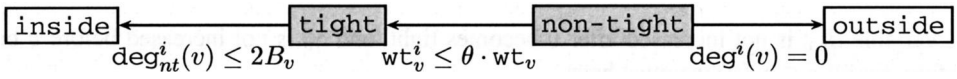


FIG. 2. *The figure shows the possible states of vertex $v \in V$ in the vertex-selection phase. The arrows indicate the possible transitions between the states. Shaded states are active while others are inactive states.*

more than $2B_v$ nontight neighbors. Intuitively, a tight vertex v will eventually be part of the computed cover. It will be assigned a subset of at most $2B_v$ of the edges to its nontight neighbors.

We will say that an edge $(u, v) \in E$ is active in round i if both u and v are active in that round. For any vertex $v \in V$, we let $\deg^i_t(v)$ and $\deg^i_{nt}(v)$ be the number of its tight and nontight neighbors in round i. We also let $\deg^i(v) = \deg^i_t(v) + \deg^i_{nt}(v)$ be the active neighbors of v in round i. An inactive vertex v can be in one of two states:

inside — A tight vertex switches its state to inside if the number $\deg^i_{nt}(v)$ of nontight neighbors is at most $2B_v$. Vertex v will be part of the final cover, and we assign all edges between v and any of its nontight neighbors to vertex v.

outside — We switch the state of a nontight vertex v to outside if it has no tight or nontight neighbors. We will later argue that all neighbors of v in G are inside in this case.

The vertex-selection phase terminates when no active vertices remain. The resulting vertex cover $\mathcal{C}$ consists of all vertices whose final state is inside. Figure 2 illustrates all possible transitions between states in the vertex-selection phase.

We proceed with a detailed description of round i of the distributed algorithm. The round has two steps.

Step 1. All nontight vertices are dormant. Each tight vertex $v \in V$ counts the number of active nontight neighbors. If this number is at most $2B_v$, then we assign all edges connecting v to nontight neighbors to v. We also switch vs state to inside and let v communicate its state-switch to all active neighbors. At this point each active vertex $v \in V$ knows the number $\deg^i(v)$ of active neighbors in G.

Step 2. The behavior of an active vertex $v \in V$ depends on its current state.

v is nontight: If $\deg^i(v)$ is 0, then we know that all edges incident to v have been assigned to other vertices. Therefore, we can switch the state of v to outside.

On the other hand, assume that v has active neighbors. Raising α_e and $\beta_{e,v}$ uniformly by $\mathrm{wt}^i_v/\deg^i(v)$ for all active edges $e \in \delta(v)$ decreases the residual weight of v to 0. Vertex v *strives* for tightness and therefore proposes to any active neighbor u to raise $\alpha_{(u,v)}$ and also $\beta_{(u,v),v}$ by its *proposal*

$$p_v = \frac{\mathrm{wt}^i_v}{\deg^i(v)}.$$

Consider an active edge $e = (u, v) \in \delta(v)$. We raise α_e and $\beta_{e,v}$ by $\min\{p_u, p_v\}$ and decrease the residual weight wt^i_v of v by the same amount.

v is tight: Notice that step 1 guarantees that v has more than $2B_v$ nontight neighbors. Vertex v receives proposals from all such neighbors and lets p_v be their minimum. Vertex v then sends p_v to all such neighbors.

For all nontight neighbors u of v we increase $\alpha_{(u,v)}$ by p_v. In order to maintain dual feasibility, we cannot increase $\beta_{(u,v),v}$ since v is tight. Hence we increase ω_v by $B_v p_v$ and γ_v by p_v.

Observe that tight vertices have to wait for the proposals of their nontight neighbors before making their own proposal. Hence two communication rounds are needed to update all of the variables.

We can show that the number of communication rounds needed to complete the vertex-selection phase is small. Recall that W denotes the ratio of largest to smallest vertex weights.

LEMMA 3. *The vertex-selection phase ends in $O(\log(nW)/\theta)$ rounds.*

Proof. We use a potential function argument in order to show the bound on the number of communication rounds. For round $j \geq 0$ we define the potential of each vertex $v \in V$ as $\Phi_v^j = \mathrm{wt}_v/\deg^j(v)$ if $\deg^j(v) > 0$ and we let $\Phi_v^j = \mathrm{wt}_{max}$ otherwise. Then let

$$\Phi^j = \min_{v \text{ nontight}} \Phi_v^j.$$

Note that Φ^j is a nondecreasing function of j. In fact, we will show that Φ^j doubles at least every $\lceil 2/\theta \rceil$ rounds. The lemma then follows since $\frac{\mathrm{wt}_{min}}{n} \leq \Phi^j \leq \mathrm{wt}_{max}$ for all rounds j.

Consider any given round i. Let V_i^j be the set of nontight vertices at the beginning of round j, $j \geq i$, with $\Phi_v^j \leq 2\Phi^i$, i.e.,

$$V_i^j = \{v \in V \,:\, \deg^j(v) > 0, \mathrm{wt}_v^j > \theta \cdot \mathrm{wt}_v, \Phi_v^j \leq 2\Phi^i\}.$$

Observe that $V_i^{j+1} \subseteq V_i^j$, since the wt_v^js and $\deg^j(v)$s are nonincreasing, while the Φ_v^js are nondecreasing. Consider any vertex $v \in V_i^i$. We will show that $v \notin V_i^{j'}$ for $j' \geq i + \lceil 2/\theta \rceil$. As a consequence, for any nontight vertex v, $\Phi_v^{j'} > 2\Phi^i$, and, hence, $\Phi^{j'} > 2\Phi^i$.

Assume by contradiction that $v \in V_i^{j'}$. Then, by the observation above, $v \in V_i^j$ for any $j \in \{i, i+1, \ldots, j'\}$. Suppose that $w \in V$ is a nontight vertex with the smallest proposal p_w in round j. Recall that $\mathrm{wt}_w^j \geq \theta\, \mathrm{wt}_w$ for nontight vertices. We then have

$$p_{\min,j} = p_w = \frac{\mathrm{wt}_w^j}{\deg^j(w)} \geq \frac{\theta \cdot \mathrm{wt}_w}{\deg^j(w)} \geq \theta \cdot \Phi^j. \tag{13}$$

It follows that the reduction of the residual weight of v in round j is at least

$$\deg^j(v) \cdot p_{\min,j} \geq \deg^j(v) \cdot \theta \cdot \Phi^j \geq \deg^j(v) \cdot \theta \cdot \Phi^i \geq \deg^j(v) \cdot \theta \cdot \Phi_v^j/2 = \theta \cdot \mathrm{wt}_v/2,$$

where the first inequality uses (13) and the third inequality uses the definition of the set V_i^j. Hence

$$\mathrm{wt}_v^{j'} \leq \mathrm{wt}_v - \frac{\theta\, \mathrm{wt}_v}{2} \lceil 2/\theta \rceil \leq 0 \leq \theta \cdot \mathrm{wt}_v,$$

which contradicts the assumption that $v \in V_i^{j'}$. ∎

We now prove that the weight of the vertices in $\mathcal{C}$ is small.

LEMMA 4. *The total weight of the vertices in $\mathcal{C}$ is at most $\frac{2}{1-\theta}$ times the optimum.*

Proof. Assume that the distributed algorithm finishes after $t \geq 0$ rounds, and let $(\alpha, \beta, \gamma, \omega)$ be the final dual. A proof very similar to that of Lemma 1 shows that the dual is indeed feasible. We proceed as in the proof of Lemma 2.

Consider a vertex $v \in \mathcal{C}$ and observe that v must have been tight before switching to the inside state. Thus $\mathrm{wt}_v^t \leq \theta \mathrm{wt}_v$ and

$$\sum_{e \in \delta(v)} \beta_{e,v} \geq \mathrm{wt}_v(1-\theta). \tag{14}$$

We will now show that

$$\sum_{e \in \delta(v)} \beta_{e,v} \leq \sum_{e \in \delta(v)} \alpha_e - 2\omega_v. \tag{15}$$

Equation (15) is trivially satisfied if we consider only the steps in which v is nontight. In fact, in these steps $\omega_v = 0$ and $\beta_{e,v} = \alpha_e$, for all $e \in \delta(v)$.

Consider now a step in which v is tight. The value of the left-hand side of (15) does not change. If ω_v increases by a quantity $B_v \cdot p_v$, then γ_v increases by a quantity p_v. It follows that, for all edges $e = (v, u)$ between v and a nontight neighbor u of v in the current step, the value of α_e also increases by at least p_v. Since there are at least $2B_v$ such neighbors, the right-hand side of (15) cannot decrease.

Let apx denote the weight of $\mathcal{C}$. Hence,

$$\mathrm{apx} = \sum_{v \in \mathcal{C}} \mathrm{wt}_v \leq \frac{1}{1-\theta} \sum_{v \in C} \sum_{e \in \delta(v)} \beta_{e,v} \leq \frac{1}{1-\theta} \sum_{v \in C} \sum_{e \in \delta(v)} (\alpha_e - 2\omega_v),$$

where the first inequality uses (14) and the second inequality (15). Since every edge is incident to at most two vertices from $\mathcal{C}$ we have that the right-hand side of the last inequality is bounded by

$$\frac{2}{1-\theta} \left(\sum_{e \in E} \alpha_e - \sum_{v \in V} \omega_v \right).$$

The claim follows by weak duality. □

For a given accuracy parameter $\epsilon \geq 0$ we now let $\theta = 1 - 2/(2+\epsilon)$. Note that this choice implies that $1/\theta = O(1/\epsilon)$ for $\epsilon \in (0, 1]$. Hence, given a feasible instance of capVC our distributed algorithm terminates within $O(\log(nW)/\epsilon)$ communication rounds with a cover of weight at most $(2+\epsilon)$opt as was claimed in Theorem 2.

3.2. Edge-assignment phase. At the end of the vertex-selection phase we are left with a subset $\mathcal{C}' \subseteq \mathcal{C}$ of the tight vertices such that all unassigned edges have both of their endpoints in $\mathcal{C}'$. In the following we let $G^0 = G[\mathcal{C}'] = (V, E)$ be the graph induced by the vertices in $\mathcal{C}'$. Assuming that the given capVC instance is feasible, there must be an assignment of the edges in G^0 to the vertices in $\mathcal{C}'$ that obeys the original capacity bounds. We describe a deterministic distributed algorithm which assigns at most $(2+\epsilon)B_v$ edges to each $v \in \mathcal{C}'$ in $O(\log n/\epsilon)$ rounds.

Our algorithm starts with all edges unassigned and computes a final assignment iteratively. In each round t we consider all vertices $v \in V$ with at most $(2+\epsilon)B_v$ incident unassigned edges, and we assign all such edges $(u, v) \in \delta(v)$ to v. We continue until no unassigned edges remain.

To prove that the number of rounds is polylogarithmic we need the following lemma. Let H be the set of vertices with a degree of more than $(2+\epsilon)B_v$, and let $E(H)$ be the set of those edges that have both of their endpoints in H. Finally use $\overline{E(H)}$ as an abbreviation for the set $E \setminus E(H)$ of edges that have at most one endpoint in H.

LEMMA 5. *If there is a feasible assignment, then we must have* $|\overline{E(H)}| \geq \epsilon|E(H)|$.

Proof. Let $\pi : E \to V$ be a feasible assignment of edges to vertices. We have that

$$\sum_{v \in H} |\delta(v)| \leq 2|E(H)| + |\overline{E(H)}| \tag{16}$$

as every edge in $E(H)$ is counted exactly twice in the sum on the left-hand side while an edge in $\overline{E(H)}$ is counted at most once. Moreover,

$$|E(H)| \leq \sum_{v \in H} |\pi^{-1}(v)| \tag{17}$$

since every edge in $E(H)$ must be assigned to some vertex in H. From (16) and (17) it follows that

$$(2+\epsilon) \sum_{v \in H} B_v \leq \sum_{v \in H} |\delta(v)| \leq 2 \sum_{v \in H} |\pi^{-1}(v)| + |\overline{E(H)}| \leq 2 \sum_{v \in H} B_v + |\overline{E(H)}|.$$

Hence

$$|\overline{E(H)}| \geq \epsilon \sum_{v \in H} B_v \geq \epsilon |E(H)|$$

which proves the lemma. ∎

LEMMA 6. *If there is a feasible assignment, then the algorithm above assigns at most* $(2+\epsilon)B_v$ *edges to each* $v \in V$. *The number of rounds required is* $O(\log n/\epsilon)$.

Proof. The capacity bound in the theorem follows immediately since for each vertex v in V there is at most one round t in which we assign at most $(2+\epsilon)B_v$ edges to it.

Let E^t be the set of unassigned edges at the beginning of iteration t, and let $G^t = G[E^t]$ be the subgraph of G induced by E^t. We also use H^t to denote the set of vertices $v \in V$ whose degree is more than $(2+\epsilon)B_v$ in G^t. Note that for any t, there must exist a feasible assignment in G^t as G^t is a subgraph of the initial graph G where a feasible assignment exists. So we can apply Lemma 5 and conclude that

$$|E^t| = |\overline{E(H^t)}| + |E(H^t)| \geq (1+\epsilon)|E(H^t)|.$$

In round t all of the edges in $\overline{E(H^t)}$ are assigned to some vertex and so $|E^{t+1}| \leq |E(H^t)|$. Hence, $|E^{t+1}| \leq \frac{1}{1+\epsilon}|E^t|$ and the number of unassigned edges decreases by a factor of $(1+\epsilon)$ in every round. ∎

Since at most $2B_u$ edges are assigned to each u during the vertex-selection phase, this concludes the proof of Theorem 2.

4. A lower bound. In this section we show that every efficient distributed approximation algorithm for capVC needs to violate the capacity constraints by a large additive term.

Consider the following two families of graphs $G^0_{B,k}$ and $G^1_{B,k}$, where $B, k \geq 1$. Graph $G^0_{B,k}$ has $k+1$ levels $L_0, L_1, \ldots, L_k$, each one containing $2B+1$ vertices. Each vertex in level L_i, $i = 0, 1, \ldots, k-1$, is adjacent to exactly B vertices in level L_{i+1}. Symmetrically, each vertex in level L_i, $i = 1, 2, \ldots, k$, is adjacent to exactly B vertices in level L_{i-1}. There are no other edges in the graph. In particular, each level L_i induces an independent set. Graph $G^1_{B,k}$ is obtained from $G^0_{B,k}$ by adding an edge between each pair of vertices in L_0. Let the capacity of all vertices in both graphs

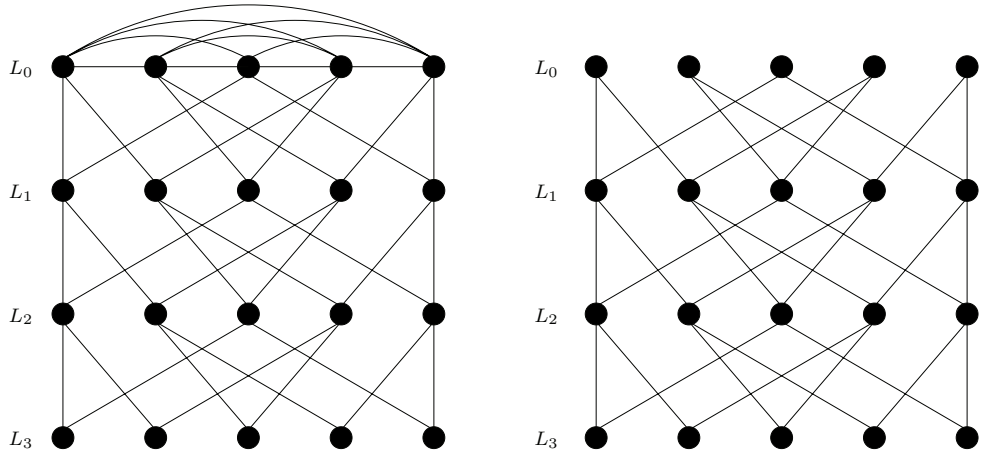


FIG. 3. *The figure shows graphs $G^1_{B,k}$ (on the left) and $G^0_{B,k}$ for $B = 2$ and $k = 3$ (on the right).*

be B. Moreover, all vertices have cost zero, except for the vertices in level L_k, which have cost one. Figure 3 shows an instance of the two graphs.

For $0 \le i \le k-1$, let δ_i be the set of edges that connect vertices in L_i to those in L_{i+1}. We obtain a feasible solution for $G^0_{B,k}$ as follows: Let

$$C = L_0 \cup L_1 \cup \cdots \cup L_{k-1}$$

and assign all edges in δ_i to the vertices in L_i for $0 \le i \le k-1$.

Graph $G^1_{B,k}$ has $n = (k+1)(2B+1)$ vertices and Bn edges. Thus, any feasible capacitated vertex cover must contain all vertices. Moreover, the edges belonging to the clique on L_0 clearly have to be assigned to the vertices in L_0. Thus, the unique feasible capVC solution for $G^0_{B,k}$ assigns all edges in δ_i to the vertices in L_{i+1} for all $0 \le i \le k-1$.

The following lemma turns out to be useful in the proof of the lower bound.

LEMMA 7. *Consider a solution for $G^1_{B,k}$ that assigns at most $(B+c)$ edges to each vertex, for some $c \ge 1$. For $0 \le i \le k-1$, let A_i be the number of edges in δ_i that are assigned to vertices in L_i. Then*

$$A_i \le (2B+1)(i+1)c$$

for all $0 \le i \le k-1$.

Proof. The proof is by induction on i. For $i = 0$, all clique edges need to be assigned to vertices in L_0. The spare capacity of the vertices in L_0 is thus $(2B+1)c$ which is the maximum number of edges in δ_0 that can be assigned to the vertices in L_0.

Now assume that the claim is true for all $0 \le i < k$. Using the induction hypothesis, at most $(2B+1)(i+1)c$ edges in δ_i are assigned to the vertices in L_i. Therefore, the remaining $(2B+1)(B-(i+1)c)$ edges need to be assigned to vertices in L_{i+1}. The remaining capacity of the vertices in L_{i+1} is thus

$$(2B+1)(B+c) - (2B+1)(B-(i+1)c) = (2B+1)(i+2)c$$

which is the maximum number of edges in δ_{i+1} that can be assigned to vertices in L_{i+1}. ∎

Armed with Lemma 7 we are now ready to provide a proof of Theorem 4. We restate it here for completeness.

THEOREM 4. *Let $B, k \geq 1$ be integer parameters. There is a* capVC *instance with uniform vertex capacities B, for which any distributed approximation algorithm that assigns less than $(1+1/k) \cdot B$ edges to all vertices must take at least k communication rounds.*

Proof. The proof is by contradiction. Let $c < B/k$ and consider a distributed approximation algorithm for capVC that assigns at most $(B+c)$ edges to each vertex for any given (uniform capacity) instance whose running time is less than k.

We first execute this algorithm on graph $G^1_{B,k}$. Lemma 7 shows that at most $(2B+1) \cdot kc$ of the edges in δ_{k-1} are assigned to the vertices in L_{k-1}. The number of edges that need to be assigned to vertices in L_k is therefore at least

$$(2B+1)(B-kc) > 0,$$

where the inequality uses our assumption on c. Hence at least one vertex in L_k needs to be in any cover of $G^1_{B,k}$ that assigns at most $B+c$ edges to each vertex. Let u be this vertex.

We now run the algorithm again on $G^0_{B,k}$. Since the graphs $G^0_{B,k}$ and $G^1_{B,k}$ are identical up to distance k from u, this vertex will be included in the cover in this case too. On the other hand, no vertex in L_k can be part of any approximate solution for $G^0_{B,k}$. ∎

For instance, consider an (efficient) distributed approximation algorithm for capVC with running time $O(\log^d n)$, where d is a positive constant. Theorem 4 then shows that there is a family of (uniform capacity) capVC instances for which this algorithm must assign at least $B + \Omega(\frac{B}{\log^d B})$ edges to some vertex.

We observe that graphs $G^0_{B,k}$ and $G^1_{B,k}$ have $n = (2B+1)(k+1)$ vertices. This implies that $B = \Theta\left(\frac{n}{k}\right)$, which is large in the interesting case when k is polylogarithmic. However, this *hitch* is easily removed by defining a graph G_0 (resp., G_1) consisting of t disjoint copies of the main building block $G^0_{B,k}$ (resp., $G^1_{B,k}$). Using the new parameter t we can now produce instances in which B is arbitrarily small in comparison to n while our proof argument goes through unchanged.

Acknowledgment. We would like to thank Volker Kaibel for pointing out a much simplified proof of Lemma 5.

REFERENCES

[1] N. ALON, D. MOSHKOVITZ, AND S. SAFRA, *Algorithmic construction of sets for k-restrictions*, ACM Trans. Algorithms, 2 (2006), pp. 153–177.

[2] R. BAR-YEHUDA AND S. EVEN, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.

[3] M. BELLARE, S. GOLDWASSER, C. LUND, AND A. RUSSELL, *Efficient probabilistically checkable proofs: Applications to approximation*, in Proceedings of the ACM Symposium on Theory of Computing, Montreal, Quebec 1994, pp. 294–304.

[4] F. CHUDAK, T. ERLEBACH, AND A. PANCONESI, *Primal-Dual Based Distributed Algorithms for Vertex Cover with Soft Capacities and Facility Location*, manuscript, 2004.

[5] J. CHUZHOY AND J. NAOR, *Covering problems with hard capacities*, SIAM J. Comput., 36 (2006), pp. 498–515.

[6] D. DUBHASHI, O. HÄGGSTRÖM, G. MAMBRINI, A. PANCONESI, AND C. PETRIOLI, *Blue pleieades, a new solution for device discovery and scatternet formation in multi-hop bluetooth networks*, ACM-Kluwer Wireless Networks (Winet), 13 (2007), pp. 107–125.

[7] M. Elkin, *Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, 2004, ACM, New York, pp. 331–340.
[8] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
[9] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan, *An improved approximation algorithm for vertex cover with hard capacities*, J. Comput. System Sci., 72 (2006), pp. 16–33.
[10] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, *Dependent rounding in bipartite graphs*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, British Columbia, 2002, pp. 323–332.
[11] D. Grable and A. Panconesi, *Nearly optimal distributed edge colouring in o(log log n) rounds*, Random Structures Algorithms, 10 (1997), pp. 385–405.
[12] S. Guha, R. Hassin, S. Khuller, and E. Or, *Capacitated vertex covering*, J. Algorithms, 48 (2003), pp. 257–270.
[13] E. Halperin, *Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs*, SIAM J. Comput., 31 (2002), pp. 1608–1623.
[14] D. S. Hochbaum, *Approximation algorithms for set covering and vertex cover problems*, SIAM J. Comput., 11 (1982), pp. 555–556.
[15] G. Karakostas, *A better approximation ratio for the vertex cover problem*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, Lisboa, Portugal, Lecture Notes in Comput. Sci., 3580, Springer, Berlin, 2005, pp. 1043–1050.
[16] S. Khuller, U. Vishkin, and N. Young, *A primal-dual parallel approximation technique applied to weighted set and vertex covers*, J. Algorithms, 17 (1994), pp. 280–289.
[17] F. Kuhn and T. Moscibroda, *Distributed approximation of capacitated dominating sets*, in Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Diego, CA, 2007, pp. 161–170.
[18] F. Kuhn, T. Moscibroda, and R. Wattenhofer, *What cannot be computed locally!*, in Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing, St. Johns, Newfoundland, 2004, pp. 300–309.
[19] F. Kuhn and R. Wattenhofer, *Constant-time distributed dominating set approximation*, in Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing, Boston, MA, 2003, pp. 25–32.
[20] L. Jia, R. Rajaraman, and T. Suel, *An efficient distributed algorithm for constructing small dominating sets*, Distributed Computing, 15 (2002), pp. 193–205.
[21] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.
[22] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.
[23] M. Luby, *Removing randomness in parallel without a processor penalty*, J. Comput. System Sci., 47 (1993), pp. 250–286.
[24] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.
[25] M. Naor, *A lower bound on probabilistic algorithms for distributive ring coloring*, SIAM J. Discrete Math., 4 (1991), pp. 409–412.
[26] A. Panconesi and A. Srinivasan, *The local nature of delta-coloring and its algorithmic applications*, Combinatorica, 15 (1995), pp. 255–280.
[27] S. Rajagopalan and V. V. Vazirani, *Primal-dual RNC approximation algorithms for set cover and covering integer programs*, SIAM J. Comput., 28 (1998), pp. 525–540.
[28] R. Rajaraman, *Topology control and routing in ad hoc networks: A survey*, SIGACT News, 33 (2002), pp. 60–73.
[29] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, EI Paso, IX, 1997, ACM, New York, 1999, pp. 475–484.
[30] P. Wan, K. M. Alzoubi, and O. Frieder, *Distributed construction of connected dominating set in wireless ad hoc networks*, in Proceedings of IEEE INFOCOM, New York, 2002.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 841–880

ON THE COMPLEXITY OF VERIFYING CONSISTENCY OF XML SPECIFICATIONS*

MARCELO ARENAS†, WENFEI FAN‡, AND LEONID LIBKIN‡

Abstract. XML specifications often consist of a type definition (typically, a document type definition (DTD)) and a set of integrity constraints. It has been shown previously that such specifications can be inconsistent, and thus it is often desirable to check consistency at compile time. It is known [W. Fan and L. Libkin, *J. ACM*, 49 (2002), pp. 368–406] that for general keys, foreign keys, and DTDs the consistency problem is undecidable; however, it becomes NP-complete when all keys are one-attribute (unary) and tractable, if no foreign keys are used. In this paper, we consider a variety of previously studied constraints for XML data and investigate the complexity of the consistency problem. Our main conclusion is that, in the presence of foreign key constraints, compile-time verification of consistency is infeasible. We look at absolute constraints that hold in the entire document and relative constraints that hold only in a part of the document. For absolute constraints, we prove decidability and establish complexity bounds for primary multiattribute keys and unary foreign keys and study unary constraints that involve regular expressions. For relative constraints, we prove that even for unary constraints the consistency problem is undecidable. We also show that results continue to hold for extended DTDs, a more expressive typing mechanism for XML.

Key words. XML, keys and foreign keys, document type definition, consistency, complexity

AMS subject classifications. 03D05, 68P15

DOI. 10.1137/050646895

1. Introduction.

XML data, just like relational and object-oriented data, can be specified in a certain data definition language. While the exact details of XML data definition languages are still being worked out, it is clear that all of them would contain a form of document description, as well as integrity constraints. Constraints are naturally introduced when one considers transformations between XML and relational databases [10, 12, 18, 19, 23, 30, 31], as well as integrating several XML documents [2, 3, 4, 15].

Document descriptions usually come in the form of DTDs (document type definitions), and constraints are typically natural analogues of the most common relational integrity constraints: keys and foreign keys. Indeed, a large number of proposals (e.g., [35, 38, 36, 5]) support specifications for keys and foreign keys.

We investigate XML specifications with DTDs and keys and foreign keys. We study the *consistency*, or *satisfiability*, of such specifications: given a DTD and a set of constraints, whether there are XML documents conforming to the DTD and satisfying the constraints. In other words, we want to validate XML specifications statically, at compile time. Invalid XML specifications are likely to be more common than invalid

*Received by the editors December 7, 2005; accepted for publication (in revised form) January 17, 2008; published electronically June 6, 2008. An extended abstract was presented at the 21st ACM Symposium on Principles of Database Systems (PODS 2002).

http://www.siam.org/journals/sicomp/38-3/64689.html

†Escuela de Ingenieria, Pontificia Universidad Católica de Chile, Santiago 22, Chile (marenas@ing.puc.cl). This author's research was partially supported by grants from NSERC, CITO, and PREA, as well as by FONDECYT grant 1050701.

‡School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK (wenfei@inf.ed.ac.uk, libkin@inf.ed.ac.uk). The second author's research was supported by NSF Career Award IIS-0093168 and grants EPSRC GR/S63205/01, EPSRC GR/T27433/01, and NSFC 60228006. The third author's research was partially supported by grants from NSERC, CITO, and PREA and by the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502 and EPSRC grant E005039.

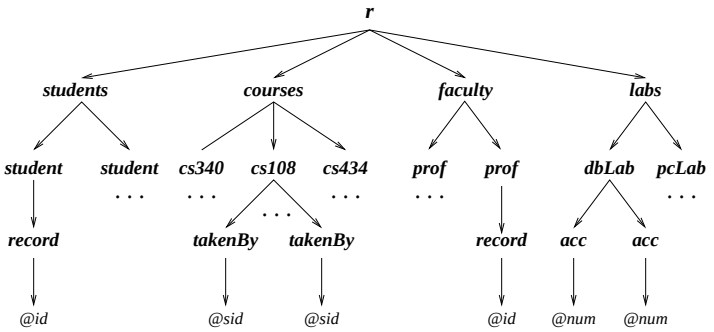


FIG. 1.1. *An XML document.*

specifications of other kinds of data, due to the rather complex interaction of DTDs and constraints. Furthermore, many specifications are not written at once, but rather in stages: as new requirements are discovered, they are added to the constraints, and thus it is quite possible that at some point they may be contradictory.

An alternative to the static validation would be a dynamic approach: simply attempt to validate a document with respect to a DTD and a set of constraints. This, however, would not tell us whether repeated failures are due to a bad specification or problems with the documents.

The consistency analysis for XML specifications is not nearly as easy as for relational data (any set of keys and foreign keys can be declared on a set of relational attributes). Indeed, [16] showed that, for DTDs and arbitrary keys and foreign keys, the consistency problem is undecidable. Furthermore, under the restriction that all keys and foreign keys are unary (single-attribute), the problem is NP-complete.

These results revealed only the tip of the iceberg, as many other flavors of XML constraints exist, and are likely to be added to future standards for XML such as XML Schema [38]. One of our goals is to study such constraints. In particular, we concentrate on constraints with regular expressions and relative constraints that hold only in a part of the document. We now give examples of new kinds of constraints considered here and explain their consistency problem.

Constraints with regular expressions. As XML data are hierarchically structured, one is often interested in constraints specified by regular expressions. For example, consider an XML document (represented as a node-labeled tree) in Figure 1.1, which conforms to the following DTD for schools:

```
<!ELEMENT  r  (students, courses, faculty, labs)>
<!ELEMENT  students   (student+)>
<!ELEMENT  courses    (cs340, cs108, cs434)>
<!ELEMENT  faculty    (prof+)>
<!ELEMENT  labs       (dbLab, pcLab)>
<!ELEMENT  student    (record)>          /* similarly for prof
<!ELEMENT  cs434      (takenBy+)         /* similarly for cs340, cs108
<!ELEMENT  dbLab      (acc+)             /* similarly for pcLab
```

Here we omit the descriptions of elements whose type is string (PCDATA). Assume that each *record* element has an attribute @*id*, each *takenBy* has an attribute @*sid* (for student id), and each *acc* has an attribute @*num*. One may impose the following

constraints over the DTD of that document:

$$r._{-}^{*}.(student \cup prof).record.@id \;\rightarrow\; r._{-}^{*}.(student \cup prof).record,$$
$$r._{-}^{*}.cs434.takenBy.@sid \subseteq_{FK} r._{-}^{*}.student.record.@id,$$
$$r._{-}^{*}.dbLab.acc.@num \subseteq_{FK} r._{-}^{*}.cs434.takenBy.@sid.$$

Here "_" is a wild card that matches any label (tag), and "_*" is its Kleene closure that matches any path. The first constraint says that *@id* is a key for all records of students and professors. The other constraints specify foreign keys, which assert that cs434 can be taken only by students and only students who are taking cs434 can have an account in the database lab. Recall that a foreign key also imposes a key constraint on the target elements; e.g., the last foreign key above also says that *@sid* is a key for students taking cs434.

Clearly, there is an XML tree satisfying both the DTD and the constraints. As was mentioned earlier, specifications are rarely written at once. Now suppose a new requirement is discovered: all faculty members must have a dbLab account. Consequently, one adds a new foreign key:

$$r.faculty.prof.record.@id \subseteq_{FK} r._{-}^{*}.dbLab.acc.@num.$$

However, this addition makes the whole specification inconsistent. This is because previous constraints postulate that *dbLab* users are students taking cs434, and no professor can be a student since *@id* is a key for both students and professors, while the new foreign key insists upon professors also being *dbLab* users and the DTD enforces the requirement that at least one professor be present in the document. Thus no XML document both conforms to the DTD and satisfies all of the constraints.

The consistency problem for regular expression constraints is at least as hard as for constraints specified for element types with simple attributes: NP-hard in the unary case and undecidable in general [16]. We use results from [1, 16, 27] to show that, in the unary case, the problem remains decidable, but the lower bound becomes PSPACE.

Relative integrity constraints. Many types of constraints are specified for an entire document. A different kind of constraints, called *relative*, was proposed in [5]—those constraints hold only in a part of a document. As an example, consider an XML document that for each country lists its administrative subdivisions (e.g., into provinces or states) as well as capitals of provinces. A DTD is given below, and an XML document conforming to it is depicted in Figure 1.2:

```
<!ELEMENT  db       (country+)>
<!ELEMENT  country  (province+, capital+)>
<!ELEMENT  province (capital, city*)>
```

Each country has a nonempty sequence of provinces and a nonempty sequence of province capitals, and for each province we specify its capital and perhaps other cities. Each country and province has an attribute *@name*, and each capital has an attribute *@inProvince*.

Now suppose that we want to define keys for countries and provinces. One can state that country *@name* is a key for *country* elements. It is also tempting to say that *@name* is a key for *province*, but this may not be the case. The example in Figure 1.2 clearly shows that which *Limburg* one is interested in probably depends on whether one's interests are in database theory or in the history of the European

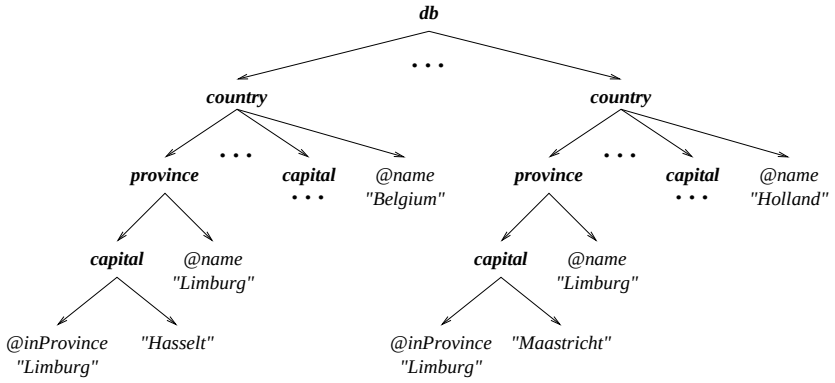


FIG. 1.2. *An XML document storing information about countries and their administrative subdivisions.*

Union. To overcome this problem, we define *@name* to be a key for *province relative* to a country; indeed, it is extremely unlikely that two provinces of the same country would have the same name. Thus, our constraints are

$$\begin{aligned} country.@name &\rightarrow country, \\ country(province.@name &\rightarrow province), \\ country(\_^*.capital.@inProvince &\rightarrow \_^*.capital), \\ country(\_^*.capital.@inProvince &\subseteq_{FK} \_^*.province.@name). \end{aligned}$$

The first constraint is like those we have encountered before: it is an *absolute* key, which applies to the entire document. The rest are *relative constraints* which are specified for subdocuments rooted at *country* elements. They assert that for each country *@name* is a key of all *province* descendants of the country element and *@inProvince* is a key of all *capital* descendants of the country element and a foreign key referring to *@name* of *province* elements in the same subdocument. The foreign keys assure that for each *capital* element in a *country* element (subdocument) its *@inProvince* attribute refers to a *province* in the same country (recall that *capital* elements immediately below *country* also denote *province* capitals). Note that these constraints are somewhat related to the notion of keys for weak entities in relational databases (cf. [33]). In contrast to regular expression constraints given earlier, these constraints are defined for element types; e.g., the first constraint is a key for all *country* elements in the entire document, and the second constraint is a (relative) key for all *capital* elements in a subdocument rooted at a *country* node.

To illustrate the interaction between constraints and DTDs, observe that the above specification—which might look reasonable at first—is actually inconsistent! To see this, let T be a tree that satisfies the specification. The constraints say that for any subdocument rooted at a country c the number of its *capital* elements is at most the number of *province* elements among c's descendants. The DTD says that each *province* has a *capital* element as a child and that each *country* element has at least one *capital* child. Thus, the number of *capital* descendants of c is larger than the number of *province* descendants of c, which contradicts the previous bound. Hence, the specification is inconsistent. We note that one can make the specification consistent by replacing $country(\_^*.capital.@inProvince \rightarrow \_^*.capital)$ with two keys: $country(capital.@inProvince \rightarrow capital)$ and $country(province.capital.@inProvince \rightarrow$

province.capital), which allow *capital.@inProvince* and *province.capital.@inProvince* to share the same value.

Relative constraints appear to be quite useful for capturing information about XML documents that cannot possibly be specified by absolute constraints. It turns out, however, that the consistency problem is much harder for them: it is undecidable even for single-attribute keys and foreign keys.

Decidable restrictions. Since expensive lower bounds, and even undecidability, were established for most versions of the consistency problem, we would like to see some interesting tractable, or decidable, restrictions. In the case of absolute constraints, the results of [16] consider either single attributes or multiattribute sets for both keys and foreign keys and thus say nothing about the intermediate case in which only keys are allowed to be multiattribute. This class of constraints is rather common and arises when relational data is translated into XML. While often identifiers are used as single-attribute keys, other sets of attributes can form a key as well (e.g., via SQL `unique` declaration), and those typically contain more than one attribute. We show that the consistency problem for this class of constraints, when every key is primary (i.e., at most one key is defined for each element type), remains decidable.

The main conclusion of this paper is that, while many proposals such as XML Schema [38] and XML Data [36] support the facilities provided by the DTDs as well as integrity constraints, and while it is possible to write inconsistent specifications, checking consistency at compile time appears to be infeasible, even for fairly small specifications.

Related work. Consistency was studied for other data models, such as object-oriented and extended relational (e.g., with support for cardinality constraints); see [8, 9, 22].

A number of specifications for XML keys and foreign keys have been proposed, e.g., XML Schema [38] and XML Data [36]. A recent proposal [5] introduced relative constraints. To the best of our knowledge, consistency of XML constraints in the presence of schema specifications was investigated only in [16]. However, [16] did not consider relative constraints, constraints defined with regular expressions, and the class of multiattribute keys and unary foreign keys. Other constraints for semistructured data, different from those considered here, were studied in, e.g., [1, 6, 17]. The latter also studies the consistency problem; the special form of constraints used there makes it possible to encode consistency as an instance of conjunctive query containment. Application of constraints in data transformations was studied in [23, 12]; usefulness of keys and foreign keys in query optimization has also been recognized [13, 14].

Organization. Section 2 defines DTDs, absolute keys, and foreign keys for XML. Section 3 studies the class of absolute multiattribute keys and unary foreign keys and the class of regular expression constraints which is an extension of absolute constraints with regular path expressions. Section 4 defines and investigates relative keys and foreign keys. Section 5 provides lower and upper bounds for the consistency problem for extended DTDs, a slight extension of DTDs which captures unranked tree automata, and several different classes of keys and foreign keys. Section 6 summarizes the main results of the paper.

2. Notations.

2.1. DTDs, XML trees, and paths. Assume that we have the following disjoint sets: *El* of element names, *Att* of attribute names, *S* of possible values of attributes and raw text, and *Vert* of node identifiers. All attribute names start with

the symbol @, and these are the only ones starting with this symbol. We let S be a reserved symbol not in any of those sets.

We formalize the notion of DTDs as follows (cf. [35, 7, 25, 16]).

DEFINITION 2.1. *A* DTD *is defined to be* $D = (E, A, P, R, r)$*, where:*

- $E \subseteq El$ *is a finite set of* element types*;*
- $A \subseteq Att$ *is a finite set of* attributes*;*
- P *is a mapping from* E *to* element type definitions*: Given* $\tau \in E$*,* $P(\tau) = \mathtt{S}$ *or* $P(\tau)$ *is a regular expression* α *defined as follows:*

$$\alpha ::= \epsilon \mid \tau' \mid \alpha|\alpha \mid \alpha,\alpha \mid \alpha^*,$$

  where S *denotes the* string *type,* $\tau' \in E$*,* ϵ *is the empty word, and "*$|$*", ",", and "*$*$*" denote union, concatenation, and the Kleene closure, respectively;*
- R *is a mapping from* E *to the powerset of* A*. If* $@l \in R(\tau)$*, we say that* $@l$ *is* defined for τ*;*
- $r \in E$ *and is called* the element type of the root*.*

We normally denote element types by τ and assume that $R(r) = \emptyset$ and r does not appear in $P(\tau)$ for any $\tau \in E$. We also assume that each τ in $E \setminus \{r\}$ is *connected to* r; i.e., either τ appears in $P(r)$, or it appears in $P(\tau')$ for some τ' that is connected to r. In this paper we also use the following shorthand for regular expressions: α^+ for (α, α^*) and $\alpha?$ for $(\epsilon|\alpha)$. Finally, notice that mixed content is not allowed in XML trees; for every $\tau \in E$, $P(\tau)$ is either S or a regular expression over E.

Example 2.2. Let us consider the DTD D given in section 1 for storing information about countries and their administrative subdivisions. In our formalism, D can be represented as (E, A, P, R, r), where $E = \{db, country, province, capital, city\}$, $A = \{@name, @inProvince\}$, $r = db$, and P, R are as follows:

$P(db)$	$=$	$country^+$,	$R(db)$	$=$	$\emptyset$,
$P(country)$	$=$	$(province^+, capital^+)$,	$R(country)$	$=$	$\{@name\}$,
$P(province)$	$=$	$(capital, city^*)$,	$R(province)$	$=$	$\{@name\}$,
$P(capital)$	$=$	S,	$R(capital)$	$=$	$\{@inProvince\}$,
$P(city)$	$=$	S,	$R(city)$	$=$	$\emptyset$.

An XML document is typically modeled as a node-labeled tree. Below we describe valid XML documents w.r.t. a DTD, along the same lines as XQuery [39], XML Schema [38], and DOM [34].

DEFINITION 2.3. *Let* $D = (E, A, P, R, r)$ *be a DTD. An* XML tree T conforming to D*, written* $T \models D$*, is defined to be* $(V, lab, ele, att, root)$*, where*

- $V \subseteq Vert$ *is a finite set of* nodes*;*
- $lab : V \to E$*; if* $lab(v) = \tau$ $(v \in V)$*,* τ *is said to be the* element type *of* v*;*
- $ele : V \to S \cup V^*$*, where* V^* *is the set of all of the finite sequences of values from* V*, such that, for every* $v \in V$*, if* $P(lab(v)) = \mathtt{S}$*, then* $ele(v) = [s]$*, where* $s \in S$*; otherwise,* $ele(v) = [v_1, \ldots, v_n]$*, and the string* $lab(v_1) \ldots lab(v_n)$ *is in the regular language defined by* $P(lab(v))$*;*
- att *is a partial function from* $V \times A$ *to* S *such that, for any* $v \in V$ *and* $@l \in A$*,* $att(v, @l)$ *is defined iff* $@l \in R(lab(v))$*;*
- $root$ *is the root of* T*:* $root \in V$ *and* $lab(root) = r$*.*

The parent-child edge relation on V*,* $\{(v_1, v_2) \mid v_2$ *occurs in* $ele(v_1)\}$*, is required to form a rooted tree.*

In an XML tree T, for each $v \in V$, there is a unique path of parent-child edges from the root to v, and each node has at most one incoming edge. The root is a

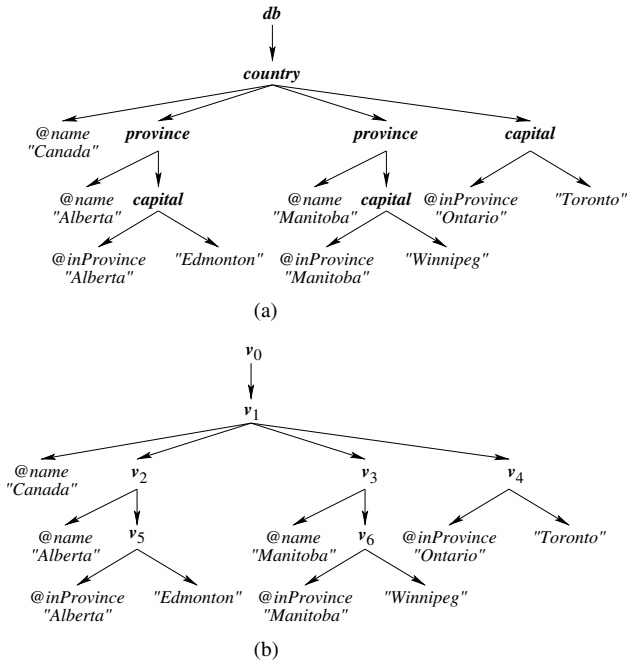


FIG. 2.1. *An XML document represented as a tree.*

unique node labeled with r. If a node x is labeled τ in E, then function ele defines the children of x and function att defines the attributes of x. The children of x are ordered, and their labels observe the regular expression $P(\tau)$. In contrast, its attributes are unordered and are identified by their labels (names).

Example 2.4. Figure 2.1(a) shows an XML document storing information about provinces in Canada and conforming to the DTD shown in Example 2.2. Figure 2.1(b) shows an XML tree $T = (V, lab, ele, att, v_0)$ representing this document. In this tree, $V = \{v_i \mid i \in [0, 6]\}$ and lab is defined as:

$$lab(v_0) = db, \quad lab(v_2) = province, \quad lab(v_4) = capital, \quad lab(v_6) = capital.$$
$$lab(v_1) = country, \quad lab(v_3) = province, \quad lab(v_5) = capital,$$

Furthermore, function ele is defined as:

$$ele(v_0) = [v_1], \quad ele(v_2) = [v_5], \quad ele(v_4) = [\text{Toronto}], \quad ele(v_6) = [\text{Winnipeg}].$$
$$ele(v_1) = [v_2, v_3, v_4], \quad ele(v_3) = [v_6], \quad ele(v_5) = [\text{Edmonton}],$$

Finally, function att is defined as:

$$att(v_1, @name) = \text{Canada}, \quad att(v_4, @inProvince) = \text{Ontario},$$
$$att(v_2, @name) = \text{Alberta}, \quad att(v_5, @inProvince) = \text{Alberta},$$
$$att(v_3, @name) = \text{Manitoba}, \quad att(v_6, @inProvince) = \text{Manitoba}.$$

Our model is simpler than the models of XQuery and XML Schema, as DTDs support only one basic type (PCDATA or string) and do not have complex type constructs. Unlike the data model of XQuery, we do not consider nodes representing namespaces, processing instructions, and references. These simplifications do not affect the lower bounds, however.

We also use the following notations. Referring to an XML tree T, if x is a τ-element in T and $@l$ is an attribute in $R(\tau)$, then $x.@l$ denotes the $@l$-attribute value

of x, i.e., $x.@l = att(x, @l)$. If X is a list $[@l_1, \ldots, @l_n]$ of attributes in $R(\tau)$, then $x[X] = [x.@l_1, \ldots, x.@l_n]$. For any element type $\tau \in E$, $ext(\tau)$ denotes the set of all of the τ-elements in T. For any $@l \in R(\tau)$, $values(\tau.@l)$ denotes $\{x.@l \mid x \in ext(\tau)\}$, the set of all of the $@l$-attribute values of τ-nodes. We write $|S|$ for the cardinality of a set S. Given a DTD D and a set Σ of constraints, we also use $|D|$ and $|\Sigma|$ to denote their sizes, respectively.

Given a DTD $D = (E, A, P, R, r)$ and element types $\tau, \tau' \in E$, a string $\tau_1.\tau_2.\ldots.\tau_n$ over E is a *path in D from τ to τ'* if $\tau_1 = \tau$, $\tau_n = \tau'$, and for each $i \in [2, n]$, τ_i is a symbol in the alphabet of $P(\tau_{i-1})$. Moreover, $paths(D) = \{p \mid$ there is $\tau \in E$ such that p is a path in D from r to $\tau\}$. We say that a DTD is *nonrecursive* if $paths(D)$ is finite and recursive otherwise. We also say that D is a *no-star* DTD if the Kleene star does not occur in any regular expression $P(\tau)$ (note that this is a stronger restriction than being $*$-free: a regular expression without the Kleene star yields a finite language, while the language of a $*$-free regular expression may still be infinite as it allows boolean operators including complement).

2.2. Keys and foreign keys. We consider two forms of constraints for XML: *absolute constraints* that hold on the entire document, denoted by $\mathcal{AC}$, and *relative constraints* that hold on certain subdocuments, denoted by $\mathcal{RC}$. Below we define absolute keys and foreign keys, and we shall define relative constraints in section 4. The constraints given in section 1 are instances of absolute constraints and relative constraints.

Regular expression constraints. To capture the hierarchical nature of XML data, absolute constraints, in their general form, are defined on a collection of elements identified by a regular path expression. It is common to find path expressions in specification and query languages for XML (e.g., XML Schema [38], XQuery [39], and XSL [40]). We define a *regular (path) expression* over a set of element types E as follows:

$$\beta \quad ::= \quad \epsilon \quad | \quad \tau \quad | \quad \beta.\beta \quad | \quad \beta \cup \beta \quad | \quad \beta^*,$$

where ϵ denotes the empty word, τ is an element type in E, and "$.$", "$\cup$", and "$*$" denote concatenation, union, and Kleene closure, respectively. A regular expression defines a language over the alphabet E, which will be denoted by β as well. Given a DTD $D = (E, A, P, R, r)$ and a regular expression β over E, we say that β is a *regular (path) expression over D* if β is of the form $r.\beta'$, where β' does not include r. In this section, we use "$\_$" as shorthand for $E \setminus \{r\}$.

Recall that a path in a DTD is a list of E symbols, that is, a string in E^*. Given an XML tree $T = (V, lab, ele, att, root)$, a pair of nodes x, y in T, with y a descendant of x, and a path $w = \tau_1.\ldots.\tau_n$ over E, we say that *w is a path from x to y* if there exists a sequence of nodes $v_1, \ldots, v_n$ in T such that (1) $v_1 = x$ and $v_n = y$, (2) v_{i+1} is a child of v_i in T, for every $i \in [1, n-1]$, and (3) $lab(v_i) = \tau_i$, for every $i \in [1, n]$. Any pair of nodes x, y in an XML tree T with y a descendant of x uniquely determines the path, denoted by $\rho(x, y)$, from x to y. We say that y is *reachable* from x by following a regular expression β over D, denoted by $T \models \beta(x,\ y)$, iff $\rho(x, y) \in \beta$. For any fixed T, let $nodes(\beta)$ stand for the set of nodes reachable from the root by following the regular expression β: $nodes(\beta) = \{y \mid T \models \beta(root,\ y)\}$. Note that, for any element type $\tau \in E \setminus \{r\}$, $nodes(r.\_^*.\tau) = ext(\tau)$.

We now define unary XML keys and foreign keys with regular path expressions. Let DTD $D = (E, A, P, R, r)$.

- A *key* over D is an expression φ of the form $\beta.\tau[X] \rightarrow \beta.\tau$, where

– $\tau \in E$;
– X is a nonempty set of attributes in $R(\tau)$; and
– β is a regular expression over D.

For any XML tree T that conforms to D, the tree T satisfies φ, denoted by $T \models \varphi$, if

$$\forall\, x, y \in nodes(\beta.\tau),\ x[X] = y[X] \rightarrow x = y.$$

- A *foreign key* over D is an expression φ of the form $\beta_1.\tau_1[X] \subseteq_{FK} \beta_2.\tau_2[Y]$, where
  – $\tau_i \in E$ for $i = 1, 2$;
  – β_i is a regular expression over D, for $i = 1, 2$; and
  – X, Y are nonempty lists of attributes in $R(\tau_1)$, $R(\tau_2)$ of the same length.

  Here $T \models \varphi$ if $T \models \beta_2.\tau_2[Y] \rightarrow \beta_2.\tau_2$, and

$$\forall\, x \in nodes(\beta_1.\tau_1)\ \exists\, y \in nodes(\beta_2.\tau_2)\ \big(x[X] = y[Y]\big).$$

We use two notions of equality to define keys: value equality is assumed when comparing attributes, and node identity is used when comparing elements. We shall use the same symbol "=" for both, as it will never lead to ambiguity.

The above constraints are generally referred to as *multiattribute* regular expression constraints as they may be defined with multiple attributes. A regular expression key (foreign key) is said to be *unary* if it is defined in terms of a single attribute; that is, $|X| = 1$ ($|X| = |Y| = 1$) in the above definition. In that case, we write $\beta.\tau.@l \rightarrow \beta.\tau$ for regular expression unary keys and $\beta_1.\tau_1.@l_1 \subseteq_{FK} \beta_2.\tau_2.@l_2$ for regular expression unary foreign keys.

From [16], we immediately obtain that the consistency problem for regular expression constraints is undecidable. Thus, in this paper we study only the consistency problem for unary constraints defined with regular expressions. We denote this class of constraints by $\mathcal{AC}^{reg}_{K,FK}$, where subscripts K and FK stand for keys and foreign keys, respectively. For example, the constraints over the school DTD that we have seen in section 1 are instances of $\mathcal{AC}^{reg}_{K,FK}$.

Constraints associated with element types. A class of absolute keys and foreign keys, denoted by $\mathcal{AC}^{*,*}_{K,FK}$ (we shall explain the notation shortly), has been studied in [16]. It is a special case of regular-expression constraints and is defined for element types as follows. An $\mathcal{AC}^{*,*}_{K,FK}$-constraint φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

- *Key.* $\tau[X] \rightarrow \tau$, where $\tau \in E$ and X is a nonempty set of attributes in $R(\tau)$. An XML tree T satisfies φ, denoted by $T \models \varphi$, if

$$\forall\, x, y \in ext(\tau)\ \big(x[X] = y[X] \rightarrow x = y\big).$$

- *Foreign key.* $\tau_1[X] \subseteq_{FK} \tau_2[Y]$, where $\tau_1, \tau_2 \in E$ and X, Y are nonempty lists of attributes in $R(\tau_1)$, $R(\tau_2)$ of the same length. It is satisfied by a tree T if $T \models \tau_2[Y] \rightarrow \tau_2$, and in addition

$$\forall\, x \in ext(\tau_1)\ \exists\, y \in ext(\tau_2)\ \big(x[X] = y[Y]\big).$$

That is, $\tau[X] \rightarrow \tau$ says that the X-attribute values of a τ-element uniquely identify the element in $ext(\tau)$. Furthermore, $\tau_1[X] \subseteq_{FK} \tau_2[Y]$ says that the list of X-attribute values of every τ_1-node in T must match the list of Y-attribute values of some τ_2-node in T and the Y-attribute values of a τ_2-element uniquely identify the element in $ext(\tau_2)$.

TABLE 2.1
Notation summary.

Notation	Meaning
$\mathcal{AC}^{*,*}_{K,FK}$	Multiattribute keys and foreign keys
$\mathcal{AC}^{*,1}_{PK,FK}$	Multiattribute primary keys, unary foreign keys
$\mathcal{AC}_{K,FK}$	Unary keys and foreign keys
$\mathcal{AC}_{PK,FK}$	Primary unary keys and unary foreign keys
$\mathcal{AC}^{reg}_{K,FK}$	Regular expression unary keys and foreign keys

Note that an $\mathcal{AC}^{*,*}_{K,FK}$-constraint can be readily expressed as a regular expression constraint, by using $r._^*.\tau$ for τ.

As for the case of regular expression constraints, an $\mathcal{AC}^{*,*}_{K,FK}$-constraint is generally referred to as a *multiattribute* constraint as it may be defined with multiple attributes. An $\mathcal{AC}^{*,*}_{K,FK}$-constraint is said to be *unary* if it is defined in terms of a single attribute; that is, $\mid X \mid = \mid Y \mid = 1$ in the above definition. In that case, we write $\tau.@l \rightarrow \tau$ for unary keys and $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$ for unary foreign keys. As in relational databases, we also consider *primary keys*: for each element type, at most one key can be defined.

We shall use the following notations for subclasses of $\mathcal{AC}^{*,*}_{K,FK}$: subscripts K and FK denote keys and foreign keys, respectively. When the primary key restriction is imposed, we use subscript PK instead of K. The superscript "$*$" denotes multiattribute, and "1" means unary. When both superscripts are left out, we mean that both keys and foreign keys are unary. We shall be dealing with the following subclasses of $\mathcal{AC}^{*,*}_{K,FK}$: $\mathcal{AC}^{*,1}_{K,FK}$ denotes the class of multiattribute keys and unary foreign keys; $\mathcal{AC}^{*,1}_{PK,FK}$ is the class of primary multiattribute keys and unary foreign keys; $\mathcal{AC}_{K,FK}$ is the class of unary keys and unary foreign keys; and $\mathcal{AC}_{PK,FK}$ is the class of primary unary keys and unary foreign keys. We note that, since a key is part of a foreign key, the restriction of $\mathcal{AC}^{*,*}_{K,FK}$ to unary keys and multiattribute foreign keys ($\mathcal{AC}^{1,*}_{K,FK}$) does not make sense.

For easy reference, in Table 2.1 we summarize our notation for absolute constraints.

2.3. The consistency problem. We are interested in the consistency or satisfiability problem for XML constraints considered together with DTDs: that is, whether a given set of constraints and a DTD are satisfiable by an XML tree. Formally, for a class $\mathcal{C}$ of integrity constraints we define the *XML specification consistency problem* $\mathsf{SAT}(\mathcal{C})$ as follows:

PROBLEM:	$\mathsf{SAT}(\mathcal{C})$
INPUT:	A DTD D, a finite set Σ of $\mathcal{C}$-constraints.
QUESTION:	Is there an XML tree T such that $T \models D$ and $T \models \Sigma$?

It is known [16] that $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable, but $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ and $\mathsf{SAT}(\mathcal{AC}_{PK,FK})$ are NP-complete. Nothing was known, however, about $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$, where only keys are allowed to be multiattribute, or about $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, where regular expressions are used to define unary keys and foreign keys. These problems will be studied in section 3.

In what follows, we write $T \models (D, \Sigma)$ instead of $T \models D$ and $T \models \Sigma$.

3. Absolute integrity constraints. In this section, we establish the decidability and lower bounds for $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ and $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ and the consistency problems for absolute primary multiattribute keys and unary foreign keys and for regular expression unary keys and unary foreign keys.

3.1. Consistency of multiattribute keys. We know that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problem for unary absolute keys and foreign keys, is NP-complete [16]. In contrast, $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable [16]. This leaves a large gap: namely, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$, where only keys are allowed to be multiattribute.

The reason for the undecidability of $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is that the implication problem for functional and inclusion dependencies can be reduced to it [16]. However, this implication problem is known to be decidable—in fact, in cubic time—for single-attribute inclusion dependencies [11], thus giving us hope to get decidability for multiattribute keys and unary foreign keys.

The problem we resolve here is $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$: the consistency problem for *primary* multiattribute keys and unary foreign keys. Recall that a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints is said to be *primary* if for each element type τ there is at most one key in Σ defined for τ-elements (including key dependencies defined by foreign key constraints). Even dealing with this version of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ one encounters considerable difficulties: with a rather involved proof, we manage to show that this problem is equivalent to a certain decidable version of Diophantine equations problem whose exact complexity has been an open problem for a while [21]:

PROBLEM:	PDE (prequadratic Diophantine equations).
INPUT:	An integer $n \times m$ matrix A, a vector $\vec{b} \in \mathbb{Z}^n$, and a set $E \subseteq \{1, \ldots, m\}^3$.
QUESTION:	Is there a vector $\vec{x} \in \mathbb{N}^m$ such that $A\vec{x} \leq \vec{b}$ and $x_i \leq x_j \cdot x_k$ for all $(i, j, k) \in E$?

Note that for $E = \emptyset$ this is exactly the integer linear programming problem [27]. Thus, PDE can be thought of as integer linear programming extended with inequalities of the form $x \leq y \cdot z$ among variables. It is therefore NP-hard, and [21] proved an NEXPTIME upper bound for PDE. The exact complexity of the problem remains unknown.

Recall that two problems P_1 and P_2 are *polynomially equivalent* if there are PTIME reductions from P_1 to P_2 and from P_2 to P_1. We now show the following.

THEOREM 3.1. $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *and PDE are polynomially equivalent.*

Proof. The proof consists of two PTIME reductions, one for each direction.

(a) *A reduction from* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *to PDE.* We first define a class of simplified DTDs called *narrow DTDs*, and we explain how to reduce the consistency problem for $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over arbitrary DTDs to that over narrow DTDs. Then we show how to encode the consistency problem for narrow DTDs and $\mathcal{AC}^{*,1}_{PK,FK}$-constraints by a prequadratic Diophantine system.

We start by explaining the process of narrowing the DTDs. Intuitively, we replace long "horizontal" regular expressions in $P(\tau)$ by shorter ones. Formally, consider a DTD $D = (E, A, P, R, r)$. D is basically an extended regular grammar (cf. [7, 25]); for each $\tau \in E$, $P(\tau)$ is a regular expression α, and, thus, $\tau \to \alpha$ can be viewed as the production rule for τ. We rewrite the regular expression by introducing a set N of new element types (nonterminals) such that the production rules of the new DTD

have one of the following forms:

$$\tau \to \tau_1,\ \tau_2, \qquad \tau \to \tau_1 \mid \tau_2, \qquad \tau \to \tau_1^*, \qquad \tau \to \tau', \qquad \tau \to\ \mathtt{S}, \qquad \tau \to \epsilon,$$

where τ, τ_1, τ_2 are element types in $E \cup N$, $\tau' \in E$, $\mathtt{S}$ is the string type, and ϵ denotes the empty word. More specifically, we conduct the following "narrowing" process on the production rule $\tau \to \alpha$:

- If $\alpha = (\alpha_1, \alpha_2)$, then we introduce two new element types τ_1 and τ_2 and replace $\tau \to \alpha$ with a new rule $\tau \to \tau_1, \tau_2$. We proceed to process $\tau_1 \to \alpha_1$ and $\tau_2 \to \alpha_2$ in the same way.
- If $\alpha = (\alpha_1 | \alpha_2)$, then we introduce two new element types τ_1 and τ_2 and replace $\tau \to \alpha$ with a new rule $\tau \to \tau_1 \mid \tau_2$. We proceed to process $\tau_1 \to \alpha_1$ and $\tau_2 \to \alpha_2$ in the same way.
- If $\alpha = \alpha_1^*$, then we introduce a new element type τ_1 and replace $\tau \to \alpha$ with $\tau \to \tau_1^*$. We proceed to process $\tau_1 \to \alpha_1$ in the same way.
- If α is one of $\tau' \in E$, $\mathtt{S}$, or ϵ, then the rule for τ remains unchanged.

We refer to the set of new element types introduced when processing $\tau \to P(\tau)$ as N_τ and the set of production rules generated/revised as P_τ. Observe that $N_\tau \cap E = \emptyset$ for any $\tau \in E$. We define a new DTD $D_N = (E_N, A, P_N, R_N, r)$, referred to as the *narrowed DTD of* D (or just a narrow DTD if D is clear from the context), where

- $E_N = E \cup \cup_{\tau \in E} N_\tau$, i.e., all element types of E and new element types introduced in the narrowing process;
- $P_N = \cup_{\tau \in E} P_\tau$, i.e., production rules generated/revised in the narrowing process;
- $R_N(\tau) = R(\tau)$ for each $\tau \in E$, and $R_N(\tau) = \emptyset$ for each $\tau \in E_N \setminus E$.

Note that the root element type r and the set A of attributes remain unchanged. Moreover, elements of any type in $E_N \setminus E$ do not have any attribute. The only kind of P_N production rules whose right-hand side contains element type E are of the form $\tau \to \tau'$, where $\tau' \in E$. It is easy to see that D_N is computable in polynomial time.

Obviously, any set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D is also a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over the narrow DTD D_N of D. The next lemma establishes the connection between D and D_N, which allows us to consider only narrow DTDs from now on.

LEMMA 3.2. *Let D be a DTD, D_N the narrowed DTD of D, and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$ iff there exists an XML tree T_2 such that $T_2 \models (D_N, \Sigma)$.*

Proof. Given an element type τ and a sequence of attributes $@l_1, \ldots, @l_n \in R(\tau)$, define $values(\tau[@l_1, \ldots, @l_n])$ as $\{(x.@l_1, \ldots, x.@l_n) \mid x \in ext(\tau)\}$.

To prove the lemma, it suffices to show the following.

Claim. Given any XML tree $T_1 \models D$ one can construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$, and vice versa. Furthermore, for every element type τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1, and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1.

If the claim holds, we can show the lemma as follows. Assume that there exists an XML tree T_1 such that $T_1 \models D$ and $T_1 \models \Sigma$. By the claim, there is T_2 such that $T_2 \models D_N$. Suppose, by contradiction, that there is $\varphi \in \Sigma$ such that $T_2 \not\models \varphi$. (1) If φ is a key $\tau[@l_1, \ldots, @l_n] \to \tau$, then there exist two distinct nodes $x, y \in ext(\tau)$ in T_2 such that $x.@l_i = y.@l_i$ for every $i \in [1, n]$. In other words, $|values(\tau[@l_1, \ldots, @l_n])| < |ext(\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|values(\tau[@l_1, \ldots, @l_n])| = |ext(\tau)|$ in T_1 because the tuple $(x.@l_1, \ldots, x.@l_n)$ of each $x \in ext(\tau)$ uniquely identifies x

among $ext(\tau)$. This contradicts the claim that $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1. (2) If φ is a unary foreign key $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then either $T_2 \not\models \tau_2.@l_2 \rightarrow \tau_2$ or there is $x \in ext(\tau_1)$ in T_2 such that, for all $y \in ext(\tau_2)$ in T_2, $x.@l_1 \neq y.@l_2$. In the first case, we reach a contradiction as in (1). In the second case, we have $x.@l_1 \notin values(\tau_2.@l_2)$ in T_2. By the claim, $x.@l_1 \in values(\tau_1.@l_1)$ in T_1. Since $T_1 \models \varphi$, $x.@l_1 \in values(\tau_2.@l_2)$ in T_1. Again by the claim, we have $x.@l_1 \in values(\tau_2.@l_2)$ in T_2, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, lab_1, ele_1, att, root)$ such that $T_1 \models D$, we construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$. Consider a τ-element v in T_1. Let $ele_1(v) = [v_1, \ldots, v_n]$ and $w = lab_1(v_1) \ldots lab_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \rightarrow P(\tau)$. Let Q_τ be the set of E symbols that appears in P_τ plus S. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context free grammar, where Q_τ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, P_τ the set of production rules, and τ the start symbol.[1] Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_N(\tau)$, it can be verified that w is in the language defined by G. Thus there is a parse tree $T(w)$ w.r.t. the grammar G for w, and w is the frontier (the list of leaves from left to right) of $T(w)$. Without loss of generality, assume that the root of $T(w)$ is v, and the leaves are $v_1, \ldots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in N_τ except that the root v is labeled τ. Intuitively, we construct T_2 by replacing each element v in T_1 by such a parse tree. More specifically, let $T_2 = (V_2, lab_2, ele_2, att, root)$. Here V_2 consists of nodes in V_1 and the internal nodes introduced in the parse trees. For each x in V_2, let $lab_2(x) = lab_1(x)$ if $x \in V_1$, and otherwise let $lab_2(x)$ be the node label of x in the parse tree where x belongs. Note that nodes in $V_2 \setminus V_1$ are elements of some type in $E_N \setminus E$. For every $x \in V_1$, let $ele_2(x)$ be the list of its children in the parse tree having x as root. For every $x \in V_2 \setminus V_1$, let $ele_2(x)$ be the list of its children in the parse tree of an element in V_1 that contains x. Note that att and $root$ remain unchanged. By the construction of T_2 it can be verified that $T_2 \models D_N$; moreover, for every element type τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1 because, among other things, (1) none of the new nodes, i.e., nodes in $V_2 \setminus V_1$, is labeled with an E-type; (2) no new attributes are defined; and (3) the attribute function att is unchanged.

Conversely, assume that there is $T_2 = (V_2, lab_2, ele_2, att, root)$ such that $T_2 \models D_N$. We construct an XML tree T_1 by modifying T_2 such that $T_1 \models D$. For every node $v \in V_2$ with $lab(v) = \tau$ and $\tau \in E_N \setminus E$, we substitute v in $ele_2(v')$ by the children of v, where v' is the parent of v. In addition, we remove v from V_2, $lab_2(v)$ from lab_2, and $ele_2(v)$ from ele_2. Observe that, by the definition of D_N, no attributes are defined for elements of any type in $E_N \setminus E$. We repeat the process until there is no node labeled with element type in $E_N \setminus E$. Now let $T_1 = (V_1, lab_1, ele_1, att, root)$, where V_1, lab_1, and ele_1 are V_2, lab_2, and ele_2 at the end of the process, respectively. Notice that att and $root$ remain unchanged. By the definition of T_1 it can be verified that $T_1 \models D$; and in addition, for every element type τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1 because, among other things, none of the nodes removed is labeled with a type of E and the attribute function att is unchanged.

[1] If τ is in $P(\tau)$, i.e., if τ is recursively defined, we need to rename τ in Q_τ to ensure that Q_τ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.

By Lemma 3.2, in the rest of this proof we consider only narrow DTDs. Next we show how to encode $\mathcal{AC}^{*,1}_{PK,FK}$-constraints by a prequadratic Diophantine system. Let $D = (E, A, P, R, r)$ be a narrow DTD and Σ be a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints, i.e., primary $\mathcal{AC}^{*,1}_{K,FK}$-constraints. We encode Σ with a set C_Σ of integer constraints, referred to as *the cardinality constraints determined by* Σ. For every $\varphi \in \Sigma$,

- if φ is a key constraint $\tau[@l_1, \ldots, @l_k] \to \tau$, then C_Σ contains $|ext(\tau)| \leq |values(\tau.@l_1)| \cdot \ldots \cdot |values(\tau.@l_k)|$;
- if φ is a unary foreign key $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then C_Σ contains $|values(\tau_1.@l_1)| \leq |values(\tau_2.@l_2)|$ and $|ext(\tau_2)| \leq |values(\tau_2.@l_2)|$;
- furthermore, for any $\tau \in E$, if $R(\tau) = \emptyset$, then $0 \leq |ext(\tau)|$ is in C_Σ. Otherwise, for every $@l \in R(\tau)$, $|values(\tau.@l)| \leq |ext(\tau)|$ and $0 \leq |values(\tau.@l)|$ are in C_Σ.

Observe that for a unary key $\tau.@l \to \tau$ we have both $|values(\tau.@l)| \leq |ext(\tau)|$ and $|ext(\tau)| \leq |values(\tau.@l)|$ in C_Σ. Thus C_Σ assures $|ext(\tau)| = |values(\tau.@l)|$.

We write $T \models C_\Sigma$ if T satisfies all of the constraints of C_Σ, and we write $T \models (D, C_\Sigma)$ if T conforms to a narrow DTD D and satisfies C_Σ. Note that C_Σ is equivalent (in fact, can be converted in polynomial time) to a prequadratic Diophantine system since $x \leq x_1 \cdot \ldots \cdot x_k$ can be written as constraints of the form $x \leq y \cdot z$ by introducing $k-2$ fresh variables; e.g., $x \leq x_1 \cdot x_2 \cdot x_3 \cdot x_4$ is equivalent to $x \leq x_1 \cdot z_1$, $z_1 \leq x_2 \cdot z_2$, and $z_2 \leq x_3 \cdot x_4$ (in the sense that the former is satisfiable iff the latter is). Thus, without loss of generality, assume that C_Σ consists of linear and prequadratic integer constraints only. It should be noted that C_Σ can be computed in time polynomial in the size of Σ and D. The lemma below shows that C_Σ characterizes the consistency of Σ if keys in Σ are primary.

LEMMA 3.3. *Let D be a narrow DTD and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then every XML tree conforming to D and satisfying Σ also satisfies C_Σ. In addition, if there exists an XML tree T_2 such that $T_2 \models (D, C_\Sigma)$, then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$.*

Proof. It is easy to see that, for every XML tree T_1 that satisfies Σ, it must be the case that $T_1 \models C_\Sigma$.

Conversely, we show that if there exists an XML tree $T_2 = (V, lab, ele, att_2, root)$ such that $T_2 \models (D, C_\Sigma)$, then we can construct an XML tree $T_1 = (V, lab, ele, att_1, root)$ such that $T_1 \models (D, \Sigma)$. We construct T_1 from T_2 by modifying the function att_2 while leaving V, lab, ele, and $root$ unchanged. More specifically, let $S = \{\tau.@l \mid \tau \in E, @l \in R(\tau)\}$. To define the new function, denoted by att_1, we first associate a set of string values with each $\tau.@l$ in S. Let N be the maximum cardinality of $values(\tau.@l)$ in T_2, i.e., $N \geq |values(\tau.@l)|$ in T_2 for all $\tau.@l \in S$. Let $V_S = \{a_i \mid i \in [1, N]\}$ be a set of distinct string values. For each $\tau.@l \in S$, let $V_{\tau.@l} = \{a_i \mid i \in [1, |values(\tau.@l)|]\}$, and for each $x \in ext(\tau)$, let $att_1(x, @l)$ be a string value in $V_{\tau.@l}$ such that, in T_1, $values(\tau.@l) = V_{\tau.@l}$. The value $att_1(x, @l)$ can be selected in such a way that, for each key $\varphi = \tau[@l_1, \ldots, @l_k] \to \tau$ in Σ, $x[@l_1, \ldots, @l_k]$ is a distinct list of string values from $V_{\tau.@l_1} \times \cdots \times V_{\tau.@l_k}$. This is possible because by the definition of T_1, (1) $ext(\tau)$ in T_1 equals $ext(\tau)$ in T_2; (2) $|values(\tau.@l)|$ in T_1 equals $|values(\tau.@l)|$ in T_2; (3) $T_2 \models C_\Sigma$ and $|ext(\tau)| \leq |values(\tau.@l_1)| \cdot \ldots \cdot |values(\tau.@l_k)|$ is in C_Σ; and (4) since φ is the only key defined for τ-elements, when we populate attributes $@l_1, \ldots, l_k$ of x, we need only to select the value of $att_1(x, @l_i)$ from $V_{\tau.@l_i}$ such that $x[@l_1, \ldots, @l_k]$ is distinct, without worrying about whether the population may hamper "other keys" defined on x (note that, in the absence of the primary key assumption, the populations of different keys may interact with each other, and, as a result, the

simple population strategy given above may no longer work; this is why we assume primary keys). It should be noted that it may be the case that $V_{\tau_1.@l_1} \subseteq V_{\tau_2.@l_2}$ even if Σ does not imply $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$. This does not lose generality as we do not intend to capture negation of foreign keys. We next show that T_1 is indeed what we want.

It is easy to verify that $T_1 \models D$ given the construction of T_1 from T_2 and the assumption that $T_2 \models D$. To show that $T_1 \models \Sigma$, we consider $\varphi \in \Sigma$ in the following cases. (1) If φ is a key $\tau[@l_1, \ldots, @l_k] \rightarrow \tau$, it is immediate from the definition of T_1 that $T_1 \models \varphi$ since for any $x \in ext(\tau)$, $x[@l_1, \ldots, @l_k]$ is a distinct list of string values from $V_{\tau.@l_1} \times \cdots \times V_{\tau.@l_k}$. (2) If φ is $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then $T_2 \models |values(\tau_1.@l_1)| \leq |values(\tau_2.@l_2)|$ by $T_2 \models C_\Sigma$. By the definition of att_1, for $i = 1, 2$, $V_{\tau_i.@l_i} = \{a_i \mid i \in [1, |values(\tau_i.@l_i)|]\}$ and in T_1, $values(\tau_i.@l_i) = V_{\tau_i.@l_i}$. Thus $values(\tau_1.@l_1) \subseteq values(\tau_2.@l_2)$ in T_1. Furthermore, given that $|ext(\tau_2)| \leq |values(\tau_2.@l_2)|$ and $|values(\tau_2.@l_2)| \leq |ext(\tau_2)|$ are both in C_Σ, $T_2 \models C_\Sigma$, $|ext(\tau_2)|$ in T_2 is equal to $|ext(\tau_2)|$ in T_1, and $|values(\tau_2.@l_2)|$ in T_2 is equal to $|values(\tau_2.@l_2)|$ in T_1, we conclude that $|ext(\tau_2)|$ is equal to $|values(\tau_2.@l_2)|$ in T_1 and, hence, $T_1 \models \tau_2.@l_2 \rightarrow \tau_2$ since each $x \in ext(\tau_2)$ in T_1 has a distinct $@l_2$-attribute value and thus the value of its $@l_2$-attribute uniquely identifies x among nodes in $ext(\tau_2)$. Therefore, $T_1 \models \varphi$ and, thus, $T_1 \models (D, \Sigma)$. This concludes the proof of the lemma. □

The above lemma takes care of coding the constraints; the next step is to code DTDs. For that, we use the technique developed in [16]: for each narrow DTD D, one can compute in polynomial time in the size of D a set Ψ_D of linear inequalities on nonnegative integers, referred to as the set of cardinality constraints determined by D, which includes $|ext(\tau)|$ as a variable for each element type τ in D, but it does not have $|values(\tau.@l)|$ as a variable for any attribute $@l$ of τ. More specifically, for each symbol $\tau \in E \cup \{\mathtt{S}\}$, $|ext(\tau)|$ is treated as a distinct variable, which keeps track of the number of all τ elements in an XML tree T conforming to D. In addition, for each occurrence of τ in the definition $P(\tau')$ of some element type τ', we also create distinct variables as follows: if $P(\tau') = \tau_1$ for $\tau_1 \in E \cup \{\mathtt{S}\}$, then we create a distinct variable $x^1_{\tau_1,\tau'}$; if $P(\tau') = (\tau_1, \tau_2)$ or $P(\tau') = (\tau_1|\tau_2)$, then we create two distinct variables $x^1_{\tau_1,\tau'}$ and $x^2_{\tau_2,\tau'}$. Intuitively, for $i \in [1, 2]$, $x^i_{\tau_i,\tau'}$ keeps track of the number of τ_i subelements at position i under all τ' elements in T. Let X_τ be the set of all variables of the form $x^i_{\tau,\tau'}$. By using these variables, for each $\tau \in E$, we define a set ψ_τ of linear integer constraints that characterizes $P(\tau)$ quantitatively, as follows:

- If $P(\tau) = \tau_1$ for $\tau_1 \in E \cup \{\mathtt{S}\}$, then ψ_τ includes $|ext(\tau)| = x^1_{\tau_1,\tau}$. Referring to an XML tree T that conforms to D, this assures that each τ element has a unique τ_1 subelement.
- If $P(\tau') = (\tau_1, \tau_2)$, then ψ_τ includes $|ext(\tau)| = x^1_{\tau_1,\tau}$ and $|ext(\tau)| = x^2_{\tau_2,\tau}$. These assure that each τ element in T must have a unique τ_1 subelement and a unique τ_2 subelement.
- If $P(\tau') = (\tau_1|\tau_2)$, then ψ_τ includes $|ext(\tau)| = x^1_{\tau_1,\tau} + x^2_{\tau_2,\tau}$. These assure that each τ element in T must have either a τ_1 subelement or a τ_2 subelement, and thus the sum of the numbers of these τ_1 and τ_2 subelements equals the number of τ elements in T.

The set Ψ_D of cardinality constraints determined by DTD D consists of the following:

- $|ext(r)| = 1$; i.e., there is a unique root in any XML tree valid w.r.t. D;
- constraints of ψ_τ for each $\tau \in E$; these assure that $P(\tau)$ is satisfied;
- $|ext(\tau)| = \sum_{x^i_{\tau,\tau'} \in X_\tau} x^i_{\tau,\tau'}$ for each $\tau \in (E \setminus \{r\}) \cup \{\mathtt{S}\}$; this indicates that the set $ext(\tau)$ includes all τ elements no matter where they occur in an XML tree;

- $x \geq 0$ for any variable x used above; i.e., the number of elements (subelements) is nonnegative.

It has been shown [16] that Ψ_D has a nonnegative integer solution iff there exists an XML tree T conforming to D such that the cardinality of $ext(\tau)$ in T equals the value of the variable $|ext(\tau)|$ in the solution for each element type τ in D.

We now combine this coding with the coding for $\mathcal{AC}^{*,1}_{PK,FK}$-constraints. Given a narrow DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D, we define *the set of cardinality constraints determined by* D *and* Σ to be

$$\Psi(D, \Sigma) \;=\; \Psi_D \cup C_\Sigma \cup \{(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0) \mid \tau \in E,\ @l \in R(\tau)\},$$

where C_Σ is the set of cardinality constraints determined by Σ, Ψ_D is the set of cardinality constraints determined by D, and constraints $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ are to ensure that every τ-element has an $@l$-attribute (note that $|values(\tau.@l)| \leq |ext(\tau)|$ is already in C_Σ). Constraints in $\Psi(D, \Sigma)$ are either linear integer constraints, or inequalities of the form $x \leq y \cdot z$, which come from C_Σ, or constraints of the form $x > 0 \rightarrow y > 0$. Note that, if we leave out constraints of the form $x > 0 \rightarrow y > 0$, $\Psi(D, \Sigma)$ is a prequadratic Diophantine system. Also note that $\Psi(D, \Sigma)$ can be computed in polynomial time in the size of D and Σ.

We say that $\Psi(D, \Sigma)$ is *consistent* iff $\Psi(D, \Sigma)$ admits a nonnegative integer solution. That is, there is a nonnegative integer assignment to the variables in $\Psi(D, \Sigma)$ such that all of the constraints in $\Psi(D, \Sigma)$ are satisfied.

LEMMA 3.4. *Let D be a narrow DTD and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then $\Psi(D, \Sigma)$ is consistent iff there is an XML tree T such that $T \models (D, \Sigma)$.*

Proof. Suppose that there exists an XML tree T such that $T \models (D, \Sigma)$. Then there is a nonnegative integer solution to Ψ_D such that for each element type τ in D, the value of the variable $|ext(\tau)|$ equals the number of τ-elements in T [16]. By Lemma 3.3 and $T \models \Sigma$, we have $T \models C_\Sigma$. We extend the solution of Ψ_D to be one to $\Psi(D, \Sigma)$ by letting the variable $|values(\tau.@l)|$ equal the number of distinct $@l$-attribute values of all τ-elements in T, for each element type τ and attribute $@l$ of τ in D. Since $T \models C_\Sigma$, this extended assignment satisfies all of the constraints in C_Σ. In addition, if $|ext(\tau)| > 0$, then $|values(\tau.@l)| > 0$ since every τ-element in T has an $@l$-attribute. Hence the assignment is indeed a nonnegative solution to $\Psi(D, \Sigma)$, and, therefore, $\Psi(D, \Sigma)$ is consistent.

Conversely, suppose that $\Psi(D, \Sigma)$ admits a nonnegative integer solution. Then there exists an XML tree T such that $T \models D$, and, moreover, for each element type τ in D, the cardinality of $ext(\tau)$ in T equals the value of the variable $|ext(\tau)|$ in the solution [16]. We construct a new tree T' from T by modifying the definition of the function att such that in T', for each element type τ and attribute $@l$ of τ, the number of distinct $@l$-attribute values of all τ-elements equals the value of the variable $|values(\tau.@l)|$ in the solution. This is possible since $|values(\tau.@l)| \leq |ext(\tau)|$ and $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ are in $\Psi(D, \Sigma)$. The assignment is also a solution to C_Σ. Thus $T' \models D$ and $T' \models C_\Sigma$. Hence by Lemma 3.3, there exists an XML tree T'' such that $T'' \models (D, \Sigma)$. This concludes the proof of the lemma. □

We now conclude the proof of reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE. By Lemma 3.2, given an arbitrary DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints, one can compute a narrow DTD D_N such that (D, Σ) is consistent iff (D_N, Σ) is consistent. By Lemma 3.4, (D_N, Σ) is consistent iff $\Psi(D_N, \Sigma)$ has a nonnegative integer solution. Such a solution requires $|values(\tau.@l)| > 0$ if $|ext(\tau)| > 0$. To ensure this, let $\Phi(D_N, \Sigma)$ be a system that includes all linear integer constraints and pre-

quadratic constraints in $\Psi(D_N, \Sigma)$ and, moreover, $|ext(\tau)| \leq |values(\tau.@l)| \cdot |ext(\tau)|$ for each $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ in $\Psi(D_N, \Sigma)$. Now $\Phi(D_N, \Sigma)$ is a prequadratic Diophantine system. In addition, $\Psi(D_N, \Sigma)$ has a nonnegative integer solution iff $\Phi(D_N, \Sigma)$ has a nonnegative integer solution. To see this, observe that, for any nonnegative integer assignment to $|ext(\tau)|$ and $|values(\tau.@l)|$, $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ iff $|ext(\tau)| \leq |values(\tau.@l)| \cdot |ext(\tau)|$. Thus, (D, Σ) is consistent iff the prequadratic Diophantine system $\Phi(D_N, \Sigma)$ has a nonnegative integer solution. Note that D_N can be computed in polynomial time in the size of D, $\Psi(D_N, \Sigma)$ can be computed in polynomial time in the size of D_N and Σ, and $\Phi(D_N, \Sigma)$ can be computed in polynomial time in the size of $\Psi(D_N, \Sigma)$. Hence, it takes polynomial time to compute $\Phi(D_N, \Sigma)$ from D and Σ. Therefore, there is a PTIME reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE.

(b) *A reduction from PDE to* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. We now move to the other direction. Given an instance of PDE, i.e., a system S consisting of a set S_L of linear equations/inequalities on integers and a set S_P of prequadratic constraints of the form $x \leq y \cdot z$, we define a DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints such that S has a nonnegative solution iff there is an XML tree T satisfying Σ and conforming to D. We use $X = \{x_i \mid i \in [1, n]\}$ to denote the set of all of the variables in S. Assume that $S_L = \{e_j \mid j \in [1, m]\}$ and e_j is of the form: $a^j_1\ x_1 + \cdots + a^j_n\ x_n + c_j \leq b^j_1\ x_1 + \cdots + b^j_n\ x_n + d_j$, where a^j_i $(i \in [1, n])$, b^j_i $(i \in [1, n])$, c_j, and d_j are nonnegative integers.[2] Also, assume that $S_P = \{p_j \mid j \in [1, l]\}$, where p_j is a prequadratic equation of the form $x \leq y \cdot z$. Then we define DTD $D = (E,\ A,\ P,\ R,\ r)$ as follows:

(1) For each variable x_i, we define an element type X_i. In addition, for each $p_s \in S_P$ of the form $x_i \leq x_j \cdot x_k$, we define an element type U^s_i. For each linear constraint e_j, we define distinct element types E_j, A^j_1, …, A^j_n, C_j, F_j, B^j_1, …, B^j_n, D_j. We use r to denote the root element type. That is,

$$E\ =\ \{r\} \cup \{X_i \mid i \in [1, n]\} \cup$$
$$\{E_j, A^j_1, \ldots, A^j_n, C_j, F_j, B^j_1, \ldots, B^j_n, D_j \mid j \in [1, m]\} \cup \{U^s_i \mid p_s = x_i \leq x_j \cdot x_k \in S_P\}.$$

Intuitively, referring to an XML tree conforming to D, we use $|ext(X_i)|$ to code the value of the variable x_i in S. For every equation e_j, we use $|ext(A^j_1)|$, …, $|ext(A^j_n)|$, $|ext(C_j)|$ to code the values of constants a^j_1, …, a^j_n, c_j; $|ext(E_j)|$ to code the value of the expression $a^j_1 x_1 + \cdots + a^j_n x_n + c_j$; $|ext(B^j_1)|$, …, $|ext(B^j_n)|$, $|ext(D_j)|$ to code the values of constants b^j_1, …, b^j_n, d_j; and $|ext(F_j)|$ to code the value of the expression $b^j_1 x_1 + \cdots + b^j_n x_n + d_j$. Furthermore, for each prequadratic equation $p_s = x_i \leq x_j \cdot x_k$ in S_P, we create a distinct copy U^s_i of X_i. The reason to use U^s_i instead of X_i is to ensure that the set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints defined below is primary.

(2) $A = \{@c, @d, @e\}$. Intuitively, we shall define $@e$ as a key and use $@c$ and $@d$ to code prequadratic constraints of the form $x \leq y \cdot z$.

(3) We define production rules as follows. For the root of the DTD:

$$P(r)\ =\ (X_1, U^{s_{1,1}}_1, \ldots, U^{s_{1,j_1}}_1)^*, \ldots, (X_n, U^{s_{n,1}}_n, \ldots, U^{s_{n,j_n}}_n)^*,$$
$$\underbrace{C_1, \ldots, C_1}_{c_1 \text{ times}}, \ldots, \underbrace{C_m, \ldots, C_m}_{c_m \text{ times}}, \underbrace{D_1, \ldots, D_1}_{d_1 \text{ times}}, \ldots, \underbrace{D_m, \ldots, D_m}_{d_m \text{ times}},$$

where $\{s_{i,1}, \ldots, s_{i,j_i}\}$ $(i \in [1, n])$ is the set of indexes $\{s \mid p_s = x_i \leq x_j \cdot x_k \in S_P\}$.

[2] For example, we represent the equation $-3x + 5y \leq -7$ as $0x + 5y + 7 \leq 3x + 0y + 0$.

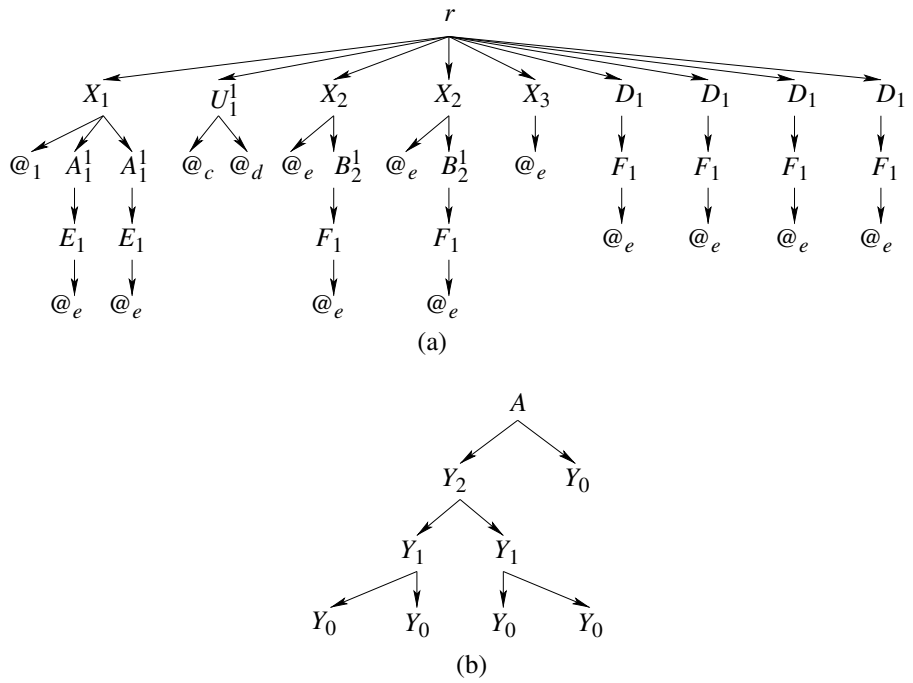


FIG. 3.1. *Trees used in the proof of Theorem* 3.1.

Furthermore, for every $i \in [1,n]$ and every $j \in [1,m]$:

$$\begin{array}{lcl} P(A_i^j) & = & E_j, \\ P(C_j) & = & E_j, \\ P(B_i^j) & = & F_j, \\ P(D_j) & = & F_j, \\ P(X_i) & = & \underbrace{A_i^1,\ldots,A_i^1}_{a_i^1 \text{ times}},\ldots,\underbrace{A_i^m,\ldots,A_i^m}_{a_i^m \text{ times}},\underbrace{B_i^1,\ldots,B_i^1}_{b_i^1 \text{ times}},\ldots,\underbrace{B_i^m,\ldots,B_i^m}_{b_i^m \text{ times}}. \end{array}$$

Finally, for every $i \in [1,n]$ and every $s \in [1,l]$ such that $p_s = x_i \le x_j \cdot x_k \in S_P$, $P(U_i^s) = \epsilon$.

(4) We define the attribute function R as follows: for every $j \in [1,m]$, $R(E_j) = R(F_j) = \{@e\}$. In addition, for every $i \in [1,n]$, $R(X_i) = \{@e\}$, and for every $s \in [1,l]$ such that $p_s = x_i \le x_j \cdot x_k \in S_P$, $R(U_i^s) = \{@c, @d\}$. For all other element types τ, let $R(\tau)$ be empty.

For example, Figure 3.1(a) shows an XML tree conforming to the DTD constructed from the set of equations $S_L = \{2x_1 \le x_2 + 4\}$ and $S_P = \{x_1 \le x_2 \cdot x_3\}$. We note that this tree codes the solution $x_1 = 1$, $x_2 = 2$, $x_3 = 1$ for this system of equations.

Given DTD D, we define a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. For each $j \in [1,m]$, Σ includes keys $E_j.@e \to E_j$ and $F_j.@e \to F_j$ and foreign key $E_j.@e \subseteq_{FK} F_j.@e$. Furthermore, for every $i,j,k \in [1,n]$ and $s \in [1,l]$ such that $p_s = x_i \le x_j \cdot x_k \in S_P$, Σ includes the following constraints:

$$U_i^s[@c, @d] \to U_i^s, \qquad U_i^s.@c \subseteq_{FK} X_j.@e, \qquad U_i^s.@d \subseteq_{FK} X_k.@e.$$

Clearly, the set Σ is primary; i.e., for any element type τ there is at most one key defined. In fact, we use copies U_i^s of X_i just to ensure that Σ is primary.

We next show that the encoding is indeed a reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. Suppose that S has a nonnegative solution. Then we construct an XML tree T conforming to D as shown in Figure 3.1(a). That is, for each $i \in [1,n]$ we let $|ext(X_i)|$ be the value of the variable x_i in the solution. We note that, by the definition of D, this implies that, for every $s \in [1,l]$ such that $p_s = x_i \le x_j \cdot x_k \in S_P$, $|ext(U_i^s)|$ is also equal to the value of x_i in the solution. For every $i \in [1,n]$ and every X_i-element x in T, we let $x.@e$ be a distinct value such that, in T, $|values(X_i.@e)| = |ext(X_i)|$. For every $j \in [1,m]$ and every E_j-element x in T, we let $x.@e$ be a distinct value such that, in T, $|values(E_j.@e)| = |ext(E_j)|$. Likewise, we assign values to the $@e$-attribute of the nodes in $ext(F_j)$ in such a way that $|values(F_j.@e)| = |ext(F_j)|$ in T. Finally, for every $i,j,k \in [1,n]$ and $s \in [1,l]$ such that $p_s = x_i \le x_j \cdot x_k \in S_P$, and for every node x in T of type U_i^s, we let $x[@c,@d]$ be a distinct list of string values from $values(X_j.@e) \times values(X_k.@e)$. This is possible since $x_i \le x_j \cdot x_k \in S_P$ and by the definition of T, $|ext(U_i^s)| = |ext(X_i)| = x_i$, $|values(X_j.@e)| = |ext(X_j)| = x_j$, and $|values(X_k.@e)| = |ext(X_k)| = x_k$. Since T codes a solution of S, it is straightforward to prove that $T \models C_\Sigma$, the set of cardinality constraints determined by Σ. Thus, by Lemma 3.3 we conclude that there exists an XML tree T' such that $T' \models (D,\Sigma)$, and, hence, (D,Σ) is consistent. Conversely, suppose that there exists an XML tree T such that $T \models (D,\Sigma)$. We construct a solution of S by letting variable x_i equal $|ext(X_i)|$ in T. By the definitions of D and Σ, it is easy to verify that this is indeed a nonnegative integer solution for S. In particular, each $p_s = x_i \le x_j \cdot x_k$ in S_P holds because $T \models (D,\Sigma)$, and, thus, $|ext(X_i)| = |ext(U_i^s)| \le |values(U_i^s.@c)| \cdot |values(U_i^s.@d)| \le |values(X_j.@e)| \cdot |values(X_k.@e)| \le |ext(X_j)| \cdot |ext(X_k)|$.

We observe that the previous reduction is not polynomial since constants a_i^j, b_i^j ($i \in [1,n]$, $j \in [1,m]$) and c_j, d_j ($j \in [1,m]$) are coded in unary. To overcome this problem, next we show how to code in a DTD the binary representation of a number. We introduce this coding separately to simplify the presentation of this proof.

Assume that $a = \sum_{i=0}^{k} a_i \cdot 2^i$, where each a_i ($i \in [0,k-1]$) is either 0 or 1 and $a_k = 1$; that is, the binary representation of a is $a_k a_{k-1} \dots a_1 a_0$. To code a in a DTD we include element types $A, Y_0, \dots, Y_k$, and we define P on these elements as follows:

$$P(Y_i) = \begin{cases} \epsilon & i = 0, \\ Y_{i-1}, Y_{i-1} & \text{otherwise}, \end{cases}$$

and $P(A) = Y_{i_1}, \dots, Y_{i_l}$, where $i_1 > \dots > i_l \ge 0$ and $\{i_1, \dots, i_l\}$ is the set of indexes $\{j \in [0,k] \mid a_j = 1\}$. We note that the size of this set of rules is polynomial in the size of a. Furthermore, if an XML tree T conforms to this DTD, then $|ext(Y_0)| = a$ in T. For example, if $a = 5$, then $P(A) = Y_2, Y_0$, $P(Y_2) = Y_1, Y_1$, $P(Y_1) = Y_0, Y_0$, and $P(Y_0) = \epsilon$, and an XML tree conforming to these rules is of the form shown in Figure 3.1(b).

Thus, by using this coding in our original reduction of PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$, we can show that there is a PTIME reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. This completes the proof of Theorem 3.1. □

It is known that the linear integer programming problem is NP-hard and PDE is in NEXPTIME. Thus from Theorem 3.1 follows immediately the corollary below.

COROLLARY 3.5. *$\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ is NP-hard and can be solved in NEXPTIME.*

Obviously we cannot obtain the exact complexity of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ without resolving the corresponding question for PDE, which appears to be quite hard [21]. The

result of Theorem 3.1 can be generalized to *disjoint* $\mathcal{AC}^{*,1}_{K,FK}$-constraints: that is, a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints in which, for every element type τ and every two distinct keys $\tau[X] \to \tau$ and $\tau[Y] \to \tau$ in Σ (including key dependencies defined by foreign key constraints), $X \cap Y = \emptyset$. The proof of Theorem 3.1 applies almost verbatim to show the following.

COROLLARY 3.6. *The restriction of* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ *to disjoint constraints is polynomially equivalent to PDE, and, thus, it is NP-hard and can be solved in NEXPTIME.*

3.2. Consistency of regular expression constraints. Specifications of $\mathcal{AC}^{*,*}_{K,FK}$-constraints are associated with element types. We next consider $\mathcal{AC}^{reg}_{K,FK}$, the class of unary keys and foreign keys defined in terms of regular path expressions. For $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we are able to establish both an upper and a lower bound. The lower bound already indicates that the problem is perhaps infeasible in practice, even for very simple DTDs. Finding the precise complexity of the problem remains open and does not appear to be easy. In fact, even the current proof of the upper bound is quite involved and relies on combining the techniques from [16] for coding DTDs and constraints with integer linear inequalities and from [1] for reasoning about constraints given by regular expressions by using the product automaton for all of the expressions involved in the constraints.

THEOREM 3.7.

(a) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *can be solved in* 2*-NEXPTIME.*

(b) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *is PSPACE-hard, even for nonrecursive no-star DTDs.*

Proof. We reduce $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ to the existence of solution of an (almost) instance of linear integer programming, which happens to be of double-exponential size; hence the 2-NEXPTIME bound. For the lower bound, we encode the quantified boolean formula problem (QBF) as an instance of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$.

Proof of (a). The proof is a bit long, so we first give a rough outline. The idea is similar to the proof of the NP membership for $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ [16]: we code both the DTD and the constraints with linear inequalities over integers. However, compared to the proof of [16], the current proof is considerably harder due to the following. First, regular expressions in DTDs ("horizontal" regular expressions) interact in a certain way with regular expressions in integrity constraints (those correspond to "vertical" paths through the trees). To eliminate this interaction, we first show how to reduce the problem to that over *narrow* DTDs, in which no wide horizontal regular expressions are allowed. The next problem is that regular expressions in constraints can interact with each other. Thus, to model them with linear inequalities, we extend the approach of [16] by taking into account all possible boolean combinations of regular languages given by expressions used in constraints. The last problem is coding the DTDs in such a way that variables corresponding to each node have the information about the path leading to the node and its relationship with the regular expressions used in constraints. For that, we adopt the technique of [1], and tag all of the variables in the coding of DTDs with states of a certain automaton (the product automaton for all of the automata corresponding to the regular expressions used in constraints).

Now it is time to fill in all of the details. First, we need some additional notation. For every regular expression β and every attribute $@l$, we write $values(\beta.@l)$ to denote the set $\{y.@l \mid y \in nodes(\beta) \text{ and } y.@l \text{ is defined}\}$. Observe that, for any $\tau \in E \setminus \{r\}$ and $@l \in R(\tau)$, $values(r.\_^*.\tau.@l)$ corresponds to our original definition of $values(\tau.@l)$.

We say that a DTD D is *one-attribute* if D contains only one attribute and no element type τ such that $P(\tau) = \mathsf{S}$. We start by showing that $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$

can be reduced to the consistency problem for regular expression constraints over one-attribute DTDs. Let $D = (E, A, P, R, r)$ be a DTD and Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D. First, define DTD $D_U = (E_U, A_U, P_U, R_U, r)$ as follows. For every $\tau \in E$ and $@l \in R(\tau)$, assume that $\tau_{@l}$ is a fresh element type symbol. Then define E_U as $E \cup \{\tau_{@l} \mid \tau \in E \text{ and } @l \in R(\tau)\}$ and $A_U = \{@e\}$, where $@e$ is a fresh attribute symbol. Furthermore, define functions P_U and R_U as follows:

- For every $\tau \in E$ such that $P(\tau) = \mathtt{S}$, if $R(\tau) = \{@l_1, \ldots, @l_n\}$, where $n \geq 0$, then $P_U(\tau) = \tau_{@l_1}, \ldots, \tau_{@l_n}$ and $R_U(\tau) = \emptyset$.
- For every $\tau \in E$ such that $P(\tau)$ is a regular expression over E, if $R(\tau) = \{@l_1, \ldots, @l_n\}$, where $n \geq 0$, then $P_U(\tau) = P(\tau), \tau_{@l_1}, \ldots, \tau_{@l_n}$ and $R_U(\tau) = \emptyset$.
- For every $\tau \in E$ and $@l \in R(\tau)$, $P_U(\tau_{@l}) = \epsilon$ and $R_U(\tau_{@l}) = \{@e\}$.

We note that if $P(\tau) = \mathtt{S}$ and $R(\tau) = \emptyset$, then $P_U(\tau) = \epsilon$.

Second, define the set Σ_U of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_U as follows. For every key constraint $\beta.\tau.@l \rightarrow \beta.\tau$ in Σ, we include $\beta.\tau.\tau_{@l}.@e \rightarrow \beta.\tau.\tau_{@l}$ in Σ_U, and, for every foreign key constraint $\beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$ in Σ, we add $\beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$ to Σ_U.

LEMMA 3.8. *Let D be a DTD, Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and D_U, Σ_U be as defined above. Then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$ iff there exists an XML tree T_2 such that $T_2 \models (D_U, \Sigma_U)$.*

Proof. ($\Rightarrow$) Let $T_1 = (V_1, lab_1, ele_1, att_1, root)$ be an XML tree such that $T_1 \models (D, \Sigma)$. We define an XML tree T_2 from T_1 such that $T_2 \models (D_U, \Sigma_U)$. More specifically, $T_2 = (V_2, lab_2, ele_2, att_2, root)$, where V_2, lab_2, ele_2, and att_2 are defined as follows. Let v be a node in T_1 such that $lab_1(v) = \tau \in E$ and $R(\tau) = \{@l_1, \ldots, @l_k\}$. Then V_2 contains node v and fresh nodes $v_{@l_1}, \ldots, v_{@l_k}$ such that $lab_2(v) = \tau$ and $lab_2(v_{@l_i}) = \tau_{@l_i}$, for every $i \in [1, k]$. Furthermore, if $ele_1(v) = [s]$, where $s \in \mathtt{S}$, then $ele_2(v) = [v_{@l_1}, \ldots, v_{@l_k}]$. Otherwise, $ele_1(v) = [v_1, \ldots, v_n]$, where $n \geq 0$ and each v_i is an element node, and $ele_2(v) = [v_1, \ldots, v_n, v_{@l_1}, \ldots, v_{@l_k}]$. Finally, $att_2(v, @e)$ is not defined and $att_2(v_{@l_i}, @e) = att_1(v, @l_i)$ for every $i \in [1, k]$. Next we show that $T_2 \models (D_U, \Sigma_U)$.

By the definition of D_U and given that $T_1 \models D$, it is easy to see that $T_2 \models D_U$. Assume that $T_2 \not\models \Sigma_U$. Then there exist $\varphi \in \Sigma_U$ such that $T_2 \not\models \varphi$. (1) If φ is a key $\beta.\tau.\tau_{@l}.@e \rightarrow \beta.\tau.\tau_{@l}$, then there exist distinct $v_1, v_2 \in nodes(\beta.\tau.\tau_{@l})$ in T_2 such that $att_2(v_1, @e) = att_2(v_2, @e)$. Let u_1 and u_2 be the parents of v_1 and v_2 in T_2, respectively. By the definition of D_U and given that $v_1 \neq v_2$, we have $u_1 \neq u_2$. Thus, by the definition of T_2, u_1 and u_2 are nodes in T_1 such that $u_1, u_2 \in nodes(\beta.\tau)$ and $att_1(u_1, @l) = att_1(u_2, @l) = att_2(v_1, @e)$. Therefore, $T_1 \not\models \beta.\tau.@l \rightarrow \beta.\tau$, which contradicts the assumption that $T_1 \models \Sigma$. (2) If φ is a foreign key $\beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$, then either $T_2 \not\models \beta'.\tau'.\tau'_{@l'}.@e \rightarrow \beta'.\tau'.\tau'_{@l'}$ or there exists $v \in nodes(\beta.\tau.\tau_{@l})$ such that $att_2(v, @e) \notin values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2. In the former case, we reach a contradiction as in (1). In the latter case, assume that u is the parent of v in T_2. By the definition of T_2, we have that u is a node in T_1 such that $u \in nodes(\beta.\tau)$ and $att_1(u, @l) = att_2(v, @e)$. Thus, given that $values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2 is equal to $values(\beta'.\tau'.@l')$ in T_1, we conclude that $att_1(u, @l) \notin values(\beta'.\tau'.@l')$ in T_1. Therefore, $T_1 \not\models \beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$, which contradicts the assumption that $T_1 \models \Sigma$.

($\Leftarrow$) Let $T_2 = (V_2, lab_2, ele_2, att_2, root)$ be an XML tree such that $T_2 \models (D_U, \Sigma_U)$. We define an XML tree T_1 from T_2 such that $T_1 \models (D, \Sigma)$. More specifically, $T_1 = (V_1, lab_1, ele_1, att_1, root)$, where V_1, lab_1, ele_1, and att_1 are defined as follows. Let v be a node in T_2 such that $lab_2(v) = \tau$, $\tau \in E$, and $R(\tau) = \{@l_1, \ldots, @l_k\}$.

Then V_1 also contains node v with $lab_1(v) = \tau$. Furthermore, if $P(\tau) = \mathtt{S}$, then $ele_2(v) = [v_{@l_1}, \ldots, v_{@l_k}]$, where $lab(v_{@l_j}) = \tau_{@l_j}$ $(j \in [1,k])$, and we define $ele_1(v)$ as $[s]$, where s is an arbitrary string in S, and we define $att_1(v, @l_i)$ as $att_2(v_{@l_i}, @e)$ for every $i \in [1,k]$. Otherwise, $P(\tau)$ is a regular expression over E and $ele_2(v) = [v_1, \ldots, v_n, v_{@l_1}, \ldots, v_{@l_k}]$, where $lab(v_i) \in E$ $(i \in [1,n])$ and $lab(v_{@l_j}) = \tau_{@l_j}$ $(j \in [1,k])$, and we define $ele_1(v)$ as $[v_1, \ldots, v_n]$ and $att_1(v, @l_i)$ as $att_2(v_{@l_i}, @e)$ for every $i \in [1,k]$. Next we show that $T_1 \models (D, \Sigma)$.

By the definition of D_U and given that $T_2 \models D_U$, it is easy to see that $T_1 \models D$. Assume that $T_1 \not\models \Sigma$. Then there exists $\varphi \in \Sigma$ such that $T_1 \not\models \varphi$. (1) If φ is a key $\beta.\tau.@l \rightarrow \beta.\tau$, then there exist distinct $u_1, u_2 \in nodes(\beta.\tau)$ in T_1 such that $att_1(u_1, @l) = att_1(u_2, @l)$. By the definition of T_1, u_1 and u_2 are also in $nodes(\beta.\tau)$ in T_2. Let v_1 and v_2 be the children of u_1 and u_2 in T_2 of type $\tau_{@l}$, respectively. Given that $u_1 \neq u_2$, we have $v_1 \neq v_2$. Thus, by the definition of T_1, v_1 and v_2 are nodes in T_2 such that $v_1, v_2 \in nodes(\beta.\tau.\tau_{@l})$ and $att_2(v_1, @e) = att_2(v_2, @e) = att_1(u_1, @l)$. Therefore, $T_2 \not\models \beta.\tau.\tau_{@l}.@e \rightarrow \beta.\tau.\tau_{@l}$, which contradicts the assumption that $T_2 \models \Sigma_U$. (2) If φ is a foreign key $\beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$, then either $T_1 \not\models \beta'.\tau'.@l' \rightarrow \beta'.\tau'$ or there exists $u \in nodes(\beta.\tau)$ such that $att_1(u, @l) \notin values(\beta'.\tau'.@l')$ in T_1. In the former case, we reach a contradiction as in (1). In the latter case, assume that v is the child of u in T_2 of type $\tau_{@l}$ (u is a node of T_2 by the definition of T_1). By the definition of T_1, we have $v \in nodes(\beta.\tau.\tau_{@l})$ and $att_2(v, @e) = att_1(u, @l)$. Thus, given that $values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2 is equal to $values(\beta'.\tau'.@l')$ in T_1, we conclude that $att_2(v, @e) \notin values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2. Therefore, $T_2 \not\models \beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$, which contradicts the assumption that $T_2 \models \Sigma_U$. This concludes the proof of the lemma. □

By Lemma 3.8, from now on we consider only one-attribute DTDs. Let $D = (E, \{@l\}, P, R, r)$ be a one-attribute DTD and $D_N = (E_N, \{@l\}, P_N, R_N, r)$ be the narrow DTD of D (defined in the proof of Theorem 3.1). Observe that D_N is also one-attribute. Furthermore, observe that an XML tree T valid w.r.t. D may not conform to D_N and vice versa. In addition, an $\mathcal{AC}^{reg}_{K,FK}$-constraint φ over D may be satisfied by T, but it may not be satisfied by any XML tree conforming to D_N. To explore the connection between XML trees conforming to D and those conforming to D_N, we replace $\mathcal{AC}^{reg}_{K,FK}$-constraints over D by new $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_N. More precisely, given a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, we define a set Σ_N of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_N, referred to as the *narrowed set of constraints of* Σ, as follows. Let f be a substitution for the element types in E defined as $f(\tau) = \tau.(E_N \setminus E)^*$ for every $\tau \in E$. Then for every key constraint $\beta.\tau.@l \rightarrow \beta.\tau$ in Σ, $f(\beta).\tau.@l \rightarrow f(\beta).\tau$ is in Σ_N, and, for every foreign key constraint $\beta_1.\tau_1.@l \subseteq_{FK} \beta_2.\tau_2.@l$ in Σ (recall that $@l$ is the only attribute of D), $f(\beta_1).\tau_1.@l \subseteq_{FK} f(\beta_2).\tau_2.@l$ is in Σ_N.

We are now ready to establish the connection between D and D_N, which allows us to consider only narrow DTDs from now on.

LEMMA 3.9. *Let D be a one-attribute DTD, D_N the narrowed DTD of D, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and Σ_N the narrowed set of constraints of Σ. Then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$ iff there exists an XML tree T_2 such that $T_2 \models (D_N, \Sigma_N)$.*

Proof. It suffices to show the following.

Claim. Given any XML tree $T_1 \models D$, one can construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$, and vice versa. Furthermore, for any regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1, and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1, where f is the substitution defined above.

If the claim holds, we can show the lemma as follows. Assume that there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$. By the claim, there is T_2 such that $T_2 \models D_N$. Suppose, by contradiction, that there is $\varphi \in \Sigma_N$ such that $T_2 \not\models \varphi$. (1) If φ is a key $f(\beta).\tau.@l \rightarrow f(\beta).\tau$, then there exist two distinct nodes $x, y \in nodes(f(\beta).\tau)$ in T_2 such that $x.@l = y.@l$. In other words, $|values(f(\beta).\tau.@l)| < |nodes(f(\beta).\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|values(\beta.\tau.@l)| = |nodes(\beta.\tau)|$ in T_1 because the value $x.@l$ of each $x \in nodes(\beta.\tau)$ uniquely identifies x among $nodes(\beta.\tau)$. This contradicts the claim that $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1 and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1. (2) If φ is a foreign key $f(\beta_1).\tau_1.@l \subseteq_{FK} f(\beta_2).\tau_2.@l$, then either $T_2 \not\models f(\beta_2).\tau_2.@l \rightarrow f(\beta_2).\tau_2$ or there is $x \in nodes(f(\beta_1).\tau_1)$ such that, for all $y \in nodes(f(\beta_2).\tau_2)$ in T_2, $x.@l \neq y.@l$. In the first case, we reach a contradiction as in (1). In the second case, we have $x.@l \notin values(f(\beta_2).\tau_2.@l)$ in T_2. By the claim, $x.@l \in values(\beta_1.\tau_1.@l)$ in T_1. Since $T_1 \models \varphi$, $x.@l \in values(\beta_2.\tau_2.@l)$ in T_1. Again by the claim, we have $x.@l \in values(f(\beta_2).\tau_2.@l)$ in T_2, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, lab_1, ele_1, att, root)$ such that $T_1 \models D$, we construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$. Consider a τ-element v in T_1. Let $ele_1(v) = [v_1, \ldots, v_n]$ and $w = lab_1(v_1) \ldots lab_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \rightarrow P(\tau)$ (see proof of Theorem 3.1), respectively. Let Q_τ be the set of E symbols that appears in P_τ. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context free grammar, where Q_τ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, P_τ the set of production rules, and τ the start symbol.[3] Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_N(\tau)$ it can be verified that w is in the language defined by G. Thus there is a parse tree $T(w)$ w.r.t. the grammar G for w, and w is the frontier (the list of leaves from left to right) of $T(w)$. Without loss of generality, assume that the root of $T(w)$ is v and the leaves are $v_1, \ldots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in N_τ except that the root v is labeled τ. Intuitively, we construct T_2 by replacing each element v in T_1 by such a parse tree. More specifically, let $T_2 = (V_2, lab_2, ele_2, att, root)$. Here V_2 consists of nodes in V_1 and the internal nodes introduced in the parse trees. For each x in V_2, let $lab_2(x) = lab_1(x)$ if $x \in V_1$, and otherwise let $lab_2(x)$ be the node label of x in the parse tree where x belongs. Note that nodes in $V_2 \setminus V_1$ are elements of some type in $E_N \setminus E$. For every $x \in V_1$, let $ele_2(x)$ be the list of its children in the parse tree having x as the root. For every $x \in V_2 \setminus V_1$, let $ele_2(x)$ be the list of its children in the parse tree of an element in V_1 that contains x. Note that att remains unchanged. By the construction of T_2 it can be verified that $T_2 \models D_N$; and moreover, for every regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1 and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1 because, among other things, (1) if a string $r.\tau_1.\ldots.\tau_n.\tau$ over E is in $\beta.\tau$, then for every sequence of strings $w_0, \ldots, w_n$ in $(E_N \setminus E)^*$, $r.w_0.\tau_1.w_1.\ldots.\tau_n.w_n.\tau$ is in $f(\beta).\tau$; (2) if a string $r.w_0.\tau_1.w_1.\ldots.\tau_n.w_n.\tau$ is in $f(\beta).\tau$, where τ_1, ..., τ_n, τ are element types in E and w_0, ..., w_n are strings in $(E_N \setminus E)^*$, then $r.\tau_1.\ldots.\tau_n.\tau$ is in $\beta.\tau$; (3) none of the new nodes, i.e., nodes in $V_2 \setminus V_1$, is labeled with an E type; (4) no new attributes are defined; and (5) the ancestor-descendant relation on T_1-elements is not changed in T_2.

[3] As in the proof of Lemma 3.2, if τ is in $P(\tau)$, then we need to rename τ in Q_τ to ensure that Q_τ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.

Conversely, assume that there is $T_2 = (V_2, lab_2, ele_2, att, root)$ such that $T_2 \models D_N$. We construct an XML tree T_1 by modifying T_2 such that $T_1 \models D$. For any node $v \in V_2$ with $lab(v) = \tau$ and $\tau \in E_N \setminus E$, we replace v in $ele_2(v')$ by the children of v, where v' is the parent of v. In addition, we remove v from V_2, $lab_2(v)$ from lab_2, and $ele_2(v)$ from ele_2. Observe that, by the definition of D_N, no attributes are defined for elements of any type in $E_N \setminus E$. We repeat the process until there is no node labeled with an element type in $E_N \setminus E$. Now let $T_1 = (V_1, lab_1, ele_1, att, root)$, where V_1, lab_1, and ele_1 are V_2, lab_2, and ele_2 at the end of the process, respectively. Observe that att and $root$ remain unchanged. By the definition of T_1 it can be verified that $T_1 \models D$; in addition, for any regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(\beta.\tau)|$ in T_1 equals $|nodes(f(\beta).\tau)|$ in T_2, and $values(\beta.\tau.@l)$ in T_1 equals $values(f(\beta).\tau.@l)$ in T_2, because of (1) and (2) above and, among other things, the fact that none of the nodes removed is labeled with a type of E and the attribute function att is unchanged.

We now move to encoding of DTDs, more specifically, narrow one-attribute DTDs. Let $D = (E, \{@l\}, P, R, r)$ be a narrow one-attribute DTD and Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D. We encode D with a system Ψ^{Σ}_{D} of integer constraints such that there exists an XML tree conforming to D iff Ψ^{Σ}_{D} admits a nonnegative solution. The coding is developed w.r.t. Σ. More specifically, assume that $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ is an enumeration of all regular expressions and attributes that appear in Σ and Θ is the set of functions $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which are not identically 0. For every $\theta \in \Theta$, define a regular expression:

$$r_\theta = \left(\bigcap_{i\,:\,\theta(i)=1} \beta_i.\tau_i \right) \cap \left(\bigcap_{j\,:\,\theta(j)=0} \overline{\beta_j.\tau_j} \right), \tag{3.1}$$

where $\overline{\beta_j.\tau_j}$ is the complement $\beta_j.\tau_j$. We allow intersection and complement operators only in regular expressions r_θ. We note that, for every $i \in [1, k]$,[4]

$$\beta_i.\tau_i = \bigcup_{\theta\,:\,\theta(i)=1} r_\theta.$$

Then to capture the interaction between D and constraints of Σ, the system Ψ^{Σ}_{D} has a variable $|nodes(\beta_i.\tau_i)|$, for every $i \in [1, k]$, and $|nodes(r_\theta)|$, for every $\theta \in \Theta$. In other words, Ψ^{Σ}_{D} specifies the dependencies imposed by D on the number of elements reachable by following $\beta_i.\tau_i$ ($i \in [1, k]$) and r_θ ($\theta \in \Theta$).

To capture $\beta_i.\tau_i$ ($i \in [1, k]$) and r_θ ($\theta \in \Theta$) in Ψ^{Σ}_{D}, consider, for each regular expression $\beta_i.\tau_i$ ($i \in [1, k]$), a deterministic automaton that recognizes that expression. Let M be the deterministic automaton equivalent to the product of all of these automata. We refer to M as the DFA for Σ. Let s_M be the start state of M and δ be its transition function. Given an XML tree T conforming to D, for each node x in T we define $state(x)$ as s, if there is a simple path ρ over D such that $T \models \rho(root, x)$ and $s = \delta(s_M, \rho)$. The connection between M and T w.r.t. $\beta_i.\tau_i$ ($i \in [1, k]$) is described by the following lemma.

LEMMA 3.10. *Let D be a narrow one-attribute DTD, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, M the DFA for Σ, and $\beta_i.\tau_i$ a regular expression in Σ. Then for every XML tree T conforming to D and every τ_i-element x in T, $x \in nodes(\beta_i.\tau_i)$ in T iff $state(x)$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$.*

[4]Recall that the regular language defined by a regular expression β is denoted by β as well.

In other words, $nodes(\beta_i.\tau_i)$ in T consists of all τ_i-elements x such that $state(x)$ (which is a tuple of states of automata corresponding to regular expressions in Σ) contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. A similar idea was exploited in [1].

Proof. Since T is a tree, there exists a unique simple path ρ over D such that $T \models \rho.\tau_i(root,\ x)$. Thus $x \in nodes(\beta.\tau_i)$ in T iff $\rho.\tau_i \in \beta.\tau_i$. If $x \in nodes(\beta.\tau_i)$ in T, then $\rho.\tau_i \in \beta.\tau_i$, and, therefore, there must be a final state $f_{\beta.\tau_i}$ in the automaton for $\beta.\tau_i$ and a state s in M such that $s = \delta(s_M, \rho.\tau_i)$ and s contains $f_{\beta_i.\tau_i}$. Thus $state(x) = s$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. Conversely, if $state(x)$ contains a final state $f_{\beta_i.\tau_i}$ in the automaton for $\beta_i.\tau_i$, then $\rho.\tau_i \in \beta.\tau_i$ since $s = \delta(s_M, \rho.\tau_i)$. Therefore, $x \in nodes(\beta_i.\tau_i)$ in T.

We next define a system Ψ_D^Σ of integer constraints. The variables used in the constraints of Ψ_D^Σ are as follows. Let $\tau \in E$ be an element type and $s = \delta(s_M, \rho.\tau)$ for some simple path $\rho.\tau \in E^*$. For each such pair we create a distinct variable x_τ^s. Intuitively, in an XML tree T conforming to D, we use x_τ^s to keep track of the number of τ-elements with state s. Furthermore, define Y_τ^s as the set of pairs (τ', s') such that $\tau' \in E$, $s' = \delta(s_M, \rho.\tau')$ for some simple path $\rho.\tau' \in E^*$, τ is mentioned in $P(\tau')$, and $s = \delta(s', \tau)$. For each such pair (τ', s'), we create a variable $x_{\tau,\tau'}^{s,s'}$. Intuitively, in an XML tree T conforming to D, we use $x_{\tau,\tau'}^{s,s'}$ to keep track of the number of τ-elements with state s that are children of a node of type τ' with state s'. There are exponentially many variables (in the size of D and Σ) in total since M is a DFA. By using these, we define an integer constraint to specify $\tau \to P(\tau)$ at state s as follows. Let us use Ψ_τ^s to denote the set of integer constraints defined for τ at s.

- If $P(\tau) = \tau_1$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$, where $s_1 = \delta(s, \tau_1)$.
- If $P(\tau) = (\tau_1, \tau_2)$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ and $x_\tau^s = x_{\tau_2,\tau}^{s_2,s}$, where $s_i = \delta(s, \tau_i)$ for $i = 1, 2$. Referring to the XML tree T, these assure that each τ-element in T must have a τ_1-subelement and a τ_2-subelement.
- If $P(\tau) = (\tau_1|\tau_2)$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s} + x_{\tau_2,\tau}^{s_2,s}$, where $s_i = \delta(s, \tau_i)$ for $i = 1, 2$. This assures that each τ-element in T must have either a τ_1-subelement or a τ_2-subelement, and thus the sum of the number of these τ_1-subelements and the number of τ_2-subelements equals the number of τ-elements.
- If $P(\tau) = \tau_1^*$, then Ψ_τ^s includes $(x_{\tau_1,\tau}^{s_1,s} > 0) \to (x_\tau^s > 0)$, where $s_1 = \delta(s, \tau_1)$.

In addition, Ψ_τ^s includes $x_\tau^s = \sum_{(\tau',s') \in Y_\tau^s} x_{\tau,\tau'}^{s,s'}$.

Recall that $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ is an enumeration of all regular expressions and attributes that appear in Σ, that Θ is the set of functions $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which are not identically 0, and that, for each such function θ, r_θ is a regular expression defined as in (3.1). For each $i \in [1, k]$, we define $F_{\beta_i.\tau_i}$ as the set of states $s = (s_1, \ldots, s_k)$ of the DFA for Σ such that s_i is a final state of the DFA for $\beta_i.\tau_i$. Notice that by Lemma 3.10, for every XML tree T conforming to D and every node x of T, $x \in nodes(\beta_i.\tau_i)$ in T iff $state(x) \in F_{\beta_i.\tau_i}$. Furthermore, for each $\theta \in \Theta$, we define F_θ as the set of states $s = (s_1, \ldots, s_k)$ of the DFA for Σ such that for every $i \in [1, k]$, s_i is a final state of the DFA for $\beta_i.\tau_i$ iff $\theta(i) = 1$. Notice that by Lemma 3.10, for every XML tree T conforming to D and every node x of T, $x \in nodes(r_\theta)$ in T iff $state(x) \in F_\theta$. Finally, for each $r_\theta \neq \emptyset$, we have that, for every $i, j \in [1, k]$, if $\theta(i) = \theta(j) = 1$, then $\tau_i = \tau_j$. In this case, we define τ_θ as τ_i, for an arbitrary $i \in [1, k]$ such that $\theta(i) = 1$.

By our restriction on regular expressions regarding element type r, there is a unique variable x_r^s associated with r, where $s = \delta(s_M, r)$. We write x_r for x_r^s. Then

we define *the set of cardinality constraints determined by DTD D* w.r.t. a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, denoted by Ψ^{Σ}_{D}, as follows:

- For each $\tau \in E$ and each state s given above, Ψ^{Σ}_{D} contains all of the constraints in Ψ^{s}_{τ}.
- Ψ^{Σ}_{D} contains constraint $x_r = 1$; i.e., there is a unique root in each XML tree conforming to D.
- For every $i \in [1,k]$, Ψ^{Σ}_{D} contains constraint $|nodes(\beta_i.\tau_i)| = \sum_{s\,:\,s\in F_{\beta_i.\tau_i}} x^{s}_{\tau_i}$.
- For every $\theta \in \Theta$ such that $r_\theta \neq \emptyset$, Ψ^{Σ}_{D} contains constraint $|nodes(r_\theta)| = \sum_{s\,:\,s\in F_\theta} x^{s}_{\tau_\theta}$.
- For every $\theta \in \Theta$ such that $r_\theta = \emptyset$, Ψ^{Σ}_{D} contains constraint $|nodes(r_\theta)| = 0$.

Note that Ψ^{Σ}_{D} can be computed in EXPTIME in the size of D and Σ. We say that Ψ^{Σ}_{D} is *consistent* iff it has a nonnegative solution. We next show that Ψ^{Σ}_{D} indeed characterizes the narrow one-attribute DTD D. □

LEMMA 3.11. *Let D be a narrow one-attribute DTD, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and Ψ^{Σ}_{D} the set of cardinality constraints determined by D w.r.t. Σ. Then Ψ^{Σ}_{D} is consistent iff there is an XML tree T such that $T \models D$. In addition, for every $i \in [1,k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T equal the value of variables $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ given by the solution to Ψ^{Σ}_{D}.*

Proof. First, assume that there is an XML tree $T = (V, lab, ele, att, root)$ conforming to D. We define a nonnegative solution of Ψ^{Σ}_{D} as follows. For each variable $x^{s,s'}_{\tau,\tau'}$ in Ψ^{Σ}_{D}, let its value be the number of τ-elements x in T such that x is a child of a node y of type τ' with $state(x) = s$ and $state(y) = s'$. Furthermore, let x_r be 1, and, for every variable x^{s}_{τ} in Ψ^{Σ}_{D}, let x^{s}_{τ} be the sum of the variables $x^{s,s'}_{\tau,\tau'}$ where $(\tau', s') \in Y^{s}_{\tau}$. Finally, for every $i \in [1,k]$ and every $\theta \in \Theta$, let $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ be $\sum_{s\,:\,s\in F_{\beta_i.\tau_i}} x^{s}_{\tau_i}$ and $\sum_{s\,:\,s\in F_\theta} x^{s}_{\tau_\theta}$, respectively. This defines a nonnegative assignment since T is finite. It can be verified that the assignment is a solution of Ψ^{Σ}_{D}. Indeed, it satisfies the constraint $x_r = 1$ and constraints of the form $x^{s}_{\tau} = \sum_{(\tau',s')\in Y^{s}_{\tau}} x^{s,s'}_{\tau,\tau'}$, $|nodes(\beta_i.\tau_i)| = \sum_{s\,:\,s\in F_{\beta_i.\tau_i}} x^{s}_{\tau_i}$, and $|nodes(r_\theta)| = \sum_{s\,:\,s\in F_\theta} x^{s}_{\tau_\theta}$ by the definition of the assignment. Moreover, one can verify that it also satisfies the constraints of each Ψ^{s}_{τ}, by considering four different cases corresponding to the four different types of regular expressions in D. In particular, it satisfies constraints of the form $(x^{s_1,s}_{\tau_1,\tau} > 0) \rightarrow (x^{s}_{\tau} > 0)$ for each $\tau \rightarrow \tau_1^*$ in P, since if $x^{s_1,s}_{\tau_1,\tau} > 0$, then there exists a τ_1-node in T having as its parent a τ-node y with $state(y) = s$. Thus, $x^{s}_{\tau} > 0$ by the definition of the assignment. Therefore, Ψ^{Σ}_{D} is consistent. Moreover, by Lemma 3.10, for every $i \in [1,k]$ and $\theta \in \Theta$, the values of variables $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in the solution are indeed $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T.

Conversely, assume that Ψ^{Σ}_{D} admits a nonnegative solution. We show that there exists an XML tree $T = (V, lab, ele, att, root)$ such that $T \models D$. To do so, for each element type τ and state s for τ, we create x^{s}_{τ} many distinct τ-elements. Let $ext(\tau)$ denote the set of all τ-elements created above and

$$V = \bigcup_{\tau\in E} ext(\tau).$$

Then function lab is defined as $lab(v) = \tau$ if $v \in ext(\tau)$, and function att is defined as follows:

$$att(v, @l) = \begin{cases} \text{empty string} & \text{if } @l \in R(lab(v)), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is easy to verify that these functions are well defined. Let *root* be the node labeled r, which is unique since $x_r = 1$ is in Ψ_D^Σ. Finally, to define function *ele*, we do the following. For each $x_{\tau,\tau'}^{s,s'}$ in Ψ_D^Σ, we choose $x_{\tau,\tau'}^{s,s'}$ many distinct vertices labeled τ and mark them with $x_{\tau,\tau'}^{s,s'}$. Note that every τ-element in V can be marked once and only once. Starting at *root*, for each τ-element x marked with $x_{\tau,\tau'}^{s,s'}$ for some $(\tau', s') \in Y_\tau^s$, consider $P(\tau)$ and constraints of Ψ_D^Σ.[5] If $P(\tau)$ is $\tau_1 \in E$, then we choose a distinct τ_1-element y marked with $x_{\tau_1,\tau}^{s_1,s}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ is in Ψ_D^Σ. If $P(\tau) = (\tau_1, \tau_2)$, then we choose a τ_1-element y_1 marked with $x_{\tau_1,\tau}^{s_1,s}$ and a τ_2-element y_2 marked with $x_{\tau_2,\tau}^{s_2,s}$ and let $ele(x) = [y_1, y_2]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ and $x_\tau^s = x_{\tau_2,\tau}^{s_2,s}$ are in Ψ_D^Σ. If $P(\tau) = (\tau_1|\tau_2)$, then we choose an element y marked with either $x_{\tau_1,\tau}^{s_1,s}$ or $x_{\tau_2,\tau}^{s_2,s}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s} + x_{\tau_2,\tau}^{s_2,s}$ is in Ψ_D^Σ. If $P(\tau) = \tau_1^*$, then we choose a list $[y_1, \ldots, y_n]$ $(n \geq 0)$ of τ_1-elements marked with $x_{\tau_1,\tau}^{s_1,s}$ and let $ele(x) = [y_1, \ldots, y_n]$, where $(x_{\tau_1,\tau}^{s_1,s} > 0) \rightarrow (x_\tau^s > 0)$ is in Ψ_D^Σ. By the constraints in Ψ_D^Σ, each element of V can be chosen once and only once. One can verify that T defined in this way is indeed an XML tree and $T \models D$. Hence, there exists an XML tree conforming to D.

Finally, to see that, for every $i \in [1, k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T equal the values of variables $|nodes(\beta.\tau)|$ and $|nodes(r_\theta)|$ in the solution, respectively, it suffices to show, by Lemma 3.10, that for each node x in T, if x is marked with $x_{\tau,\tau'}^{s,s'}$ in the construction, then $state(x) = s$. Since T is a tree, there is a unique simple path $\rho \in E^*$ such that $T \models \rho(root, x)$. We show the claim by induction on the length $|\rho|$ of ρ. If $|\rho| = 1$, i.e., $\rho = r$, then x is the root and, obviously, $state(x) = \delta(s_M, r)$. Assume the claim for ρ, and we show that the claim holds for $\rho.\tau$. Let y be the τ'-element in T such that $T \models \rho(root, y)$ and y is the parent of x. Suppose that y is marked with $x_{\tau',\tau''}^{s',s''}$ in the construction. By the induction hypothesis, $state(y) = s'$. It is easy to see that $state(x) = \delta(s', \tau)$. By the definition of $\Psi_{\tau'}^{s'}$, we have that s is precisely the state $\delta(s', \tau)$. Thus $state(x) = s$. This proves the claim and thus the lemma. □

We now move to encoding $\mathcal{AC}_{K,FK}^{reg}$-constraints in terms of integer constraints. Let D be a DTD $(E, \{@l\}, P, R, r)$ and Σ a set of $\mathcal{AC}_{K,FK}^{reg}$-constraints over D. By Lemmas 3.8 and 3.9, we assume, without loss of generality, that D is a narrow one-attribute DTD. To encode Σ, let $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ be an enumeration of all regular expressions and attributes that appear in Σ, and, for every function $\theta : \{1, \ldots, k\} \rightarrow \{0, 1\}$ which is not identically 0, let regular expression r_θ be defined as in (3.1). Then for every nonempty $\Omega \subseteq \Theta$, we introduce a new variable z_Ω. In any XML tree conforming to D, the intended interpretation of z_Ω is the cardinality of

$$\left(\bigcap_{\theta\,:\,\theta \in \Omega} values(r_\theta.@l) \right) \setminus \left(\bigcup_{\theta\,:\,\theta \in \Theta \setminus \Omega} values(r_\theta.@l) \right). \tag{3.2}$$

Note that the number of new variables is double-exponential in the number of regular expression in Σ. By using these variables, we define the set of *the cardinality constraints determined by* Σ, denoted by C_Σ, which consists of the following:

[5] We assume that *root* is marked with x_r^s, where $s = \delta(s_M, r)$ and s_M is the initial state of the DFA for Σ.

$$\sum_{\Omega\,:\,\theta\in\Omega} z_\Omega = |values(r_\theta.@l)| \quad \text{for every } \theta\in\Theta,$$

$$\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset} z_\Omega = |values(\beta_i.\tau_i.@l)| \quad \text{for every } i\in[1,k],$$

$$|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)| \quad \text{for every } \beta_i.\tau_i.@l\to\beta_i.\tau_i \text{ in } \Sigma,$$

$$|values(\beta_j.\tau_j.@l)| = |nodes(\beta_j.\tau_j)| \quad \text{for every } \beta_i.\tau_i.@l\subseteq_{FK}\beta_j.\tau_j.@l \text{ in } \Sigma,$$

$$\sum_{\substack{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\\ \Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset}} z_\Omega = 0 \quad \text{for every } \beta_i.\tau_i.@l\subseteq_{FK}\beta_j.\tau_j.@l \text{ in } \Sigma,$$

$$|values(\beta_i.\tau_i.@l)| \le |nodes(\beta_i.\tau_i)| \quad \text{for every } i\in[1,k],$$

$$|values(r_\theta.@l)| \le |nodes(r_\theta)| \quad \text{for every } \theta\in\Theta.$$

Note that the size of C_Σ is double-exponential in the size of Σ.

We now combine the encodings for constraints and the DTDs and present a system $\Psi(D,\Sigma)$ of linear integer constraints for a DTD D and a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints. Assuming that D and Σ are as above, the set $\Psi(D,\Sigma)$, called the *set of cardinality constraints determined by D and Σ*, is defined to be:

$$\Psi_D^\Sigma \cup C_\Sigma \cup \{(|nodes(\beta_i.\tau_i)|>0)\to(|values(\beta_i.\tau_i.@l)|>0) \mid i\in[1,k]\} \cup \{(|nodes(r_\theta)|>0)\to(|values(r_\theta.@l)|>0)\mid\theta\in\Theta\},$$

where C_Σ is the set of cardinality constraints determined by Σ and Ψ_D^Σ is the set of cardinality constraints determined by D w.r.t. Σ. The system $\Psi(D,\Sigma)$ is said to be *consistent* iff it has a nonnegative solution that satisfies all of its constraints. Observe that $\Psi(D,\Sigma)$ can be partitioned into two sets: $\Psi(D,\Sigma)=\Psi^l(D,\Sigma)\cup\Psi^d(D,\Sigma)$, where $\Psi^l(D,\Sigma)$ consists of linear integer constraints and $\Psi^d(D,\Sigma)$ consists of constraints of the form $(x>0\to y>0)$. Also note that the size of $\Psi(D,\Sigma)$ is double-exponential in the size of D and Σ.

We next show that $\Psi(D,\Sigma)$ indeed characterizes the consistency of D and Σ.

LEMMA 3.12. *Let D be a narrow one-attribute DTD, Σ a finite set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and $\Psi(D,\Sigma)$ the set of cardinality constraints determined by D and Σ. Then $\Psi(D,\Sigma)$ is consistent iff there is an XML tree T such that $T\models(D,\Sigma)$.*

Proof. Suppose that there exists an XML tree T such that $T\models(D,\Sigma)$. Then by Lemma 3.11, there exists a nonnegative solution for Ψ_D^Σ such that, for every $i\in[1,k]$ and $\theta\in\Theta$, the values of variables $|nodes(r_\theta)|$ and $|nodes(\beta_i.\tau_i)|$ in this solution coincide with $|nodes(r_\theta)|$ and $|nodes(\beta_i.\tau_i)|$ in T, respectively. From this solution, it is easy to generate a solution to $\Psi(D,\Sigma)$ by assigning to variable $|values(r_\theta.@l)|$ the size of $values(r_\theta.@l)$ in T, for every $\theta\in\Theta$, assigning to variable $|values(\beta_i.\tau_i.@l)|$ the size of $values(\beta_i.\tau_i.@l)$ in T, for every $i\in[1,k]$, and then assigning to each variable z_Ω the cardinality of set (3.2) above. It is straightforward to verify that this assignment is a solution to $\Psi(D,\Sigma)$.

Conversely, suppose that $\Psi(D,\Sigma)$ has an integer solution. We show that there is an XML tree T such that $T\models(D,\Sigma)$. By Lemma 3.11, given an integer solution to $\Psi(D,\Sigma)$, we can construct an XML tree $T'=(V,lab,ele,att',root)$ such that

$T' \models D$. Moreover, for every $i \in [1,k]$, there are exactly $n_{\beta_i.\tau_i}$ elements in T' reachable by following $\beta_i.\tau_i$, where $n_{\beta_i.\tau_i}$ is the value of the variable $|nodes(\beta_i.\tau_i)|$ in $\Psi(D, \Sigma)$, and, for every $\theta \in \Theta$, there are exactly n_{r_θ} elements in T' reachable by following r_θ, where n_{r_θ} is the value of the variable $|nodes(r_\theta)|$ in $\Psi(D, \Sigma)$. We modify the definition of the function att', while leaving V, lab, ele, and $root$ unchanged, to generate a tree $T = (V, lab, ele, att, root)$ such that $T \models (D, \Sigma)$. More specifically, we modify $att'(v, @l)$ if v is in $nodes(\beta.\tau)$ for some regular expression $\beta.\tau$ mentioned in Σ and leave $att'(v, @l)$ unchanged otherwise. To do this, for each variable z_Ω we create a set s_Ω of distinct string values such that $|s_\Omega| = z_\Omega$ and $s_\Omega \cap s_{\Omega'} = \emptyset$ if $\Omega \neq \Omega'$. Then for every $\Omega \subseteq \Theta$, we let $values(r_\theta.@l)$ in T contain s_Ω iff $\theta \in \Omega$. This is possible because (1) $\sum_{\Omega\,:\,\theta\in\Omega} z_\Omega = |values(r_\theta.@l)|$ is in C_Σ for every $\theta \in \Theta$; (2) $\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset} z_\Omega = |values(\beta_i.\tau_i.@l)|$ is in C_Σ for every $i \in [1,k]$; (3) if $r_\theta = \emptyset$, then $|nodes(r_\theta)| = 0$ is in Ψ_D^Σ for every $\theta \in \Theta$; (4) $|values(\beta_i.\tau_i.@l)| \leq |nodes(\beta_i.\tau_i)|$ is in C_Σ for every $i \in [1,k]$; (5) $|values(r_\theta.@l)| \leq |nodes(r_\theta)|$ is in C_Σ for every $\theta \in \Theta$; and (6) $nodes(\beta)$ in T equals $nodes(\beta)$ in T' for every regular expression β over D.

We next show that T has the desired properties. It is easy to verify $T \models D$ given the construction of T from T' and the assumption $T' \models D$. By the definition of T, we have that, for every $i \in [1,k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$, $|values(\beta_i.\tau_i.@l)|$, $|nodes(r_\theta)|$, and $|values(r_\theta.@l)|$ in T equal the value of variables $|nodes(\beta_i.\tau_i)|$, $|values(\beta_i.\tau_i.@l)|$, $|nodes(r_\theta)|$, and $|values(r_\theta.@l)|$, respectively, given by the solution to $\Psi(D, \Sigma)$. We use this property to show that $T \models \Sigma$. Let φ be a constraint in Σ. (1) If φ is a key $\beta_i.\tau_i.@l \rightarrow \beta_i.@l$, it is immediate from the definition of T that $T \models \varphi$ since $|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)|$ is a constraint in C_Σ and, hence, $|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)|$ in T. That is, each $x \in nodes(\beta_i.\tau_i)$ in T has a distinct $@l$-attribute value, and thus the value of its $@l$-attribute uniquely identifies x among nodes in $nodes(\beta_i.\tau_i)$. (2) If φ is $\beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l$, it is easy to see that in T:

$$values(\beta_i.\tau_i.@l) \setminus values(\beta_j.\tau_j.@l) = \bigcup_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\,\Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset} s_\Omega.$$

Since $s_\Omega \cap s_{\Omega'} = \emptyset$ if $\Omega \neq \Omega'$,

$$|values(\beta_i.\tau_i.@l) \setminus values(\beta_j.\tau_j.@l)| = \sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\,\Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset} z_\Omega.$$

Thus, given that

$$\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\,\Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset} z_\Omega = 0$$

is in C_Σ (since $\beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l \in \Sigma$), we have $|values(\beta_i.\tau_i.@l) \setminus values(\beta_j.\tau_j.@l)| = 0$ in T, that is, $values(\beta_i.\tau_i.@l) \subseteq values(\beta_j.\tau_j.@l)$ in T. Furthermore, $T \models \beta_j.\tau_j.@l \rightarrow \beta_j.\tau_j$ since $|values(\beta_j.\tau_j.@l)| = |nodes(\beta_j.\tau_j)|$ is a constraint in C_Σ. Thus $T \models \varphi$. This concludes the proof of the lemma. □

We need another lemma for a mild generalization of linear integer constraints.

LEMMA 3.13. *Given a system $A\vec{x} \leq \vec{b}$ of linear integer constraints together with conditions of the form $(x_i > 0) \rightarrow (x_j > 0)$, where A is an $n \times m$ matrix on integers, $\vec{b}$ is an n-vector on integers, and $1 \leq i, j \leq m$, the problem of determining whether the system admits a nonnegative integer solution is in NP.*

Proof. Let $c_1, \ldots, c_p$ enumerate the conditions of the form $(x > 0) \rightarrow (y > 0)$, c_k being $(x_k^1 > 0) \rightarrow (x_k^2 > 0)$. Consider 2^p instances $\mathcal{I}_j$ of integer linear programming

obtained by adding, for each $k \leq p$, either $x_k^1 = 0$ or $x_k^2 > 0$ to $A\vec{x} \leq \vec{b}$. Clearly, the original system of constraints has a solution iff some $\mathcal{I}_j$ has a solution. By [27], $\mathcal{I}_j$ has a solution iff it has a solution whose size is polynomial in A, $\vec{b}$, and p. Hence, to check if the original system of constraints has a solution, it suffices to guess a system $\mathcal{I}_j$ and then guess a polynomial size solution for it; thus, the problem is in NP.

We now conclude the proof of the first part of the theorem. By Lemma 3.8, given an arbitrary DTD D and a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, it is possible to compute a one-attribute DTD D' and a set Σ' of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D' such that (D, Σ) is consistent iff (D', Σ') is consistent. By Lemma 3.9, one can compute a narrow one-attribute DTD D'_N and a set Σ'_N of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D'_N such that (D', Σ') is consistent iff (D'_N, Σ'_N) is consistent. By Lemma 3.12, (D'_N, Σ'_N) is consistent iff $\Psi(D'_N, \Sigma'_N)$ has a nonnegative integer solution. Thus, (D, Σ) is consistent iff $\Psi(D'_N, \Sigma'_N)$ has a nonnegative integer solution. Note that (D', Σ') can be computed in polynomial time on $|D| + |\Sigma|$, (D'_N, Σ'_N) can be computed in polynomial time on $|D'|+|\Sigma'|$, and $\Psi(D'_N, \Sigma'_N)$ can be computed in double-exponential time on $|D'_N|+|\Sigma'_N|$. Thus, by Lemma 3.13, one can check in 2-NEXPTIME whether there exists an XML tree T such that $T \models (D, \Sigma)$. □

Proof of (b). We establish the PSPACE-hardness by reduction from the QBF-CNF problem. An instance of QBF-CNF is a quantified boolean formula in prenex conjunctive normal form. The problem is to determine whether this formula is valid. QBF-CNF is known to be PSPACE-complete [20, 28].

Let θ be a formula of the form

$$Q_1 x_1 \dots Q_m x_m \psi, \tag{3.3}$$

where each $Q_i \in \{\forall, \exists\}$ $(1 \leq i \leq m)$ and ψ is a propositional formula in conjunctive normal form, say, $C_1 \wedge \dots \wedge C_n$, that mentions variables $x_1, \dots, x_m$. We construct a DTD D_θ and a set Σ_θ of $\mathcal{AC}^{reg}_{K,FK}$-constraint such that θ is valid iff there is an XML tree conforming to D_θ and satisfying Σ_θ.

We construct a DTD $D_\theta = (E, A, P, R, r)$ as follows. $E = \{r, C\} \cup \bigcup_{i=1}^{m} \{x_i, \bar{x}_i, N_{x_i}, P_{x_i}\}$, $A = \{@l\}$, and P is defined by considering the quantifiers of θ. We use Q_1 to define P on the root:

$$P(r) = \begin{cases} (N_{x_1}|P_{x_1}), C & Q_1 = \exists, \\ (N_{x_1}, P_{x_1}), C & Q_1 = \forall. \end{cases}$$

In general, for each $1 \leq i \leq m-1$, we consider quantifier Q_{i+1} to define $P(N_{x_i})$ and $P(P_{x_i})$:

$$P(N_{x_i}) = P(P_{x_i}) = \begin{cases} N_{x_{i+1}}|P_{x_{i+1}} & Q_{i+1} = \exists, \\ N_{x_{i+1}}, P_{x_{i+1}} & Q_{i+1} = \forall. \end{cases}$$

We represent formula ψ as a regular expression. Given a clause $C_j = \bigvee_{i=1}^{p} y_i \vee \bigvee_{i=1}^{q} \neg z_i$ $(j \in [1, n])$, $tr(C_j)$ is defined to be the regular expression $y_1| \dots |y_p|\bar{z}_1| \dots |\bar{z}_q$. We define P on element types N_{x_m} and P_{x_m} as $P(N_{x_m}) = P(P_{x_m}) = tr(C_1), \dots, tr(C_n)$. For the remaining elements of E, we define P as ϵ. We define function R as follows:

$$\begin{array}{lllllll} R(r) & = & R(P_{x_i}) & = & R(N_{x_i}) & = & \emptyset, \quad 1 \leq i \leq m, \\ R(C) & = & R(x_i) & = & R(\bar{x}_i) & = & \{@l\}, \quad 1 \leq i \leq m. \end{array}$$

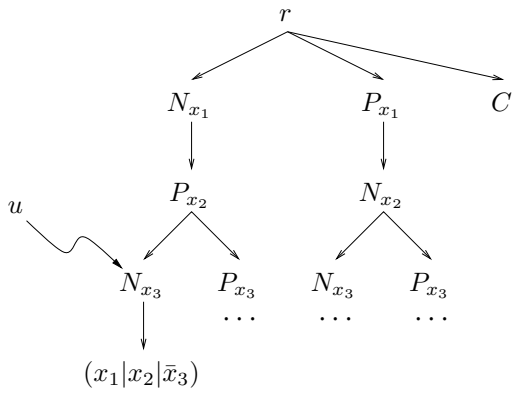


FIG. 3.2. *An XML tree conforming to the DTD constructed from* $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$.

Finally, Σ_θ contains the following foreign keys:

$$r._^*.N_{x_i}._^*.x_i.@l \subseteq_{FK} r.C.C.@l, \qquad r._^*.P_{x_i}._^*.\bar{x}_i.@l \subseteq_{FK} r.C.C.@l, \qquad i \in [1,m].$$

For instance, for the formula $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$, an XML tree conforming to D is shown in Figure 3.2. In this tree, a node of type N_{x_i} represents a negative value (0) for the variable x_i, and a node of type P_{x_i} represents a positive value (1) for this variable. Thus, given that the root has two children of types N_{x_1} and P_{x_1}, the values 0 and 1 are assigned to x_1 (representing the quantifier $\forall x_1$). Nodes of type N_{x_1} have one child of type either N_{x_2} or P_{x_2}, and, therefore, either 0 or 1 is assigned to x_2 (representing the quantifier $\exists x_2$). The same holds for nodes of type P_{x_2}. The fourth level of the tree represents the quantifier $\forall x_3$. Note that, in any XML tree T conforming to D, there is *no* node in T reachable by following the path $r.C.C$.

In Figure 3.2, every path from the root r to a node of type either N_{x_3} or P_{x_3} represents a truth assignment for the variables x_1, x_2, x_3. For example, the path from the root to the node u represents the truth assignment σ_u: $\sigma_u(x_1) = 0$, $\sigma_u(x_2) = 1$, and $\sigma_u(x_3) = 0$. To verify that all of these assignments satisfy the formula $x_1 \vee x_2 \vee \neg x_3$ we use the set of constraints Σ_θ.

Next we prove that θ, defined in (3.3), is valid iff there is an XML tree T conforming to D_θ and satisfying Σ_θ. We show only the "if" direction. The "only if" direction is similar.

Suppose that there is an XML tree T such that $T \models (D_\theta, \Sigma_\theta)$. To prove that θ is valid, it suffices to prove that each path from the root r to a node of type either N_{x_m} or P_{x_m} represents a truth assignment satisfying ψ. Let p be one of these paths, and let v be the node of type either N_{x_m} or P_{x_m} reachable from the root by following p. We define the truth assignment σ_p as follows:

$$\sigma_p(x_i) = \begin{cases} 1 & \text{if } p \text{ contains a node of type } P_{x_i}, \\ 0 & \text{otherwise.} \end{cases}$$

We have to prove that $\sigma_p(C_i) = 1$ for each $i \in [1,n]$. Given that $T \models D_\theta$, v has as a child a node v' whose type is in $tr(C_i)$. If the type of v' is x_j, then, given that $T \models r._^*.N_{x_j}._^*.x_j.@l \subseteq_{FK} r.C.C.@l$ and that there exists no node in T reachable by following the path $r.C.C$, p contains a node of type P_{x_j}, and, therefore, $\sigma_p(C_i) = 1$ since $\sigma_p(x_j) = 1$. If the type of v' is $\bar{x}_j$, then, given that $T \models r._^*.P_{x_j}._^*.\bar{x}_j.@l \subseteq_{FK} r.C.C.@l$, p contains a node of type N_{x_j} and it does not contain a node of type P_{x_j},

and, therefore, $\sigma_p(C_i) = 1$ since $\sigma_p(\neg x_j) = 1$. Thus, we conclude that θ is valid. This concludes the proof of part (b) of the theorem. □

4. Relative integrity constraints. Since XML documents are hierarchically structured, one may be interested in the entire document as well as in its subdocuments. The latter gives rise to *relative integrity constraints* [5] that hold only on certain subdocuments. Below we define relative keys and foreign keys. Recall that we use $\mathcal{RC}$ to denote various classes of such constraints. We use the notation $x \prec y$ when x and y are two nodes in an XML tree and y is a descendant of x.

We first define unary relative keys and foreign keys associated with element types. Let $D = (E, A, P, R, r)$ be a DTD. A *relative key* is an expression φ of the form $\tau(\tau_1.@l \rightarrow \tau_1)$, where $@l \in R(\tau_1)$. It says that, relative to each node x of element type τ, $@l$ is a key for all of the τ_1-nodes that are descendants of x. That is, if a tree T conforms to D, then $T \models \varphi$ if

$$\forall x \in ext(\tau)\ \forall y, z \in ext(\tau_1)\ \big((x \prec y) \wedge (x \prec z) \wedge (y.@l = z.@l) \rightarrow y = z\big).$$

A *relative foreign key* is an expression φ of the form $\tau(\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2)$, where $@l_i \in R(\tau_i), i = 1, 2$. It indicates that, for each x in $ext(\tau)$, $@l_1$ is a foreign key of descendants of x of type τ_1 that references a key $@l_2$ of τ_2-descendants of x. That is, $T \models \varphi$ if $T \models \tau(\tau_2.@l_2 \rightarrow \tau_2)$ and T satisfies

$$\forall x \in ext(\tau)\ \forall y_1 \in ext(\tau_1)\ \big((x \prec y_1) \rightarrow \exists y_2 \in ext(\tau_2)\ \ ((x \prec y_2) \wedge y_1.@l_1 = y_2.@l_2)\big).$$

Here τ is called the *context type* of φ. Note that absolute constraints are a special case of relative constraints when $\tau = r$: i.e., $r(\tau.@l \rightarrow \tau)$ is the usual absolute key. Thus, the consistency problem for multiattribute relative constraints is undecidable [16], and hence we consider only unary relative constraints here.

Following the notations for $\mathcal{AC}$, we use $\mathcal{RC}_{K,FK}$ to denote the class of all unary relative keys and foreign keys defined for element types; $\mathcal{RC}_{PK,FK}$ means the primary key restriction. For example, the constraints given in section 1 over the country/province/capital DTD can be expressed in $\mathcal{RC}_{K,FK}$ as follows:

$$\begin{aligned} country.@name &\rightarrow country, \\ country(province.@name &\rightarrow province), \\ country(capital.@inProvince &\rightarrow capital), \\ country(capital.@inProvince &\subseteq_{FK} province.@name). \end{aligned}$$

A more general form of unary relative constraints is defined in terms of regular path expressions, along the same lines as $\mathcal{AC}^{reg}_{K,FK}$. For example, the constraints given in section 1 over the country/province/capital DTD are instances of this general form of relative constraints. Since $\mathcal{RC}_{K,FK}$ constraints are a special case of the general regular-expression relative constraints (by substituting $\_^*.\tau$ for τ), the lower bound for $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ carries over to the consistency problem for unary relative constraints defined in terms of regular path expressions.

Recall that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problem for absolute unary constraints, is NP-complete. One would be tempted to think that $\mathsf{SAT}(\mathcal{RC}_{K,FK})$, the consistency problems for relative unary constraints, is decidable as well. We next show, however, that there is an enormous difference between unary absolute constraints and unary relative constraints: while clearly $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ is recursively enumerable, it turns out that one cannot lower this bound.

THEOREM 4.1. $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ *is undecidable.*

Proof. We establish the undecidability of the consistency problem for unary relative keys and foreign keys by reduction from the Hilbert's 10th problem [24]. To do this, we consider a variation of the Diophantine problem, referred to as the *positive Diophantine quadratic system problem*. An instance of the problem is

$$\begin{aligned} P_1(x_1,\ldots,x_k) &= Q_1(x_1,\ldots,x_k) + c_1,\\ P_2(x_1,\ldots,x_k) &= Q_2(x_1,\ldots,x_k) + c_2,\\ &\cdots\\ P_n(x_1,\ldots,x_k) &= Q_n(x_1,\ldots,x_k) + c_n, \end{aligned}$$

where, for $1 \le i \le n$, P_i and Q_i are polynomials in which all coefficients are positive integers; the degree of P_i is at most 2, and the degree of each of its monomials is at least 1; each polynomial Q_i satisfies the same condition, and each c_i is a nonnegative integer constant. The problem is to determine, given any positive Diophantine quadratic system, whether it has a nonnegative integer solution.

The positive Diophantine quadratic system problem is undecidable. To prove this, it is straightforward to reduce to it another variation of the Diophantine problem, the *positive Diophantine equation problem*, which is known to be undecidable. An instance of this problem is $R(\bar{y}) = S(\bar{y})$, where R and S are polynomials in which all coefficients are positive integers, and the problem is to determine whether it has a nonnegative integer solution.

In what follows, we show a reduction from the positive Diophantine quadratic system problem to $\mathsf{SAT}(\mathcal{RC}_{K,FK})$. More precisely, given a quadratic equation we show how to represent it by using a DTD and a set of constraints. It is straightforward to extend this representation to consider an arbitrary number of quadratic equations.

Consider the following equation:

$$\sum_{i=1}^{m} a_i x_{\alpha_i} + \sum_{i=m+1}^{n} a_i x_{\alpha_i} x_{\beta_i} = \sum_{i=1}^{p} b_i x_{\gamma_i} + \sum_{i=p+1}^{q} b_i x_{\gamma_i} x_{\delta_i} + o. \tag{4.1}$$

In this equation, for every $i \in [1,n]$ and $j \in [m+1,n]$, a_i is a positive integer and x_{α_i} and x_{β_j} represent variables, where $\alpha_i, \beta_j \in [1,k]$. Furthermore, for every $i \in [1,q]$ and $j \in [p+1,q]$, b_i is a positive integer and x_{γ_i} and x_{δ_j} are variables, where $\gamma_i, \delta_j \in [1,k]$. Finally, o is a nonnegative integer.

To code the previous equation, we need to define a DTD $D = (E, A, P, R, r)$ and a set of $\mathcal{RC}_{K,FK}$-constraints Σ. Here D includes the following elements and attributes:

$$\begin{aligned} E = {} & \{r, X, Y\} \cup \bigcup_{i=1}^{k} \{n_i\} \cup \bigcup_{i=1}^{n} \{\alpha_i\} \cup \bigcup_{i=m+1}^{n} \{\alpha_i', \beta_i, c_i, d_i, e_i\}\\ & \cup \bigcup_{i=1}^{q} \{\gamma_i\} \cup \bigcup_{i=p+1}^{q} \{\gamma_i', \delta_i, f_i, g_i, h_i\},\\ A = {} & \{@v\}. \end{aligned}$$

In this DTD, r is the root. Intuitively, by referring to an XML tree conforming to D, we use $|ext(n_i)|$ to code the value of the variable x_i, and we use $|ext(X)|$ and $|ext(Y)|$ to code the values of the left- and the right-hand sides of (4.1), respectively.

We define $P(r)$ as follows:

$$P(r) = n_1^*, \ldots, n_k^*, \alpha_1^*, \ldots, \alpha_m^*, (\epsilon|\alpha_{m+1}), \ldots, (\epsilon|\alpha_n), \\ \gamma_1^*, \ldots, \gamma_p^*, (\epsilon|\gamma_{p+1}), \ldots, (\epsilon|\gamma_q), \underbrace{Y, \ldots, Y}_{o \text{ times}}.$$

We define the function P on α_i and β_i as follows:

$$\begin{aligned}
P(\alpha_i) &= \underbrace{X, \ldots, X}_{a_i \text{ times}}, && 1 \le i \le m,\\
P(\alpha_i) &= (\beta_i, c_i, c_i, \underbrace{X, \ldots, X}_{a_i \text{ times}})^*, \alpha_i', && m+1 \le i \le n,\\
P(\gamma_i) &= \underbrace{Y, \ldots, Y}_{b_i \text{ times}}, && 1 \le i \le p,\\
P(\gamma_i) &= (\delta_i, f_i, f_i, \underbrace{Y, \ldots, Y}_{b_i \text{ times}})^*, \gamma_i', && p+1 \le i \le q.
\end{aligned}$$

To code (4.1) we need to capture the multiplication operator. To do this, we use α_i' and γ_i':

$$\begin{aligned}
P(\alpha_i') &= (\beta_i, d_i, d_i)^*, (\alpha_i|(c_i, e_i)^*), && m+1 \le i \le n,\\
P(\gamma_i') &= (\delta_i, g_i, g_i)^*, (\gamma_i|(f_i, h_i)^*), && p+1 \le i \le q.
\end{aligned}$$

For all of the other element types τ in D, $P(\tau)$ is defined as ϵ:

$$\begin{array}{llllll}
P(\beta_i) = \epsilon, & m+1 \le i \le n, & P(\delta_i) = \epsilon, & p+1 \le i \le q, & P(X) = \epsilon, & \\
P(c_i) = \epsilon, & m+1 \le i \le n, & P(f_i) = \epsilon, & p+1 \le i \le q, & P(Y) = \epsilon, & \\
P(d_i) = \epsilon, & m+1 \le i \le n, & P(g_i) = \epsilon, & p+1 \le i \le q, & P(n_i) = \epsilon, & 1 \le i \le k,\\
P(e_i) = \epsilon, & m+1 \le i \le n, & P(h_i) = \epsilon, & p+1 \le i \le q. &&
\end{array}$$

Finally, we include the following attributes:

$$\begin{aligned}
&R(r) = \emptyset,\\
&R(n_i) = \{@v\}, && 1 \le i \le k,\\
&R(X) = R(Y) = \{@v\},\\
&R(\alpha_i) = \{@v\}, && 1 \le i \le n,\\
&R(\gamma_i) = \{@v\}, && 1 \le i \le q,\\
&R(\beta_i) = R(c_i) = R(d_i) = R(e_i) = \{@v\}, && m+1 \le i \le n,\\
&R(\delta_i) = R(f_i) = R(g_i) = R(h_i) = \{@v\}, && p+1 \le i \le q,\\
&R(\alpha_i') = \emptyset, && m+1 \le i \le n,\\
&R(\gamma_i') = \emptyset, && p+1 \le i \le q.
\end{aligned}$$

To ensure that XML documents that conform to D indeed code (4.1), we need to define a set of $\mathcal{RC}_{K,FK}$-constraints Σ. This set contains the following absolute keys:

$$\begin{aligned}
&r(X.@v \to X),\\
&r(\alpha_i.@v \to \alpha_i) && \text{for every } 1 \le i \le n,\\
&r(\beta_i.@v \to \beta_i) && \text{for every } m+1 \le i \le n,\\
&r(c_i.@v \to c_i) && \text{for every } m+1 \le i \le n,\\
&r(d_i.@v \to d_i) && \text{for every } m+1 \le i \le n,\\
&r(e_i.@v \to e_i) && \text{for every } m+1 \le i \le n,\\
&r(n_i.@v \to n_i) && \text{for every } 1 \le i \le k,\\
&r(Y.@v \to Y),\\
&r(\gamma_i.@v \to \gamma_i) && \text{for every } 1 \le i \le q,
\end{aligned}$$

$$\begin{array}{ll} r(\delta_i.@v \to \delta_i) & \text{for every } p+1 \le i \le q, \\ r(f_i.@v \to f_i) & \text{for every } p+1 \le i \le q, \\ r(g_i.@v \to g_i) & \text{for every } p+1 \le i \le q, \\ r(h_i.@v \to h_i) & \text{for every } p+1 \le i \le q. \end{array}$$

Σ contains the following absolute foreign keys:

$$\begin{array}{lll} r(X.@v \subseteq_{FK} Y.@v), & r(Y.@v \subseteq_{FK} X.@v), & \\ r(n_s.@v \subseteq_{FK} \alpha_i.@v), & r(\alpha_i.@v \subseteq_{FK} n_s.@v), & 1 \le i \le n \text{ and the value of } \alpha_i \text{ in (4.1) is equal to } s, \\ r(n_s.@v \subseteq_{FK} e_i.@v), & r(e_i.@v \subseteq_{FK} n_s.@v), & m+1 \le i \le n \text{ and the value of } \beta_i \text{ in (4.1) is equal to } s, \\ r(n_s.@v \subseteq_{FK} \gamma_i.@v), & r(\gamma_i.@v \subseteq_{FK} n_s.@v), & 1 \le i \le q \text{ and the value of } \gamma_i \text{ in (4.1) is equal to } s, \\ r(n_s.@v \subseteq_{FK} h_i.@v), & r(h_i.@v \subseteq_{FK} n_s.@v), & p+1 \le i \le q \text{ and the value of } \delta_i \text{ in (4.1) is equal to } s. \end{array}$$

Finally, Σ contains the following relative foreign keys:

$$\begin{array}{lll} \alpha_i(\beta_i.@v \subseteq_{FK} d_i.@v), & \alpha_i(d_i.@v \subseteq_{FK} \beta_i.@v), & m+1 \le i \le n, \\ \alpha'_i(\beta_i.@v \subseteq_{FK} c_i.@v), & \alpha'_i(c_i.@v \subseteq_{FK} \beta_i.@v), & m+1 \le i \le n, \\ \gamma_i(\delta_i.@v \subseteq_{FK} g_i.@v), & \gamma_i(g_i.@v \subseteq_{FK} \delta_i.@v), & p+1 \le i \le q, \\ \gamma'_i(\delta_i.@v \subseteq_{FK} f_i.@v), & \gamma'_i(f_i.@v \subseteq_{FK} \delta_i.@v), & p+1 \le i \le q. \end{array}$$

We show next that there is an XML tree T such that $T \models (D, \Sigma)$ iff there exists a nonnegative integer solution for (4.1). To do this, we prove that every XML tree T satisfying D and Σ codifies (4.1). More precisely, if the value of every variable x_i is v_i and $|ext(n_i)| = v_i$, for $i \in [1, k]$, then

$$|ext(X)| = \sum_{i=1}^{m} a_i v_{\alpha_i} + \sum_{i=m+1}^{n} a_i v_{\alpha_i} v_{\beta_i}, \tag{4.2}$$

$$|ext(Y)| = \sum_{i=1}^{p} b_i v_{\gamma_i} + \sum_{i=p+1}^{q} b_i v_{\gamma_i} v_{\delta_i} + o. \tag{4.3}$$

Let T be an XML tree conforming to D. Then every node of type X in T appears as a child of some node of type α_i ($i \in [1, n]$). Thus, to prove (4.2) it suffices to show that the number of X-nodes that are children of some node of type α_i ($i \in [1, n]$) is equal to the ith term of (4.2), that is,

$$\begin{array}{l} |\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| \\ \quad = a_i v_{\alpha_i}, \quad 1 \le i \le m, \\ |\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| \\ \quad = a_i v_{\alpha_i} v_{\beta_i}, \quad m+1 \le i \le n. \end{array}$$

Analogously, to show that (4.3) holds, we have to prove that the number of Y-nodes that are children of some node of type γ_i ($i \in [1, q]$) is equal to the ith term of (4.3). We will consider here only the case of X-nodes, the other case being similar.

First, let $i \in [1, m]$ and s be the value of α_i in (4.2). Given that $r(n_s.@v \subseteq_{FK} \alpha_i.@v)$ and $r(\alpha_i.@v \subseteq_{FK} n_s.@v)$ are in Σ, by the definition of $P(\alpha_i)$ the total number of X-nodes that are children of a node of type α_i is equal to $a_i v_{\alpha_i}$. Second, let $i \in [m+1, n]$ and s and t be the values of α_i and β_i in (4.1), respectively. Next we prove that $|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| = a_i v_s v_t$.

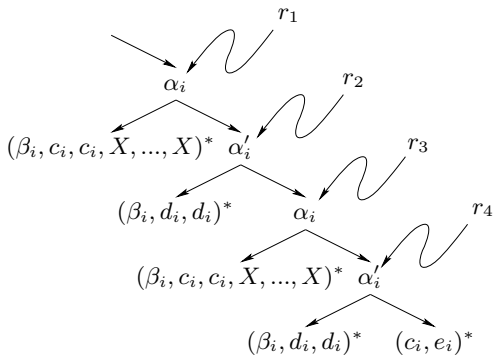


FIG. 4.1. *Part of the XML tree used in the proof of Theorem* 4.1.

Given that $r(n_s.@v \subseteq_{FK} \alpha_i.@v)$ and $r(\alpha_i.@v \subseteq_{FK} n_s.@v)$ are in Σ, $|ext(\alpha_i)|$ in T is equal to $|ext(n_s)| = v_s$. Thus, in T there are exactly v_s nodes of type α_i, each of them having exactly one child of type α_i'. Hence, there are exactly v_s nodes of type α_i', the last one being of the form shown in Figure 4.1 (see node r_4). By the definition of $P(\alpha_i')$, $|\{x \mid x \text{ is a child of } r_4 \text{ of type } c_i\}| = |\{x \mid x \text{ is a child of } r_4 \text{ of type } e_i\}|$. Given that $r(n_t.@v \subseteq_{FK} e_i.@v)$ and $r(e_i.@v \subseteq_{FK} n_t.@v)$ are in Σ and that every node of type e_i in T is a child of r_4, $|\{x \mid x \text{ is a child of } r_4 \text{ of type } c_i\}| = |ext(n_t)|$. Thus, since r_4 is a node of type α_i' and $\alpha_i'(\beta_i.@v \subseteq_{FK} c_i.@v)$ and $\alpha_i'(c_i.@v \subseteq_{FK} \beta_i.@v)$ are in Σ, $|\{x \mid x \text{ is a child of } r_4 \text{ of type } \beta_i\}| = |ext(n_t)| = v_t$. In addition, by the definition of $P(\alpha_i')$, the number of children of r_4 of type d_i is $2v_t$.

Given that r_3 is a node of type α_i and $\alpha_i(\beta_i.@v \subseteq_{FK} d_i.@v)$ and $\alpha_i(d_i.@v \subseteq_{FK} \beta_i.@v)$ are in Σ, $|\{x \mid x \text{ is a child of } r_3 \text{ of type } \beta_i\}| = v_t$, since there are $2v_t$ descendants of r_3 of type d_i and v_t children of r_4 of type β_i. Furthermore, by the definition of $P(\alpha_i)$, the number of children of r_3 of type X is $a_i v_t$, and the number of children of r_3 of type c_i is $2v_t$. We can use the same argument to prove that the number of children of r_2 of types β_i and d_i are v_t and $2v_t$, respectively. Thus, the number of children of r_1 of type X is $a_i v_t$, and the number of descendants of r_1 of type X is $2a_i v_t$. If we continue with this process, we can prove, by induction, that the number of X-nodes in T that are children of some node of type α_i is $v_s a_i v_t$, since there are v_s nodes of type α_i in T. This concludes the proof, since $|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| = a_i v_s v_t$. □

In the proof of Theorem 4.1, all relative keys are primary. Thus, we obtain the following.

COROLLARY 4.2. $\mathsf{SAT}(\mathcal{RC}_{PK,FK})$, *the restriction of* $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ *to primary keys, is undecidable.*

5. Extended DTDs. In this section, we consider a slight extension of DTDs which captures unranked tree automata. An *extended DTD* [32, 29] *ED* is a tuple (D', f, E), where $D' = (E', A', P', R', r')$ is a DTD, E is a finite set of element types such that $E \cap E' = \emptyset$, and f is a surjective mapping $f : E' \to E$ such that, for every $\tau_1, \tau_2 \in E'$, we have $R'(\tau_1) = R'(\tau_2)$ if $f(\tau_1) = f(\tau_2)$. We say that a tree T conforms to *ED* if there exists a tree T' that conforms to D' such that $T = f(T')$; that is, T can be obtained by replacing each label τ in T' by $f(\tau)$. Extended DTDs support a subtyping mechanism (specialization) and have proven useful in data migration and integration, among other things (see, e.g., [29] for examples of extended DTDs and their applications in data integration). It is also known that extended DTDs

capture precisely MSO (monadic second-order logic) definable trees and the regular tree languages of finite unranked trees [26, 29].

A constraint φ is said to be defined over extended DTD ED if every element type τ mentioned in φ is in E.

The consistency problem for extended DTDs is defined exactly as in the case of DTDs: given a specification (ED, Σ), the problem is to verify whether there exists a tree T conforming to ED and satisfying Σ. Next we show that the consistency problem for extended DTDs can be efficiently reduced to the consistency problem for DTDs.

Let $ED = (D', f, E)$ be an extended DTD, where $D' = (E', A', P', R', r')$, and Σ be either a set of $\mathcal{AC}^{*,*}_{K,FK}$-constraints over ED or a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over ED. We consider here only absolute constraints since, by Theorem 4.1, the consistency problem for extended DTDs and relative constraints is already undecidable for unary keys and foreign keys without regular expressions. We define DTD $D(ED) = (E_{ED}, A', P_{ED}, R_{ED}, r')$ as follows. Let $E_{ED} = E' \cup E$ and

$$P_{ED}(\tau) = \begin{cases} f(\tau), P'(\tau), & \tau \in E', \\ \epsilon, & \tau \in E, \end{cases} \qquad R_{ED}(\tau) = \begin{cases} \emptyset, & \tau \in E', \\ R'(\tau'), & \tau \in E \text{ and } f(\tau') = \tau. \end{cases}$$

Notice that R_{ED} is well defined since f is a surjective mapping such that $R'(\tau_1) = R'(\tau_2)$ whenever $f(\tau_1) = f(\tau_2)$. Moreover, we define a set of keys and foreign keys $\Gamma(ED, \Sigma)$ over DTD $D(ED)$ as follows. If Σ is a set of $\mathcal{AC}^{*,*}_{K,FK}$-constraints, then $\Gamma(ED, \Sigma) = \Sigma$. Otherwise, Σ is a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints, and for every $\varphi \in \Sigma$ we have the following constraint ψ in $\Gamma(ED, \Sigma)$. For every element type $\tau \in E$, define the image of substitution h on τ as $h(\tau) = (\tau_1 \cup \cdots \cup \tau_n)$, where $\{\tau_1, \ldots, \tau_n\}$ is the set of element types $\tau' \in E'$ such that $f(\tau') = \tau$. If φ is a key $\beta.\tau.@l \to \beta.\tau$, then $\psi = h(\beta.\tau).\tau.@l \to h(\beta.\tau).\tau$. If φ is a foreign key $\beta_1.\tau_1.@l_1 \subseteq_{FK} \beta_2.\tau_2.@l_2$, then $\psi = h(\beta_1.\tau_1).\tau_1.@l_1 \subseteq_{FK} h(\beta_2.\tau_2).\tau_2.@l_2$. The following simple lemma shows that the consistency problems for (ED, Σ) and $(D(ED), \Gamma(ED, \Sigma))$ are equivalent.

LEMMA 5.1. *Let $ED = (D', f, E)$ be an extended DTD and Σ be either a set of $\mathcal{AC}^{*,*}_{K,FK}$-constraints over ED or a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over ED. Then (ED, Σ) is consistent iff $(D(ED), \Gamma(ED, \Sigma))$ is consistent.*

We note that if $\mathcal{C}$ is one of $\mathcal{AC}_{PK,FK}$, $\mathcal{AC}_{K,FK}$, $\mathcal{AC}^{*,1}_{PK,FK}$, and $\mathcal{AC}^{reg}_{K,FK}$ and Σ is a set of $\mathcal{C}$-constraints over an extended DTD ED, then $\Gamma(ED, \Sigma)$ is a set of $\mathcal{C}$-constraints over $D(ED)$. Thus, given that $D(ED)$ and $\Gamma(ED, \Sigma)$ can be constructed in polynomial time from (ED, Σ), we obtain the following corollary from the previous lemma, Theorem 4.7 and Corollary 4.8 in [16], Corollary 3.5, and Theorem 3.7.

COROLLARY 5.2.

(a) *The consistency problem for extended DTDs and $\mathcal{AC}_{K,FK}$-constraints ($\mathcal{AC}_{PK,FK}$-constraints) is NP-complete.*
(b) *The consistency problem for extended DTDs and $\mathcal{AC}^{*,1}_{PK,FK}$-constraints is NP-hard and can be solved in NEXPTIME.*
(c) *The consistency problem for extended DTDs and $\mathcal{AC}^{reg}_{K,FK}$-constraints is PSPACE-hard and can be solved in 2-NEXPTIME.*

6. Conclusion. We have studied the problem of statically checking XML specifications, which may include various schema definitions as well as integrity constraints. As observed earlier, such static validation is quite desirable as an alternative to dynamic checking, which would attempt to validate each document; indeed, in the case of repeated failures, one does not know whether the problems lies in the documents or in the specification. Our main conclusion is that, however desirable, the static

TABLE 6.1
Complexity of the consistency problem for absolute constraints.

Class	$\mathcal{AC}^{*,*}_{K,FK}$ [16]	$\mathcal{AC}^{*,1}_{PK,FK}$	$\mathcal{AC}^{reg}_{K,FK}$	$\mathcal{AC}_{K,FK}$ [16]
Description	Multiattribute keys and foreign keys	Primary multiattribute keys, unary foreign keys	Unary regular path constraints (keys, foreign keys)	Unary keys and foreign keys
Upper bound	Undecidable	NEXPTIME	2-NEXPTIME	NP
Lower bound	Undecidable	NP	PSPACE	NP

TABLE 6.2
Complexity of the consistency problem for relative constraints.

Class	$\mathcal{RC}^{*,*}_{K,FK}$ [16]	$\mathcal{RC}_{K,FK}$	$\mathcal{RC}_{PK,FK}$
Description	Multiattribute keys and foreign keys	Unary keys and foreign keys	Primary unary keys and foreign keys
Lower bound	Undecidable	Undecidable	Undecidable

checking is hard: even with very simple document definitions given by DTDs, and (foreign) keys as constraints, the complexity ranges from NP-hard to undecidable.

The main results are summarized in Tables 6.1 and 6.2 (we also included the main results from [16] in those tables for completeness). When one deals with absolute constraints, which hold in an entire document, the general consistency problem is undecidable. It is solvable in NEXPTIME if foreign keys are single-attribute and is NP-complete if so are all of the keys as well. However, if regular expressions are allowed in single-attribute constraints, the lower bounds become at least PSPACE. For relative constraints, which are required to hold only in a part of a document, the situation is quite bleak, as even the very simple case of single-attribute constraints is undecidable.

Although most of the results of the paper are negative, the techniques developed in the paper help study consistency of individual XML specification with type and constraints. These techniques include, e.g., the connection between regular expression constraints and integer linear programming and automata.

One open problem is to close the complexity gaps. However, these are by no means trivial: for example, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ was proved to be equivalent to a problem related to Diophantine equations whose exact complexity remains unknown. In the case of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we think that it is more likely that our lower bounds correspond to the exact complexity of those problems. However, the algorithms are quite involved, and we do not yet see a way to simplify them to prove the matching upper bounds.

Another topic for future work is to study the interaction between more complex XML constraints, e.g., those defined in terms of XPath [37], and more complex schema specifications such as XML Schema [38] and the type system of XQuery [39]. Our lower bounds apply to those settings, but it is open whether upper bounds remain intact.

Acknowledgments. We are grateful to anonymous referees for their comments. Most of this work was done while Arenas and Libkin were at the University of Toronto.

REFERENCES

[1] S. ABITEBOUL AND V. VIANU, *Regular path queries with constraints*, J. Comput. System Sci., 58 (1999), pp. 428–452.

[2] C. K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, *XML-based information mediation with MIX*, in Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, Philadelphia, PA, 1999, pp. 597–599.

[3] C. Beeri and T. Milo, *Schemas for integration and translation of structured and semi-structured data*, in Proceedings of the 7th International Conference on Database Theory, Lecture Notes in Comput. Sci. 1540, Springer, New York, 1999, pp. 296–313.

[4] M. Benedikt, C. Y. Chan, W. Fan, J. Freire, and R. Rastogi, *Capturing both types and constraints in data integration*, in Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA, 2003, pp. 277–288.

[5] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan, *Keys for XML*, Comput. Networks, 39 (2002), pp. 473–487.

[6] P. Buneman, W. Fan, and S. Weinstein, *Path constraints in semistructured databases*, J. Comput. System Sci., 61 (2000), pp. 146–193.

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini, *Representing and reasoning on XML documents: A description logic approach*, J. Logic Comput., 9 (1999), pp. 295–318.

[8] D. Calvanese and M. Lenzerini, *Making object-oriented schemas more expressive*, in Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Minneapolis, MN, 1994, pp. 243–254.

[9] D. Calvanese and M. Lenzerini, *On the interaction between ISA and cardinality constraints*, in Proceedings of the 10th International Conference on Data Engineering, IEEE, Piscataway, NJ, 1994, pp. 204–213.

[10] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian, *XPERANTO: Publishing object-relational data as XML*, in International Workshop on the Web and Databases (Informal Proceedings), Lecture Notes in Comput. Sci. 1997, Springer, New York, 2000, pp. 105–110.

[11] S. S. Cosmadakis, P. C. Kanellakis, and M. Y. Vardi, *Polynomial-time implication problems for unary inclusion dependencies*, J. ACM, 37 (1990), pp. 15–46.

[12] S. B. Davidson, W. Fan, and C. S. Hara, *Propagating XML constraints to relations*, J. Comput. System Sci., 73 (2007), pp. 316–361.

[13] A. Deutsch, L. Popa, and V. Tannen, *Physical data independence, constraints, and optimization with universal plans*, in Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1999, pp. 459–470.

[14] A. Deutsch and V. Tannen, *Reformulation of XML queries and constraints*, in Proceedings of the 9th International Conference on Database Theory, Lecture Notes in Comput. Sci. 2572, Springer, New York, 2003, pp. 225–241.

[15] A. Eyal and T. Milo, *Integrating and customizing heterogeneous e-commerce applications*, The VLDB Journal, 10 (2001), pp. 16–38.

[16] W. Fan and L. Libkin, *On XML integrity constraints in the presence of DTDs*, J. ACM, 49 (2002), pp. 368–406.

[17] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu, *Verifying integrity constraints on web sites*, in Proceedings of the 16th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Francisco, 1999, pp. 614–619.

[18] M. F. Fernandez, A. Morishima, D. Suciu, and W. C. Tan, *Publishing relational data in XML: The SilkRoute approach*, IEEE Data Engineering Bulletin, 24 (2001), pp. 12–19.

[19] D. Florescu and D. Kossmann, *Storing and querying XML data using an RDMBS*, IEEE Data Engineering Bulletin, 22 (1999), pp. 27–34.

[20] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[21] R. Givan, D. A. McAllester, C. Witty, and D. Kozen, *Tarskian set constraints*, Inform. and Comput., 174 (2002), pp. 105–131.

[22] P. C. Kanellakis, *On the computational complexity of cardinality constraints in relational databases*, Inform. Process. Lett., 11 (1980), pp. 98–101.

[23] D. Lee and W. W. Chu, *Constraints-preserving transformation from XML document type definition to relational schema*, in Proceedings of the 19th International Conference on Conceptual Modeling, Lecture Notes in Comput. Sci. 1920, Springer, New York, 2000, pp. 323–338.

[24] Y. Matiyasevich, *Hilbert's 10th Problem*, MIT Press, Cambridge, MA, 1993.

[25] F. Neven, *Extensions of attribute grammars for structured document queries*, in Proceedings of the 7th International Workshop on Database Programming Languages, Lecture Notes in Comput. Sci. 1949, Springer, New York, 1999, pp. 99–116.

[26] F. Neven and T. Schwentick, *Query automata over finite trees*, Theoret. Comput. Sci., 275 (2002), pp. 633–674.

[27] C. Papadimitriou, *On the complexity of integer programming*, J. ACM, 28 (1981), pp. 765–768.
[28] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.
[29] Y. Papakonstantinou and V. Vianu, *DTD inference for views of XML data*, in Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Dallas, TX, 2000, pp. 35–46.
[30] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald, *Efficiently publishing relational data as XML documents*, The VLDB Journal, 10 (2001), pp. 133–154.
[31] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt, and J. Naughton, *Relational databases for querying XML documents: Limitations and opportunities*, in Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1999, pp. 302–314.
[32] J. W. Thatcher, *Characterizing derivation trees of context-free grammars through a generalization of finite automata theory*, J. Comput. System Sci., 1 (1967), pp. 317–322.
[33] J. Ullman, *Principles of Database and Knowledge-Base Systems, Volume* I, Computer Science Press, Rockville, MD, 1988.
[34] W3C, *Document Object Model (DOM) Level* 1 *Specification*, W3C Recommendation, October 1998, http://www.w3.org/TR/REC-DOM-Level-1/.
[35] W3C, *Extensible Markup Language (XML)* 1.0 (third edition), W3C Recommendation, February 2004, http://www.w3.org/TR/2004/REC-xml-20040204/.
[36] W3C, *XML-Data*, W3C Note, January 1998, http://www.w3.org/TR/1998/NOTE-XML-data-0105/.
[37] W3C, *XML Path Language (XPath) Version* 1.0, W3C Recommendation, November 1999, http://www.w3.org/TR/xpath.
[38] W3C, *XML Schema*, W3C Recommendation, October 2004, http://www.w3.org/XML/Schema.
[39] W3C, *XQuery* 1.0 : *An XML Query Language*, W3C Working Draft 29, October 2004, http://www.w3.org/TR/2004/WD-xquery-20041029/.
[40] W3C, *XSL Transformations (XSLT) Version* 1.0, W3C recommendation, November 1999, http:// www.w3.org/TR/xslt.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 881–898

COMPETITIVE ONLINE APPROXIMATION OF THE OPTIMAL SEARCH RATIO*

RUDOLF FLEISCHER†, TOM KAMPHANS‡, ROLF KLEIN§, ELMAR LANGETEPE§, AND GERHARD TRIPPEN¶

Abstract. How efficiently can we search an unknown environment for a goal in an unknown position? How much would it help if the environment were known? We answer these questions for simple polygons and for undirected graphs by providing online search strategies that are as good as the best offline search algorithms, up to a constant factor. For other settings we prove that no such online algorithms exist. We introduce a natural measure which gives reasonable results and is more realistic than pure pessimistic competitive analysis.

Key words. online motion planning, competitive ratio, searching, exploration, natural measure

AMS subject classifications. 68Q17, 68Q25, 68W40

DOI. 10.1137/060662204

1. Introduction.

One of the recurring tasks in life is to search one's environment for an object whose location is—at least temporarily—unknown. This problem comes in different variations. The searcher may have vision, or be limited to sensing by touch. The environment may be a simple polygon, for example, an apartment, or a graph, like a street network. Finally, the environment may be known to the searcher or be unknown.

Such search problems have attracted a lot of interest in online motion planning; see, for example, the survey by Berman [5]. Usually the cost of a search is measured by the length of the search path traversed; this, in turn, is compared against the length of the shortest path from the start position to the point where the goal is reached. The maximum quotient of these values, taken over all environments and all goal positions within an environment, is the *competitive ratio* of the search algorithm.

Most prominent is the problem of searching two half-lines emanating from a common start point. The "doubling" strategy visits the half-lines alternatingly, each time doubling the depth of exploration. This way, the goal point is reached after traversing a path at most 9 times as long as its distance from the start, and the competitive ratio of 9 is optimal for this problem; see Baeza-Yates, Culberson, and Rawlins [4] and Alpern and Gal [2]. This doubling approach frequently appears as a subroutine

*Received by the editors June 7, 2006; accepted for publication (in revised form) February 8, 2008; published electronically June 6, 2008. A preliminary version of this paper appeared at ESA 2004 [15]. The work described in this paper was partially supported by a grant from the National Natural Science Fund China (grant 60573025), by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (project HKUST6010/01E), and by a grant from the Germany/Hong Kong Joint Research Scheme sponsored by the Research Grants Council of Hong Kong and the German Academic Exchange Service (project G-HK024/02).

http://www.siam.org/journals/sicomp/38-3/66220.html

†Shanghai Key Laboratory of Intelligent Information Processing, Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China (rudolf@fudan.edu.cn).

‡Computer Science, Algorithms Group, Technical University of Braunschweig, 38106 Braunschweig, Germany (tom@kamphans.de).

§Institute of Computer Science I, University of Bonn, D-53117 Bonn, Germany (rolf.klein@uni-bonn.de, elmar.langetepe@informatik.uni-bonn.de).

¶Sauder School of Business, The University of British Columbia, 2329 West Mall, Vancouver, BC Canada (gerhard.trippen@sauder.ubc.ca).

in more complex navigation strategies.

In searching $m > 2$ half-lines, a constant ratio with respect to the distance from the start can no longer be achieved. Indeed, even if the half-lines were replaced by segments of the same finite length, the goal could be placed at the end of the segment visited last, causing the ratio to be at least $2m - 1$. Exponentially increasing the exploration depth by $m/m-1$ is known [4, 2] to lead to an optimal competitive ratio of

$$C(m) = 1 + 2m\left(\frac{m}{m-1}\right)^{m-1} \leq 1 + 2me.$$

Much less is known about more realistic settings. Suppose a searcher with vision wants to search an unknown simple polygon for a goal in unknown position. He could employ the m-way technique from above: By exploring the shortest paths from the start to the m reflex vertices of the polygon—ignoring their tree structure—a competitive ratio of $C(m)$ can easily be achieved [23]. Schuierer [27] has refined this method and obtained a ratio of $C(2k)$, where k denotes the smallest number of convex and concave chains into which the polygon's boundary can be decomposed.

But these results do not completely settle the problem. For one, it is not clear why the numbers m or k should measure the difficulty of searching a polygon. Also, human searchers can often outperform m-way search, because they make educated guesses about the shape of those parts of the polygon not yet visited.

In this paper we take the following approach: Let π be a *search path* for a fixed polygon P, i.e., a path from the start point, s, through P from which each point p inside P will eventually be visible. Let p_π be the first point on π where this happens for a point p. The cost of getting to p via π is equal to the length of π from s to p_π, plus the Euclidean distance from p_π to p. We divide this value by the length of the shortest s-to-p path in P. The supremum of these ratios, over all $p \in P$, is the *search ratio* of π. The lowest search ratio possible, over all search paths, is the *optimum search ratio* of P; it measures the "searchability" of P.

Apparently, this definition was first given by Koutsoupias, Papadimitriou, and Yannakakis [24]. They studied graphs with unit length edges where the goal can be located only at vertices, and they studied only the offline case where the graph is completely known a priori. They showed that computing the optimal search ratio offline is an NP-complete problem, and gave a polynomial time 8-approximation algorithm based on the doubling heuristic.

The crucial question we are considering in this paper is the following: Is it possible to design an *online* search strategy whose search ratio stays within a constant factor of the optimum search ratio for arbitrary instances of the environment? Surprisingly, the answer is positive for simple polygons as well as for undirected graphs. (However, for polygons with holes, and for graphs with unit edge length, where the goal positions are restricted to the vertices, no such online strategy exists.)

Observe that this way of measuring performance is one step beyond competitivity. Although the definitions of the search ratio and the competitive factor are quite similar, the concepts are different. In the competitive framework, we simply compare the online path from the start to the goal to the shortest s-to-t path. For an approximation of the optimal search ratio, we compare the online path to the best possible offline path, which, in turn, may already have a bad competitive ratio.

To exemplify the given concept let us consider another simple environment: arbitrary trees with a common root as a starting point. For a fixed tree, T, there will be

a strategy $S(T)$ which attains the best competitive search performance $C(T)$ among all possible goals on T. We know that an optimal strategy $S(T)$ exists, but we do not know what it looks like. Furthermore, for arbitrary trees it can be shown that the performance $C(T)$ of $S(T)$ can increase arbitrarily, though $S(T)$ is not known. Therefore, constant competitive searching is impossible for arbitrary trees T. But we can try to approximate the strategy $S(T)$ for any tree T by a simple strategy $A(T)$. In this paper we will show that we can design an approximation strategy $A(T)$ so that $A(T)$ is only a constant time worse than $S(T)$. This means that the competitive ratio of $A(T)$ is smaller than $c \cdot C(T)$ for every T and a fixed constant c. Note that $c = 4$ will be shown for this example in section 4.2.1. Thus, although the best search strategy is not known, we can approximate it efficiently. If the environment is known (but the goal is still unknown), sometimes the factor c can be slightly improved.

The idea of considering realistic ratios rather than pure pessimistic competitive ratios was also examined in the field of scheduling; see, for example, Edmonds et al. [13], Kalyanasundaram and Pruhs [22], or Berman and Coulston [6]. To compete with the optimal offline algorithm in a constant competitive sense, the online scheduler is allowed to have more power in speed or number of processors (or whatever). The increase of power is mainly represented by a parameter which, in turn, influences the competitive ratio. In our framework we increase the power of the agent only in the *offline* setting where the environment is known but the goal still has to be found. In the *online* setting we do not change the power balance between agent and adversary at all.

The *search* strategies we will present use, as building blocks, modified versions of constant-competitive strategies for online *exploration*, namely, the exploration strategy by Hoffmann et al. [21] for simple polygons and the tethered graph exploration strategy by Duncan, Kobourov, and Kumar [12].

At first glance it seems quite natural to employ an exploration strategy in searching—after all, either task involves looking at each point of the environment. But there is a serious difference in performance evaluation! In searching an environment, we compete against shortest start-to-goal paths, so we have to proceed in a breadth first search (BFS) manner.[1] In exploration, we are up against the shortest round trip from which each point is visible; this means that once we have entered some remote part of the environment we should finish it, in a depth first search (DFS) manner, before moving on.[2] However, we can fit these exploration strategies to our search problem by restricting them to a bounded part of the environment. This will be shown in section 3, where we present our general framework, which turns out to be quite elegant despite the complex definitions. The framework can be applied to both online and offline search ratio approximations. In section 2 we review basic definitions and notation. Our framework will then be applied to searching in various environments like trees, (planar) graphs, and (rectilinear) polygonal environments with and without holes in sections 4 and 5. In section 6 we give a construction scheme for lower bounds on the search ratio. Finally, in section 7, we conclude with a summary of our results.

2. Definitions. We want to find a good search path in some given environment $\mathcal{E}$. This may be a tree, a (planar) graph, or a (rectangular) polygon with or without

[1] But, as opposed to searching a data structure, we do not have pointers that allow us to jump to a different location for free.

[2] Observe, however, that neither plain BFS nor DFS would work! BFS is lacking the doubling element, and DFS, in a simple polygon, would tend to follow a long convex chain even though a small step to the side could be sufficient to see its endpoint.

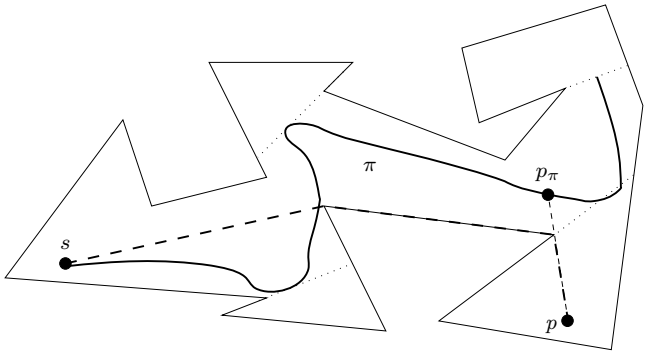


FIG. 2.1. *A search path π in a polygon, visiting all essential cuts (the dotted lines). The dashed path is the shortest path $sp(p)$ from s to the goal p. Moving along π, p can first be seen from p_π.*

holes. In a graph environment, the edges may have either unit length or arbitrary length. Edge lengths do not necessarily represent Euclidean distances, not even in embedded planar graphs. In particular, we do not assume that the triangle inequality holds. The only restriction to the type of environments required by our framework in section 3 is that there is a shortest path from every point, p, in $\mathcal{E}$ back to the start point, s, that is of the same length as a shortest path from s to p;[3] that is, for all p in $\mathcal{E}$ it holds that $|\mathrm{sp}(s,p)| = |\mathrm{sp}(p,s)|$.

In most cases, we want to search the whole environment, but there are special kinds of search problems where we know that the goal may be hidden only in some parts of the scene. Let the *goal set* $\mathcal{G} \subseteq \mathcal{E}$ denote the part of the environment where the stationary goal may be located. For example, if we search in a graph, $G = (V, E)$, the goal may be located anywhere along the edges of the graph; we call this setting *geometric search* and set $\mathcal{G} := V \cup E$. On the other hand, we may consider a *vertex search*, where the goal is restricted to be hidden in a vertex; in this case, we set $\mathcal{G} := V$. Now, *exploring* $\mathcal{E}$ means to move around in $\mathcal{E}$ until all potential goal positions $\mathcal{G}$ have been seen, whereas *searching* in $\mathcal{E}$ means to follow some exploration path in $\mathcal{E}$, until the goal has been seen. We require—as usual—that the distance to the goal is at least 1; otherwise, no search strategy is able to achieve a bounded competitive factor. Further, we assume that agents have perfect localization abilities; that is, they always know a map of the already explored part of $\mathcal{E}$, and they can always recognize when they visit some point for the second time (the robot localization problem is actually a difficult problem by itself; see, for example, [16]).

The searcher either can be *blind* (i.e., it can sense only its very close neighborhood) or can have *vision*; that is, it can see objects far away from its current position if the line of sight is not blocked by an obstacle. The model of visibility depends on the type of environment; see sections 4 and 5 for more details.

We now introduce some notation: Given a start point, $s \in \mathcal{E}$, let π be a path in the environment $\mathcal{E}$ starting in s. For a given point $q \in \pi$ let $\pi(q)$ denote the part of π from s to q. For an arbitrary point $p \in \mathcal{E}$ let $\mathrm{sp}(p)$ denote a shortest path from s to p in the given environment, and let $p_\pi \in \pi$ denote the point from which a searcher following π sees p for the first time; see Figure 2.1. We denote the length of a path

[3]Note that this is not the case for directed graphs, but it holds for undirected graphs and polygonal environments. We will see later that there is no constant-competitive online search algorithm for directed graphs.

segment $\pi(p)$ by $|\pi(p)|$. Paths computed by some algorithm $\mathcal{A}$ will be named $\mathcal{A}$, too. In particular, we write $|\mathcal{A}|$ for the length of the tour computed by $\mathcal{A}$. The main concept in our paper is the following.

DEFINITION 2.1. *Let $\mathcal{E}$ be an environment, $\mathcal{G} \subseteq \mathcal{E}$ a goal set, and $s \in \mathcal{E}$ a point in the environment. A* search path*, π, with start point s is a path which starts in s and allows a searcher following π to see every goal position in $\mathcal{G}$ from at least one point on π. The* search ratio*, $\mathrm{sr}(\pi)$, is defined as*

$$\mathrm{sr}(\pi) := \sup_{p \in \mathcal{G}} \frac{|\pi(p_\pi)| + |p_\pi p|}{|\mathrm{sp}(p)|}\,.$$

In other words, we compare the path walked by a searcher to the shortest path and take the worst ratio among all possible targets as the search ratio of our search path.

An optimal search path*, π_{opt}, is a search path with a minimum search ratio among all possible paths in the environment. We denote the* optimal search ratio *by sr_{opt}; that is, $\mathrm{sr}_{opt} := \mathrm{sr}(\pi_{opt})$. For blind agents, $p = p_\pi$ holds for every $p \in \mathcal{E}$; therefore, the search ratio can be computed as $\mathrm{sr}(\pi) := \sup_{p \in \mathcal{G}} \frac{|\pi(p)|}{|\mathrm{sp}(p)|}$.*

As the optimal search path seems hard to compute [24], we are interested in finding good approximations of the optimal search path in offline and online scenarios. We say a search algorithm $\mathcal{A}$ is *search-competitive* with factor C—or C–search-competitive for short—if there are constants $C \geq 1$ and $B \geq 0$, so that $\mathrm{sr}(\pi_\mathcal{A}) \leq C \cdot \mathrm{sr}_{\mathrm{opt}} + B$ holds for every path $\pi_\mathcal{A}$ computed by $\mathcal{A}$. Note that $\pi_\mathcal{A}$ is then a $C \cdot \mathrm{sr}(\pi_{\mathrm{opt}})$-competitive search path in the usual competitive sense. We use the term C–search-competitive also for the approximation factor of offline approximation algorithms. If there is no C–search-competitive algorithm for any constant C, we call this type of environment *hard-searchable*.

In the following, we want to use existing exploration algorithms to approximate the optimal search path. We assume that every exploration algorithm returns to the start point when the whole environment is explored.

DEFINITION 2.2. *For a given environment $\mathcal{E}$ and $d \geq 1$, let $\mathcal{E}(d)$ denote the part of $\mathcal{E}$ within distance at most d from s, and $\mathrm{OPT}(d)$ the optimal exploration for $\mathcal{E}(d)$. Further, let* Expl *be an—online or offline—algorithm for the exploration of environments of the given type.* Expl *is called* depth-restrictable *if for every $d \geq 1$ it is possible to modify the algorithm* Expl *to an algorithm* Expl(d) *that explores $\mathcal{E}(d)$ (i.e., the algorithm sees at least all potential goal positions of distance at most d—and maybe some more—before it returns to the start point), and there are constants $\beta > 0$ and $C_\beta \geq 1$, so that*

$$|\mathrm{Expl}(d)| \leq C_\beta \cdot |\mathrm{OPT}(\beta \cdot d)| \tag{2.1}$$

holds for every environment of the given type (i.e., Expl(d) *is C_β-competitive with respect to* $\mathrm{OPT}(\beta \cdot d)$*).*

For example, the DFS traversal for trees is depth-restrictable with $\beta = 1$ and $C_\beta = 1$: In every step we know exactly the distance to the tree's root. Thus, we can decide whether we can explore the children of the current node or cannot explore the tree more deeply, because we have reached depth d. Obviously, DFS is optimal also for depth-restricted exploration.

In the usual competitive framework, we would compare Expl(d) to the optimal algorithm $\mathrm{OPT}(d)$ (i.e., $\beta = 1$). As we will see later, our more general definition sometimes makes it easier to find depth-restrictable exploration algorithms. Usually,

we cannot just take an exploration algorithm Expl for $\mathcal{E}$ and restrict it to points within distance at most d from s. This way, we might miss useful shortcuts outside of $\mathcal{E}(d)$. Even worse, it may not be possible to determine in an online setting which parts of the environment belong to $\mathcal{E}(d)$, making it difficult to explore the right part of $\mathcal{E}$. In sections 4 and 5 we will derive depth-restricted exploration algorithms for graphs and polygons by carefully adapting existing exploration algorithms for the entire environment.

3. A general approximation framework. In this section, we show how to transform a depth-restrictable exploration algorithm, offline or online, into a search algorithm, without losing too much on the approximation factor.

Let $\mathcal{E}$ be the given environment and π_{opt} an optimal search path. Remember that we assume that, for any point p, we can reach s from p on a path of length at most $\text{sp}(p)$.

Let Expl be an exploration algorithm for $\mathcal{E}$, and for $d \geq 1$, let Expl(d) be the corresponding depth-restricted exploration algorithm for $\mathcal{E}(d)$. Let OPT and OPT(d) denote the corresponding optimal offline depth-restricted exploration algorithms. We assume that the exploration strategies will always return to the start.

To obtain a search algorithm for $\mathcal{E}$, we use the well-known *doubling paradigm* and successively apply our given exploration strategy with increasing exploration depth; that is, we successively run Expl(2^i), each iteration starting and ending in the start point, s.

THEOREM 3.1. *The doubling strategy based on a depth-restrictable exploration algorithm with factors C_β and β is a $4\beta C_\beta$–search-competitive search algorithm for blind agents and an $8\beta C_\beta$–search-competitive search algorithm for agents with vision.*

Proof. Consider one iteration of the doubling strategy with search radius $d \geq 1$. The optimal search path π_{opt} for $\mathcal{E}$ must in particular explore all possible goal positions within distance at most d from s. Let $last_d$ be the point on π_{opt} from which we see the last point within distance at most d from s when moving along π_{opt}. Returning from $last_d$ to s closes an exploration tour of $\mathcal{E}(d)$; therefore,

$$|\text{OPT}(d)| \leq |\pi_{\text{opt}}(last_d)| + |\text{sp}(last_d)|\,.$$

In contrast to blind searchers, $last_d$ may be located outside $\mathcal{E}(d)$ for agents with vision; thus, we distinguish between blind agents and agents with vision to bound $|\text{sp}(last_d)|$. A blind agent can detect $last_d$ only by visiting it, so we have $\text{sp}(last_d) \leq d$. Thus,

$$\text{sr}_{\text{opt}} \geq \frac{|\pi_{\text{opt}}(last_d)|}{d} \geq \frac{|\text{OPT}(d)| - d}{d} \iff |\text{OPT}(d)| \leq d \cdot (\text{sr}_{\text{opt}} + 1)\,. \tag{3.1}$$

The worst case for the search ratio of the doubling strategy occurs when we explore the environment up to some distance 2^{j+1} while the goal is within distance $2^j + \epsilon$ for some small $\epsilon > 0$. Thus, the search ratio of the doubling strategy is bounded by

$$\text{sr}(\pi) \leq \frac{\sum_{i=1}^{j+1} |\text{Expl}(2^i)|}{2^j + \epsilon}\,.$$

Expl is depth-restrictable with factor C_β, so we can apply (2.1):

$$\text{sr}(\pi) \leq \frac{C_\beta}{2^j} \cdot \sum_{i=1}^{j+1} |\text{OPT}(\beta \cdot 2^i)|\,.$$

Finally, with (3.1) we get

$$\begin{aligned}\mathrm{sr}(\pi) &\le \frac{C_\beta}{2^j} \cdot \sum_{i=1}^{j+1} \beta \cdot 2^i \cdot (\mathrm{sr}_{\mathrm{opt}} + 1) \\ &\le \beta C_\beta \cdot \left(\frac{2^{j+2} - 2}{2^j} \right) \cdot (\mathrm{sr}_{\mathrm{opt}} + 1) \\ &\le 4\beta\, C_\beta \cdot (\mathrm{sr}_{\mathrm{opt}} + 1)\,.\end{aligned}$$

If the agent has vision, it may see the last point within distance at most d from somewhere else, so we cannot guarantee $\mathrm{sp}(\mathit{last}_d) \le d$. We know only $|\mathrm{sp}(\mathit{last}_d)| \le |\pi_{\mathrm{opt}}(\mathit{last}_d)|$ (i.e., the return path to s cannot be longer than the path the agent traveled along π_{opt}). Thus,

$$\mathrm{sr}_{\mathrm{opt}} \ge \frac{|\pi_{\mathrm{opt}}(\mathit{last}_d)|}{d} \ge \frac{|\mathrm{OPT}(d)|}{2d} \iff |\mathrm{OPT}(d)| \le 2d \cdot \mathrm{sr}_{\mathrm{opt}}\,.$$

Further, we detect the goal by applying the exploration strategy with depth 2^{j+1} and return to the start; then we still have to move to the goal. Similar to the case of blind agents, we obtain for the search ratio an upper bound of

$$\begin{aligned}\frac{2^j + \sum_{i=1}^{j+1} |\mathrm{Expl}(2^i)|}{2^j} &\le 1 + C_\beta \cdot \frac{\sum_{i=1}^{j+1} |\mathrm{OPT}(\beta 2^i)|}{2^j} \le 1 + 2C_\beta \cdot \frac{\sum_{i=1}^{j+1} \beta 2^i\, \mathrm{sr}_{\mathrm{opt}}}{2^j} \\ &\le 1 + 8\beta C_\beta \cdot \mathrm{sr}_{\mathrm{opt}}\,. \qquad \square\end{aligned}$$

In the next two sections we will apply our framework to various types of environments and agents. The difficult part is always to find good depth-restrictable exploration algorithms.

4. Searching graphs. We distinguish between graphs with unit length and arbitrary length edges, planar and nonplanar graphs, directed and undirected graphs, as well as vertex and geometric search. We consider only blind agents: Located at a vertex of a (directed) graph, the agent senses only the number of outgoing edges, but neither their lengths nor the positions of the other vertices are known. Incoming edges cannot be sensed; see Deng and Papadimitriou [10]. Blind agents must eventually visit all points in the goal set. In the vertex search problem, we assume w.l.o.g. that graphs do not have parallel edges. Otherwise, there can be no constant–search-competitive vertex search algorithm: In Figure 4.1(i), the optimal search path $s \to v \to t \to s$ has length 3, whereas any online search path can be forced to cycle often between s and v before traversing the edge $v \to t$. Note that we can also use undirected edges.

4.1. Hard-searchable graphs. First, we show that for many graph classes there is no constant–search-competitive online search algorithm. Incidentally, there is also no constant-competitive online exploration algorithm for these graph classes. Note that we have the following implications for hard-searchable graphs:

- If planar graphs are hard-searchable, then so are nonplanar graphs.
- If graphs with unit length edges are hard-searchable, then so are graphs with arbitrary length edges.
- If undirected graphs are hard-searchable, then so are directed graphs (we can replace each undirected edge with directed edges in both directions).

THEOREM 4.1. *For blind agents, there is no constant–search-competitive online algorithm in the following settings:*

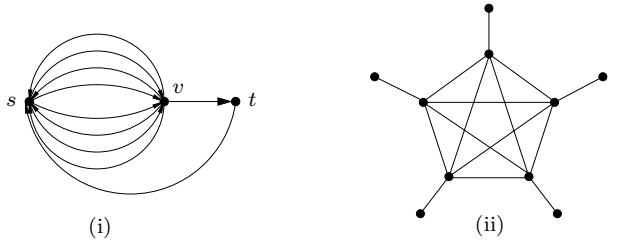


FIG. 4.1. *There is no constant–search-competitive vertex search algorithm for* (i) *graphs with parallel edges,* (ii) *vertex search in nonplanar graphs.*

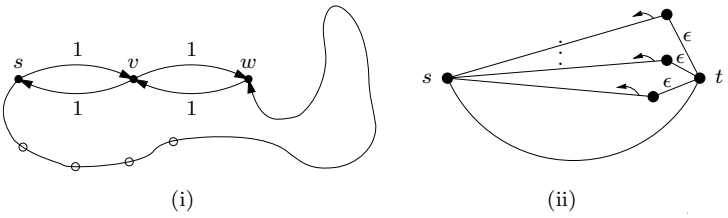


FIG. 4.2. *There is no constant–search-competitive vertex search algorithm for* (i) *vertex search and geometric search in directed graphs (the nodes ∘ occur only in the case of unit-length edges),* (ii) *vertex search in planar graphs with arbitrary edge length (the arrows denote the points on the edges where the searcher decides to return to s).*

1. *Vertex search in nonplanar graphs.*
2. *Vertex search and geometric search in directed planar graphs.*
3. *Vertex search in planar graphs with arbitrary edge lengths.*

Proof.

1. Consider the graph in Figure 4.1(ii) with unit length edges. It is a k-clique, where each vertex has a sibling that has only one connection to the clique. The optimal search ratio is $\Theta(k)$: We successively visit every clique vertex and explore its sibling before proceeding to the next clique vertex. This yields a path of length $3k$. On the other hand, any online search algorithm can be forced to travel $\Omega(k^2)$ before reaching the last vertex, so it has search ratio $\Omega(k^2)$. Thus, it is not better than k-competitive.
   Note that in a k-clique without siblings a BFS traversal achieves the optimal search ratio.
2. For vertex search, consider the planar graph in Figure 4.2(i) with a very long edge from s to w. The optimal search path, $s \to v \to w$, has search ratio 1. However, any online algorithm can be forced to first explore the long edge from s to w, resulting in a very high search ratio.
   If we are restricted to unit length edges, we can add more vertices along the long edge (marked with ∘ in Figure 4.2(i)). Then the optimal search path explores this long path after exploring the short cycle $s \to w \to s$. Because in the worst case the goal is hidden on the first vertex of the long path, this path achieves a search ratio of 5.
   For a geometric search, with unit length edges or arbitrary edges, we can use the same planar graph. Again, we force the online algorithm to explore the long edge at first. In contrast, the optimal strategy visits the cycle $s \to w \to s$ before exploring the long edge. The optimal strategy achieves its worst case

if the goal is hidden within distance 1 on the long edge.

3. For any given online search algorithm, we construct a planar graph as in Figure 4.2(ii) with k outgoing edges at the start vertex, s. An online algorithm visits one of the k edges at first. If the algorithm does not return to s while exploring this edge, the currently visited edge is the only long edge in the graph and all other edges are very short. The optimal strategy visits only short edges; thus, this algorithm can be arbitrarily bad.
   Hence, to achieve a good approximation factor, an online algorithm must at some point decide to stop exploring the first edge and to return to s (the small arrows in the figure indicate this point). We place the endpoint of the currently visited edge immediately behind the last visited point and continue similarly with the next $k-2$ edges. The last edge ends in vertex t, and its length is the minimum length among the first $k-1$ edges. Then we connect the endpoints of the first $k-1$ edges to t by an edge of length ϵ, where $\epsilon > 0$ is some very small number. The optimal search path in this graph first travels the last edge and then visits every other vertex quickly from t. Thus, its search ratio is close to 1. On the other hand, the online algorithm has search ratio at least k. Note that we introduced all the edge endpoints (instead of having the edges ending in t) because we assumed that there are no parallel edges. □

Note that there is an $O(D^8)$-competitive exploration for directed graphs by Fleischer and Trippen [17]. D denotes the *deficiency* of the given graph, that is, the minimum number of edges that must be added to get an Euler graph. This example shows that there are settings where there is no constant–search-competitive online algorithm, although there is an $O(D^8)$–competitive exploration algorithm. The problem is that this algorithm is not depth-restrictable. Besides, directed graphs do not fulfill $|\mathrm{sp}(s,p)| = |\mathrm{sp}(p,s)|$ for all p in $\mathcal{E}$, so we cannot apply our framework to directed graphs, anyway.

4.2. Competitive search in graphs. In this subsection, we present search-competitive online and offline search algorithms for the remaining graph classes. Note that we consider only undirected graphs.

4.2.1. Trees. On trees, DFS is a 1-competitive online exploration algorithm for vertex and geometric search that is depth-restrictable; it is still 1-competitive when restricted to search depth d for any $d \geq 1$. Thus, the doubling strategy gives a polynomial time 4–search-competitive search algorithm for trees—offline as well as online. On the other hand, it is an open problem whether the computation of an optimal vertex or geometric search path in trees with unit length edges is NP-complete [24].

4.2.2. Vertex search in graphs with unit length edges. Now, we give competitive search algorithms for vertex search in planar graphs with unit length edges and—in the next section—for geometric search in undirected graphs with arbitrary length edges. Both algorithms are based on an online algorithm for tethered graph exploration.

In the *tethered exploration* problem the agent is fixed to the start point by a rope of restricted length. An optimal solution to this problem was given by Duncan, Kobourov, and Kumar [12]. Their algorithm, closest first exploration (CFX), explores an unknown graph with unit length edges in $2|E| + (4 + \frac{16}{\alpha})|V|$ edge traversals, using a rope length of $(1+\alpha)d$, where d is the distance of the point farthest away from the

start point and $\alpha > 0$ is some parameter. Note that CFX explores the whole graph in spite of the *tethered* restriction.

CFX explores the graph in a mixture of depth-bounded DFS on G, DFS on spanning trees of parts of G, and recursive calls to explore certain large subgraphs. The basic idea of CFX is to maintain a set, $\mathcal{T}$, of edge-disjoint, partially[4] explored subtrees—more precisely, spanning trees of incomplete graph parts—that fulfill certain conditions on minimum size and maximal depth. Initially, $\mathcal{T}$ consists of one tree containing only the start node, s. The strategy successively selects the subtree nearest to s and walks to its root. Then, CFX traverses the subtree using DFS. For each encountered, incompletely explored vertex, CFX starts a depth-bounded DFS. In this process, new vertices are discovered and new subtrees are added to $\mathcal{T}$, possibly split into several smaller subtrees if they do not fulfill the conditions.

The costs for applying CFX sum up from relocation from s to the roots of the subtrees and back to s, the DFS traversals, and the costs for the depth-bounded DFS starting in unexplored vertices. The latter traverses only unexplored edges; thus, we have costs $2|E|$ for this part. For a subtree, T, with $|T|$ vertices, we have costs $2|T|$ for the DFS traversal. The size restrictions for the subtrees ensure that we can bound the relocation costs by $\frac{8}{\alpha}|T|$. As the subtrees may overlap, we can bound the sum of all vertices in the subtrees by $2|V|$. Altogether, we get

$$2|E| + \left(2 + \frac{8}{\alpha}\right) \sum_T |T| \le 2|E| + \left(2 + \frac{8}{\alpha}\right) \cdot 2|V| = 2|E| + \left(4 + \frac{16}{\alpha}\right) |V| \,.$$

As Duncan, Kobourov, and Kumar pointed out, the algorithm can be used even if the necessary rope length, d, is not known: They explore the whole graph by successively applying CFX and doubling d in every step. The important part is that the analysis still holds in this case. Particularly, we can apply CFX for a depth-restricted exploration (i.e., explore only a subgraph of G). For $d \ge 1$, let $G(d)$ denote the subgraph[5] of $G = (V, E)$ where all points $p \in V \cup E$ have distance at most $(1+\alpha)d$ from s. For convenience, let $G^* = (V^*, E^*) := G((1+\alpha)d)$. To explore all vertices in $G(d)$—and maybe some additional vertices from $G((1+\alpha)d)$—using a rope of length $(1+\alpha)d$, the number of edge traversals is bounded by

$$2|E^*| + \left(4 + \frac{16}{\alpha}\right) |V^*| \,.$$

Let us call this algorithm CFX(d). We have the following lemma.

LEMMA 4.2. *In planar graphs with unit length edges,* CFX *is a depth-restrictable algorithm for online vertex exploration with* $\beta = 1 + \alpha$ *and* $C_\beta = 10 + \frac{16}{\alpha}$.

Proof. As G^* is planar, we have $|E^*| \le 3|V^*| - 6$ by Euler's formula. Thus, the number of edge traversals of CFX(d) is at most

$$2\,|E^*| + \left(4 + \frac{16}{\alpha}\right) |V^*| \le 6\,|V^*| + \left(4 + \frac{16}{\alpha}\right) |V^*| = \left(10 + \frac{16}{\alpha}\right) |V^*| \,.$$

On the other hand, we have $\mathrm{OPT}((1+\alpha)d) \le |V^*|$, because the optimal algorithm must visit each vertex in V^* at least once. □

[4] That is, there are vertices that are already discovered but still have unvisited incident edges.

[5] In the case of unit-length edges, we omit all edges with length < 1 in the subgraph $G(d)$. Such edges occur if d is not an integer value.

Now, we can apply our framework with CFX.

THEOREM 4.3. *The doubling strategy based on* CFX(d) *is a* 206*–search-competitive online vertex search algorithm for blind agents in planar graphs with unit length edges.*

Proof. By Lemma 4.2, CFX(d) is depth-restrictable with $\beta = 1 + \alpha$ and $C_\beta = 10 + \frac{16}{\alpha}$. By Theorem 3.1, the doubling strategy based on CFX is $4\beta C_\beta$-competitive. Altogether, we have

$$4 \cdot \beta \cdot C_\beta = 4 \cdot (1 + \alpha) \cdot \left(10 + \frac{16}{\alpha}\right) = 104 + 40\alpha + \frac{64}{\alpha}\,.$$

By simple analysis, we get the minimal competitive ratio of 205.192... attained for $\alpha = \sqrt{\frac{8}{5}}$. ∎

4.2.3. Geometric search in graphs with arbitrary length edges. We note that CFX(d) can be modified to work on graphs with arbitrary length edges: Instead of counting the number of edge traversals, the algorithm has to track the length of the traversed edges. Now, it may happen that the maximal rope length is reached on an edge somewhere between two vertices. In this case, we interrupt the edge traversal, add an auxiliary vertex, and split the edge into two parts. Note that the added vertex is incompletely explored, so CFX will return to this vertex in a successive stage. Let $\ell(E)$ denote the total length of all edges in E. It is possible to adapt the proofs by Duncan, Kobourov, and Kumar [12] to prove the following lemma.

LEMMA 4.4. *In graphs with arbitrary length edges, CFX*(d) *explores all edges and vertices in* $G(d)$ *using a rope of length* $(1 + \alpha)d$ *at a cost of at most* $(4 + \frac{8}{\alpha}) \cdot \ell(E^*)$.

Proof (sketch). The proof is similar to the unit-length case, but we can no longer use the number of visited vertices to bound the number of traversed edges. Instead, we bound the costs for the depth-bounded DFS by $2\,\ell(E^*)$. As the subtrees are edge disjoint, we can bound the costs for the DFS traversals by $2\,\ell(E^*)$, too. The size restriction on the subtrees still ensures that the relocation costs are bound by $\frac{8}{\alpha}\,\ell(E^*)$. Altogether, we get $(4 + \frac{8}{\alpha}) \cdot \ell(E^*)$. Note that we have no additional costs for the auxiliary vertices, because we count only edge lengths, and by inserting auxiliary vertices we split one edge into two smaller edges whose total length is the same as the original edge. ∎

LEMMA 4.5. *In undirected graphs with arbitrary length edges, CFX is a depth-restrictable online geometric exploration algorithm with* $\beta = 1 + \alpha$ *and* $C_\beta = 4 + \frac{8}{\alpha}$.

Proof. The total cost of CFX(d) is at most $(4 + \frac{8}{\alpha}) \cdot \ell(E^*)$ by Lemma 4.4. On the other hand, OPT($(1 + \alpha)d$) must traverse each edge in E^* at least once. ∎

THEOREM 4.6. *The doubling strategy based on CFX*(d) *is a* 94*–search-competitive online geometric search algorithm for blind agents in undirected graphs with arbitrary length edges.*

Proof. By Lemma 4.5, CFX(d) is depth-restrictable with $\beta = 1+\alpha$ and $C_\beta = 4+\frac{8}{\alpha}$. Thus, by Theorem 3.1 the doubling strategy is $4\beta C_\beta$-competitive, and we have

$$4 \cdot \beta \cdot C_\beta = 4 \cdot (1 + \alpha) \cdot \left(4 + \frac{8}{\alpha}\right) = 48 + 16\alpha + \frac{32}{\alpha}\,.$$

Simple analysis shows that this factor is minimal for $\alpha = \sqrt{2}$, yielding a factor of 93.254.... ∎

4.2.4. Offline searching. In the offline setting, the searcher knows the graph but still does not know the location of the target. Thus, we want to compute (or approximate) an optimal search path in a known graph.

Computing an optimal search path in a known graph is NP-hard [24], but we can use our framework to give an approximation. Given a graph $G = (V, E)$ and an exploration depth $d \geq 1$, we can compute the subgraph $G(d)$. Now, we have to find an appropriate exploration strategy for $G(d)$. In the case of a vertex search, exploring $G(d)$ amounts to finding a traveling salesperson (TSP) tour on $G(d)$. This problem is NP-hard, too, but we can approximate a TSP tour within factor $C_\beta = 2$ using the mimimum–spanning-tree heuristic,[6] or we can use one of the $1+\epsilon$ approximations by Grigni, Koutsoupias, and Papadimitriou [18], Arora [3], or Mitchell [25].

The problem of finding a minimum-length tour that visits every edge of a given graph at least once is known as the Chinese postman problem and can be solved in polynomial time for graphs that are either directed or undirected [14, 26]. In this case, we have $C_\beta = \beta = 1$. Altogether we have the following theorem.

THEOREM 4.7. *There is a* 4*–search-competitive strategy for offline geometric search and an* 8*–search-competitive strategy for offline vertex search.*

5. Searching polygons.

5.1. Simple polygons. A simple polygon, P, is given by a closed, nonintersecting polygonal chain. Our searcher is equipped with ideal, unlimited vision; that is, it is provided with the full *visibility polygon* with respect to the searcher's current position.

To apply our framework, we need a depth-restrictable online exploration algorithm. The best known algorithm, PolyExplore, for the online exploration of a simple polygon by Hoffmann et al. [21] achieves a competitive ratio of 26.5.

Let $P(d) \subseteq P$ denote the part of the polygon P where all points have a distance at most d from the start. We can modify PolyExplore to explore $P(d)$: During the exploration, an unseen part of P always lies behind a *cut* c_v emanating from a reflex vertex, v. These reflex vertices are called *unexplored* as long as we have not visited the corresponding cut c_v. The algorithm maintains a list of unexplored reflex vertices and successively visits the corresponding cuts. While exploring a reflex vertex (along a sequence of line segments and circular arcs), more unexplored reflex vertices may be detected or unexplored reflex vertices may become explored. These vertices are inserted into or deleted from the list, respectively. In PolyExplore(d), unexplored reflex vertices within a distance greater than d from the start are simply ignored; that is, although they may be detected they will not be inserted into the list. Let OPT(d) be the shortest path that sees all points in $P(d)$.

Note that both PolyExplore(d) and OPT(d) may exceed $P(d)$, as shown in Figure 5.1. In (i) PolyExplore(d) successively explores the vertices v_r and v_ℓ, but OPT(d) visits the cuts outside $P(d)$. In (ii) PolyExplore(d) leaves $P(d)$ in e_4. However, we can enlarge $P(d)$ to $P'(d)$ by a well-defined region so that the resulting polygon contains PolyExplore(d) as well as OPT(d); see Figure 5.1.

More precisely, let d' be the maximal distance from PolyExplore(d) *and* OPT(d) to s; then we simply define $P'(d) := P(d')$. Newly inserted reflex vertices in $P'(d)$ cannot influence the paths of OPT(d) and PolyExplore(d) in $P'(d)$. Thus, the analysis of PolyExplore by Hoffmann et al. still holds for OPT(d) and PolyExplore(d) in $P'(d)$, and we have the following lemma.

LEMMA 5.1. *In a simple polygon,* PolyExplore *is a depth-restrictable online exploration algorithm with* $\beta = 1$ *and* $C_\beta = 26.5$.

[6] Note that we cannot apply the Christofides heuristic [8], because the triangle-inequality is not fulfilled in arbitrary graphs.

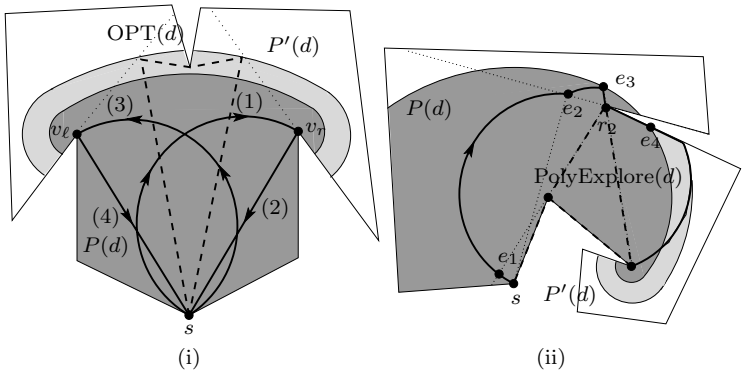


FIG. 5.1. (i) PolyExplore(d) *explores the right reflex vertex* v_r *along a circular arc* (1)*, returns to the start* (2)*, and explores the left reflex vertex* v_ℓ *likewise* (3)–(4). OPT(d) *(dashed line) leaves* $P(d)$. (ii) PolyExplore(d) *leaves* $P(d)$*, whereas the shortest exploration path for* $P(d)$ *lies inside* $P(d)$. *In both cases, we can extend* $P(d)$ *(dark gray) to* $P'(d)$ *(light gray) containing both* PolyExplore(d) *and* OPT(d).

THEOREM 5.2. *The doubling strategy based on* PolyExplore(d) *is a* 212*–search-competitive online search algorithm for an agent with vision in a simple polygon. There is also a polynomial time* 8*–search-competitive offline search algorithm.*

Proof. The online search-competitiveness follows from Lemma 5.1 and Theorem 3.1.

If we know the polygon, we can compute OPT(d) in polynomial time by adapting a corresponding algorithm for P. Every known polynomial time offline exploration algorithm visits the essential cuts in a certain sequence; see, for example, [7, 29, 28, 11]. Any of these algorithms can be used in our framework. As an optimal algorithm has approximation factor $C = 1$, our framework yields an approximation of the optimal search path with a factor of 8.

The overall running time of the algorithm seems to depend on the distance to the farthest reflex vertex of the polygon. However, we skip a step with distance 2^i if there is no reflex vertex within a distance between 2^{i-1} and 2^i. Thus, we always explore at least one new vertex in every iteration of the doubling strategy. Altogether, the total running time is bounded by a polynomial in the number of the vertices of P. ∎

Note that there is a considerable gap between the upper bound given by Hoffmann et al. [21] and the best known lower bound of 1.2825 [20]. The authors conjecture that the actual performance of PolyExplore is below 10 [21]; the worst case known so far is 5 [19]. Under this assumption, the search-competitivity of a doubling strategy based on PolyExplore can be expected to be below 80.

Now the question arises whether there is a polynomial time algorithm that computes the optimal search path in a simple polygon. We have to visit every essential cut, so we can try to visit them in any possible order. Anyway, we do not know exactly which point on the cut we should visit. We are not sure whether there are only a few possibilities as in the shortest watchman route problem. In other words, it is unknown whether this subproblem is discrete at all. So the problem of computing an optimal search path in a polygon is still open.

Even for rectilinear simple polygons no polynomial time algorithm for the optimal search path is known, but we can find better online algorithms.

THEOREM 5.3. *For an agent with vision in a simple rectilinear polygon there is*

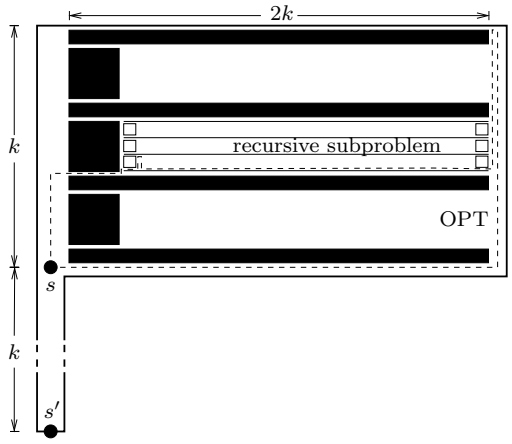


FIG. 5.2. *Lower bound construction for approximating the optimal search path in polygons with holes (start point s'). The upper half shows the lower bound for the exploration task by Albers, Kursawe, and Schuierer (start point s).*

an $8\sqrt{2}$–search-competitive online search algorithm. There is also a polynomial time 8–search-competitive offline search algorithm.

Proof. For a rectilinear simple polygon, P, Deng, Kameda, and Papadimitriou [9] introduced a simple $\sqrt{2}$-competitive online exploration algorithm. Obviously, this algorithm is depth-restrictable—we simply ignore reflex vertices farther away than d—while remaining $\sqrt{2}$-competitive compared to the restricted optimal path $\mathrm{OPT}(d)$. The optimal path never leaves $P(d)$, because we have only 90° reflex vertices. Our framework gives an $8\sqrt{2}$–search-competitive online search algorithm.

In the offline setting, we can obtain a polynomial time 8–search-competitive search algorithm based on an optimal depth-restricted exploration algorithm similar to Theorem 5.2. □

5.2. Polygons with holes. In this section, we show that there is no constant–search-competitive online search algorithm for polygons with (rectangular) holes. Albers, Kursawe, and Schuierer [1] showed that there is no constant-competitive online exploration algorithm for polygons with holes. They filled a rectangle of height k and width $2k$, $k \geq 2$, with $O(k^2)$ rectangular holes such that the optimal exploration tour has a length in $O(k)$, whereas any online exploration algorithm needs to travel a distance within $\Omega(k^2)$. Additionally, every point $p \in \mathcal{E}$ has at most the distance $3k$ from the start point, s; see Figure 5.2 and Albers, Kursawe, and Schuierer [1] for a detailed description.

THEOREM 5.4. *For an agent with vision in a polygon with holes there is no constant–search-competitive online search algorithm.*

Proof. Unfortunately, in the lower bound construction of Albers, Kursawe, and Schuierer the optimal exploration paths yields a bad search ratio. Thus, we enlarge the setting by a thin corridor of length k that leads to the former start point, s. Our new start point, s', is located at the end of the new corridor; see Figure 5.2. Now, every point that is not visible from s' is at least k steps away from s'; that is, $\mathrm{sp}(s', p) \geq k$ holds for such a point p. The optimal exploration path is still never longer than $C \cdot k$ for a constant C; thus, the optimal exploration path is a C-approximation of the optimal search path. In the new scene, every online exploration algorithm is forced to walk

a path of length in $\Omega(k^2)$. Because every online approximation of the optimal search path is also an online exploration algorithm, there are points that are discovered after walking a path length in $\Omega(k^2)$, although their distance to s' is within $O(k)$. Thus, no online approximation is able to achieve a constant approximation factor. □

Since the offline exploration problem is NP-complete (by straightforward reduction from planar TSP) we cannot use our framework to get a polynomial time approximation algorithm of the optimal search path. However, there is an exponential time 8-approximation algorithm. We can list the essential cuts of $\mathrm{OPT}(d)$ in any order to find the best one. Applying our framework gives an approximation factor of 8 for the optimal search ratio.

The results of Koutsoupias, Papadimitriou, and Yannakakis [24] imply that the offline problem of computing an optimal search path in a known polygon with holes is NP-complete.

6. A general lower bound. We have seen that for certain types of environments there exists an approximation for the optimal search path if there exists a depth-restrictable, competitive exploration strategy. Further, we have seen that polygons with holes are hard-searchable. Now, we want to generalize the latter result; that is, we want to show that—under a certain condition—there is no approximation up to a constant factor if there is no competitive exploration strategy for environments of the given type.

Usually, the nonexistence of competitive exploration strategies is shown by giving a lower bound—a scenario in which every exploration strategy is forced to walk a path whose length exceeds the length of the optimal exploration path by more than a constant factor. To transfer such a result to search path approximations, we require that the scenario can be extended around the start point, such that the start point moves further away from the original scenario. We used this technique in section 5.2. One might think of an arbitrary large narrow corridor which is added to a given environment.

DEFINITION 6.1. *Let $\mathcal{E}$ be an environment of arbitrary type and s be a start point in $\mathcal{E}$. We call $\mathcal{E}$ s-*extendable *if we can enlarge $\mathcal{E}$ locally around the start point; that is, it is possible to choose a new start point, s', outside $\mathcal{E}$ and enlarge $\mathcal{E}$ to $\mathcal{E}'$ such that s' is contained in $\mathcal{E}'$ and every path from s' to a point in $\mathcal{E}$ passes s. Additionally, we require that along any path from s' to s all targets in the extension will be (successively) detected—but no target ouside the extension. And we can design $\mathcal{E}'$ so that the shortest path from s to s' has arbitrary length.*

Beyond this rather technical condition some additional properties of a lower bound construction should hold.

We consider environments $\mathcal{E}$ of a given type such that $|\mathrm{sp}(s,p)| = |\mathrm{sp}(p,s)|$ holds. This was already one of our main requirements; see section 2. Let us now assume that there is an optimal exploration strategy OPT for $\mathcal{E}$ but in general there is no constant competitive online exploration strategy. We assume that this is shown by a lower bound construction $L(n)$, where n indicates the *size* of $L(n)$. More precisely, we assume that any online exploration strategy $\mathcal{A}$ has path length $|\mathcal{A}(L(n))| \geq C(n) \cdot |\mathrm{OPT}(L(n))|$ in the environment $L(n)$ and $C(n)$ is unbounded in n.

Furthermore, the length of the shortest path to any goal in $L(n)$ should be bounded by $K \cdot |\mathrm{OPT}(L(n))|$, where K is a constant. Obviously, the last condition always holds for blind agents.

Under the given conditions we can prove the following general result.

THEOREM 6.2. *If there is no constant-competitive online exploration algorithm*

for environments of a given type as indicated above, and the corresponding lower bound is s-extendable and additionally fulfills the properties above, then there is no competitive online approximation of the optimal search path.

Proof. Any online approximation, $\mathcal{A}$, of the optimal search path is also an online exploration strategy; therefore, we have $|\mathcal{A}(L(n))| \geq C(n) \cdot |\mathrm{OPT}(L(n))|$, and $C(n)$ is unbounded in n. We extend $L(n)$ by placing a new start point, s', *outside* $L(n)$ with distance $|\mathrm{OPT}(L(n))|$ and connecting it to the former start point s. Adding the shortest path from s to s' to $\mathrm{OPT}(L(n))$, this offline algorithm obviously has constant search ratio. The targets in the extension are detected optimally, and every target in $L(n)$ has a shortest distance of at least $|\mathrm{OPT}(L(n))|$ from s'. For visiting a detected goal t in $L(n)$ the extended version of $\mathrm{OPT}(L(n))$ might first run to its end and back to the start and then runs along the shortest path to t. This always gives a constant ratio.

On the other hand, for every online approximation, $\mathcal{A}$, there are some goals t which are detected after a path length greater than or equal to $C(n) \cdot |\mathrm{OPT}(L(n))|$, but the shortest path to the goal t is not longer than $(K+1) \cdot |\mathrm{OPT}(L(n))|$ for a constant K. Thus, there is a search ratio of at least $\frac{C(n)}{K+1}$ which is still unbounded in n. ∎

7. Conclusion and open problems. There are environments where no online search strategy can achieve a constant competitive factor. Therefore, we used the *search ratio* as a parameter of a given environment that gives a measure for the environment's searchability. A search strategy is considered "good" if it achieves a good approximation of the optimal search ratio; that is, the search ratio of an online strategy is at most a constant factor worse than the optimal search ratio.

We showed that we can use depth-restrictable exploration strategies—exploration strategies that can be modified to explore the environment only up to a certain depth while they are still competitive—to approximate the optimal search path by successively applying the exploration with exponentially increasing exploration depths. For blind agents we showed that there are $4\beta C_\beta$-approximations and for searchers with vision $8\beta C_\beta$-approximations, where β and C_β are parameters that depend on the modifications to turn an exploration algorithm into a depth-restricted exploration. We applied our results to various types of graphs and polygons; see Table 7.1.

Further, we showed that there is no constant–search-competitive strategy for polygons with holes. The main idea for this proof—enlarging the environment close to the start point—can be generalized for environments that fulfill a certain condition we called s-extendable. We also showed that some graph settings—including directed graphs—are hard-searchable.

Altogether, we showed a close relation between searching and exploring: For environments fulfilling $|\mathrm{sp}(s,p)| = |\mathrm{sp}(p,s)|$ for all p in $\mathcal{E}$ there is some equivalence between constant-competitive exploring and searching if the exploration strategy is depth-restrictable and the lower bounds are s-extendable (among some other natural properties). Naturally, these results lead to the question of whether there is a stronger connection. More precisely, can we omit at least the prerequisites "depth-restrictable" and "s-extendable" and show the following conjecture?

Conjecture 1. *For a given type of environment that fulfills* $\forall p \in \mathcal{E} : |\mathrm{sp}(s,p)| = |\mathrm{sp}(p,s)|$*, there is a constant–search-competitive strategy* if and only if *there exists a constant-competitive online exploration for environments of this type.*

Proving this conjecture would show a closer relation between exploration and searching: We are able to approximate the optimal search path—in other words,

TABLE 7.1

*Summary of our approximation results. The entry marked with * had earlier been proven by Koutsoupias, Papadimitriou, and Yannakakis* [24]. *They had also shown that computing the optimal search path is NP-complete for (planar) graphs. It is also NP-complete for polygons with holes, whereas it is not known to be NP-complete for trees and polygons without holes.*

Environment	Edge length	Goal	Polytime approximation ratio	
			Online	Offline
Tree	unit, arbitrary	vertex, geometric	4	4
Planar graph	arbitrary	vertex	no search-compet. alg.	8
Planar graph	unit	vertex	205.192...	8
Undirected graph	unit, arbitrary	vertex	no search-compet. alg.	8*
Undirected graph	arbitrary	geometric	93.254...	4
Simple polygon			212	8
Rect. simple polygon			$8\sqrt{2}$	8
Polygon with holes			no search-compet. alg.	?

we can find a good search strategy—if there is a constant-competitive exploration strategy. And, vice versa, we have no chance of finding a good search strategy if no constant-competitive exploration is possible. Note that the sp-condition is necessary, anyway, not only because our approximation framework relies on it, but also because it seems to be hard to find depth-restrictable exploration strategies for environments without the sp-condition. For example, there is a competitive exploration strategy for directed graphs (see Fleischer and Trippen [17]) which does not fulfill the sp-condition and is not depth-restrictable.

Acknowledgment. We would like to thank the anonymous referees for helpful comments and suggestions for improvements.

REFERENCES

[1] S. ALBERS, K. KURSAWE, AND S. SCHUIERER, *Exploring unknown environments with obstacles*, in Proceedings of the 10th ACM-SIAM Symposium of Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 842–843.

[2] S. ALPERN AND S. GAL, *The Theory of Search Games and Rendezvous*, Kluwer Academic Publishers, Boston, 2003.

[3] S. ARORA, *Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems*, J. ACM, 45 (1998), pp. 753–782.

[4] R. BAEZA-YATES, J. CULBERSON, AND G. RAWLINS, *Searching in the plane*, Inform. and Comput., 106 (1993), pp. 234–252.

[5] P. BERMAN, *On-line searching and navigation*, in Online Algorithms (Schloss Dagstuhl, 1996), A. Fiat and G. Woeginger, eds., Springer-Verlag, Berlin, 1998, pp. 232–241.

[6] P. BERMAN AND C. COULSTON, *Speed is more powerful than clairvoyance*, Nordic. J. Comput., 6 (1999), pp. 181–193.

[7] W.-P. CHIN AND S. NTAFOS, *Shortest watchman routes in simple polygons*, Discrete Comput. Geom., 6 (1991), pp. 9–31.

[8] N. CHRISTOFIDES, *Worst-case analysis of a new heuristic for the traveling salesman problem*, in Proceedings of the Symposium on New Directions and Recent Results in Algorithms and Complexity, J. F. Traub, ed., Academic Press, New York, 1976, p. 441.

[9] X. Deng, T. Kameda, and C. Papadimitriou, *How to learn an unknown environment* I: *The rectilinear case*, J. ACM, 45 (1998), pp. 215–245.
[10] X. Deng and C. H. Papadimitriou, *Exploring an unknown graph*, J. Graph Theory, 32 (1999), pp. 265–297.
[11] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell, *Touring a sequence of polygons*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 473–482.
[12] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar, *Optimal constrained graph exploration*, ACM Trans. Algorithms, 2 (2006), pp. 380–402.
[13] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng, *Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics*, J. Scheduling, 6 (2003), pp. 231–250.
[14] J. Edmonds and E. L. Johnson, *Matching, Euler tours and the Chinese postman*, Math. Programming, 5 (1973), pp. 88–124.
[15] R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen, *Competitive online approximation of the optimal search ratio*, in Proceedings of the 12th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 3221, Springer-Verlag, New York, 2004, pp. 335–346.
[16] R. Fleischer, K. Romanik, S. Schuierer, and G. Trippen, *Optimal robot localization in trees*, Inform. and Comput., 171 (2001), pp. 224–247.
[17] R. Fleischer and G. Trippen, *Exploring an unknown graph efficiently*, in Proceedings of the 13th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 3669, Springer-Verlag, New York, 2005, pp. 11–22.
[18] M. Grigni, E. Koutsoupias, and C. H. Papadimitriou, *An approximation scheme for planar graph TSP*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1995, pp. 640–645.
[19] R. Hagius, *Untere Schranken für das Online-Explorationsproblem*, Diplomarbeit, FernUniversität Hagen, Fachbereich Informatik, Hagen, Germany, 2002.
[20] R. Hagius, C. Icking, and E. Langetepe, *Lower bounds for the polygon exploration problem*, in Abstracts of the 20th European Workshop on Computational Geometry, Universidad de Sevilla, Seville, Spain, 2004, pp. 135–138.
[21] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel, *The polygon exploration problem*, SIAM J. Comput., 31 (2001), pp. 577–600.
[22] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, J. ACM, 47 (2000), pp. 617–643.
[23] R. Klein, *Algorithmische Geometrie*, 2nd ed., Springer-Verlag, Heidelberg, 2005.
[24] E. Koutsoupias, C. H. Papadimitriou, and M. Yannakakis, *Searching a fixed graph*, in Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 1099, Springer-Verlag, New York, 1996, pp. 280–289.
[25] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. Comput., 28 (1999), pp. 1298–1309.
[26] C. H. Papadimitriou, *On the complexity of edge traversing*, J. ACM, 23 (1976), pp. 544–554.
[27] S. Schuierer, *On-line searching in simple polygons*, in Sensor Based Intelligent Robots, Lecture Notes in Artificial Intelligence 1724, H. Christensen, H. Bunke, and H. Noltemeier, eds., Springer-Verlag, New York, 1999, pp. 220–239.
[28] X. Tan, T. Hirata, and Y. Inagaki, *Corrigendum to "An incremental algorithm for constructing shortest watchman routes,"* Internat. J. Comput. Geom. Appl., 9 (1999), pp. 319–323.
[29] X. H. Tan, T. Hirata, and Y. Inagaki, *An incremental algorithm for constructing shortest watchman routes*, Internat. J. Comput. Geom. Appl., 3 (1993), pp. 351–365.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 899–921

ON APPROXIMATING THE DEPTH AND RELATED PROBLEMS*

BORIS ARONOV† AND SARIEL HAR-PELED‡

Abstract. We study the question of finding a deepest point in an arrangement of regions and provide a fast algorithm for this problem using random sampling, showing it sufficient to solve this problem when the deepest point is shallow. This implies, among other results, a fast algorithm for approximately solving linear programming problems with violations. We also use this technique to approximate the disk covering the largest number of red points, while avoiding all the blue points, given two such sets in the plane. Using similar techniques implies that approximate range counting queries have roughly the same time and space complexity as emptiness range queries.

Key words. approximation, randomized algorithms, computational geometry

AMS subject classifications. 68W25, 68W40

DOI. 10.1137/060669474

1. Introduction.

In this paper, we study the problem of efficiently computing the deepest point in a collection of regions. Here, the *depth* of the point is the number of regions containing it. This is a natural problem that arises in optimization, where a constraint induces a region in space where it holds, and we would like to find the point that satisfies all the constraints; see Figure 1. If no such point exists, we would like to find a point that satisfies the maximum number of constraints. As a concrete example, a linear program with n inequalities over d variables induces a set of n halfspaces in $\mathbb{R}^d$. The given linear program is feasible (i.e., it has a feasible solution) if there is a point of depth n in this arrangement of n halfspaces. Furthermore, if the given linear program is not feasible, then a point with maximum depth in the arrangement is a solution violating the least number of constraints.

In this paper, we provide several reductions for the problem of finding the deepest point and show how to solve some depth problems using these techniques.

Computing a point of maximum depth via depth thresholding. In section 3, we present a general reduction that allows one to quickly compute an approximately deepest point in an arrangement of objects given a slower exact procedure for computing the depth exactly. More precisely, suppose we are given a *depth thresholding procedure* that can find, given an integer k, a deepest point for a set of n regions in $T_{\mathrm{DT}}(n,k)$ time, provided the depth does not exceed k; if it does, the procedure outputs "depth $> k$." The resulting approximation algorithm, given a set $\mathcal{S}$ of n regions, can find a point of depth at least $(1-\varepsilon)k_{\mathrm{opt}}$, where k_{opt} is the maximum number of regions covering any point. The running time of this new algorithm is

*Received by the editors September 11, 2006; accepted for publication (in revised form) February 15, 2008; published electronically June 6, 2008. The latest version of this paper is available from http://www.uiuc.edu/~sariel/papers/04/depth. A preliminary version of this paper appeared as [AH05].

http://www.siam.org/journals/sicomp/38-3/66947.html

†Department of Computer and Information Science, Polytechnic University, Brooklyn, New York, NY 11201 (aronov@ziggy.poly.edu, http://cis.poly.edu/~aronov). This author's research was supported in part by NSF ITR grant CCR-00-81964, by a grant from the U.S.-Israel Binational Science Foundation, and by NSA MSP grant H98230-06-1-0016. Part of the work was carried out while this author visited UIUC in 2004.

‡Department of Computer Science, University of Illinois, 201 N. Goodwin Avenue, Urbana, IL 61801 (sariel@uiuc.edu, http://www.uiuc.edu/~sariel/). This author's research was partially supported by NSF CAREER award CCR-0132901.

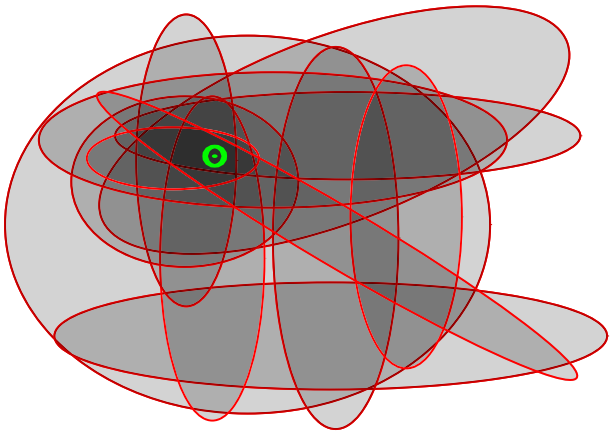


FIG. 1.

$O(T_{\mathrm{DT}}(n, \varepsilon^{-2} \log n) + n)$, assuming $T_{\mathrm{DT}}(n, k) = \Omega(n)$ for any k. This reduction is applicable to any set of "well-behaved" regions in constant dimension.

Unless explicitly stated otherwise, all bounds on running time and space hold with high probability (i.e., at least $1 - 1/n^{O(1)}$) and also in expectation. The results returned by these algorithms are correct with high probability.

Next, in section 4, we present several applications of this reduction. In section 4.1, we show that one can solve linear programming problems with violations approximately in near-linear time. Explicitly, consider a linear program L with n constraints in $\mathbb{R}^d$ and a linear objective function f. Suppose k_{opt} is the minimum number of constraints that have to be violated to make L feasible, and let v be the optimal solution in this case; namely, v is the point minimizing $f(v)$ with exactly k_{opt} violated constraints. Then, one can find a point u that violates at most $(1 + \varepsilon)k_{\mathrm{opt}}$ constraints of L and such that $f(u) \leq f(v)$. The running time of the new algorithm is $O(n(\varepsilon^{-2} \log n)^{d+1})$. This compares favorably with the previous exact algorithm of Matoušek [Mat95] which requires $O(nk^{d+1})$ running time. (In two and three dimensions faster exact algorithms exist [Cha05]. See section 4.1.) To appreciate this result, consider the case where $k = \sqrt{n}$. A natural approach to approximate linear programming problems with violations is to compute a δ-approximation [VC71] to the set of constraints in L and apply the exact algorithm on this sample. However, in this case, $\delta = \varepsilon/\sqrt{n}$, and the required random sample would include (almost) all the constraints of L, thus achieving no speedup.

An application: Finding a disc covering red points, but no blue points. Consider two finite sets of points in the plane, *red* and *blue*. As another application of the aforementioned technique, we investigate the problem of finding a disk containing the largest number of red points, while avoiding all the blue points. This is a natural problem related to learning and clustering [DHS01]. For example, one might try to learn a concept believed to be a disk, from examples, where the red points represent positive examples and the blue ones represent negative ones. A possible criterion to optimize is to say that the best concept (i.e., disk) is the one that classifies the blue points correctly (i.e., avoids them), while minimizing the error on the red points (i.e., the disk covers as many red points as possible).

The corresponding problem, in high dimensions, of computing the best such ball seems to be computationally hard, since computing the separating hyperplane min-

imizing the number of outliers is NP-hard [AK95]. In fact, approximating it within a factor of $2^{\log^{0.5-\varepsilon} n}$, for any fixed $\varepsilon > 0$, is "almost" NP-hard; see [ABSS97] for details. In constant dimension d, this ball can be computed by brute-force enumeration of all possibilities, in $n^{O(d)}$ time. Motivated by this unattractive option, we develop approximation algorithms for this problem, where we look for the ball containing (approximately) the largest number of red points, while avoiding all the blue points. More specifically, we observe that every red point p induces a feasible region which is the locus of the centers of the maximal balls containing p and not containing any blue points in their interior. Thus, the problem reduces to that of finding a point in the plane (or in space) having the largest number of such regions covering it, namely, a *deepest* point in an arrangement of such regions.

In section 4.2, we show that computing the deepest point in an arrangement of disks in the plane is 3SUM-hard [GO95] and present a $(1-\varepsilon)$-approximation algorithm with $O(n\varepsilon^{-2}\log n)$ expected running time. In section 4.3, we investigate the geometry of the problem of, given a set of blue and a set of red points, computing the disk containing the largest number of red points while avoiding all blue points. In particular, we show that this is equivalent to, given a set of red planes and a set of blue planes in three dimensions, computing the point having the largest number of red planes below it, while having all the blue planes above it. The points lying below all blue planes and above a specific red plane form the *feasible region* of that plane. Projecting the feasible regions to the plane, we obtain a set of pseudodisks which can be manipulated efficiently using an implicit representation. Plugging this into the algorithm for computing the depth of an arrangement of pseudodisks results in an approximation algorithm, with $O(n\varepsilon^{-2}\log^2 n)$ expected running time, which returns a disk covering $(1-\varepsilon)k_{\text{opt}}$ points, where k_{opt} is the number of red points covered by the optimal disk. We also study this problem in higher dimensions.

Approximate counting. Given a set of points P in $\mathbb{R}^d$ and a class of ranges, the problem of *range searching* is preprocessing P so that, given a range τ, one can quickly answer (i) *emptiness queries*: test whether $\tau \cap P = \emptyset$; (ii) *counting queries*: report the size of $\tau \cap P$; or (iii) *reporting queries*: report the elements of $\tau \cap P$. This problem and its variants have numerous applications and have received substantial attention. See [AE99] for a survey of the subject.

Interestingly, there is a complexity gap between emptiness queries and counting queries. In the (somewhat reasonable) computation model assumed by Brönnimann, Chazelle, and Pach [BCP93], halfspace counting queries require $\Omega^*(n^{1-2/(d+1)})$ time per query if only linear space is allowed.[1] Matoušek [Mat93] showed how to answer such queries in $O(n^{1-1/d})$ time. On the other hand, halfspace emptiness queries can be answered in logarithmic time in two and three dimensions, and in $O^*(n^{1-1/\lfloor d/2\rfloor})$ time in higher dimensions [Mat92], using near-linear space; this is slightly faster than the aforementioned lower bound for the counting problem. Especially interesting is the situation in dimensions two and three, where the gap is between logarithmic query time for emptiness queries and polynomial query time for counting queries.

In section 5, we show that this gap disappears if one is willing to get an approximate answer to the counting query. In particular, given a prespecified ε, $0 < \varepsilon < 1/2$, we show how to reduce a counting query to a sequence of emptiness queries of length polynomial in $\log n$ and ε^{-1}. For any range τ, this data structure reports a number μ_τ, such that $(1-\varepsilon)\,|\tau \cap P| \le \mu_\tau \le |\tau \cap P|$. Thus, approximate counting and empti-

[1] We use $f(n) = O^*(g(n))$ to express $f(n) = O(g(n)\log^c n)$ and $f(n) = \Omega^*(g(n))$ to mean $f(n) = \Omega(g(n)\log^{-c} n)$ for some constant $c > 0$.

ness range searching are computationally nearly equivalent. Since this circumvents the aforementioned gap, we believe it to be of independent interest.

We contrast our results with the work of Arya and Mount [AM00], which considers a different notion of approximation for the range searching problem. They approximate the range (using a distance which depends on the diameter of the query shape) and return the *exact* count inside this *approximate* shape. In our results, on the other hand, the shape is fixed but the count returned is approximated. In particular, our results apply to halfspace range searching, while Arya–Mount methods cannot be applied here, as a halfspace has infinite diameter.

Our results are based on careful application of random sampling, using several multiresolution samples. By performing the computation at the right resolution, we are able to achieve fast running time. This technique is by now standard in the area of randomized algorithms. Recent results that use similar techniques include [IM98, Ind00, HI00]; this list is by no means exhaustive. In the context of halfspace range searching, Chan [Cha00] used multiresolution random samples to roughly estimate the depth of the query, and speed up halfspace range *reporting* queries, in the expected sense.

2. Preliminaries. Given a set of objects $\mathcal{S}$ in $\mathbb{R}^d$ and a point $q \in \mathbb{R}^d$, let the *depth of q in $\mathcal{S}$*, $\operatorname{depth}(q, \mathcal{S})$, be the number of objects of $\mathcal{S}$ containing q. The *depth of $\mathcal{S}$* is defined as $\max_q \operatorname{depth}(q, \mathcal{S})$, with q ranging over all of $\mathbb{R}^d$. Finally, let $\operatorname{core}(\mathcal{S})$ denote the locus of points realizing $\operatorname{depth}(\mathcal{S})$.

In the remainder of the paper, we assume that the sets $\mathcal{S}$ are well behaved. In particular, we require that the complexity of the arrangement formed by $\mathcal{S}$ be bounded by $|\mathcal{S}|^{O(d)}$. The *arrangement* formed by $\mathcal{S}$ is the decomposition of the plane (or, more generally, $\mathbb{R}^d$) induced by a collection of such shapes [SA95]. The *combinatorial complexity* of an arrangement is the total number of edges, faces, and vertices in the arrangement. A *k-level* in an arrangement of curves is the closure of the set of points on the curves that have exactly k curves below them. For a set of regions R, *the union of R* is the set of points in the plane covered by at least one region of R. For a union of regions, its *complexity* is the number of edges and vertices of the arrangement lying on the boundary of the union (i.e., this is the descriptive complexity of the union). Analogous definitions apply in higher dimensions.

In the following, we need the Chernoff inequality; see [MR95].

THEOREM 2.1 (Chernoff inequality). *Let $X_1, \ldots, X_n$ be n independent Bernoulli trials, where $\mathbf{Pr}[X_i = 1] = p_i$, $\mathbf{Pr}[X_i = 0] = 1 - p_i$, $Y = \sum_i X_i$, and $\mu = \mathbf{E}[Y]$. Then, for any $\delta > 0$,*

$$\mathbf{Pr}[Y > (1+\delta)\mu] < \begin{cases} \exp\left(-\mu\delta^2/4\right), & \delta \le 2e-1 \text{ (case (a))}, \\ 2^{-\mu(1+\delta)}, & \delta \ge 2e-1 \text{ (case (b))}, \end{cases} \tag{1}$$

and

$$\mathbf{Pr}[Y < (1-\delta)\mu] \le \exp\left(-\mu\delta^2/2\right). \tag{2}$$

3. From exact depth to fast approximate depth. Given a set $\mathcal{S}$ of n objects and $\varepsilon > 0$, we consider the following two problems:

Problem 1: Computing $\operatorname{depth}(\mathcal{S})$ exactly, perhaps also producing a *witness point*, i.e., a point in $\operatorname{core}(\mathcal{S})$.

Problem 2: Estimating $\delta = \operatorname{depth}(\mathcal{S})$, i.e., producing an integer k with the $(1-\varepsilon)\delta \le k \le \delta$ together with a witness point of that depth.

We will also need, as a subroutine, a procedure DEPTHTHRESHOLD($\mathcal{S}', k$) with the following behavior: Given a collection $\mathcal{S}' \subset \mathcal{S}$ of n' objects and an integer $k > 0$, determine depth($\mathcal{S}'$) exactly (and produce a witness point of this depth) *if* it does not exceed k; otherwise, just return "depth($\mathcal{S}'$) $> k$." We refer to this as *depth thresholding*. Let $T_{\text{DT}}(n', k)$ be its running time.

Solving Problem 1 seems to require computing the whole arrangement $\mathcal{A}(\mathcal{S})$ (at least in general settings). Therefore, a running time better than $O(n^d)$ for this problem seems unlikely. As such, we will concentrate in this section on the approximation version of this problem, namely, Problem 2. In particular, in this section, we show how to solve Problem 2 by performing a logarithmic number of depth thresholding calls.

3.1. An approximate decision procedure. The intuitive idea behind approximating $\delta := \text{depth}(\mathcal{S})$ for a collection $\mathcal{S}$ of n objects is to perform a binary search on the value of the depth. However, we cannot afford to use an exact decision procedure (i.e., a test comparing δ to a given number k) in each round, as all known methods of computing the depth exactly essentially compute the full arrangement $\mathcal{A} = \mathcal{A}(\mathcal{S})$. (Actually, using the depth thresholding idea, one can test whether $\delta > k$ for any given k, without necessarily computing the whole arrangement, but the cost $T_{\text{DT}}(n, k)$ of doing this usually grows with k, so it is too expensive to perform the test for large values of k. This is precisely the case where the randomized procedure given below is more efficient.) The idea is to first perform a "rough" search for the right depth (up to a constant factor). Once we are in the right range, one can find the approximate depth by applying the depth thresholding procedure to a single random sample, using the relation between the depth of $\mathcal{S}$ and that of the sample.

To this end, we develop an approximate decision procedure, which, given a value k, returns "$k < \delta$" with high probability if k is significantly smaller than δ and "$k > \delta$" with high probability if k is significantly larger than δ. We start by proving the following technical lemma.

LEMMA 3.1. *Let $\mathcal{S}$ be a set of n objects in $\mathbb{R}^d$, ε, $0 < \varepsilon < 1/2$, be fixed, $k > 0$ be an integer, and $R \subseteq \mathcal{S}$ be a random subset formed by picking each object with probability*

$$\psi = \psi(\varepsilon, k) := \min\left(c_1 \frac{\log n}{k\varepsilon^2},\ 1\right)$$

independently, where c_1 is a sufficiently large constant. Then, for a point $p \in \mathbb{R}^d$, we have, with high probability, that

(i) *if* $\text{depth}(p, R) \geq \alpha_+(\varepsilon, k) := 2k\psi$, *then* $\text{depth}(p, \mathcal{S}) \geq (3/2)k$;
(ii) *if* $\text{depth}(p, R) \leq \alpha_-(\varepsilon, k) := (1 - \varepsilon)k\psi$, *then* $\text{depth}(p, \mathcal{S}) \leq k$;
(iii) *if* $\text{depth}(p, R) \geq \alpha_-(\varepsilon, k)$, *then*

$$(1 - \varepsilon)\,\text{depth}(p, \mathcal{S}) \leq \frac{\text{depth}(p, R)}{\psi} \leq (1 + \varepsilon)\,\text{depth}(p, \mathcal{S});$$

namely, the depth of a "deep" point in $\mathcal{A}(R)$ is a good estimate of its depth in $\mathcal{A}(\mathcal{S})$.

Proof. If $\psi = 1$, there is nothing to prove, so we assume $\psi < 1$ in what follows.

(i) Consider a point $p \in \mathcal{A}(\mathcal{S})$ such that $r := \text{depth}(p, \mathcal{S}) < (3/2)k$. We have that

$\mu := \mathbf{E}[\mathrm{depth}(p,R)] = r\psi$ and $r(1+\varepsilon/4) < 2k$. Therefore,

$$\begin{aligned}\nu := \mathbf{Pr}[\mathrm{depth}(p,R) \geq 2k\psi] &= \mathbf{Pr}\left[\mathrm{depth}(p,R) \geq \frac{2k}{r}\mu\right] \\ &= \mathbf{Pr}\left[\mathrm{depth}(p,R) \geq \left(1+\left(\frac{2k}{r}-1\right)\right)\mu\right].\end{aligned}$$

If $\frac{2k}{r} - 1 \leq 2e-1$, i.e., $k/e \leq r$, then, by the Chernoff inequality ((1), case (a)), we have

$$\begin{aligned}\nu &\leq \exp\left(-\mu\left(\frac{2k}{r}-1\right)^2/4\right) = \exp\left(-r\psi\left(\frac{2k}{r}-1\right)^2/4\right) \leq \exp\left(-r\left(c_1\frac{\log n}{k\varepsilon^2}\right)\frac{\varepsilon^2}{36}\right) \\ &\leq \exp\left(-\frac{k}{e}\left(c_1\frac{\log n}{k\varepsilon^2}\right)\frac{\varepsilon^2}{36}\right) = \frac{1}{n^{\Omega(1)}}.\end{aligned}$$

If $r \geq k/e$ then, by the Chernoff inequality ((1), case (b)), we have

$$\nu \leq 2^{-2k\mu/r} = 2^{-2k\psi} \leq \frac{1}{n^{\Omega(1)}}.$$

(ii) Let p be a point of $\mathbb{R}^d$ such that $r = \mathrm{depth}(p,\mathcal{S}) \geq k$. We have that $\mu = \mathbf{E}[\mathrm{depth}(p,R)] = r\psi$. Arguing as above, we have

$$\mathbf{Pr}[\mathrm{depth}(p,R) < (1-\varepsilon)k\psi] \leq \mathbf{Pr}[\mathrm{depth}(p,R) < (1-\varepsilon)r\psi] \leq \exp\left(-\frac{\varepsilon^2}{2}r\psi\right) \leq \frac{1}{n^{\Omega(1)}}.$$

(iii) Arguing as in case (i), we have that if $\mathrm{depth}(p,R) \geq \alpha_-(\varepsilon,k)$, then $\mathrm{depth}(p,\mathcal{S}) \geq k/2$, and this holds with high probability. As such, we have that $\mu = \mathbf{E}[\mathrm{depth}(p,R)] \geq k\psi/2$. This implies, using the Chernoff inequality (Theorem 2.1), that

$$\mathbf{Pr}[\mathrm{depth}(p,R) > (1+\varepsilon)\mu] \leq \exp\left(-\frac{\varepsilon^2}{4}\mu\right) \leq \exp\left(-\frac{\varepsilon^2}{8}k\psi\right) \leq \frac{1}{n^{\Omega(1)}}.$$

Similarly,

$$\mathbf{Pr}[\mathrm{depth}(p,R) < (1-\varepsilon)\mu] \leq \exp\left(-\frac{\varepsilon^2}{2}\mu\right) \leq \frac{1}{n^{\Omega(1)}},$$

implying the claim. □

COROLLARY 3.2. *Let $\mathcal{S}$ be a set of n objects, with $\delta = \mathrm{depth}(\mathcal{S})$. Let ε, $0 < \varepsilon < 1/2$, be fixed, $k \geq \delta/4$ be an integer, and $R \subseteq \mathcal{S}$ be a random subset formed by picking each object with probability*

$$\psi = \psi(\varepsilon,k) := \min\left(c_1\frac{\log n}{k\varepsilon^2},\ 1\right),$$

independently, where c_1 is a sufficiently large constant. Then, we have that

(i) *if $\mathrm{depth}(R) \geq \alpha_+(\varepsilon,k) := 2k\psi$, then with high probability $\delta \geq (3/2)k$;*
(ii) *if $\mathrm{depth}(R) \leq \alpha_-(\varepsilon,k) := (1-\varepsilon)k\psi$, then with high probability $\delta \leq k$; and*
(iii) *for all $p \in \mathbb{R}^d$, such that $\mathrm{depth}(p,R) \geq \alpha_-(\varepsilon,k)$, we have with high probability that*

$$(1-\varepsilon)\,\mathrm{depth}(p,\mathcal{S}) \leq \frac{\mathrm{depth}(p,R)}{\psi} \leq (1+\varepsilon)\,\mathrm{depth}(p,\mathcal{S}).$$

Proof. The proof follows immediately from Lemma 3.1, since the complexity of $\mathcal{A}(\mathcal{S})$ is polynomial. Indeed, place a point inside each face of $\mathcal{A}(\mathcal{S})$, and apply the lemma to all these points. ∎

3.1.1. If we know where to look, we can find it. Assume that we are given an estimate y of $\delta := \operatorname{depth}(\mathcal{S})$, with $y \leq \delta \leq 8y$. Then we can compute approximately the depth of $\mathcal{S}$. Indeed, compute a random sample R of $\mathcal{S}$ as specified by Lemma 3.1 for $k = y/2$. Next, perform depth thresholding on R with threshold $M := 16y\psi = O(\varepsilon^{-2} \log n)$, where $\psi = \psi(\varepsilon/4, k)$. It is easy to verify that, with high probability (by the Chernoff inequality), the depth of R is bounded from above by M (as the expected maximum depth is $M/2$). Furthermore, $\delta > k$, and by Corollary 3.2 (ii), we have $\operatorname{depth}(R) > \alpha_-(\varepsilon/4, k)$ (again, with high probability). Let v be the witness point returned by DepthThreshold(R, M) with the maximum depth in R. The depth of v in $\mathcal{A}(R)$ exceeds $\alpha_-(\varepsilon/4, k)$, and by Corollary 3.2 (iii), $\operatorname{depth}(v, R)/\psi$ is a $(1 \pm \varepsilon/4)$-approximation to $\operatorname{depth}(v, \mathcal{S})$. In particular,

$$\operatorname{depth}(v, R) \leq (1 + \varepsilon/4)\psi \operatorname{depth}(v, \mathcal{S}).$$

Now, consider a point $q \in \operatorname{core}(\mathcal{S})$ that is with $\operatorname{depth}(q, \mathcal{S}) = \operatorname{depth}(\mathcal{S})$. As before, with high probability, $\operatorname{depth}(q, R) \geq \alpha_-(\varepsilon/4, k)$, and we have

$$(1 - \varepsilon/4)\psi \operatorname{depth}(\mathcal{S}) \leq \operatorname{depth}(q, R) \leq \operatorname{depth}(v, R) \leq (1 + \varepsilon/4)\psi \operatorname{depth}(v, \mathcal{S}).$$

Namely, $\operatorname{depth}(v, \mathcal{S}) \geq (1 - \varepsilon) \operatorname{depth}(\mathcal{S})$ and

$$(1 - \varepsilon/4) \operatorname{depth}(\mathcal{S}) \leq \frac{\operatorname{depth}(v, R)}{\psi} \leq (1 + \varepsilon/4) \operatorname{depth}(\mathcal{S}).$$

Computing v takes $O\big(n + T_{\text{DT}}(2n\psi(\varepsilon/4, k), O(\varepsilon^{-2} \log n))\big)$ time and succeeds with high probability, where $2n\psi(\varepsilon/4, k)$ is an upper bound on the size of R that holds with high probability. Let FindDeepPoint$(\mathcal{S}, y)$ denote this algorithm.

3.1.2. Testing the water. It remains to find a good estimate of the maximum depth of $\mathcal{S}$. Assume that we have a guess k such that $\delta \leq 4k$. We construct procedure DepthTest that outputs either "guess too big" or a range $[y, 4y]$ containing δ with high probability.

The procedure DepthTest works by computing the sample R, according to Corollary 3.2, and applying the depth thresholding to R; formally, we execute DepthThreshold$(R, \alpha_+(\varepsilon, k))$. Consider the random variable $X = \operatorname{depth}(R)$.

If $X \geq \alpha_+(\varepsilon, k)$ (i.e., DepthThreshold returned "depth exceeds threshold"), then it follows that $\delta \geq (3/2)k$ by Corollary 3.2(i). In addition, since we assumed that $\delta \leq 4k$, we have that $(3/2)k \leq \delta \leq 4k$, and DepthTest returns $[k, 4k]$ as the range containing δ.

If $X \leq \alpha_-(\varepsilon, k)$, then by Corollary 3.2(ii), with high probability, $\delta \leq k$. Namely, our guess k of the depth is too large, and DepthTest outputs "guess too high."

If $\alpha_-(\varepsilon, k) < X < \alpha_+(\varepsilon, k)$, then, by Lemma 3.1(iii), with high probability, we have that $(1 - \varepsilon)k \leq \delta \leq 2k(1 + \varepsilon)$, so DepthTest returns $[k/2, 4k]$ as the range containing δ.

The running time of DepthTest is dominated by the running time of DepthThreshold.

3.1.3. The algorithm. The procedure DepthTest can tell us if the guess k for the maximum depth is in the right range or, alternatively, that the guess is too large. We use it to perform an exponential decreasing search for the right range, as follows.

Theorem 3.3. *Let $\mathcal{S}$ be a set of n objects in $\mathbb{R}^d$ of depth δ. Suppose there exists an algorithm implementing depth thresholding for depth at most k in a set of m objects in time $T_{\text{DT}}(m,k)$. Then, given a prespecified parameter ε, $0 < \varepsilon < 1/2$, one can compute a point of depth at least $(1-\varepsilon)\delta$ in $O(n + T_{\text{DT}}(r',\delta')\log n)$ expected time, where $\delta' = \min(c_2\varepsilon^{-2}\log n, n)$ and $r' = \min(n, \delta' n/\delta)$. In fact, the running time is $O(n + T_{\text{DT}}(r',\delta'))$ if $T_{\text{DT}}(r',k)$ is $\Omega(r')$ for any k. The output is correct with high probability.*

Proof. Let $k_i = n/2^i$ for $i = 1, \ldots, \lceil \lg n \rceil$. In the ith iteration, we estimate whether $\delta \leq k_i$ using DepthTest. This requires $O(n + T_{\text{DT}}(r_i, \delta_i))$ time, where δ_i and r_i are the depth and the size of the ith random sample, respectively. If k_i is too big according to DepthTest, then we continue to the next iteration.

Otherwise, we know that $k_i \leq \delta \leq 4k_i$. We now use FindDeepPoint$(\mathcal{S}, k_i)$, and this returns the required point and its depth; see section 3.1.1. Note that as for a running time, this invocation of FindDeepPoint takes (asymptotically) the same time as the last call to DepthTest (as both procedures essentially "just" call DepthThreshold).

Since at the last call to FindDeepPoint we have $k_i \leq \delta \leq 4k_i$, it follows that the expected depth of the random sample used is $O(\varepsilon^{-2}\log n)$. Note, that the sizes of the samples used in DepthTest form an increasing geometric sequence. Thus, the last iteration uses the largest sample. Furthermore, the expected depth of the sample increases with each sample. Therefore, using the Chernoff inequality again (we omit the straightforward but tedious details), we deduce that, with high probability, $\delta_i = O(\varepsilon^{-2}\log n)$ for $i \geq 1$.

Since we use $O(\log n)$ calls to DepthTest, the claim follows. As for the improved running time, observe that in those $O(\log n)$ calls, as mentioned above, we use samples of geometrically increasing sizes. Thus the overall running time is dominated by the cost of FindDeepPoint, which results in a call to the depth thresholding algorithm. With high probability, the sample size in the last iteration is $O(c_2(n/\delta)\varepsilon^{-2}\log n)$, as claimed.

To achieve further speedup, observe that we can compute all the random samples in advance. Let $R_1 := \mathcal{S}$, and let R_i be the sample computed from R_{i-1}, by picking each element with probability $1/2$ (this is known as *gradation*). Clearly, this takes $O(n)$ time, with high probability, and we can use the appropriate random sample when needed. Thus the overall time to compute the random samples is linear. (Note that, although the samples are no longer independent, the analysis still works since we have used the union bound to bound the probability of failure.) ∎

In the above algorithm the call to FindDeepPoint can be carried out using the results of the last DepthTest call issued by the algorithm. Therefore, the procedure FindDeepPoint can be merged with DepthTest to create a simpler algorithm. We believe, however, that the current presentation is more intuitive (and we hope that the reader will agree).

3.2. Finding a point of minimum depth. For a set of objects $\mathcal{S}$ in $\mathbb{R}^d$, let $\text{depth}_{\min}(\mathcal{S})$ denote the depth of a point in $\mathbb{R}^d$ covered by the fewest objects in $\mathcal{S}$. An algorithm computes a *minimum depth thresholding* for a set X of objects if it returns a point of minimum depth at most k in $\mathcal{A}(X)$, in time $T_{\text{minDT}}(|X|,k)$, or alternatively

reports that all points in $\mathcal{A}(X)$ are of depth larger than k.

It is easy to verify that the above discussion can be adopted to this "complementary" setting.

THEOREM 3.4. *Let $\mathcal{S}$ be a set of n objects in $\mathbb{R}^d$ of minimum depth $\delta = \mathrm{depth}_{\min}(\mathcal{S})$. Suppose there exists an algorithm implementing depth thresholding for depth at most k in a set of m objects in time $T_{\mathrm{minDT}}(m,k)$. Then, given a prespecified parameter ε, $0 < \varepsilon < 1/2$, one can compute a point of depth at most $(1+\varepsilon)\delta$ in $O(n+T_{\mathrm{minDT}}(r',\delta')\log n)$ expected time, where $\delta' = \min(c_2\varepsilon^{-2}\log n, n)$ and $r' = \min(n, \delta' n/\delta)$. In fact, the running time is $O(n + T_{\mathrm{minDT}}(r',\delta'))$ if $T_{\mathrm{minDT}}(r',k)$ is $\Omega(r')$ for any k. The output is correct with high probability.*

In fact, if we are given a target depth k, there is no need to perform the binary search for the right depth, and we obtain the following theorem.

THEOREM 3.5. *Let $\mathcal{S}$ be a set of n objects in $\mathbb{R}^d$. Let $f(\cdot)$ be a function defined over $\mathbb{R}^d$. Let* `Alg` *be an algorithm that, given $X \subseteq \mathcal{S}$, and a parameter t, can report the point $x \in \mathbb{R}^d$ such that $f(x)$ is the minimum among all points contained in t or fewer regions of X (if no such point exists,* `Alg` *outputs "infeasible"). Suppose that the running time of* `Alg` *is $T_{\mathrm{minDT}}(|X|,t)$. Then, given prespecified parameters $\varepsilon > 0$ and k, one can compute a point $u \in \mathbb{R}^d$ of depth at most $(1+\varepsilon)k$ in $\mathcal{S}$, in $O(n+T_{\mathrm{minDT}}(r',\delta'))$ expected time, where $\delta' = \min(c_2\varepsilon^{-2}\log n, n)$ and $r' = \min(n, \delta' n/k)$. Furthermore, $f(u) \le f(v_k)$, where v_k is a point minimizing f among all points in $\mathbb{R}^d$ of depth k or less in $\mathcal{A}(\mathcal{S})$s. The output is correct with high probability.*

4. Applications.

4.1. Linear programming with violations. Given a linear program L with n constraints in $\mathbb{R}^d$, one can determine whether there exists a point that violates at most k constraints of L in $O(nk^{d+1})$ time [Mat95]. This problem can be solved in $O((n+k^2)\log n)$ time in two dimensions and in $O(n+k^{11/4}n^{1/4})$ time in three dimensions [Cha05]. Using these algorithms to implement depth thresholding in Theorems 3.4 and 3.5 results in the following.

THEOREM 4.1. *Let L be a linear program with n constraints in $\mathbb{R}^d$, and let f be the objective function to be minimized. Let k_{opt} be the minimum number of constraints that must be violated to make L feasible, and let v be the point minimizing $f(v)$ with k_{opt} constraints violated. Then one can output a point $u \in \mathbb{R}^d$ such that u violates at most $(1+\varepsilon)k_{\mathrm{opt}}$ constraints of L, and $f(u) \le f(v)$. The results returned are correct with high probability. The expected running time (which also holds with high probability) of this algorithm is*

- *$O(n\log(\varepsilon^{-1}\log n) + (\varepsilon^{-1}\log n)^{O(1)})$ for $d = 2, 3$ and*
- *$O(n(\varepsilon^{-2}\log n)^{d+1})$ for $d > 3$.*

Remark 4.2. For the algorithm of Theorem 4.1, if k is specified in advance, one can find a point u violating at most $(1+\varepsilon)k$ constraints of L, with $f(u) \le f(v_k)$, where v_k is the optimal solution violating at most k constraints. If L is not feasible when violating only k constraints, the algorithm may output "infeasible."

Note that this observation implies a number of new results; see the introduction of Chan [Cha05] for a list of problems that can be solved approximately using our techniques. For example, in $\mathbb{R}^d$, given n points, an integer k, $0 < k < n$, and $\varepsilon > 0$, one can find a spherical shell containing all but $(1+\varepsilon)k$ points which has a volume smaller than the minimum-volume spherical shell containing all but k points. The resulting running time is $O(n(\varepsilon^{-1}\log n)^{O(d)})$.

Remark 4.3. Observe that if k_{opt} is sufficiently large in Theorem 4.1, then the running time is in fact linear, as the samples used by the algorithm are sufficiently

small. In particular, for $k_{\mathrm{opt}} = \Omega((\varepsilon^{-2}\log n)^{d+2})$, the running time of the algorithm of Theorem 4.1 is $O(n)$.

Remark 4.4 (handling outliers). Using Theorem 3.5 and plugging it into the techniques of Har-Peled and Wang [HW04], one can solve shape fitting problems with outliers in near-linear time. For example, given a set P of n points in the plane and parameters $k, \varepsilon, \delta > 0$, one can compute an annulus A, in expected $O(n+(\varepsilon^{-1}\delta^{-1}\log n)^{O(1)})$ time. With high probability, the annulus A contains all but at most $(1+\varepsilon)k$ points of P and its width is at most $(1+\delta)w_{\mathrm{opt}}(P,k)$, where $w_{\mathrm{opt}}(P,k)$ is the minimum width of an annulus containing all but k points of P. Note that the approximation here is in terms of both the width of the annulus and the number of outliers. This result was known before only for the case when the number of outliers was large (i.e., close to n). Our methods can be applied to all the problems studied by Har-Peled and Wang [HW04]. To prevent this paper from deteriorating into a shopping list, we omit any further details.

4.2. Approximating the deepest point in a pseudodisk arrangement.

LEMMA 4.5. *Determining a point of maximum depth in an arrangement of n disks is* 3SUM*-hard.*

Proof. It is 3SUM-hard to decide, given a set of lines in the plane with integer coefficients, whether any three of the lines have a point in common [GO95]. Consider such a set L of n lines. We assume they are given by equations of the form $ax+by=c$, with $a, b, c \in \mathbb{Z}$.

One can compute in $O(n\log n)$ time an axis-parallel square Q containing all vertices of the arrangement $\mathcal{A}(L)$. Indeed, let M be the coefficient with the largest absolute value appearing in the n given lines. For two lines $ax+by=c$ and $a'x+b'y=c'$, the coordinates of their intersection point is, by Cramer's rule, a ratio of two 2×2 determinants with entries from among these six numbers. So the x coordinate of such an intersection point is in $[-2M^2, 2M^2]$. Arguing similarly for y, we conclude that the square $Q = \left[-2M^2, 2M^2\right]^2$ contains all the intersection points of the lines (and it can be computed in $O(n)$ time). Alternatively, one can compute the leftmost, rightmost, topmost, and bottommost vertices in the arrangement of the lines L in $O(n\log n)$ time [Mat91].

Furthermore, one can compute a lower bound Δ on the distance between a vertex of $\mathcal{A}(L)$ and a line of L not passing through it. Indeed, consider a line $\ell\colon ax+by=c$, where $|a|, |b|, |c| \le M$, and a point $p=(\alpha,\beta)$, which is formed by the intersection of two given lines. The distance of p from ℓ is

$$x = \frac{a\alpha + b\beta - c}{\sqrt{a^2+b^2}}.$$

Now, $a\alpha+b\beta-c$ is a rational number with denominator smaller than $4M^4$, since the denominator of α and β are smaller than $2M^2$ by the above argument, and a, b, c are integers. Thus, if the numerator of x is nonzero, its absolute value is at least $1/(4M^4)$. Therefore, if x is not zero, then

$$|x| = \frac{|a\alpha+b\beta-c|}{\sqrt{a^2+b^2}} \ge \frac{1/4M^4}{\sqrt{2M^2}} > \frac{1}{8M^5} =: \Delta.$$

Next, we replace each line $\ell \in L$ by two sufficiently large disks D_ℓ, D'_ℓ of equal radii such that $\partial D_\ell \cap \partial D'_\ell = \ell \cap \partial Q$ (so that $D_\ell \cap D'_\ell$ is symmetric with respect to ℓ) and $D_\ell \cap D'_\ell$ lies in the strip of half-width $\Delta/4$ centered at ℓ. Namely, we replace the

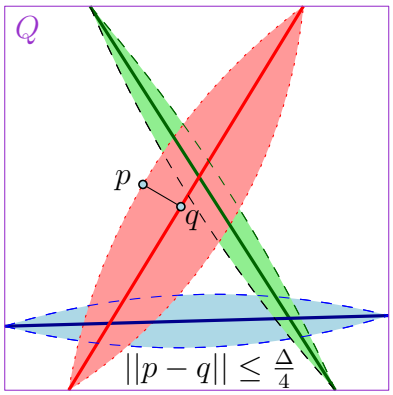


FIG. 2.

segment $\ell \cap Q$ by a lens formed by the intersection of two "large" disks. The union of the two disks that form the lens contains the square Q. By construction, three original lines intersect if and only if the three corresponding lenses share a point. See Figure 2.

Let $\mathcal{D}$ be the resulting set of $2n$ disks. Note that no point outside Q is covered by more than n of the disks. Moreover, there is a point in Q contained in at least $n+3$ disks of $\mathcal{D}$ if and only if some three lines of L share a point. The reduction takes linear time. ∎

Having argued that computing the depth exactly appears difficult, we turn to approximating it. Consider a family $\mathcal{S}$ of n *x-monotone pseudodisks*, i.e., a collection of regions, each bounded by a connected closed curve of constant algebraic complexity, such that any vertical line intersects a region in a connected interval, if at all, and such that the boundaries of any two regions cross at most twice. In the following, we assume that we can perform the geometric primitives on pseudodisks in constant time per operation. We need the following facts:

(i) The complexity of the union $U := \bigcup \mathcal{S}$ of n x-monotone pseudodisks is $O(n)$ [KLPS86].

(ii) Using a randomized incremental construction, one can compute U in $O(n \log n)$ expected time [MMP+94]. Furthermore, in the same amount of time one can preprocess U for $O(\log n)$-time point-location queries.

(iii) For a set $\mathcal{S}$ of n pseudodisks, the total complexity of faces of depth at most k in their arrangement $\mathcal{A} = \mathcal{A}(\mathcal{S})$ is $O(nk)$ [CS89, Sha03, Sha91].

(iv) Using a randomized incremental construction, one can compute all faces at depth at most k in $\mathcal{A}$ in time $O(nk + n \log n)$ [BY98]. (In fact, if we are interested in "depth thresholding" rather than construction of a subset of $\mathcal{A}$, this can be done with a standard plane sweep in $O(nk \log n)$ deterministic time [dBvKOS00].)

THEOREM 4.6. *Given a set $\mathcal{S}$ of n pseudodisks, one can compute a point q of depth at least $(1-\varepsilon)\,\mathrm{depth}(\mathcal{S})$ in $O(n\varepsilon^{-2}\log n)$ expected time. The output is correct with high probability.*

Proof. Given any set X of m pseudodisks, we can compute the deepest point in $\mathcal{A}(X)$, in $T(m,\delta) = O(m\delta + m\log m)$ time, using the algorithm of fact (iv) above, where $\delta := \mathrm{depth}(X)$. The theorem now follows from Theorem 3.3. ∎

4.3. Finding a ball with the maximum number of red points. We consider the following two problems.

Problem 4.7. Given two sets of red and blue points in $\mathbb{R}^d$, denoted by R and B, respectively, with a total of n points, find a ball that contains the maximum number of red points while not containing any blue point in its interior.

Problem 4.8. Given two sets of red and blue hyperplanes in $\mathbb{R}^d$, denoted by R and B, respectively, of total size n, find a point lying below (or on) all blue hyperplanes and below (or on) the maximum number of red hyperplanes.

The second problem appears more abstract and less natural. However, it is more general, as the former reduces to the latter, albeit in one higher dimension. Indeed, if one applies the standard "lifting transformation" [Ede87], which maps balls in $\mathbb{R}^d$ to hyperplanes in $\mathbb{R}^{d+1}$ and points in $\mathbb{R}^d$ to points on the standard paraboloid in $\mathbb{R}^{d+1}$ in such a manner that a point lies within a ball if and only if the corresponding lifted point lies below the corresponding hyperplane, an instance of Problem 4.7 is transformed into that of Problem 4.8. More precisely, we lifted the two points sets by one dimension, such that now we look for a hyperplane lying below all the blue points and that has as many red points below it as possible. Now, using point/hyperplane duality that preserves the above/below relationship between points and hyperplanes, we get Problem 4.8.

We now explain how to solve Problem 4.8 efficiently for $d = 3$ (which corresponds to solving Problem 4.7 for disks in the plane), both exactly and approximately, and later discuss higher-dimensional extensions.

THEOREM 4.9. *Given sets R and B of red and blue planes in $\mathbb{R}^3$, respectively, with a total of n planes, one can compute a point that lies below (or on) all blue planes and above (or on) the maximum number k_{opt} of red planes in time $O(n^2 \log n)$.*

Given in addition a parameter $\varepsilon > 0$, one can compute a point with the property that it lies below (or on) all blue planes and above (or on) $(1-\varepsilon)k_{\text{opt}}$ red planes, in expected time $O(n\varepsilon^{-2} \log^2 n)$.

Proof. We reduce the problem to that of depth in the pseudodisk arrangement. Let $\mathcal{B}$ be the (convex) set of points lying below (or on) all of the blue planes. Without loss of generality, we can restrict our attention to points on $\partial\mathcal{B}$. A point $p \in \partial\mathcal{B}$ lies on or above a plane $\pi \in R$ if and only if its projection p' to the xy-plane lies in the projection of $\pi \cap \mathcal{B}$, which is a convex set in the plane; we denote it by C_π.

We claim that $\{C_\pi \mid \pi \in R\}$ is a family of (convex, and therefore x-monotone) pseudodisks. Indeed, the intersection points of ∂C_π and $\partial C_{\pi'}$ are the projections of the points in $\pi \cap \pi' \cap \partial\mathcal{B}$, which is the intersection of a convex surface with a straight line.

At this point, we observe that $\mathcal{B}$ can be computed in time $O(n \log n)$ time (for example, by computing the convex hull of the dual points; see [Ede87]) and preprocessed into the Dobkin–Kirkpatrick hierarchy [DK90], so that C_π can be manipulated implicitly. More specifically, it is easily checked that the following operations can be performed on the resulting pseudodisks in logarithmic time: computation of boundary intersection points of two pseudodisks, computation of x-extreme points of a pseudodisk, computation of points of intersection of a pseudodisk with a y-vertical line. Armed with these operations, we can compute the entire arrangement and thus its depth in time $O(n^2 \log n)$ as in fact (ii) above. Similarly, Theorem 3.3 together with the implicit representation yields the desired approximation algorithm. □

We now turn to the higher-dimensional version of the problem. Recall that we aim to compute a point above a maximum number of red hyperplanes, while lying below all blue hyperplanes. Check if such a point of depth k exists can be performed

as follows: Compute the $(\leq k)$-levels of the arrangement $\mathcal{A}(R \cup B)$ [AES99, Mul91]. Traverse the 1-skeleton of the resulting structure, clipping away the portions of those levels that lie above the lower envelope of B and record the highest-depth unclipped vertex. This takes time proportional to the time spent on computing the $(\leq k)$-levels of $\mathcal{A}(R \cup B)$, which is $T(n,k) = O\big(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil}\big)$ [AES99, Mul91]. Using this observation to implement depth thresholding in the algorithm of Theorem 4.6, we obtain the following theorem.

THEOREM 4.10. *Given sets R and B of red and blue hyperplanes in $\mathbb{R}^d$, for $d \geq 4$, $|R| + |B| = n$, and a parameter $\varepsilon > 0$, one can compute a point that lies below (or on) all blue hyperplanes and above (or on) at least $(1-\varepsilon)k_{\mathrm{opt}}$ hyperplanes, where k_{opt} is the maximum number achievable. The running time of the algorithm is $O(n^{\lfloor d/2 \rfloor}(\varepsilon^{-2}\log n)^{\lceil d/2 \rceil})$ with high probability.*

COROLLARY 4.11. *Given sets R and B of red and blue points in $\mathbb{R}^d$ for $d \geq 3$, $|R| + |B| = n$, and a parameter $\varepsilon > 0$, one can compute a ball $\mathcal{B}$ that contains $(1-\varepsilon)k_{\mathrm{opt}}$ red points, while avoiding all blue points, where k_{opt} is the maximum number of red points that can be covered by such a ball. The running time of the algorithm is $O(n^{\lceil d/2 \rceil}(\varepsilon^{-2}\log n)^{\lceil (d+1)/2 \rceil})$.*

Can the algorithm running time in Theorem 4.10 be further improved? We leave this as an open problem for further research.

5. From emptiness to approximate range counting. Assume that there exists an *emptiness testing* data structure that can be constructed in $T(n)$ time for a set $\mathcal{S}$ of n objects such that, given a query range τ, we can check in $Q(n)$ time whether τ intersects any of the objects in $\mathcal{S}$. Let $S(n)$ be the space required to store this data structure. In this section, we show how to build a data structure that quickly returns an approximate number of objects in $\mathcal{S}$ intersecting τ using emptiness testing as a subroutine.

In particular, let μ_τ denote the number of objects of $\mathcal{S}$ intersected by τ. Below we use $\varepsilon > 0$ to denote the required approximation quality; namely, we would like the data structure to output a number α_τ such that $(1-\varepsilon)\mu_\tau \leq \alpha_\tau \leq \mu_\tau$.

5.1. The decision procedure. Given parameters $z \in [1,n]$ and ε, with $1/2 > \varepsilon > 0$, we construct a data structure, such that, for any δ, with $1/2 > \delta \geq \varepsilon$, and a query range τ, we can decide, with high probability, whether $\mu_\tau < z$ or $\mu_\tau \geq z$. The twist that makes it efficiently solvable is that the data structure is allowed to make a mistake if $\mu_\tau \in [(1-\delta)z, (1+\delta)z]$.

The data structure. Let $\mathcal{R}_1, \ldots, \mathcal{R}_M$ be M independent random samples of $\mathcal{S}$, formed by picking every element with probability $1/z$, where $M := N(\varepsilon)$,

$$N(\varepsilon) := \left\lceil c_3 \varepsilon^{-2} \log n \right\rceil,$$

and c_3 is a sufficiently large absolute constant. Build M separate emptiness-query data structures $D_1, \ldots, D_M$ for the sets $\mathcal{R}_1, \ldots, \mathcal{R}_M$, respectively, and put $\mathcal{D} = \mathcal{D}(z,\varepsilon) := \{D_1, \ldots, D_M\}$.

Answering a query. Consider a query range τ and approximation quality δ, $1/2 > \delta \geq \varepsilon$, and let $X_i = 1$ if τ intersects any of the objects of R_i and $X_i = 0$ otherwise for $i = 1, \ldots, N = N(\delta)$; note that we use N rather than all M of the samples R_i in answering the query. The value of X_i can be determined using a single emptiness query in D_i. Compute $Y_\tau := \sum_{1 \leq i \leq N} X_i$.

For a range σ of depth k, the probability that σ does not avoid all the elements

of $\mathcal{R}_i$ is

$$\rho(k) := 1 - \left(1 - \frac{1}{z}\right)^k.$$

If a range σ has depth z, $\mathbf{E}[Y_\sigma] = N\rho(z) =: \Delta$. Our data structure returns "depth$(\tau, \mathcal{S})$ $< z$" if $Y_\tau < \Delta$, and "depth$(\tau, \mathcal{S}) \geq z$" otherwise.

Correctness. In the following, we show that with high probability the data structure indeed returns the correct answer if the depth of the query range is outside the "uncertainty" range $[(1-\delta)z, (1+\delta)z]$. For simplicity of exposition, we assume in the following that $z \geq 10$ (the case $z < 10$ follows by similar arguments). Consider a range τ of depth at most $(1-\delta)z$. The data structure returns a wrong answer if $Y_\tau > \Delta$. We will show that the probability of this event is polynomially small. The other case, where τ has depth at least $(1+\delta)z$ but $Y_\tau < \Delta$, is handled in a similar fashion.

LEMMA 5.1. *The probability* $\mathbf{Pr}[Y_\tau > \Delta \mid \text{depth}(\tau, \mathcal{S}) \leq (1-\delta)z]$ *does not exceed* n^{-c_6}*, where* $c_6 = c_6(c_3) > 0$ *depends only on* c_3 *and can be made arbitrarily large by a choice of a sufficiently large* $c_3 > 0$*.*

The proof of this lemma follows from the Chernoff inequality. It is tedious and not very insightful; thus it is delegated to Appendix A. This implies the following lemma.

LEMMA 5.2. *Given a set* $\mathcal{S}$ *of* n *objects, a parameter* $0 < \varepsilon < 1/2$*, and* $z \in [0, n]$*, one can construct a data structure* $\mathcal{D}(z)$ *which, given a range* τ *and a parameter* $1/2 > \delta \geq \varepsilon$*, returns either* LOW *or* HIGH*. If it returns* LOW*, then* $\mu_\tau \leq (1+\delta)z$*, and if it returns* HIGH*, then* $\mu_\tau \geq (1-\delta)z$*. The data structure might return either answer if* $\mu_\tau \in [(1-\delta)z, (1+\delta)z]$*.*

The data structure $\mathcal{D}$ *consists of* $M = O(\varepsilon^{-2} \log n)$ *emptiness data structures. The space and time needed are* $O(S(2n/z)\varepsilon^{-2} \log n)$ *and* $O(Q(2n/z)\delta^{-2} \log n)$*, where* $S(m)$ *and* $Q(m)$ *are the space needed for a single emptiness data structure storing* m *objects and the time needed for a single query in such a structure, respectively. All bounds hold with high probability.*

Proof. The lemma follows immediately from the above discussion. The only missing part is observing that, by the Chernoff inequality, $|\mathcal{R}_i| \leq 2n/z$, with high probability. □

The path to answering approximate counting query is now clear. We use Lemma 5.2 to perform binary search, repeatedly narrowing the range containing the answer. We stop when the size of the range is within our error tolerances. At the start of the process, this range is large, so we use large values of δ. As the range narrows, δ is reduced. (Recall that δ determines the query cost in Lemma 5.2 by controlling the number of samples whose range emptiness data structures are consulted to answer the query.)

One difficulty is that Lemma 5.2 works only for $\delta < 1/2$. We next strengthen it considerably for much larger values of δ. Intuitively, the number of "experiments" (i.e., emptiness queries) we need to perform is only $\Theta((\log n)/\log \delta)$ to answer a LOW/HIGH query and have a high probability guarantee.

LEMMA 5.3. *Given the data structure of Lemma* 5.2*,* z *a value to compare to, and* $\delta > c_5$*, one can decide for a query range* τ *if* $\mu_\tau < z/(1+\delta)$ *or* $\mu_\tau \geq z(1+\delta)$*. Here* c_5 *is a sufficiently large constant. The data structure is allowed to return any answer if* $\mu_\tau \in [z/(1+\delta), (1+\delta)z]$*. This requires* $N = \lceil c_6(\log n)/\ln \delta \rceil$ *emptiness queries, and the answer returned is correct with high probability, where* c_6 *is an appropriate absolute constant.*

The proof of this lemma is a tedious but careful application of the Chernoff inequality and is deferred to Appendix B.

Remark 5.4. Note, that there is a gap between the ranges of δ to which Lemmas 5.2 and 5.3 are applicable, namely, when δ is between $1/2$ and c_5. If we need to apply either lemma in such a situation, we use Lemma 5.2 with $\delta = 1/4$ a constant number of times.

5.2. The data structure. Lemmas 5.2 and 5.3 provide us with a tool for performing a "binary" search for the count value μ_τ of a range τ. For small values of i, we just build a separate data structure $\mathcal{D}_i = \mathcal{D}(v_i, \varepsilon_i)$ of Lemma 5.3 for depth values $v_i = i/2$ for $i = 1, \ldots, U = O(\varepsilon^{-1})$. For depth i, we use accuracy

$$\varepsilon_i = 1/8i \tag{3}$$

(i.e., this is the value of ε when using Lemma 5.2). Using these data structures, we can decide whether the query range count is at least U or smaller than U. If it is smaller than U, then we can perform a binary search to find its exact value. The result is correct with high probability.

Next, consider the values $v_j = (U/4)(1 + \varepsilon/16)^j$ for $j = U + 1, \ldots, W$, where $W := c \log_{1+\varepsilon/16} n = O(\varepsilon^{-1} \log n)$ for an appropriate choice of an absolute constant $c > 0$, so that $v_W = n$. We build a data structure $\mathcal{D}_j = \mathcal{D}(v_j, \varepsilon/16)$ for each $z = v_j$, using Lemma 5.2.

In the end of this process, we have built $\mathcal{D}_1, \ldots, \mathcal{D}_W$.

Answering a query. Given a range query τ, each data structure in our list returns LOW or HIGH. Moreover, with high probability, if we were to query all the data structures, we would get a sequence of HIGHs, followed by a sequence of LOWs. It is easy to verify that the value associated with the last data structure returning HIGH (rounded to the nearest integer) yields the required approximation. We can use binary search on $\mathcal{D}_1, \ldots, \mathcal{D}_W$ to locate this changeover value using a total of $O(\log W) = O(\log(\varepsilon^{-1} \log n))$ queries in the structures of $\mathcal{D}_1, \ldots, \mathcal{D}_W$. Namely, the overall query time is $O(Q(n)\varepsilon^{-2}(\log n) \log(\varepsilon^{-1} \log n))$.

Answering queries more efficiently. To speed up the query, we organize the search into $O(\log(n/\varepsilon))$ rounds. Treat $\mathcal{D}_1, \ldots, \mathcal{D}_W$ as a linked list L_M, where $M = \lceil \lg W \rceil = O(\log(\varepsilon^{-1} \log n))$. Next, we build a data structure (which is somewhat similar to a skip-list), where L_{i-1} is formed from L_i by picking every other element of L_i, such that the base list L_1 has, say, 4 to 8 elements.

The search now continues in a top-down fashion starting from the list L_1. At the ith stage, we maintain pointers to four consecutive data structures in L_i, such that the left two return HIGH and the right two return LOW. Thus the exact answer μ_τ must lie between the depth associated with the first and the fourth of these data structures in L_i. The corresponding portion of L_{i+1} delimited by these two data structures in L_i is a sublist of (at most) seven data structures in L_{i+1}. We now query the at most three new data structures in the list (i.e., the ones we have not queried on previous rounds) using Lemmas 5.2 and 5.3 to determine the sublist of four consecutive data structures whose range contains the query depth. The key observation, leading to a more efficient query time, is that we answer the queries, on the ith level, using an error parameter δ_i which is as large as possible, such that the error intervals of all these data structures are disjoint.

We continue in this process until we reach the bottom level. Clearly, we have the required approximation and we return the value associated with, say, the leftmost data structure in this sublist as the approximation.

We can set $\delta_1 := n^{1/4}$ and $(1+\delta_i/c)^2 = 1+\delta_{i-1}/c$, where $\delta_{i-1} > 1$ and c is some constant. Therefore, $\delta_i = O(\sqrt{\delta_{i-1}})$ as long as, say, $\delta_i > c^2$. To bound the number of emptiness queries used in this range, where the error is large, we use Lemma 5.3, implying that the number of emptiness queries used is

$$\sum_{i,\delta_i>c^2} O\left(\frac{\log n}{\log \delta_i}\right) = \sum_{i=O(1)}^{\lg\lg n} O\left(\frac{\log n}{\log\left(2^{2^{\lg\lg n - i}}\right)}\right) = \sum_{i=O(1)}^{\lg\lg n} O\left(\frac{\log n}{2^{\lg\lg n - i}}\right) = O(\log n).$$

Next, we bound the number of emptiness queries used when $\delta < 1/2$. In the jth level of this more refined search, we use $\delta'_j = 1/2^j$. Thus, the number of emptiness queries is

$$\sum_{j=2}^{\lg(1/\varepsilon)} O\left(\frac{\log n}{\delta'^2_j}\right) = \sum_{j=2}^{\lg(1/\varepsilon)} O\left(2^{2j}\log n\right) = O(\varepsilon^{-2}\log n),$$

by Lemma 5.2. The number of queries with δ between $1/2$ and c_5 is constant and therefore does not affect our asymptotic analysis; see Remark 5.4.

LEMMA 5.5. *Given a set $\mathcal{S}$ of n objects and a data structure that answers emptiness queries in $Q(n)$ time and can be built in $T(n)$ time, one can construct in $O(T(n)\varepsilon^{-3}\log^2 n)$ time a data structure that, given a range τ, outputs a number α_τ, with $(1-\varepsilon)\mu_\tau \le \alpha_\tau \le \mu_\tau$, where μ_τ is the number of objects in $\mathcal{S}$ intersecting τ. The query time is*

$$O\left(\varepsilon^{-2}Q(n)\log n\right).$$

The output of the query is correct with high probability for all queries, and the running time bounds hold with high probability.

(In the above lemma, we assumed the number of queries is polynomial.)

Space and time requirements. Assume that the emptiness data structure uses $S(n)$ space to store n elements, and it can be constructed in $T(n)$ preprocessing time.

We need a technical definition: If a data structure D can be constructed for n elements using $S(n)$ space in $T(n)$ time, such that $S(n/i) = O(S(n)/i^\lambda)$ and $T(n/i) = O(T(n)/i^\lambda)$, for all n and $1 \le i \le n$, for a constant $\lambda \ge 1$, then the data structure has *degree* λ. (The O notation might hide constants that depend on λ.)

The space requirements to implement the data structure of Lemma 5.5 for the low range $[1, O(1/\varepsilon)]$ can be reduced by using precision $\delta := \varepsilon_i = 1/8i$ (see (3)) when considering numbers in the interval $I_i = [i/4, (i+1)/4]$, which contains only a single integer. Therefore, the data structure returns the exact answer in this case. The space needed is

$$\begin{aligned} S_1 &= \sum_{i=1}^{O(1/\varepsilon)} O\left(i^2 S(n/8i)\log n\right) = \sum_{i=1}^{O(1/\varepsilon)} O\left(i^2\frac{S(n)}{i^\lambda}\log n\right) \\ &= O\left(S(n)\log n \sum_{i=1}^{O(1/\varepsilon)} \frac{1}{i^{\lambda-2}}\right) = O(H_{\lambda-2}(1/\varepsilon)S(n)\log n), \end{aligned}$$

where $H_d(k) = \sum_{i=1}^{k} 1/i^d$. For the intervals $I_{U+1}, \ldots, W$, the space is at most proportional to

$$\sum_{i=1}^{W-U} \frac{S(n/(\varepsilon^{-1}(1+\varepsilon/16)^i))}{\varepsilon^2}\log n \le \sum_{i=1}^{W-U} \frac{(\varepsilon/(1+\varepsilon/16)^i)^\lambda}{\varepsilon^2} O(S(n)\log n)$$

$$\leq \sum_{i=1}^{W-U} \frac{O(S(n)\log n)}{\varepsilon^{2-\lambda}(1+\varepsilon/16)^{i\lambda}}$$
$$\leq \frac{O(S(n)\log n)}{\varepsilon^{2-\lambda}} \sum_{i=1}^{W-U} \frac{1}{(1+\varepsilon/16)^{i\lambda}}$$
$$\leq \frac{O(S(n)\log n)}{\varepsilon^{2-\lambda}} \sum_{i=1}^{\infty} \frac{1}{(1+\varepsilon)^{i}} = \frac{O(S(n)\log n)}{\varepsilon^{3-\lambda}}.$$

Thus, the overall space required is $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(1/\varepsilon))S(n)\log n)$. A similar bound holds on the preprocessing time.

The result. Putting everything together, we have the following result.

THEOREM 5.6. *We are given a set of $\mathcal{S}$ of n objects. Assume that one can construct, in $T(n)$ time, a data structure of degree λ that answers emptiness queries in $Q(n)$ time using $S(n)$ space. Then one can construct a data structure in $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(1/\varepsilon))T(n)\log n)$ time, using $O((\varepsilon^{\lambda-3} + H_{\lambda-2}(1/\varepsilon))S(n)\log n)$ space, such that, given a range τ, it outputs a number α_τ, such that $(1-\varepsilon)\mu_\tau \leq \alpha_\tau \leq \mu_\tau$, and the query time is $O\big(\varepsilon^{-2}Q(n)\log n\big)$, where $H_d(k) := \sum_{i=1}^{k} 1/i^d$ and μ_τ is the exact answer to the counting query. The result returned is correct with high probability for all queries.*

(We assumed in the above theorem that the number of queries is polynomial.)

The number of emptiness queries used by Theorem 5.6 is probably tight. Indeed, just deciding if the query depth lies in the range $[z/(1+\varepsilon),\, z(1+\varepsilon)]$ requires $O(\varepsilon^{-2}\log n)$ emptiness queries by the Chernoff inequality. Since the Chernoff inequality is tight, it seems unlikely that the number of emptiness queries can be improved. Notice that this argument applies only to the case where emptiness queries are treated as "black box" operations. If this assumption is removed, certain improvements are possible [AHS07].

5.3. Applications.

5.3.1. Halfplane and halfspace range counting. Using the data structure of Dobkin and Kirkpatrick [DK85], one can answer emptiness halfspace range searching queries in logarithmic time when $d = 2, 3$. In this case, we have $S(n) = O(n)$, $T(n) = O(n\log n)$, $Q(n) = O(\log n)$, and $\lambda = 1$.

COROLLARY 5.7. *Given a set P of n points in two (resp., three) dimensions and a parameter $\varepsilon > 0$, one can construct in expected $O(n\varepsilon^{-2}\log^2 n)$ time a data structure of size $O(n\varepsilon^{-2}\log n)$, such that, given a halfplane (resp., halfspace) τ, it outputs a number α, such that $(1-\varepsilon)\,|\tau \cap P| \leq \alpha \leq |\tau \cap P|$, and the query time is $O\big(\varepsilon^{-2}\log^2 n\big)$. The result returned is correct with high probability for all queries.*

Using the standard lifting of points in $\mathbb{R}^2$ to the paraboloid in $\mathbb{R}^3$ implies a similar result for approximate range counting for disks, as a disk range query in the plane reduces to a halfspace range query in three dimensions.

COROLLARY 5.8. *Given a set of P of n points in two dimensions and a parameter ε, one can construct a data structure in $O(n\varepsilon^{-2}\log^2 n)$ expected time, using $O(n\varepsilon^{-2}\log n)$ space, such that given a disk τ, it outputs a number α, such that $(1-\varepsilon)\,|\tau \cap P| \leq \alpha \leq |\tau \cap P|$, and the query time is $O\big(\varepsilon^{-2}\log^2 n\big)$. The result returned is correct with high probability for all possible queries.*

These algorithms are faster by a polynomial factor in n than previously known exact procedures for answering the corresponding questions.

Notice that our techniques also apply to the higher dimensions settings in an obvious way. Since the resulting bounds are somewhat less appealing, we refrain from

stating them explicitly.

5.3.2. Depth queries. By computing the union of a set of n pseudodisks in the plane and preprocessing the union for point-location queries, one can perform "emptiness" queries in logarithmic time. This is done by computing the union [EHS04] and then performing a point-location query in it [dBvKOS00]. (Again, we are assuming here that we can implement the geometric primitives on the pseudodisks in constant time.) The space needed is $O(n)$, and it takes $O(n \log n)$ time to construct the data structure. Thus, we get the following result.

COROLLARY 5.9. *A set of $\mathcal{S}$ of n pseudodisks in the plane can be preprocessed in expected $O(n\varepsilon^{-2} \log^2 n)$ time, using $O(n\varepsilon^{-2} \log n)$ space, such that given a query point q one can output a number α, such that $(1-\varepsilon)\operatorname{depth}(p, \mathcal{S}) \leq \alpha \leq \operatorname{depth}(p, \mathcal{S})$, and the query time is $O\bigl(\varepsilon^{-2} \log^2 n\bigr)$. The result returned is correct with high probability for all possible queries.*

6. Conclusions. In this paper, we have shown the connection between depth thresholding, emptiness, and approximate depth computation. Note that Theorem 5.6 is better than the result in the conference version of this paper [AH05].

There are several natural problems for further research:

- Computing a square containing the largest number of red points while avoiding the blue points. Of course, the same question can be asked for other families of shapes, such as rectangles, ellipses, translates, or homothets of a fixed shape.
- Proving a lower bound on the problem of computing the ball containing the largest number of red points while avoiding the blue points. In particular, is this problem 3SUM-hard in two dimensions?

We would also like to point out that the approximate depth computation in the plane does not require "pseudodiskness." Indeed, it is sufficient to have a family of objects with small complexity of the union, such as the one described by [Efr05]. An analogous statement holds in three dimensions, though there are few known classes of objects with small union complexity there for which the computation of the union can be performed efficiently.

6.1. Permutations, emptiness, and randomized incremental construction. In a followup to this work, Haim Kaplan and Micha Sharir [KS06] showed that one can use the "permutations technique" of Cohen [Coh97] to approximate the depth of a query range. The idea is to randomly permute the set of objects $\mathcal{S}$ and compute the first index in the permutation whose corresponding object in $\mathcal{S}$ meets the query range. If the index is i, the estimate of the depth is n/i. Repeating this process a sufficient number of times implies, by the Chernoff inequality, that the estimate is within the acceptable error range. To find the lowest index i where this happens, one can perform the query inside a history graph of a randomized incremental construction algorithm for computing the union of the objects. This yields polylogarithmic improvement over the proceedings version of this paper for some specific applications (but no improvement over the current version). See [KS06] for details.

We observe that up to polylogarithmic factors, the two techniques are equivalent. Indeed, it is sufficient to approximate the first index in the permutation that contains the query range up to a factor of $1+\varepsilon$, which can be reduced to a binary search over emptiness data structures.

The other direction is somewhat more interesting. Our data structure has $O(\varepsilon^{-1} \log n)$ data structures at different resolutions, each comprised of $O(\varepsilon^{-2} \log n)$

emptiness data structures built over random samples. If we were to use instead of independent random subsets of the input in our data structures a gradation (we remind the reader that a gradation is a sequence of random samples, where the ith sample is a random sample of the higher $(i-1)$th-level sample), then the sequence of queries performed along the emptiness data structures associated with this gradation is essentially equivalent to the permutation query. Thus, in some intuitive, rough, and very informal sense, the data structure of Kaplan and Sharir performs the "same" computations as our data structure. The "sameness" here should be taken with several sizeable grains of salt, as everything looks similar from afar. It is interesting and surprising that these two different approaches to the problem are so similar once one inspects the underlying machinery.

Appendix A. Proof of Lemma 5.1.

Proof. The probability $\alpha := \mathbf{Pr}[Y_\tau > \Delta \mid \operatorname{depth}(\tau, \mathcal{S}) \leq (1-\delta)z]$ is maximized when $\operatorname{depth}(\tau, \mathcal{S}) = (1-\delta)z$. Thus

$$\alpha \leq \beta := \mathbf{Pr}[Y_\tau > \Delta \mid \operatorname{depth}(\tau, \mathcal{S}) = (1-\delta)z].$$

We argue that this probability is polynomially small.

Observe that $\mathbf{Pr}[X_i = 1] = \rho((1-\delta)z)$, where $\rho(k) := 1-(1-1/z)^k$. Therefore, we have

$$\mathbf{E}[Y_\tau] = N\rho((1-\delta)z) = N \cdot \left(1 - \left(1 - \frac{1}{z}\right)^{(1-\delta)z}\right) \geq N \cdot \left(1 - e^{-(1-\delta)}\right) \geq \frac{N}{3}, \tag{4}$$

since $1 - 1/z \leq \exp(-1/z)$. The last inequality holds since $\delta \leq 1/2$. By definition, $\Delta = N\rho(z)$; therefore,

$$\begin{aligned}\xi := \frac{\Delta}{\mathbf{E}[Y_\tau]} &= \frac{1-\left(1-\frac{1}{z}\right)^z}{1-\left(1-\frac{1}{z}\right)^{(1-\delta)z}} = 1 + \frac{\left(1-\frac{1}{z}\right)^{(1-\delta)z} - \left(1-\frac{1}{z}\right)^z}{1-\left(1-\frac{1}{z}\right)^{(1-\delta)z}} \\ &= 1 + \left(1-\frac{1}{z}\right)^{(1-\delta)z} \frac{1-\left(1-\frac{1}{z}\right)^{\delta z}}{1-\left(1-\frac{1}{z}\right)^{(1-\delta)z}}.\end{aligned}$$

Since $(1-1/z)/e \leq (1-1/z)^z \leq 1/e$ [MR95], $(1-1/z)^{(1-\delta)z} \geq ((1-1/z)/e)^{1-\delta} \geq (1-1/z)/e$. This implies that

$$\begin{aligned}\xi &\geq 1 + \frac{1}{e}\left(1-\frac{1}{z}\right)\frac{1-\exp(-\delta)}{1-\left(1-\frac{1}{z}\right)\frac{1}{e}} \\ &\geq 1 + \frac{1}{2}\cdot\frac{1}{e}\cdot\frac{1}{1/2}\cdot(1-\exp(-\delta)),\end{aligned}$$

since $(1-1/x)^x \leq 1/e$ (for $x \geq 1$) [MR95] and $z \geq 10$. Observe that $\exp(-\delta) \leq 1 - \delta + \delta^2/2$, using the Taylor expansion of $e^{-\delta}$ for $\delta > 0$. Therefore,

$$\xi \geq 1 + \frac{1}{e}\left(\delta - \delta^2/2\right) \geq 1 + \frac{\delta}{6}.$$

Deploying the Chernoff inequality, we have that if $\operatorname{depth}(\tau, \mathcal{S}) = (1-\delta)z$, then

$$\begin{aligned}\beta = \mathbf{Pr}[Y_\tau > \Delta] &\leq \mathbf{Pr}[Y_\tau > \xi\,\mathbf{E}[Y_\tau]\,] \leq \mathbf{Pr}[Y_\tau > (1+\delta/6)\,\mathbf{E}[Y_\tau]\,] \\ &\leq \exp\left(-\,\mathbf{E}[Y_\tau]\,(\delta/6)^2/4\right) \leq \exp\left(-\frac{N\delta^2}{3\cdot 36\cdot 4}\right) \leq \exp\left(-\frac{\delta^2\left\lceil c_3\delta^{-2}\log n\right\rceil}{432}\right) \leq n^{-c_3/432}\end{aligned}$$

by ((1), case (a)) and (4). ∎

Appendix B. Proof of Lemma 5.3.

Proof. Recall that $Y_\tau = \sum_{i=1}^{N} X_i$. Again, we can assume that $\operatorname{depth}(\tau, \mathcal{S}) = z/(1+\delta)$. We want to bound the probability that this "light" range would be considered to be "heavy" (i.e., $Y_r > N\rho(z)$). We have that

$$\mathbf{E}[Y_\tau] = N\left(1 - \left(1 - \frac{1}{z}\right)^{z/(1+\delta)}\right) \leq N\left(1 - \exp\left(-\frac{2z/(1+\delta)}{z}\right)\right) \leq N\frac{2}{1+\delta}, \tag{5}$$

since $1-1/z \geq \exp(-2/z)$ and $1-x \leq \exp(-x)$ for $0 \leq x \leq 1/2$. By similar reasoning,

$$\mathbf{E}[Y_\tau] \geq N\left(1 - \exp\left(-\frac{z/(1+\delta)}{z}\right)\right) \geq \frac{N}{2(1+\delta)}. \tag{6}$$

Observe that

$$\lim_{z\to\infty} \rho(z) = \lim_{n\to\infty}\left(1 - \left(1 - \frac{1}{z}\right)^{z}\right) = 1 - \frac{1}{e}.$$

Thus $\rho(z) > 1/2$ since z is sufficiently large (we remind the reader that we assumed that $z \geq 10$ for the sake of simplicity of exposition). Thus

$$\begin{aligned}\zeta = \mathbf{Pr}[Y_\tau > N\rho(z)] &\leq \mathbf{Pr}[Y_\tau > N/2] \leq \mathbf{Pr}\left[Y_\tau > \frac{1+\delta}{4}\,\mathbf{E}[Y_\tau]\right] \\ &\leq \mathbf{Pr}\left[Y_\tau > \left(1 + \frac{\delta}{5}\right)\mathbf{E}[Y_\tau]\right]\end{aligned}$$

by (5) and since δ is sufficiently large. Observe that since $\delta \geq c_5$, we have that

$$\frac{e^{\delta/5}}{(1+\delta/5)^{(1+\delta/5)/3}} \leq \frac{1}{e}.$$

Now, by the Chernoff inequality, we have

$$\zeta < \left(\frac{e^{\delta/5}}{(1+\delta/5)^{1+\delta/5}}\right)^{\mathbf{E}[Y_\tau]} \leq \left((1+\delta/5)^{-(1+\delta/5)/2}\right)^{\mathbf{E}[Y_\tau]},$$

since $\delta \geq c_5$. Observe that

$$\frac{(1+\delta/5)\,\mathbf{E}[Y_\tau]}{2} \geq \frac{(1+\delta/5)}{4(1+\delta)}N \geq \frac{1}{20}N$$

by (6). As $\ln(1+\delta/5) \geq (\ln\delta)/2$, we have

$$\zeta \leq (1+\delta/5)^{-N/20} \leq \exp\left(-\frac{N}{40}\ln\delta\right) < \frac{1}{n^{\Omega(1)}}$$

for c_6 sufficiently large, since $N = \lceil c_6(\log n)/\ln\delta \rceil$.

The other direction is slightly more challenging. Assume that $\operatorname{depth}(\tau, \mathcal{S}) = z(1+\delta)$, and let $Z_\tau = N - Y_\tau$. Observe that if $z/(1+\delta) \leq 1$, then any answer returned by the algorithm would be valid in this case (here we are trying to bound

the case that a heavy range is reported as being "light"). Therefore, we assume that $z \geq 1+\delta$. Observe that

$$\mathbf{E}[Z_\tau] = N\left(1-\frac{1}{z}\right)^{z(1+\delta)} \leq N\exp(-(1+\delta)).$$

(Specifically, $\mathbf{E}[Z_\tau] \leq N/10^6$ since $\delta \geq c_5$, and thus $\mathbf{E}[Y_\tau]$ is very close to N in this case.) Now, since z is large enough, we have that $\rho(z) < 1 - 1/10$ as $\rho(z) \approx 1 - 1/e$. Therefore,

$$\begin{aligned}\mathbf{Pr}\Big[Y_\tau < N\rho(z)\Big] &\leq \mathbf{Pr}\left[Y_\tau < \frac{9}{10}N\right] = \mathbf{Pr}\left[N - Z_\tau < \frac{9}{10}N\right] = \mathbf{Pr}\left[Z_\tau > \frac{N}{10}\right] \\ &\leq \mathbf{Pr}\left[Z_\tau > \frac{\mathbf{E}[Z_\tau]}{10\exp(-(1+\delta))}\right] \\ &= \mathbf{Pr}\Big[Z_\tau > (1+\xi)\,\mathbf{E}[Z_\tau]\Big],\end{aligned}$$

where $\xi := (1/10)\exp(1+\delta) - 1 > 10^8$. The variable Z_τ can be interpreted as the sum of N independent 0/1 variables (i.e., it is the complement of the variables forming Y_τ). By applying the Chernoff inequality to Z_τ, we obtain

$$\varrho := \mathbf{Pr}[Y_\tau < N\rho(z)] \leq \mathbf{Pr}\Big[Z_\tau > (1+\xi)\,\mathbf{E}[Z_\tau]\Big] \leq \left(\frac{e^\xi}{(1+\xi)^{(1+\xi)}}\right)^{\mathbf{E}[Z_\tau]}.$$

Now, we have $(1-1/z)^z \geq (1/e)(1-1/z) \geq \exp(-1-2/z) \geq \exp(-(1+2/\delta))$, since $z \geq 1+\delta \geq \delta$. As such,

$$\begin{aligned}\mathbf{E}[Z_\tau] = N\left(1-\frac{1}{z}\right)^{z(1+\delta)} &\geq N\exp(-(1+\delta)(1+2/\delta)) \\ &\geq \frac{N}{\exp(1+\delta+2/\delta+2)} \geq \frac{N}{c_7\xi},\end{aligned}$$

where $c_7 = 20e^3$. Thus, as $\ln(1+\xi) = \ln(1/10) + 1 + \delta \geq \ln\delta$, we have

$$\begin{aligned}\varrho &\leq \Big((1+\xi)^{-(1+\xi)/2}\Big)^{\mathbf{E}[Z_\tau]} \leq (1+\xi)^{-(1+\xi)N/c_7\xi} \leq \exp\left(-\frac{N\ln(1+\xi)}{c_7}\right) \\ &\leq \exp\left(-\frac{N}{c_7}\ln\delta\right) < \frac{1}{n^{\Omega(1)}},\end{aligned}$$

since $N = \lceil c_6(\log n)/\ln\delta \rceil$ and c_6 is sufficiently large. □

Acknowledgments. The authors would like to thank Jeff Erickson and Edgar Ramos for useful discussion on the problems studied in this paper. Haim Kaplan and Micha Sharir have pointed out that the space requirement of depth queries, as described in an earlier version of this paper [AH05] can be improved. We then also improved the query time in Theorem 5.6.

The authors would also like to thank the anonymous referees for their detailed and insightful comments that lead to what we hope are significant improvements in the presentation of this paper.

REFERENCES

[ABSS97] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331.

[AE99] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.

[AES99] P. K. Agarwal, A. Efrat, and M. Sharir, *Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications*, SIAM J. Comput., 29 (1999), pp. 912–953.

[AH05] B. Aronov and S. Har-Peled, *On approximating the depth and related problems*, in Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 886–894.

[AHS07] B. Aronov, S. Har-Peled, and M. Sharir, *On approximate halfspace range counting and relative epsilon-approximations*, in Proceedings of the 23rd Annual ACM Symposium on Computational Geometry, ACM, New York, 2007, pp. 327–336.

[AK95] E. Amaldi and V. Kann, *The complexity and approximability of finding maximum feasible subsystems of linear relations*, Theoret. Comput. Sci., 147 (1995), pp. 181–210.

[AM00] S. Arya and D. M. Mount, *Approximate range searching*, Comput. Geom., 17 (2000), pp. 135–152.

[BCP93] H. Brönnimann, B. Chazelle, and J. Pach, *How hard is halfspace range searching?*, Discrete Comput. Geom., 10 (1993), pp. 143–155.

[BY98] J.-D. Boissonnat and M. Yvinec, *Algorithmic Geometry*, Cambridge University Press, Cambridge, UK, 1998; also available online from http://www.inria.fr/prisme/personnel/yvinec/livre.html.

[Cha00] T. M. Chan, *Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions*, SIAM J. Comput., 30 (2000), pp. 561–575.

[Cha05] T. M. Chan, *Low-dimensional linear programming with violations*, SIAM J. Comput., 34 (2005), pp. 879–893; also available online from http://www.math.uwaterloo.ca/~tmchan/pub.html.

[Coh97] E. Cohen, *Size-estimation framework with applications to transitive closure and reachability*, J. Comput. System Sci., 55 (1997), pp. 441–453.

[CS89] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[dBvKOS00] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, New York, 2000; also available online from http://www.cs.uu.nl/geobook/.

[DHS01] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley-Interscience, New York, 2001.

[DK85] D. P. Dobkin and D. G. Kirkpatrick, *A linear algorithm for determining the separation of convex polyhedra*, J. Algorithms, 6 (1985), pp. 381–392.

[DK90] D. P. Dobkin and D. G. Kirkpatrick, *Determining the separation of preprocessed polyhedra—a unified approach*, in Proceedings of the 17th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 443, Springer-Verlag, New York, 1990, pp. 400–413.

[Ede87] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Heidelberg, 1987.

[Efr05] A. Efrat, *The complexity of the union of (α, β)-covered objects*, SIAM J. Comput., 34 (2005), pp. 775–787.

[EHS04] E. Ezra, D. Halperin, and M. Sharir, *Speeding up the incremental construction of the union of geometric objects in practice*, Comput. Geom., 27 (2004), pp. 63–85.

[GO95] A. Gajentaan and M. H. Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Comput. Geom., 5 (1995), pp. 165–185; also available online from http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-1993/1993-15.ps.gz.

[HI00] S. Har-Peled and P. Indyk, *When crossings count—approximating the minimum spanning tree*, in Proceedings of the 16th Annual ACM Symposium on Computational Geometry, ACM, New York, 2000, pp. 166–175; also available online from http://www.uiuc.edu/~sariel/papers/99/mst.html.

[HW04] S. Har-Peled and Y. Wang, *Shape fitting with outliers*, SIAM J. Comput., 33 (2004), pp. 269–285.

[IM98] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 604–613; also available online from http://theory.lcs.mit.edu/~indyk/nndraft.ps.

[Ind00] P. Indyk, *High-Dimensional Computational Geometry*, Ph.D. thesis, Stanford University, Stanford, CA, 2000; also available online from http://theory.lcs.mit.edu/~indyk/thesis.html.

[KLPS86] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete Comput. Geom., 1 (1986), pp. 59–71.

[KS06] H. Kaplan and M. Sharir, *Randomized incremental constructions of three-dimensional convex hulls and planar voronoi diagrams, and approximate range counting*, in Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2006, pp. 484–493.

[Mat91] J. Matoušek, *Randomized optimal algorithm for slope selection*, Inform. Process. Lett., 39 (1991), pp. 183–187.

[Mat92] J. Matoušek, *Reporting points in halfspaces*, Comput. Geom., 2 (1992), pp. 169–186.

[Mat93] J. Matoušek, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.

[Mat95] J. Matoušek, *On geometric optimization with few violated constraints*, Discrete Comput. Geom., 14 (1995), pp. 365–384.

[MMP+94] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, *Fat triangles determine linearly many holes*, SIAM J. Comput., 23 (1994), pp. 154–169.

[MR95] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995; also available online from http://us.cambridge.org/titles/catalogue.asp?isbn=0521474655.

[Mul91] K. Mulmuley, *On levels in arrangements and Voronoi diagrams*, Discrete Comput. Geom., 6 (1991), pp. 307–338.

[SA95] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995; also available online from http://us.cambridge.org/titles/catalogue.asp?isbn=0521470250.

[Sha91] M. Sharir, *On k-sets in arrangements of curves and surfaces*, Discrete Comput. Geom., 6 (1991), pp. 593–613.

[Sha03] M. Sharir, *The Clarkson-Shor technique revisited and extended*, Combin. Probab. Comput., 12 (2003), pp. 191–201.

[VC71] V. N. Vapnik and A. Y. Chervonenkis, *On the uniform convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 922–945

EFFICIENT ALGORITHMS FOR DESCRIPTION PROBLEMS OVER FINITE TOTALLY ORDERED DOMAINS*

ÁNGEL J. GIL†, MIKI HERMANN‡, GERNOT SALZER§, AND BRUNO ZANUTTINI¶

Dedicated to the memory of Peter Ružička (1947–2003)

Abstract. Given a finite set of vectors over a finite totally ordered domain, we study the problem of computing a constraint in conjunctive normal form such that the set of solutions for the produced constraint is identical to the original set. We develop an efficient polynomial-time algorithm for the general case, followed by specific polynomial-time algorithms producing Horn, dual Horn, and bijunctive formulas for sets of vectors closed under the operations of conjunction, disjunction, and median, respectively. Our results generalize the work of Dechter and Pearl on relational data, as well as the papers by Hébrard and Zanuttini. They complement the results of Hähnle et al. on multivalued logics and Jeavons et al. on the algebraic approach to constraints.

Key words. finite domain, description problems, algorithms, complexity

AMS subject classifications. 68Q25, 68T27, 68W40

DOI. 10.1137/050635900

1. Introduction and summary of results.

Constraint satisfaction problems constitute today a well-studied topic on the frontier of complexity, logic, combinatorics, and artificial intelligence. It is indeed well known that this framework allows us to encode many natural problems or knowledge bases. In principle, an instance of a constraint satisfaction problem is a finite set of variable vectors associated with an allowed set of values. A *model* is an assignment of values to all variables that satisfies every constraint. When a constraint satisfaction problem encodes a decision problem, the models represent its *solutions*. When it encodes some knowledge, the models represent possible combinations that the variables can assume in the described universe.

Constraints can be represented by means of a set of variable vectors associated with an allowed set of values. This representation is not always well suited; therefore, other representations have been introduced. The essence of the most studied alternative is the notion of a *relation*, making it easy to apply it within the database or knowledge base framework. We focus on the representation by formulas in conjunctive normal form with the literals taking the form $(x \leq d)$ and $(x \geq d)$, where x is a variable and d is an element from the given finite domain D, totally ordered by the relation $\leq$ (see Hähnle et al. [5, 6, 16]). We study in this paper the constraint *description* problem, i.e., the problem of converting a set of vectors M to a formula φ

*Received by the editors July 13, 2005; accepted for publication (in revised form) February 20, 2008; published electronically June 6, 2008. This work was supported by ÉGIDE 06606ZF, ÖAD Amadeus 18/2004, ANR CANAR ANR-06-BLAN-0383-02, and Acciónes Integradas Hispano-Austriacas HU2003-0043 and HU2005-0024. A preliminary extended abstract of this paper appeared as [13].

http://www.siam.org/journals/sicomp/38-3/63590.html

†Department of Economics, Universitat Pompeu Fabra, 08005 Barcelona, Spain (angel.gil@upf.edu).

‡LIX (UMR 7161), École Polytechnique, 91128 Palaiseau, France (hermann@lix.polytechnique.fr).

§Department of Computer Science, Technische Universität Wien, 1040 Wien, Austria (salzer@logic.at).

¶GREYC (UMR 6072), Université de Caen, 14032 Caen, France (zanutti@info.unicaen.fr).

in conjunctive normal form, such that its satisfying assignments $\mathrm{Sol}(\varphi)$ equals the original set M. We consider this problem first in its general setting without any restrictions imposed on the initial set of vectors. We continue by imposing several properties on the initial set, like the closure under the minimum, maximum, and median operations. We subsequently discover that these closure properties induce the description by Horn, dual Horn, and bijunctive constraints, respectively. Moreover, we give an elegant and unified solution to the *structure identification* problem of these three classes, as it was defined by Dechter and Pearl [11]. Given a set of vectors, this problem asks whether it can be represented by a formula of a special type, computing such a formula if the answer is affirmative.

The motivations to study description and identification problems are numerous. From the artificial intelligence point of view, description problems formalize the notion of exact acquisition of knowledge from examples. This means that they formalize situations where a system is given access to a set of examples and it is asked to compute a formula describing it exactly. Moreover, this representation takes usually less space than the original set of examples; thus it can be stored more easily in a knowledge base.

Satisfiability poses a keystone problem in artificial intelligence, automated deduction, databases, and verification. It is well known that the satisfiability problem for arbitrary constraints is an NP-complete problem. Therefore, it is important to look for restricted classes of constraints that admit polynomial algorithms deciding satisfiability. Horn, dual Horn, bijunctive, and affine constraints, in the Boolean case, constitute exactly these tractable classes, as was proved by Schaefer [24]. Thus the description problem for these four classes can be seen as storing specific knowledge into a knowledge base while we are required to respect its format. This problem is also known as *structure identification*, studied by Dechter with Pearl [11] and by Hébrard with Zanuttini [17, 25]. Another motivation for studying description problems comes from combinatorics. Indeed, since finding a solution for an instance of a constraint satisfaction problem is difficult in general but tractable in the four aforementioned cases, it is important to be able to recognize constraints belonging to these tractable cases.

The study of Boolean constraint satisfaction problems, especially their complexity questions, was started by Schaefer in [24], although he did not yet consider constraints explicitly. During the last ten years, constraints gained considerable interest in theoretical computer science. An excellent complexity classification of existing Boolean constraint satisfaction problems can be found in the monograph [10]. Jeavons et al. [9, 19, 20] started to study constraint satisfaction problems from an algebraic viewpoint. Feder, Kolaitis, and Vardi [12, 22] posed a general framework for the study of constraint satisfaction problems. A part of the research in constraint satisfaction problems requires the existence of efficient description and identification methods for special constraint classes.

Recently, there has been much progress on constraint satisfaction problems over domains with larger cardinality. Hell and Nešetřil [18] studied constraint satisfaction problems by means of graph homomorphisms. Bulatov [7] made a significant breakthrough with a generalization of Schaefer's result to three-element domains. He also proved a dichotomy theorem for conservative constraints over arbitrary domains [8]. On the other hand, Hähnle et al. [5, 6, 16] studied the complexity of satisfiability problems for many-valued logics that present yet another viewpoint of constraint satisfaction problems. We realized reading the previous articles on many-valued logics that in the presence of a total order the satisfiability problems for the Horn, dual

Horn, and bijunctive many-valued formulas of signed logic are decidable in polynomial time. We also noticed that Jeavons and Cooper [21] studied some aspects of tractable constraints on finite ordered domains from an algebraic standpoint. This led us to the idea to look more carefully at constraint description problems over finite totally ordered domains, developing a new formalism for constraints based on an already known concept of inequalities.

The purpose of our paper is manifold. We want to generalize the work of Dechter and Pearl [11], based on the more efficient algorithms for Boolean description problems by Hébrard and Zanuttini [17, 25]. We also want to complement the work of Hähnle et al. on many-valued logics. Finally, we want to provide a characterization by closure properties of polynomial-time decidable subcases of constraint satisfaction problems over finite totally ordered domains, which are straightforward generalizations of the known polynomial-time decidable Boolean cases.

2. Preliminaries. Let $D = \{0, \dots, n-1\}$ be a finite, totally ordered domain of cardinality n, and let V be a set of variables. For a variable $x \in V$ and a value $d \in D$, the inequalities $x \geq d$ and $x \leq d$ are called positive and negative *literal*, respectively. The set of *formulas* over D and V is inductively defined as follows:

- the logical constants *false* and *true* are formulas;
- literals are formulas;
- if φ and ψ are formulas, then the expressions $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$ are formulas.

We write $\varphi(x_1, \dots, x_\ell)$ to indicate that formula φ contains exactly the variables $x_1, \dots, x_\ell$. For convenience, we use the following shorthand notation:

- $x > d$ means $x \geq d+1$ for $d \in \{0, \dots, n-2\}$, and *false* otherwise;
- $x < d$ means $x \leq d-1$ for $d \in \{1, \dots, n-1\}$, and *false* otherwise;
- $x = d$ means $x \geq d \wedge x \leq d$;
- $\neg$*false* and $\neg$*true* mean *true* and *false*, respectively;
- $\neg(x \geq d)$, $\neg(x \leq d)$, $\neg(x > d)$, and $\neg(x < d)$ mean $x < d$, $x > d$, $x \leq d$, and $x \geq d$, respectively;
- $\neg(x = d)$ and $x \neq d$ both mean $x < d \vee x > d$;
- $\neg(\varphi \wedge \psi)$ and $\neg(\varphi \vee \psi)$ mean $\neg\varphi \vee \neg\psi$ and $\neg\varphi \wedge \neg\psi$, respectively.

Note that $x = d$ and $x \neq d$ asymptotically require the same space as their alternative notation, i.e., $O(\log n)$. Indeed, since d is bounded by n, its binary coding has length $O(\log n)$.

A *clause* is a disjunction of literals. It is a *Horn* clause if it contains at most one positive literal, *dual Horn* if it contains at most one negative literal, and *bijunctive* if it contains at most two literals. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. It is a Horn, a dual Horn, or a bijunctive formula if it is a conjunction of Horn, dual Horn, or bijunctive clauses, respectively. Since the considered formulas in what follows are all in CNF, we will use the expression "formula" with a slight abuse of terminology also for CNF formulas, without explicitly specifying it.

Note that contrary to the Boolean case, a clause in our formalism can contain the same variable twice, in both a negative and a positive literal, without being reducible. However, such a clause can be assumed to contain not more than twice the same variable, since $x \geq d \vee x \geq d'$ can be reduced to $x \geq \min(d, d')$ and, dually, $x \leq d \vee x \leq d'$ can be reduced to $x \leq \max(d, d')$.

Example 2.1. Let $D = \{0, 1, 2, 3, 4\}$ be the domain for our running example. The expressions $x \leq 2$ and $y \geq 4$ are a negative and a positive literal, respectively. Instead of $x \leq 2$ and $y \geq 4$, we can also write $x < 3$ and $y > 3$, respectively. The disjunction

of literals $(x \le 2 \vee x \ge 4 \vee y \le 4)$ is a Horn clause, $(x \le 2 \vee x \ge 4 \vee y \ge 3)$ is a dual Horn clause, and $(x \le 2 \vee x \ge 4)$ is a bijunctive clause. The formula

$$\varphi(x,y) = (x \le 2 \vee x \ge 4 \vee y \le 4) \wedge (x \le 2 \vee x \ge 4 \vee y \ge 4) \wedge (x \le 2 \vee x \ge 3)$$

is in CNF. □

An *assignment* for a formula $\varphi(x_1, \ldots, x_\ell)$ is a mapping $m\colon \{x_1, \ldots, x_\ell\} \to D$ assigning a domain element $m(x)$ to each variable x. The *satisfaction relation* $m \models \varphi$ is inductively defined as follows:

- $m \models \mathit{true}$ and $m \not\models \mathit{false}$;
- $m \models x \le d$ if $m(x) \le d$, and $m \models x \ge d$ if $m(x) \ge d$;
- $m \models \varphi \wedge \psi$ if $m \models \varphi$ and $m \models \psi$;
- $m \models \varphi \vee \psi$ if $m \models \varphi$ or $m \models \psi$.

The set of all assignments satisfying φ is denoted by $\mathrm{Sol}(\varphi)$, also called *models* of φ. If we arrange the variables in some arbitrary but fixed order, say, as a vector $x = (x_1, \ldots, x_\ell)$, then the models can be identified with the vectors in D^ℓ. The jth component of a vector m, denoted by $m[j]$, gives the value of the jth variable, i.e., $m(x_j) = m[j]$. The operations of conjunction, disjunction, addition, and median on vectors $m, m', m'' \in D^\ell$ are defined as follows:

$$\begin{aligned} m \wedge m' &= (\min(m[1], m'[1]), \ldots, \min(m[\ell], m'[\ell])), \\ m \vee m' &= (\max(m[1], m'[1]), \ldots, \max(m[\ell], m'[\ell])), \\ \mathrm{med}(m, m', m'') &= (\mathrm{med}(m[1], m'[1], m''[1]), \ldots, \mathrm{med}(m[\ell], m'[\ell], m''[\ell])). \end{aligned}$$

The ternary *median* operator is defined as follows: for each choice of three values $a, b, c \in D$ such that $a \le b \le c$, we have $\mathrm{med}(a,b,c) = b$. Moreover, median is a permutative operator; i.e., the identity $\mathrm{med}(a,b,c) = \mathrm{med}(\pi(a), \pi(b), \pi(c))$ holds for every permutation π on all domain elements $a, b, c \in D$. Note that the median can also be defined by $\mathrm{med}(a,b,c) = \min(\max(a,b), \max(b,c), \max(c,a))$ as well as by $\mathrm{med}(a,b,c) = \max(\min(a,b), \min(b,c), \min(c,a))$, which implies the identities

$$\begin{aligned} \mathrm{med}(m_1, m_2, m_3) &= (m_1 \vee m_2) \wedge (m_2 \vee m_3) \wedge (m_3 \vee m_1) \\ &= (m_1 \wedge m_2) \vee (m_2 \wedge m_3) \vee (m_3 \wedge m_1) \end{aligned}$$

for all vectors $m_1, m_2, m_3 \in D^\ell$.

Example 2.2. Consider the set of vectors $M = \{010, 013, 220, 440, 444\}$. It is closed under conjunction, since for each pair of vectors $m, m' \in M$ we have $m \wedge m' \in M$. It is *not* closed under disjunction, since $013 \vee 220 = 223 \notin M$. It is also *not* closed under median, since $\mathrm{med}(013, 220, 444) = 223 \notin M$. □

3. Formulas in conjunctive normal form. We investigate first the description problem for arbitrary sets of vectors.

Problem: DESCRIPTION.
Input: A finite set of vectors $M \subseteq D^\ell$ over a finite totally ordered domain D.
Output: A formula $\varphi(x_1, \ldots, x_\ell)$ over D in CNF such that $\mathrm{Sol}(\varphi) = M$.

The naive approach to this problem is to compute first the complement set $\bar{M} = D^\ell \setminus M$, followed by the construction of a clause $c(\bar{m})$ for each vector $\bar{m} \in \bar{M}$ missing from M such that $\bar{m}$ is the unique vector falsifying $c(\bar{m})$. The formula φ is then the conjunction of the clauses $c(\bar{m})$ for all missing vectors $\bar{m} \in \bar{M}$. However, this algorithm is essentially exponential, since the complement set $\bar{M}$ can be exponentially bigger than the original set of vectors M.

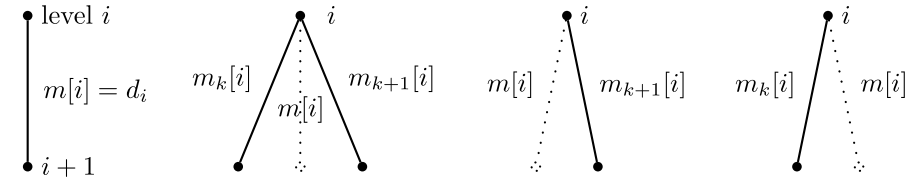
Example 3.1. Consider again the set of vectors $M = \{010, 013, 220, 440, 444\}$ as in Example 2.2. The complement set $\bar{M}$ contains $5^3 - 5 = 120$ vectors, where the lexicographically first ones are $000, 001, \ldots, 004, 011, 012$ and the last ones are $434, 441, 442, 443$. For example, the clause $c(001)$ is $(x_1 \neq 0 \vee x_2 \neq 0 \vee x_3 \neq 1)$, equivalent to $(x_1 \geq 1 \vee x_2 \geq 1 \vee x_3 \leq 0 \vee x_3 \geq 2)$. The only assignment to the variables x_1, x_2, x_3 which does not satisfy the clause $c(001)$ is the vector 001. Similarly, the clause $c(223)$ is $(x_1 \leq 1 \vee x_1 \geq 3 \vee x_2 \leq 1 \vee x_2 \geq 3 \vee x_3 \leq 2 \vee x_3 \geq 4)$. The formula φ describing M and built in this manner contains exactly 120 clauses. ∎

We present a new algorithm running in polynomial time and producing a CNF formula of polynomial length with respect to the cardinality of the set M, the dimension of vectors ℓ, and the size $O(\log |D|)$ of the domain elements in binary notation.

In what follows we assume without loss of generality the set of vectors M to be nonempty, which simplifies the presentation. Note, however, that the empty set $\emptyset$ is easily recognized and described by the formula $(x_1 \leq 0) \wedge (x_1 \geq 1)$, which is logically equivalent to *false*.

To construct the formula φ we proceed in the following way. We arrange the set M into an ordered n-ary semantic tree T_M [14], with branches corresponding to the vectors in M. In case M contains all possible vectors, i.e., $M = D^\ell$, T_M is a complete tree of branching factor $|D|$ and depth ℓ. Otherwise, some branches are missing, leading to gaps in the tree. We characterize these gaps by conjunctions of literals. Their disjunction yields a complete description of all vectors that are *missing* from M. Finally, by negation and de Morgan's laws we obtain φ.

Let T_M be an ordered tree with edges labeled by domain elements such that each path from the root to a leaf corresponds to a vector in M. The tree T_M contains a path labeled $d_1.\cdots.d_i$ from the root to some node if there is a vector $m \in M$ such that $m[j] = d_j$ holds for every $j = 1, \ldots, i$. The level of a node is its distance to the root plus 1; i.e., the root is at level 1 and a node reachable via $d_1.\cdots.d_i$ is at level $i + 1$ (Figure 3.1(a)). Note that all leaves are at level $\ell + 1$. If the edges between a node and its children are sorted in ascending order according to their labels, then traversing the leaves from left to right enumerates the vectors of M in lexicographic order, say, $m_1, \ldots, m_{|M|}$. A vector m is lexicographically smaller than a vector m' if there is a level i such that $m[i] < m'[i]$ holds, and for all $j < i$ we have $m[j] = m'[j]$.



(a) Level i (b) Missing between (c) ... left (d) ... right

FIG. 3.1. *Tree representation of vectors.*

Example 3.2. Let $M = \{m_1 = 010, m_2 = 013, m_3 = 220, m_4 = 440, m_5 = 444\}$ be the set of vectors over the domain $D = \{0, 1, 2, 3, 4\}$ for which we want to construct a formula φ in CNF satisfying the condition $\mathrm{Sol}(\varphi) = M$. The tree T_M is depicted in Figure 3.2 in solid lines. ∎

Suppose that m_k and m_{k+1} are immediate neighbors in the lexicographic enumeration of M, and let m be a vector lexicographically in between, thus missing from M. There are three possibilities for the path corresponding to m. It may leave the tree

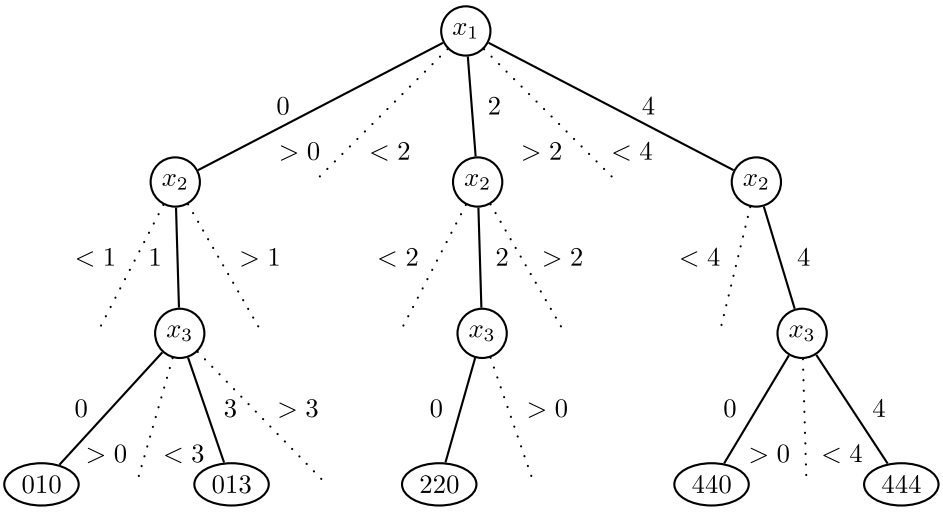


FIG. 3.2. *Tree T_M and the missing parts for the set $M = \{010, 013, 220, 440, 444\}$.*

at the fork *between* m_k and m_{k+1} (Figure 3.1(b)), or at the fork to the *left* of m_{k+1} (Figure 3.1(c)), or at the fork to the *right* of m_k (Figure 3.1(d)). Let i be the least position in which the consecutive vectors m_k and m_{k+1} differ. In other words, we have $m_k[i] \neq m_{k+1}[i]$ and $m_k[j] = m_{k+1}[j]$ for all $j < i$. The set of missing vectors can be characterized by the conjunctions middle(k, i), left$(k + 1, i)$, and right(k, i), defined as follows:

$$\begin{aligned}
\text{middle}(k,i) &= \bigwedge_{j<i}(x_j = m_k[j]) \quad \wedge \quad (x_i > m_k[i]) \ \wedge \ (x_i < m_{k+1}[i]),\\
\text{left}(k+1,i) &= \bigwedge_{j<i}(x_j = m_{k+1}[j]) \quad \wedge \quad (x_i < m_{k+1}[i]),\\
\text{right}(k,i) &= \bigwedge_{j<i}(x_j = m_k[j]) \quad \wedge \quad (x_i > m_k[i]).
\end{aligned}$$

The situation depicted in Figure 3.1 is a snapshot at level i of the tree T_M.

Example 3.3. The missing parts in the tree T_M, displayed in Figure 3.2 by dotted lines, are described by the following conjunctions. First are the parts missing between two vectors,

$$\begin{aligned}
\text{middle}(1,3) &= (x_1 = 0) \wedge (x_2 = 1) \wedge (x_3 > 0) \wedge (x_3 < 3),\\
\text{middle}(2,1) &= (x_1 > 0) \wedge (x_1 < 2),\\
\text{middle}(3,1) &= (x_1 > 2) \wedge (x_1 < 4),\\
\text{middle}(4,3) &= (x_1 = 4) \wedge (x_2 = 4) \wedge (x_3 > 0) \wedge (x_3 < 4),
\end{aligned}$$

followed by the parts missing to the left,

$$\begin{aligned}
\text{left}(1,2) &= (x_1 = 0) \wedge (x_2 < 1),\\
\text{left}(3,2) &= (x_1 = 2) \wedge (x_2 < 2),\\
\text{left}(4,2) &= (x_1 = 4) \wedge (x_2 < 4),
\end{aligned}$$

and finally the parts missing to the right,

$$\begin{aligned}
\mathrm{right}(2,2) &= (x_1 = 0) \wedge (x_2 > 1),\\
\mathrm{right}(2,3) &= (x_1 = 0) \wedge (x_2 = 1) \wedge (x_3 > 3),\\
\mathrm{right}(3,2) &= (x_1 = 2) \wedge (x_2 > 2),\\
\mathrm{right}(3,3) &= (x_1 = 2) \wedge (x_2 = 2) \wedge (x_3 > 0).
\end{aligned}$$

There are no other missing parts in the tree T_M. □

To describe *all* vectors missing from M we form the disjunction of the above conjunctions for appropriate values of k and i. We need to determine the levels at which neighboring models fork by means of the following function:

$$\mathrm{fork}(k) \quad = \quad \begin{cases} 0 & \text{for } k = 0,\\ \min\{i \mid m_k[i] \neq m_{k+1}[i]\} & \text{for } k = 1, \ldots, |M| - 1,\\ 0 & \text{for } k = |M|. \end{cases}$$

The values $\mathrm{fork}(0)$ and $\mathrm{fork}(|M|)$ correspond to imaginary models m_0 and $m_{|M|+1}$ forking at a level above the root. They allow us to write the conditions below in a concise way at the left and right borders of the tree. The three situations in Figure 3.1 can now be specified by the following conditions:

$$\begin{array}{lcll}
i = \mathrm{fork}(k) & \wedge & m_k[i] + 1 < m_{k+1}[i] & \text{(edges missing in between)},\\
\mathrm{fork}(k) < i & \wedge & m_{k+1}[i] > 0 & (\ldots \text{ to the left}),\\
\mathrm{fork}(k) < i & \wedge & m_k[i] < |D| - 1 & (\ldots \text{ to the right}).
\end{array}$$

The second condition in each line ensures that there is at least one missing edge. It avoids the conjunctions $\mathrm{middle}(k,i)$, $\mathrm{left}(k+1,i)$, and $\mathrm{right}(k,i)$ to evaluate to false.

Example 3.4. The function fork for $M = \{m_1 = 010,\ m_2 = 013,\ m_3 = 220,\ m_4 = 440,\ m_5 = 444\}$ is given by the following table:

	0	1	2	3	4	5
fork	0	3	1	1	3	0

The aforementioned fork conditions for missing edges are satisfied in the following cases.

(i) For the first condition, implying edges missing in between, we have four cases where it is satisfied:

$$\begin{array}{lcl}
3 = \mathrm{fork}(1) & \wedge & 1 = m_1[3] + 1 < m_2[3] = 3,\\
1 = \mathrm{fork}(2) & \wedge & 1 = m_2[1] + 1 < m_3[1] = 2,\\
1 = \mathrm{fork}(3) & \wedge & 3 = m_3[1] + 1 < m_4[1] = 4,\\
3 = \mathrm{fork}(4) & \wedge & 1 = m_4[3] + 1 < m_5[3] = 4.
\end{array}$$

(ii) For the second condition, implying edges missing to the left, we have three cases where it is satisfied:

$$\begin{array}{lcl}
\mathrm{fork}(0) < 2 & \wedge & 1 = m_1[2] > 0,\\
\mathrm{fork}(2) < 2 & \wedge & 2 = m_3[2] > 0,\\
\mathrm{fork}(3) < 2 & \wedge & 4 = m_4[2] > 0.
\end{array}$$

(iii) Finally, for the third condition, implying edges missing to the right, we have also three cases where it is satisfied:

$$\begin{aligned}
&\mathrm{fork}(2) < 2 \quad \wedge \quad 1 = m_2[2] < 4,\\
&\mathrm{fork}(2) < 3 \quad \wedge \quad 3 = m_2[3] < 4,\\
&\mathrm{fork}(3) < 2 \quad \wedge \quad 2 = m_3[2] < 4,\\
&\mathrm{fork}(3) < 3 \quad \wedge \quad 0 = m_3[3] < 4.
\end{aligned}$$

It can be easily seen that these conditions trigger the middle, left, and right formulas shown in Example 3.3. □

The disjunction of terms $\mathrm{middle}(k,i)$, $\mathrm{left}(k+1,i)$, and $\mathrm{right}(k,i)$ that satisfy the first, second, and third condition, respectively, for all models and all levels represents a disjunctive formula satisfied by the models *missing* from M. After applying negation and de Morgan's laws, we arrive at the required formula in CNF,

$$\begin{aligned}
\varphi(M) \quad = \quad & \bigwedge\{\, \neg\,\mathrm{middle}(k,i) \mid 0 < k < |M|,\ i = \mathrm{fork}(k),\ m_k[i]+1 < m_{k+1}[i] \,\}\\
\wedge \quad & \bigwedge\{\, \neg\,\mathrm{left}(k+1,i) \mid 0 \le k < |M|,\ \mathrm{fork}(k) < i \le \ell,\ m_{k+1}[i] > 0 \,\}\\
\wedge \quad & \bigwedge\{\, \neg\,\mathrm{right}(k,i) \quad \mid 0 < k \le |M|,\ \mathrm{fork}(k) < i \le \ell,\ m_k[i] < |D|-1 \,\},
\end{aligned}$$

where the condition $\mathrm{Sol}(\varphi) = M$ holds. Note that we use the negation symbol not as an operator on the syntax level but as a metanotation expressing that the formula following the negation sign has to be replaced by its dual. Note also that the conjunct $\mathrm{left}(k+1,i)$ is defined and used with the shifted parameter $k+1$ since it characterizes a gap lexicographically before the vector m_{k+1}.

Example 3.5. The set of vectors $M = \{010, 013, 220, 440, 444\}$ is described by the following CNF formula:

$$\begin{aligned}
\varphi(M) \ = \ & \neg\,\mathrm{middle}(1,3) \ \wedge \ \neg\,\mathrm{middle}(2,1) \ \wedge \ \neg\,\mathrm{middle}(3,1) \ \wedge \ \mathrm{middle}(4,3)\\
& \wedge \ \neg\,\mathrm{left}(1,2) \ \wedge \ \neg\,\mathrm{left}(3,2) \ \wedge \ \neg\,\mathrm{left}(4,2)\\
& \wedge \ \neg\,\mathrm{right}(2,2) \ \wedge \ \neg\,\mathrm{right}(2,3) \ \wedge \ \neg\,\mathrm{right}(3,2) \ \wedge \ \neg\,\mathrm{right}(3,3).
\end{aligned}$$

After substitution and application of de Morgan laws, this amounts to

$$\begin{aligned}
\varphi(M) \ = \ & (x_1 \neq 0 \vee x_2 \neq 1 \vee x_3 \le 0 \vee x_3 \ge 3) \ \wedge \ (x_1 \le 0 \vee x_1 \ge 2)\\
& \wedge \ (x_1 \le 2 \vee x_1 \ge 4) \ \wedge \ (x_1 \neq 4 \vee x_2 \neq 4 \vee x_3 \le 0 \vee x_3 \ge 4)\\
& \wedge \ (x_1 \neq 0 \vee x_2 \ge 1) \ \wedge \ (x_1 \neq 2 \vee x_2 \ge 2) \ \wedge \ (x_1 \neq 4 \vee x_2 \ge 4)\\
& \wedge \ (x_1 \neq 0 \vee x_2 \le 1) \ \wedge \ (x_1 \neq 0 \vee x_2 \neq 1 \vee x_3 \le 3) \ \wedge \ (x_1 \neq 2 \vee x_2 \le 2)\\
& \wedge \ (x_1 \neq 2 \vee x_2 \neq 2 \vee x_3 \le 0).
\end{aligned}$$

Replacing the shorthand notation $\neq$ by proper literals gives the following final formula:

$$\begin{aligned}
\varphi(M) \ = \ & (x_1 \ge 1 \vee x_2 \le 0 \vee x_2 \ge 2 \vee x_3 \le 0 \vee x_3 \ge 3) \ \wedge \ (x_1 \le 0 \vee x_1 \ge 2)\\
& \wedge \ (x_1 \le 2 \vee x_1 \ge 4) \ \wedge \ (x_1 \le 3 \vee x_2 \le 3 \vee x_3 \le 0 \vee x_3 \ge 4)\\
& \wedge \ (x_1 \ge 1 \vee x_2 \ge 1) \ \wedge \ (x_1 \le 1 \vee x_1 \ge 3 \vee x_2 \ge 2) \ \wedge \ (x_1 \le 3 \vee x_2 \ge 4)\\
& \wedge \ (x_1 \ge 1 \vee x_2 \le 1) \ \wedge \ (x_1 \ge 1 \vee x_2 \le 0 \vee x_2 \ge 2 \vee x_3 \le 3)\\
& \wedge \ (x_1 \le 1 \vee x_1 \ge 3 \vee x_2 \le 2) \wedge (x_1 \le 1 \vee x_1 \ge 3 \vee x_2 \le 1 \vee x_2 \ge 3 \vee x_3 \le 0).
\end{aligned}$$

Algorithm: DESCRIPTION
Input: Nonempty set $M \subseteq D^\ell$ of vectors.
Output: Formula $\varphi(M)$ in CNF, satisfying $\mathrm{Sol}(\varphi) = M$.
Method:

```
1: let M = (m_1, ..., m_|M|) be lexicographically sorted
2: φ(M) ← true
3: fork ← FORK(M)
4: for k ← 0 to |M| do
5:     f ← fork[k]
6:     if f > 0 and m_k[f] + 1 < m_{k+1}[f] then
7:         φ(M) ← φ(M) ∧ ¬middle(k, f)
8:     end if
9:     for i ← f + 1 to ℓ do
10:        if k < |M| and m_{k+1}[i] > 0 then
11:            φ(M) ← φ(M) ∧ ¬left(k + 1, i)
12:        end if
13:        if k > 0 and m_k[i] < |D| − 1 then
14:            φ(M) ← φ(M) ∧ ¬right(k, i)
15:        end if
16:    end for
17: end for
18: return φ(M)
```

Algorithm: FORK
Input: Lexicographically sorted nonempty list $M \subseteq D^\ell$ of vectors without duplicates.
Output: Array fork: $[0 \dots |M|]$ containing the fork function for M.
Method:

```
1: fork[0] ← 0
2: fork[|M|] ← 0
3: for k ← 1 to |M| − 1 do
4:     i ← 1
5:     while m_k[i] = m_{k+1}[i] do
6:         i ← i + 1
7:     end while
8:     fork[k] ← i
9: end for
10: return fork
```

FIG. 3.3. *Algorithm for the description problem.*

It can be easily checked that the constructed formula $\varphi(M)$ satisfies the condition $\mathrm{Sol}(\varphi(M)) = M$. □

The main algorithm that implements the construction of a formula φ in CNF for a given set of vectors M over a finite totally ordered domain D, satisfying the condition $\mathrm{Sol}(\varphi) = M$, and the algorithm computing the fork function are displayed in Figure 3.3.

THEOREM 3.6. *For each set of vectors $M \subseteq D^\ell$ over a finite ordered domain D there exists a formula φ in CNF such that $M = \mathrm{Sol}(\varphi)$. It contains at most $2\,|M|\,\ell$ clauses and its length is $O(|M|\,\ell^2 \log |D|)$. The algorithm constructing φ runs in time*

$O(|M|\,\ell^2 \log|D|)$.

Proof. The formula $\varphi(M)$ contains at most $|M|-1$ middle-clauses and at most $\ell + (|M|-1)(\ell-1)$ left/right-clauses. Summing up these partial bounds, we obtain for the total number of clauses the bound

$$|M| - 1 + 2(\ell + (|M|-1)(\ell-1)) = 2\,|M|\,\ell - |M| + 1 \;\le\; 2\,|M|\,\ell \quad (\text{for } M \neq \emptyset).$$

Each clause contains at most 2ℓ literals, namely at most one positive and one negative for each variable. Each literal has length $O(\log|D|)$, since the domain elements are written in binary notation. Hence, the overall length of the formula $\varphi(M)$ is $O(|M|\,\ell^2 \log|D|)$.

The vectors in M can be lexicographically sorted in time $O(|M|\,\ell \log|D|)$ using a decision tree (trie) or a radix sort. The factor $\log|D|$ stems from the comparison of domain elements. The fork levels can also be computed in time $O(|M|\,\ell \log|D|)$, in parallel with sorting the set M. The formula $\varphi(M)$ is produced by two loops, where the outer loop is going through each vector in M and the inner loop through the variables. The three clauses $\neg\,\mathrm{middle}(k,i)$, $\neg\,\mathrm{left}(k+1,i)$, and $\neg\,\mathrm{right}(k,i)$ are potentially written in each step inside the combined loops. This makes an algorithm with time complexity $O(|M|\,\ell^2 \log|D|)$. □

An important property of our algorithm is its linearity with respect to the number of models $|M|$ being the most relevant parameter. In fact, the paper by Amilhastre, Fargier, and Marquis [1] mentions an industrial problem provided by Renault DVI, where the cardinality of the set of vectors is $|M| = 1.5 \cdot 10^{12}$ with the vector arity $\ell = 101$ over a domain of size $|D| = 43$. The complement set $\bar{M}$ contains $43^{101} - 1.5 \cdot 10^{12}$ vectors; therefore, the naive algorithm is inapplicable in this situation.

4. Prime formulas. The formulas in CNF computed by Algorithm DESCRIPTION in section 3 are of a particular form: The variables in each clause form a prefix of the variable vector $(x_1, \ldots, x_\ell)$. As a consequence, although polynomial in the size of the initial relation M, the size of the formulas is not minimal. In Example 3.5, for instance, it can be easily seen that several literals and even clauses can be removed from the formula. We investigate in this section a way to shorten formulas. To this aim we generalize the notion of a *prime* formula in propositional logic to the case of finite domains and show how to obtain such a prime formula in CNF describing the given relation M. Note that although we apply the minimization process to the formulas produced by our algorithm, it can be applied to any formula in CNF.

The notion of primality and prime clauses in many-valued logic was considered for the first time by Murray and Rosenthal in [23].

4.1. Notions of primality. Recall that a clause of a propositional formula φ in CNF is said to be *prime* (with respect to φ) if φ implies none of its proper subclauses. This leads to the following straightforward generalization. Let φ be a CNF formula over some finite totally ordered domain. A clause $c = (l_1 \vee \cdots \vee l_q)$ of φ is said to be *prime* (with respect to φ) if for each $i = 1, \ldots, q$ there exists a model $m_i \in \mathrm{Sol}(\varphi)$ not satisfying the reduced clause $c \setminus l_i = (l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \cdots \vee l_q)$. The formula φ is said to be prime if all its clauses are prime.

However, this notion of primality considers each literal as a whole. This is adequate in the case of classical propositional logic but does not meet our requirements in the case of larger domains. The following more sophisticated notion of primality also considers the value d involved in a literal $x \le d$ or $x \ge d$.

DEFINITION 4.1 (primality). *For a variable x and a pair of values $d, d' \in D$ satisfying the relation $d' > d$ (resp., $d' < d$), the literal $x \ge d'$ (resp., $x \le d'$) is said*

to be stronger *than the literal $x \geq d$ (resp., $x \leq d$). The constant false is stronger than any other nonfalse literal. Removal of a literal from a clause is a particular case of strengthening, namely, of this literal to the constant false.*

Let φ be a CNF formula over a finite totally ordered domain D. A clause c in φ is prime *with respect to φ if strengthening any literal in c yields a clause which is not implied by φ. A formula φ is* prime *if all its clauses are prime.*

As in the propositional case, it is easily seen that for a given formula φ there always exists at least one prime formula φ' which is logically equivalent to φ and can be obtained from φ by strengthening and removing some of its literals. If φ is already prime, then φ and φ' are identical.

The interest in this new, stronger notion of primality comes from efficiency requirements. In fact, the presence of a literal $x \leq d$ or $x \geq d$ instead of $x \leq d'$ or $x \geq d''$, respectively, for $d < d'$ or $d > d''$ reduces the search space during a search for a suitable satisfying assignment.

4.2. Algorithm. Given a CNF formula φ, we show how to efficiently compute a prime formula φ' satisfying the equality $\mathrm{Sol}(\varphi) = \mathrm{Sol}(\varphi')$. Our algorithm, specified in Figure 4.1, is inspired by the one presented in [25] for the Boolean domain. The algorithm considers each clause $l_1 \vee \cdots \vee l_q$ of φ separately. First, it determines for each model m_k the last literal satisfied by it and stores the index of the literal in the array $last[k]$ (lines 3–9). Then the literals are strengthened in turn, starting with the first one.

Suppose that the literal is positive; i.e., it is of the form $x_i \geq d$ (lines 12–20). Strengthening means to increase the value of d. Some models that satisfied the literal before might not satisfy the literal after strengthening. This is a problem only for those models for which the literal was the last possibility to make the clause true (remember that for a model to satisfy a clause it suffices to satisfy a single literal). Therefore, we choose the new value d' as the minimum of all such models (lines 14–19) and construct the new literal as $x_i \geq d'$. Negative literals are handled dually by taking the maximum (lines 21–30). If the literal is redundant, i.e., if no model depends on it as its last literal, the minimum (maximum) would have to be taken over the empty set; in this case we set d' to n (resp., -1).

Line 31 checks whether the literal is redundant. If it is not redundant, it is added to the new clause constructed to replace the old one (line 32). Finally, all models m_k satisfying the new literal are marked by setting $last[k]$ to zero (lines 33–37). As a consequence, the minimum/maximum computations for the remaining literals will ignore these models.

Example 4.2. Let $x_1 \leq 3 \vee x_2 \leq 3 \vee x_3 \leq 0 \vee x_3 \geq 4$ be the clause under consideration, and let M be as in Example 3.5. The array *last* is set to the values

	010	013	220	440	444
last	3	2	3	3	4

The first literal is eliminated since there is no model m_k such that $last[k] = 1$. The second literal is strengthened to $x_2 \leq 1$ since $d' = 1$ for $j = 2$. The last two literals remain unchanged, since the maximum for $j = 3$ is $d' = 0$ and the minimum for $j = 4$ is $d' = 4$. Hence the reduced clause is equal to $x_2 \leq 1 \vee x_3 \leq 0 \vee x_3 \geq 4$.

The PRIMALITY algorithm applied to the whole formula $\varphi(M)$ from Example 3.5

Algorithm: PRIMALITY
Input: A formula φ in conjunctive normal form and a nonempty set $M \subseteq D^\ell$ of vectors such that $\mathrm{Sol}(\varphi) = M$.
Output: A reduced prime formula φ' such that $\mathrm{Sol}(\varphi) = \mathrm{Sol}(\varphi')$.
Method:

```
1: φ' ← true
2: for all clauses c = (l_1 ∨ ··· ∨ l_q) ∈ φ do                ▷ compute the vector last
3:     for k ← 1 to |M| do
4:         for j ← 1 to q do
5:             if m_k satisfies l_j then
6:                 last[k] ← j
7:             end if
8:         end for
9:     end for
10:    c' ← false                                              ▷ greedy strengthening of literals
11:    for j ← 1 to q do
12:        if l_j is positive then
13:            let l_j = x_i ≥ d
14:            d' ← n                  ▷ d' = min({n} ∪ {m_k[i] | 1 ≤ k ≤ |M|, last[k] = j})
15:            for k ← 1 to |M| do
16:                if last[k] = j then
17:                    d' ← min(d', m_k[i])
18:                end if
19:            end for
20:            l' ← x_i ≥ d'
21:        else
22:            let l_j = x_i ≤ d
23:            d' ← −1             ▷ d' = max({−1} ∪ {m_k[i] | 1 ≤ k ≤ |M|, last[k] = j})
24:            for k ← 1 to |M| do
25:                if last[k] = j then
26:                    d' ← max(d', m_k[i])
27:                end if
28:            end for
29:            l' ← x_i ≤ d'
30:        end if
31:        if 0 ≤ d' ≤ n − 1 then
32:            c' ← c' ∨ l'
33:            for k ← 1 to |M| do
34:                if m_k satisfies l' then
35:                    last[k] ← 0
36:                end if
37:            end for
38:        end if
39:    end for
40:    φ' ← φ' ∧ c'
41: end for
42: return φ'
```

FIG. 4.1. *Reduction to a prime formula.*

returns the reduced formula

$$\begin{aligned}\varphi'(M) = {} & (x_3 \leq 0 \vee x_3 \geq 3) \wedge (x_1 \leq 0 \vee x_1 \geq 2) \wedge (x_1 \leq 2 \vee x_1 \geq 4) \\ & \wedge (x_2 \leq 1 \vee x_3 \leq 0 \vee x_3 \geq 4) \wedge (x_2 \geq 1) \wedge (x_1 \leq 0 \vee x_2 \geq 2) \\ & \wedge (x_1 \leq 2 \vee x_2 \geq 4) \wedge (x_1 \geq 2 \vee x_2 \leq 1) \wedge (x_2 \geq 4 \vee x_3 \leq 3) \\ & \wedge (x_1 \geq 4 \vee x_2 \leq 2) \wedge (x_2 \leq 1 \vee x_2 \geq 4 \vee x_3 \leq 0)\,.\end{aligned}$$

Note that $\varphi'(M)$ is a Horn formula with at most three literals per clause. □

THEOREM 4.3. *For a formula φ in conjunctive normal form (CNF) there exists a logically equivalent prime formula φ' such that* $\mathrm{Sol}(\varphi) = \mathrm{Sol}(\varphi')$*, which can be computed in time $O(|\varphi|\,|M|\,\ell \log |D|)$, where $|\varphi|$ is the number of clauses in φ.*

Proof. The time complexity directly follows from Figure 4.1. To prove the correctness of the algorithm, let c be a clause of φ and c' be the clause obtained from c by running Algorithm PRIMALITY.

We first show that the models satisfying c are the same as those satisfying c'. The lines 3–9 set $last[k]$ to a value in $\{1, \ldots, q\}$ for every k, since every model in M satisfies at least one literal in c. Moreover, $last[k]$ is set to zero if and only if the corresponding model satisfies the literal added to c'. Obviously every element of *last* will eventually be set to zero: either the model "accidentally" satisfies a new literal before the last one, or otherwise the new literal derived from the literal identified by $last[k]$ is satisfied by m_k. Therefore, we have $\mathrm{Sol}(\varphi) = \mathrm{Sol}(\varphi')$.

It remains to show that c' is prime. According to Definition 4.1 we have to prove that no literal from c' can be removed or strengthened. Consider the start of the jth iteration of the **for**-loop in lines 11–39. Construct the set $M^{(j)} = \{m_k \mid 1 \leq k \leq |M|\,, last[k] = j\}$. The models in $M^{(j)}$ satisfy none of the literals added to c' so far (otherwise their *last*-entry would have been set to zero, preventing their inclusion into the set $M^{(j)}$), nor will they satisfy any future literal since the jth literal is the last one satisfied by the models. Therefore, the literal constructed in this iteration cannot be dropped from c' without changing the set of satisfying models. Now suppose that the literal considered in this round is $l_j = x_i \geq d$ (the other case is dual). Let $m^{(j)}$ be one of the models in $M^{(j)}$ for which the ith component is minimal, i.e., $m^{(j)}[i] = d'$. It satisfies $x_i \geq d'$ but clearly no other literal $x_p \geq d_p$ satisfying $d_p > d'$. We conclude that the literals added to c' can be neither removed nor strengthened, which implies that c' is prime. □

Combining Algorithms DESCRIPTION and PRIMALITY, i.e., first describing M by means of a CNF formula $\varphi(M)$, followed by a reduction of $\varphi(M)$ to a prime formula, we get the following result.

COROLLARY 4.4. *For each set of vectors $M \subseteq D^\ell$ over a finite totally ordered domain D there exists a prime formula φ in CNF such that $M = \mathrm{Sol}(\varphi)$. The algorithm constructs φ in time $O(|M|^2\,\ell^2 \log |D|)$.*

5. Horn formulas. Horn clauses and formulas constitute a frequently studied subclass of propositional formulas. This is due to the fact that there exists a polynomial-time algorithm for deciding their satisfiability problem. It turns out that this is still the case for Horn formulas over finite domains [4], which motivates our study of their description and identification problems. As we will see, the sets of vectors described by Horn formulas are closed under the minimum operation.

A question may arise about the usefulness and practical implications of computing a Horn formula $\varphi(M)$ for a given set of vectors M whenever it is possible. Since the complexity of the description algorithm is determined by the cardinality of the

set of vectors M, it may seem superfluous to compute a Horn formula describing them. However, imagine a two-stage procedure, where first a describing formula φ is computed offline for the set of vectors M, followed by its extensive use during a second stage for online reasoning. It is obvious that we prefer a structurally simpler formula φ for the second stage reasoning process. There exist numerous examples in logic and automated deduction (see, e.g., the survey [3] in case of many-valued logics), like resolution or several tableau methods, where it is more efficient to work with Horn clauses or Horn formulas, compared with general formulas in CNF.

Problem: DESCRIPTION[HORN].

Input: A finite set of vectors $M \subseteq D^\ell$, closed under conjunction, over a finite totally ordered domain D.

Output: A Horn formula φ over D such that $\mathrm{Sol}(\varphi) = M$.

The general construction in section 3 does not guarantee that the final formula is Horn whenever the set M is closed under conjunction. For instance, there exists a Horn formula describing the set M presented in Example 2.2, but the formula $\varphi(M)$ computed by the DESCRIPTION algorithm in Example 3.5 is *not* Horn. Therefore, we must reduce the clauses of the formula φ, produced in section 3, to obtain only Horn clauses. For this, we will modify a construction proposed by Jeavons and Cooper in [21]. Their method is exponential, since it proposes to construct a Horn clause for each vector in the complement set $D^\ell \setminus M$. We will first adapt the method of Jeavons and Cooper to get a polynomial-time algorithm and then propose a more sophisticated implementation of the approach that will guarantee us an algorithm with even lower asymptotic complexity.

Let $\varphi(M)$ be a formula produced by the DESCRIPTION algorithm in section 3, and let c be a clause from $\varphi(M)$. We denote by c^- the disjunction of the negative literals in c. The vectors in M satisfying a negative literal in c also satisfy the restricted clause c^-. Hence we have only to care about the vectors that satisfy a positive literal but no negative literals in c, described by the set

$$M_c \;=\; \{m \in M \mid m \not\models c^-\}.$$

If M_c is empty, we can replace the clause c by $h(c) = c^-$ in the formula $\varphi(M)$ without changing the set of models $\mathrm{Sol}(\varphi)$. Otherwise, note that M_c is closed under conjunction, since M is already closed under this operation. Indeed, if the vectors m and m' falsify every negative literal $x \le d$ of c^-, then the conjunction $m \wedge m'$ falsifies the same negative literals. Hence M_c contains a unique minimal model $m_* = \bigwedge M_c$. Every positive literal in c satisfied by m_* is also satisfied by all vectors in M_c. Let l be a positive literal from c and satisfied by m_*. There exists at least one such literal since otherwise m_* would satisfy neither c^- nor any positive literal in c; hence it would not be in M_c. Then c can be replaced with the Horn clause $h(c) = l \vee c^-$, without changing the set of models $\mathrm{Sol}(\varphi)$. We obtain a Horn formula $h(M)$ for a Horn set M by replacing every non-Horn clause c in $\varphi(M)$ by its Horn restriction $h(c)$.

Example 5.1. Consider again the set of vectors $M = \{010, 013, 220, 440, 444\}$ and the non-Horn clause $c = (x_1 \ge 1 \vee x_2 \le 0 \vee x_2 \ge 2 \vee x_3 \le 0 \vee x_3 \ge 3)$ from Example 3.5. We have $c^- = (x_2 \le 0 \vee x_3 \le 0)$; thus any subclause of c containing c^- is already satisfied by the vectors 010, 220, and 440. We need to keep a positive literal from c in order to satisfy the reduced clause also by the vectors $013, 444 \in M$. In other words, we have $M_c = \{013, 444\}$. The unique minimal model is $m_* = 013 \wedge 444 = 013$. Since the minimal model m_* satisfies the positive literal $(x_3 \ge 3)$ in c, we can reduce the clause c to $h(c) = (x_3 \ge 3 \vee x_2 \le 0 \vee x_3 \le 0)$. ∎

The length of $h(M)$ is basically the same as that of $\varphi(M)$. The number of clauses is the same and the length of clauses is $O(\ell \log |D|)$ in both cases. There are at most 2ℓ literals in each clause of $\varphi(M)$ (one positive and one negative literal per variable) versus $\ell + 1$ literals in each clause of $h(M)$ (one negative literal per variable plus a single positive literal).

The construction of each Horn clause $h(c)$ requires time $O(|M|\,\ell \log |D|)$. Indeed, for every vector $m \in M$ we have to evaluate at most ℓ negative literals in c to find out whether m belongs to M_c. The evaluation of a literal takes time $O(\log |D|)$. Hence the computation of the set M_c takes time $O(|M|\,\ell \log |D|)$. To obtain $m_* = \bigwedge M_c$, we have to compute $|M_c|-1$ conjunctions between vectors of length ℓ, each of the ℓ conjunctions taking time $O(\log |D|)$. Therefore, m_* can also be computed in time $O(|M|\,\ell \log |D|)$. Since there are at most $2\,|M|\,\ell$ clauses in $\varphi(M)$, the transformation of $\varphi(M)$ into $h(M)$ can be done in time $O(|M|^2\,\ell^2 \log |D|)$. Hence, the whole algorithm producing the Horn formula $h(M)$ from the set of vectors M runs in time $O(|M|^2\,\ell^2 \log |D|)$.

Note that we can also use the PRIMALITY algorithm to reduce a CNF formula φ to a Horn formula $h(\varphi)$ whenever there exists a Horn formula logically equivalent to φ. The application of the PRIMALITY algorithm yields the same asymptotic time complexity as the aforementioned method according to Corollary 4.4. However, neither the application of the PRIMALITY algorithm nor the aforementioned method are asymptotically optimal.

Another interest for using the primality algorithm comes from the fact that this method does not need the assumption that M is closed under conjunction. Indeed, once we compute a prime formula describing M, we can conclude that M is Horn if the obtained prime formula is Horn. We will return to this issue in section 6 on bijunctive formulas.

We now describe a new algorithm that significantly outperforms the previous methods in terms of running time. This new algorithm is inspired by the one from [17] for the Boolean case. The basic idea is to describe the set M directly by means of a CNF formula as in section 3, but keeping only one or no positive literal in each obtained clause. For this purpose, we define the terms hmiddle(k, i), hleft$(k+1, i)$, and hright(k, i) that replace the corresponding terms defined in section 3. The replacement of the previous terms by the new ones is based on the following lemma.

LEMMA 5.2. *Let $M \subseteq D^\ell$ be a finite set of vectors closed under conjunction and let c be a clause satisfied by each model $m \in M$. Then there exists a Horn subclause $h(c)$ of c that is satisfied by every model from M.*

Proof. If c contains only one or no positive literals, then we set $h(c)$ equal to c. Otherwise, let $c = (x_i \geq a \vee x_j \geq b \vee c')$ be a clause containing two different positive literals, where $i \neq j$ since otherwise one of the two positive literals would be implied by the other and could therefore be eliminated. Assume that neither $x_i \geq a$ nor $x_j \geq b$ can be removed from c if the truth of c with respect to M has to be preserved. Then there must be two models $m, m' \in M$ satisfying the conditions $m[i] \geq a$, $m[j] < b$ and $m'[i] < a$, $m'[j] \geq b$. Moreover, neither m nor m' satisfy the rest of the clause c'. Then it can be easily seen that $(m \wedge m')[i] < a$, $(m \wedge m')[j] < b$, and that the model $m \wedge m'$ does not satisfy c'. Hence, the model $m \wedge m' \in M$ does not satisfy the clause c. This contradicts the hypothesis because M is closed under conjunction. ∎

We now define the terms for the Horn formulas by distinguishing the cases hmiddle, hleft, and hright. The conditions of their application are inherited from

section 3. The first two cases are relatively easy to determine:

$$\begin{aligned}\text{hmiddle}(k,i) &= \bigwedge_{j<i}(x_j \geq m_k[j]) \ \wedge \ (x_i > m_k[i]) \wedge (x_i < m_{k+1}[i]),\\ \text{hleft}(k+1,i) &= \bigwedge_{j<i}(x_j \geq m_{k+1}[j]) \ \wedge \ (x_i < m_{k+1}[i]).\end{aligned}$$

We can easily see that $\neg\,\text{hmiddle}(k,i)$ and $\neg\,\text{hleft}(k+1,i)$ are Horn clauses. Observe that $\neg\,\text{hleft}(k+1,i)$ implies $\neg\,\text{left}(k+1,i)$. We will show that the rightmost literal cannot be removed from $\neg\,\text{left}(k+1,i)$ when we compute the Horn subclause $\neg\,\text{hleft}(k+1,i)$. From the tree T_M and the term $\text{left}(k+1,i)$ observe that the clause $c = \neg\,\text{left}(k+1,i) \setminus (x_i \geq m_{k+1}[i])$ is falsified by at least one model in M. Therefore, there is no Horn subclause $h(c)$ of c that is satisfied by all models in M. Since M is closed under conjunction, from Lemma 5.2 it follows that there must be a Horn subclause $h(c)$ of c that is satisfied by all models of M. Hence, the clause $h(c)$ must contain the literal $x_i \geq m_{k+1}[i]$ and since it is Horn, it must be a subclause of $\neg\,\text{hleft}(k+1,i)$. Similarly for $\text{hmiddle}(k,i)$, the construction ensures that the rightmost literal cannot be removed from $\neg\,\text{middle}(k,i)$, since otherwise the clause $\neg\,\text{middle}(k,i)$ would be equal to $\neg\,\text{right}(k,i)$. Thus all other positive literals can be removed from $\neg\,\text{middle}(k,i)$ to obtain the Horn subclause $\neg\,\text{hmiddle}(k,i)$.

The construction of the term $\text{hright}(k,i)$ is more involved. Recall that for all convenient parameters k and i we have the clause

$$\neg\,\text{right}(k,i) \ = \ \bigvee_{j<i}(x_j < m_k[j] \ \vee \ x_j > m_k[j]) \ \vee \ (x_i \leq m_k[i]).$$

We look for the positive literals that can be removed from $\neg\,\text{right}(k,i)$ in order to derive the term $\text{hright}(k,i)$. For this purpose, construct the set

$$M(k,i) \ = \ \{m \in M \mid m[i] > m_k[i] \text{ and } \forall j < i, \ m[j] \geq m_k[j]\}$$

for the given parameters k and i. Clearly, $M(k,i)$ is the set of all vectors from M that falsify all negative literals in $\neg\,\text{right}(k,i)$. The set $M(k,i)$ corresponds to the previously defined set M_c for $c = \neg\,\text{right}(k,i)$.

We distinguish the cases $M(k,i) = \emptyset$ and $M(k,i) \neq \emptyset$. When the set $M(k,i)$ is empty, this means that every model $m \in M$ satisfies at least one negative literal in $\neg\,\text{right}(k,i)$. Thus we can construct $\neg\,\text{hright}(k,i)$ from $\neg\,\text{right}(k,i)$ by removing all positive literals. In other words, $\text{hright}(k,i)$ is obtained from $\text{right}(k,i)$ by removing all negative literals.

When the set $M(k,i)$ is nonempty, we know from Lemma 5.2 that there exists a positive literal in $\neg\,\text{right}(k,i)$ which is satisfied by all models in $M(k,i)$. This amounts to the computation of intersection $\bigwedge M(k,i)$ followed by a choice of a model from it, but we need to do it in a more sophisticated way than in the previous Horn method if we wish to obtain an algorithm with lower asymptotic complexity. Let us define a function that will compute the position of the kept literal:

$$\text{pos}(k,i) \ = \ \max_{1\leq j\leq k}\{j \mid \exists m \in M(k,i) \text{ such that } \forall p, \ p < j \text{ implies } m[p] \leq m_k[p]\}.$$

Note that since every model $m \in M(k,i)$ satisfies the clause $\neg\,\text{right}(k,i)$ and thus at least one of its positive literals, the condition $\text{pos}(k,i) < i$ is satisfied.

Algorithm: POS
Input: Nonempty set $M \subseteq D^\ell$ of vectors and a parameter k.
Output: The position function $pos(k)$.
Method:

```
 1: for i ← 1 to ℓ do
 2:     pos(k)[i] ← 0
 3: end for
 4: for k' ← 1 to |M| do
 5:     if k' ≠ k then
 6:         i_0 ← 1
 7:         while i_0 < ℓ and m_{k'}[i_0] ≤ m_k[i_0] do
 8:             i_0 ← i_0 + 1
 9:         end while
10:         i ← 1
11:         while i ≤ ℓ and m_{k'}[i] ≥ m_k[i] do
12:             if i > fork(k) and m_k[i] < |D| − 1 and m_{k'}[i] > m_k[i] then
13:                 pos(k)[i] ← max(pos(k)[i], i_0)
14:             end if
15:             i ← i + 1
16:         end while
17:     end if
18: end for
19: return pos(k)
```

FIG. 5.1. *Position function* $\mathrm{pos}(k,i)$.

The term $\mathrm{hright}(k,i)$ is defined as follows:

$$\mathrm{hright}(k,i) = \begin{cases} \bigwedge_{j<i}(x_j \geq m_k[j]) \ \wedge \ (x_i > m_k[i]) & \text{if } M(k,i) = \emptyset, \\ \bigwedge_{j<i}(x_j \geq m_k[j]) \ \wedge \ (x_i > m_k[i]) \ \wedge \ (x_{\mathrm{pos}(k,i)} \leq m_k[\mathrm{pos}(k,i)]) & \\ \qquad \text{otherwise.} & \end{cases}$$

We can finally present the Horn formula $h(M)$ for a set of vectors M closed under conjunction that satisfies the equality $\mathrm{Sol}(h(M)) = M$. It resembles the formula $\varphi(M)$ from section 3 modulo some syntactic changes.

$$\begin{aligned} h(M) \quad &= \quad \bigwedge\{ \neg\,\mathrm{hmiddle}(k,i) \mid 0 < k < |M|\,,\ i = \mathrm{fork}(k),\ m_k[i]+1 < m_{k+1}[i] \} \\ &\wedge \quad \bigwedge\{ \neg\,\mathrm{hleft}(k+1,i) \mid 0 \leq k < |M|\,,\ \mathrm{fork}(k) < i \leq \ell,\ m_{k+1}[i] > 0 \} \\ &\wedge \quad \bigwedge\{ \neg\,\mathrm{hright}(k,i) \quad \mid 0 < k \leq |M|\,,\ \mathrm{fork}(k) < i \leq \ell,\ m_k[i] < |D|-1 \}. \end{aligned}$$

Our algorithm computing the Horn formula $h(M)$ for a given set of vectors M is equivalent to Algorithm DESCRIPTION, where the terms $\mathrm{middle}(k,i)$, $\mathrm{left}(k+1,i)$, and $\mathrm{right}(k,i)$ are replaced by the terms $\mathrm{hmiddle}(k,i)$, $\mathrm{hleft}(k+1,i)$, and $\mathrm{hright}(k,i)$, respectively. Computing the clauses $\neg\,\mathrm{hmiddle}(k,i)$ and $\neg\,\mathrm{hleft}(k+1,i)$ does not pose any problems, whereas the clause $\neg\,\mathrm{hright}(k,i)$ heavily depends on the position function $\mathrm{pos}(k,i)$ which must be computed efficiently. For a given k, Algorithm POS in Figure 5.1 computes the values of $\mathrm{pos}(k,i)$ for all positions $i = 1,\ldots,\ell$ at the same

time. This is what makes our algorithm efficient. The result is returned in the form of an array $pos(k)$, where the equality $pos(k)[i] = \mathrm{pos}(k, i)$ holds for all positions i.

The algorithm is based on the following principle. Given a vector m_k, it considers every vector $m_{k'} \in M$ different from m_k. For a pair of vectors m_k and $m_{k'}$, the algorithm considers each index i in increasing order. While the condition $m_{k'}[i] \geq m_k[i]$ (line 11) holds, we are sure that the model $m_{k'}$ belongs to $M(k, i)$, and therefore $m_{k'}$ is taken into account for the value of $\mathrm{pos}(k, i)$ (line 13). On the other hand, as soon as the condition does not hold any more, then we are sure that for all $j > i$ the vector $m_{k'}$ does not belong to $M(k, i)$, and therefore $m_{k'}$ does not need to be taken into account for the value of $\mathrm{pos}(k, i)$. Note that the value $pos[k, i]$ has no meaning for a nonconvenient i, and therefore it is set to 0, which also means $M(k, i) = \emptyset$.

Example 5.3. We already know from Example 4.2 that there exists a Horn formula describing the set of vectors $M = \{010, 013, 220, 440, 444\}$. Let us transform the terms middle, left, and right from Example 3.3 to hmiddle, hleft, and hright, respectively. We will get the middle terms

$$\begin{aligned}
\mathrm{hmiddle}(1,3) &= (x_1 \geq 0) \wedge (x_2 \geq 1) \wedge (x_3 > 0) \wedge (x_3 < 3),\\
\mathrm{hmiddle}(2,1) &= (x_1 > 0) \wedge (x_1 < 2),\\
\mathrm{hmiddle}(3,1) &= (x_1 > 2) \wedge (x_1 < 4),\\
\mathrm{hmiddle}(4,3) &= (x_1 \geq 4) \wedge (x_2 \geq 4) \wedge (x_3 > 0) \wedge (x_3 < 4)
\end{aligned}$$

and the left terms

$$\begin{aligned}
\mathrm{hleft}(1,2) &= (x_1 \geq 0) \wedge (x_2 < 1),\\
\mathrm{hleft}(3,2) &= (x_1 \geq 2) \wedge (x_2 < 2),\\
\mathrm{hleft}(4,2) &= (x_1 \geq 4) \wedge (x_2 < 4)
\end{aligned}$$

easily. To transform the terms $\mathrm{right}(2,2)$, $\mathrm{right}(3,2)$, and $\mathrm{right}(3,3)$ to $\mathrm{hright}(2,2)$, $\mathrm{hright}(3,2)$, and $\mathrm{hright}(3,3)$, respectively, we need first to compute the arrays $pos(2)$ and $pos(3)$. We get

pos	1	2	3
2	0	1	1
3	0	1	1

which implies the terms

$$\begin{aligned}
\mathrm{hright}(2,2) &= (x_1 \geq 0) \wedge (x_2 > 1) \wedge (x_1 \leq 0),\\
\mathrm{hright}(2,3) &= (x_1 \geq 0) \wedge (x_2 \geq 1) \wedge (x_3 > 3) \wedge (x_1 \leq 0),\\
\mathrm{hright}(3,2) &= (x_1 \geq 2) \wedge (x_2 > 2) \wedge (x_1 \leq 2),\\
\mathrm{hright}(3,3) &= (x_1 \geq 2) \wedge (x_2 \geq 2) \wedge (x_3 > 0) \wedge (x_1 \leq 2).
\end{aligned}$$

The final Horn formula will be

$$\begin{aligned}
h(M) &= (x_2 \leq 0 \vee x_3 \leq 0 \vee x_3 \geq 3) \wedge (x_1 \leq 0 \vee x_1 \geq 2) \wedge (x_1 \leq 2 \vee x_1 \geq 4)\\
&\wedge\ (x_1 \leq 3 \vee x_2 \leq 3 \vee x_3 \leq 0 \vee x_3 \geq 4) \wedge (x_2 \geq 1) \wedge (x_1 \leq 1 \vee x_2 \geq 2)\\
&\wedge\ (x_1 \leq 3 \vee x_2 \geq 4) \wedge (x_1 \geq 1 \vee x_2 \leq 1) \wedge (x_1 \geq 1 \vee x_2 \leq 0 \vee x_3 \leq 3)\\
&\wedge\ (x_1 \leq 1 \vee x_1 \geq 3 \vee x_2 \leq 2) \wedge (x_1 \leq 1 \vee x_1 \geq 3 \vee x_2 \leq 1 \vee x_3 \leq 0). \qquad \square
\end{aligned}$$

THEOREM 5.4. *For each set of vectors $M \subseteq D^\ell$ over a finite totally ordered domain D that is closed under conjunction, there exists a Horn formula φ such that $M = \mathrm{Sol}(\varphi)$. The formula φ contains at most $2\,|M|\,\ell$ clauses, its length is $O(|M|\,\ell^2 \log |D|)$, and it can be computed in time $O(|M|\,\ell(|M| + \ell) \log |D|)$.*

Proof. Time complexity is straightforward since we essentially run the DESCRIPTION algorithm from section 3 with the computation of $pos(k)$ added. As for correctness, it is a consequence of the previously written reasoning in this section. ∎

Using Theorem 5.4, we are able to prove a generalization of a well-known characterization of the set of models $\mathrm{Sol}(\varphi)$ of a Horn formula φ. We wish to point out that this characterization is not new and was proved before. Indeed, a related characterization in a different setting can be found in [15], and a similar proof can be found in [21]. We mention the result here for completeness.

PROPOSITION 5.5. *A set of vectors M over a finite totally ordered domain is closed under conjunction if and only if there exists a Horn formula φ satisfying the identity $\mathrm{Sol}(\varphi) = M$.*

Proof. Theorem 5.4 shows that there exists a Horn formula φ describing a set of vectors M if M is closed under conjunction. It remains to show the converse, namely, that the set $\mathrm{Sol}(\varphi)$ is closed under conjunction if φ is a Horn formula. We need to show that for any two models m and m' of φ, their conjunction $m \wedge m'$ is also a model of φ. A model satisfies a Horn formula φ if and only if it satisfies every clause of φ. Therefore, we need to show for each clause c that $m \wedge m'$ satisfies c whenever both m and m' satisfy c. We distinguish two cases.

(i) Clause c contains a positive literal $x \geq d$ that is satisfied by both m and m'. Then we have $m(x) \geq d$ and $m'(x) \geq d$, which implies $(m \wedge m')(x) = \min(m(x), m'(x)) \geq d$. This proves that $m \wedge m'$ satisfies the clause c.

(ii) Clause c contains no positive literal satisfied by both m and m'. Then at least one model, say, m, must satisfy a negative literal $x \leq d$ in c, i.e., $m(x) \leq d$. We obtain $(m \wedge m')(x) = \min(m(x), m'(x)) \leq m(x) \leq d$. Hence also $m \wedge m'$ satisfies c. ∎

If we interchange conjunctions with disjunctions of models, as well as positive and negative literals throughout section 5, we obtain identical results for dual Horn formulas.

THEOREM 5.6. *A set of vectors $M \subseteq D^\ell$ over a finite ordered domain D is closed under disjunction if and only if there exists a dual Horn formula φ satisfying the identity $M = \mathrm{Sol}(\varphi)$. Given M closed under disjunction, the dual Horn formula φ contains at most $2\,|M|\,\ell$ clauses, and its length is $O(|M|\,\ell^2 \log |D|)$. It can be constructed in time $O(|M|\,\ell(|M| + \ell) \log |D|)$.*

6. Bijunctive formulas. Bijunctive clauses and formulas present another frequently studied subclass of propositional formulas, once more because there exists a polynomial-time algorithm for deciding their satisfiability which generalizes to the finite-domain case [5]. We investigate in this section the description problem for a generalization of bijunctive formulas to ordered finite domains, namely, for sets of vectors closed under the median operation.

Problem: DESCRIPTION[BIJUNCTIVE].
Input: A finite set of vectors $M \subseteq D^\ell$, closed under median, over a finite totally ordered domain D.
Output: A bijunctive formula φ over D such that $\mathrm{Sol}(\varphi) = M$.

Once again, the general construction in section 3 does not guarantee that the final formula is bijunctive whenever the set M is closed under median. Therefore,

we add a postprocessing step that transforms the formula φ into a bijunctive one $b(\varphi)$. Let $\varphi(M)$ be the formula produced by the method of section 3, and let c be a clause from $\varphi(M)$. We construct a bijunctive restriction $b(\varphi)$ by removing appropriate literals from φ such that no more than two literals remain in each clause. Since φ is a CNF, any model of $b(\varphi)$ is still a model of φ. The converse does not hold in general. However, if $\mathrm{Sol}(\varphi)$ is closed under median, the method presented below preserves the models; i.e., every model of φ remains a model of $b(\varphi)$. In the proof we need the following simple lemma.

LEMMA 6.1. *The model* $\mathrm{med}(m_1, m_2, m_3)$ *satisfies a literal* l *if and only if at least two of the models* m_1*,* m_2*, and* m_3 *satisfy* l*.*

Proof. Recall the identities $\mathrm{med}(m_1, m_2, m_3) = (m_1 \vee m_2) \wedge (m_2 \vee m_3) \wedge (m_3 \vee m_1) = (m_1 \wedge m_2) \vee (m_2 \wedge m_3) \vee (m_3 \wedge m_1)$. Recall also that $m \wedge m'$ and $m \vee m'$ are shorthand for the more cumbersome prefix notation $\min(m, m')$ and $\max(m, m')$, respectively. Let l be satisfied by at least two models, say, m_1 and m_2. If the literal l is positive, then it is also satisfied by the models $m_1 \vee m_2$, $m_2 \vee m_3$, and $m_3 \vee m_1$. If the literal l is negative, then it is also satisfied by the models $m_1 \wedge m_2$, $m_2 \wedge m_3$, and $m_3 \wedge m_1$. Hence, in both cases, l is also satisfied by $\mathrm{med}(m_1, m_2, m_3)$.

Conversely, if l is satisfied only by one model, say, m_1, or if l is not satisfied by any of the three models, then it is falsified by the model $m_2 \vee m_3$ if l is positive and by the model $m_2 \wedge m_3$ if l is negative. Hence, the literal l cannot be satisfied by $\mathrm{med}(m_1, m_2, m_3)$. □

DEFINITION 6.2. *We say that a literal* l *is* essential *for a clause* c *with respect to a set of models* M *if there is a model* $m \in M$ *that satisfies* l*, but no other literal in* c*. We also say that* m *is a* justification *for* l *with respect to* M*.*

Obviously, we may remove nonessential literals from c without losing models. It remains to show that no clause from φ contains more than two essential literals.

To derive a contradiction, suppose that c is a clause from φ containing at least three essential literals, say, l_1, l_2, and l_3. Let m_1, m_2, and m_3 be their justifications; i.e., for each i we have $m_i \models l_i$ and m_i does not satisfy any other literal in c. According to Lemma 6.1, in this case the model $\mathrm{med}(m_1, m_2, m_3)$ satisfies no literal at all. Hence $\mathrm{med}(m_1, m_2, m_3)$ satisfies neither c nor φ, which contradicts the assumption that $\mathrm{Sol}(\varphi)$ is closed under median.

The previous discussion suggests applying the following algorithm to every clause c of φ. For every literal l in $c = c' \vee l$, check whether the remaining clause c' is still satisfied by all models in M. If the answer is yes, the literal is not essential and can be removed. Otherwise, it is one of the (at most) two literals in the final bijunctive clause $b(c)$. As we can easily see, this operation is performed by the PRIMALITY algorithm from section 4.

THEOREM 6.3. *For each set of vectors* $M \subseteq D^\ell$ *over a finite ordered domain* D *that is closed under median, there exists a bijunctive formula* φ *such that* $M = \mathrm{Sol}(\varphi)$*. The length of* φ *is* $O(|M|\,\ell(\log \ell + \log |D|))$*, and it contains at most* $2\,|M|\,\ell$ *clauses. The algorithm constructing* φ *runs in time* $O(|M|^2\,\ell^2 \log |D|)$*.*

Proof. The main part of the result follows from Theorems 3.6 and 4.3. Concerning the length of φ, note that each clause contains at most two literals. Each literal consists of a variable and one domain element. Hence we can represent a literal by the index of its variable and the domain value, both written in binary. Therefore, the length of a literal and subsequently of each bijunctive clause is $O(\log \ell + \log |D|)$. Since there are at most $2\,|M|\,\ell$ clauses, this implies the length of φ. □

Similarly to the Horn and dual Horn formulas, we get a nice relation between bijunctive formulas by means of a closure property.

PROPOSITION 6.4. *A set of vectors M over a finite totally ordered domain is closed under median if and only if there exists a bijunctive formula φ satisfying the identity $M = \mathrm{Sol}(\varphi)$.*

Proof. Theorem 6.3 shows that there exists a bijunctive formula for every set of vectors M closed under median. It remains to show the converse, namely, that $\mathrm{Sol}(\varphi)$ is closed under median if φ is a bijunctive formula. Since φ is a conjunction of clauses, it is sufficient to show the closure property for a bijunctive clause $c = l \vee l'$. Let m_1, m_2, and m_3 be three models of c. From the pigeonhole principle it follows that one of the two literals l or l' of the clause c is satisfied by at least two models. Hence, by Lemma 6.1, at least one of the two literals l and l', and therefore also the clause c, is satisfied by $\mathrm{med}(m_1, m_2, m_3)$. □

We wish to point out that contrary to the Horn case, the most efficient known algorithm for the bijunctive description problem does not seem to lift well from the Boolean to the finite domain. Dechter and Pearl [11] showed that in the Boolean case this problem can be solved in time $O(|M|\,\ell^2)$, which is better than our result even when ignoring the unavoidable factor $\log|D|$. Their algorithm generates first all the $O(\ell^2)$ bijunctive clauses built from the variables of the formula, followed by an elimination of those falsified by a vector from M, where the bijunctive formula is the conjunction of the retained clauses. However, there are $O(\ell^2\,|D|^2)$ bijunctive clauses for a finite domain D yielding an algorithm with time complexity $O(|M|\,\ell^2\,|D|^2)$, which is exponential in the size $O(\log|D|)$ of the domain elements.

Another idea, not applicable more efficiently in the finite domain case, is that of projecting M onto each pair of variables and then computing a bijunctive formula for each projection. This requires time $O(|M|\,\ell^2)$ in the Boolean case, since we need only to compute a CNF for each projection. A CNF for a projection is always bijunctive; thus only the general Algorithm DESCRIPTION has to be used. However, in the finite domain case, computing a formula with Algorithm DESCRIPTION does not necessarily yield a bijunctive CNF. Each clause can contain up to four literals, a positive and a negative one for each variable. Thus we need to use an algorithm for computing a *bijunctive* CNF, like that of Theorem 6.3, yielding an overall time complexity of $O(|M|^2\,\ell^2\log|D|)$.

Finally, let us return to the identification problem. As was mentioned in the introduction, our results present an elegant and unified algorithm for identification of Horn, dual Horn, and bijunctive sets of vectors. Given a set of vectors M, the method presented by the PRIMALITY algorithm computes a prime formula φ satisfying the identity $\mathrm{Sol}(\varphi) = M$. Then we check in linear time whether it is Horn, dual Horn, or bijunctive. The results in sections 5 and 6 ensure that the resulting formula φ is Horn, dual Horn, or bijunctive if and only if M is a Horn, dual Horn, or bijunctive set of vectors, respectively. This generalizes the result presented by Zanuttini and Hébrard in [25].

7. Changing the literals. We finish the paper with a short discussion of the description problems for a slightly different formalism, where only the literals are changed.

If we change the underlying notion of literals, using the expressions $x = d$ and $x \neq d$ as basic building blocks, the situation changes drastically. Former positive literal $x \geq d$ becomes shorthand for the disjunction $(x = d) \vee (x = d+1) \vee \cdots \vee (x = n-1)$, whereas the former negative literal $x \leq d$ now represents the disjunction $(x = 0) \vee (x = 1) \vee \cdots \vee (x = d)$. Even if we compress literals containing the same variable into a bit vector, the new representation still needs n bits; i.e., its size is $O(n)$.

Compared to the former literals of size $O(\log n)$, this amounts to an exponential blow-up. As an immediate consequence, the algorithms given in the preceding sections become exponential, since we have to replace literals like $x_i < m_k[i]$, $x_i > m_k[i]$, and $x_i < m_{k+1}[i]$ by disjunctions of equalities.

The satisfiability problem for formulas in CNF over finite totally ordered domains with basic operators $\leq$ and $\geq$ is defined similarly to Boolean satisfiability. The complexity of these problems was studied for fixed domain cardinalities, from the standpoint of many-valued logics, by Béjar, Hähnle, and Manyà [6] and Hähnle [16]. The NP-completeness proof for Boolean satisfiability generalizes uniformly to finite ordered domains. Béjar, Hähnle, and Manyà [6] and Hähnle [16] proved that the satisfiability problems restricted to Horn, dual Horn, and bijunctive formulas are decidable in polynomial time for a fixed domain cardinality. These algorithms can be generalized to arbitrary domain cardinalities, adding only the unavoidable factor $\log |D|$.

The satisfiability of formulas in CNF is also affected when switching to $=$ and $\neq$ as basic operators. While the satisfiability problem for general formulas remains NP-complete, the restrictions to Horn, dual Horn, and bijunctive formulas change from polynomially solvable to NP-complete for $|D| \geq 3$. This can be shown by encoding, for example, the graph problem of k-COLORING [2, 9]. When we use the Horn and bijunctive clause $(u \neq d \vee v \neq d)$, we can express by $C(u, v) = (u \neq 0 \vee v \neq 0) \wedge \cdots \wedge (u \neq k-1 \vee v \neq k-1)$ that the adjacent vertices of the edge (u, v) are "colored" by different "colors." On the other hand, Beckert, Hähnle, and Manyà [5] proved that bijunctive formulas restricted to positive literals can be solved in linear time.

8. Concluding remarks. The studied formula description problems constitute a generalization of the Boolean structure identification problems, studied by Dechter and Pearl [11], with more efficient algorithms as a byproduct. Our paper presents a complement to the work of Hähnle et al. [5, 6, 16] on the complexity of the satisfiability problems in many-valued logics. It also completes the study of tractable formulas [9, 20, 21] by Jeavons and his group.

We have constructed an efficient polynomial-time algorithm for the formula description problem over a finite totally ordered domain, where the produced formula is in CNF. We have then presented a subsequent algorithm that eliminates in polynomial time redundancies from the previously computed formula, given the original set of vectors, producing in this way the prime formula. The notion of primality that we use is an extension of the same notion used in Boolean formulas. It not only captures irrelevant literals in clauses but also strengthens the value d in the literals $x \leq d$ or $x \geq d$, respectively. If the original set of vectors is closed under the operation of conjunction, disjunction, or median, we have presented specific algorithms that produce a Horn, a dual Horn, or a bijunctive formula, respectively. These algorithms generalize the well-known ones from the Boolean domain. It is interesting to note that they are compatible, with respect to asymptotic complexity, with known algorithms for the Boolean case presented in [17, 25]. This means that the restriction of the new algorithms presented in our paper to domains D with cardinality $|D| = 2$ produces the aforementioned algorithms for the Boolean case. We also found that in the case when the finite domain is totally ordered, the three well-known special cases, namely, Horn, dual Horn, and bijunctive, display the same behavior as in the Boolean case: (1) they have polynomial satisfiability algorithms and (2) they have the same closure properties.

It would be interesting to know if more efficient algorithms exist or whether

our algorithms are asymptotically optimal. Certainly a more involved lower bound analysis is necessary to answer this open question. Another possible extension of our work would be a generalization of our algorithms to partially ordered domains and to domains with a different structure, like lattices. An additional direction for future work is to look at infinite domains. Some related complexity results for satisfiability problems in the infinite domain case can be found in [4].

REFERENCES

[1] J. Amilhastre, H. Fargier, and P. Marquis, *Consistency restoration and explanations in dynamic CSPs—application to configuration*, Artificial Intelligence, 135 (2002), pp. 199–234.

[2] C. Ansótegui and F. Manyà, *New logical and complexity results for signed-SAT*, in Proceedings of the 33rd IEEE International Symposium on Multiple-Valued Logic (ISMVL 2003), Tokyo, Japan, 2003, IEEE Computer Society, Washington, DC, 2003, pp. 181–187.

[3] M. Baaz, C. G. Fermüller, and G. Salzer, *Automated deduction for many-valued logics*, in Handbook of Automated Reasoning, Vol. 2, J. A. Robinson and A. Voronkov, eds., Elsevier Science, New York, 2001, pp. 1355–1402.

[4] B. Beckert, R. Hähnle, and F. Manyà, *Transformations between signed and classical clause logic*, in Proceedings of the 29th IEEE International Symposium on Multiple-Valued Logic (ISMVL 1999), Freiburg im Breisgau, Germany, 1999, IEEE Computer Society, Washington, DC, 1999, pp. 248–255.

[5] B. Beckert, R. Hähnle, and F. Manyà, *The 2-SAT problem of regular signed CNF formulas*, in Proceedings of the 30th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2000), Portland, OR, 2000, IEEE Computer Society, Washington, DC, 2000, pp. 331–336.

[6] R. Béjar, R. Hähnle, and F. Manyà, *A modular reduction of regular logic to classical logic*, in Proceedings of the 31st IEEE International Symposium on Multiple-Valued Logic (ISMVL 2001), Warsaw, Poland, 2001, IEEE Computer Society, Washington, DC, 2001, pp. 221–226.

[7] A. A. Bulatov, *A dichotomy theorem for constraints on a three-element set*, in Proceedings of the 43rd ACM Symposium on Foundations of Computer Science (FOCS 2002), Vancouver, British Columbia, Canada, 2002, pp. 649–658.

[8] A. A. Bulatov, *Tractable conservative constraint satisfaction problems*, in Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS 2003), Ottawa, Canada, 2003, pp. 321–330.

[9] M. C. Cooper, D. A. Cohen, and P. Jeavons, *Characterising tractable constraints*, Artificial Intelligence, 65 (1994), pp. 347–361.

[10] N. Creignou, S. Khanna, and M. Sudan, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, SIAM Monographs on Discrete Mathematics and Applications 7, SIAM, Philadelphia, 2001.

[11] R. Dechter and J. Pearl, *Structure identification in relational data*, Artificial Intelligence, 58 (1992), pp. 237–270.

[12] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[13] A. Gil, M. Hermann, G. Salzer, and B. Zanuttini, *Efficient algorithms for constraint description problems over finite totally ordered domains*, in Proceedings of the 2nd International Joint Conference on Automated Reasoning (IJCAR'04), Cork, Ireland, 2004, Lecture Notes in Comput. Sci. 3097, D. Basin and M. Rusinowitch, eds., Springer-Verlag, New York, 2004, pp. 244–258.

[14] R. Hähnle, *Short conjunctive normal forms in finitely valued logics*, J. Logic Comput., 4 (1994), pp. 905–927.

[15] R. Hähnle, *Exploiting data dependencies in many-valued logics*, J. Appl. Non-Classical Logics, 6 (1996), pp. 49–69.

[16] R. Hähnle, *Complexity of many-valued logics*, in Proceedings of the 31st IEEE International Symposium on Multiple-Valued Logic (ISMVL 2001), Warsaw, Poland, 2001, IEEE Computer Society, Washington, DC, 2001, pp. 137–148.

[17] J.-J. Hébrard and B. Zanuttini, *An efficient algorithm for Horn description*, Inform. Process. Lett., 88 (2003), pp. 177–182.

[18] P. Hell and J. Nešetřil, *Graphs and Homomorphisms*, Oxford University Press, Oxford, UK, 2004.
[19] P. Jeavons, *On the algebraic structure of combinatorial problems*, Theoret. Comput. Sci., 200 (1998), pp. 185–204.
[20] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.
[21] P. Jeavons and M. C. Cooper, *Tractable constraints on ordered domains*, Artificial Intelligence, 79 (1995), pp. 327–339.
[22] P. G. Kolaitis and M. Y. Vardi, *Conjunctive-query containment and constraint satisfaction*, J. Comput. System Sci., 61 (2000), pp. 302–332.
[23] N. V. Murray and E. Rosenthal, *Adapting classical inference techniques to multiple-valued logics using signed formulas*, Fund. Inform., 21 (1994), pp. 237–253.
[24] T. J. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th ACM Symposium on Theory of Computing (STOC'78), San Diego, CA, 1978, pp. 216–226.
[25] B. Zanuttini and J.-J. Hébrard, *A unified framework for structure identification*, Inform. Process. Lett., 81 (2002), pp. 335–339.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 946–962

APPROXIMATING THE SPANNING STAR FOREST PROBLEM AND ITS APPLICATION TO GENOMIC SEQUENCE ALIGNMENT*

C. THACH NGUYEN†, JIAN SHEN‡, MINMEI HOU§, LI SHENG¶, WEBB MILLER‖, AND LOUXIN ZHANG**

Abstract. This paper studies the algorithmic issues of the spanning star forest problem. We prove the following results: (1) There is a polynomial-time approximation scheme for planar graphs; (2) there is a polynomial-time $\frac{3}{5}$-approximation algorithm for graphs; (3) it is NP-hard to approximate the problem within ratio $\frac{259}{260}+\epsilon$ for graphs; (4) there is a linear-time algorithm to compute the maximum star forest of a weighted tree; (5) there is a polynomial-time $\frac{1}{2}$-approximation algorithm for weighted graphs. We also show how to apply this spanning star forest model to aligning multiple genomic sequences over a tandem duplication region.

Key words. dominating set, spanning star forest, approximation algorithm, genomic sequence alignment

AMS subject classifications. 68Q17, 68Q25, 68R10, 68W25

DOI. 10.1137/070682150

1. Introduction.

A star is a tree in which some vertex is incident to each of the edges in the graph. Let G be a graph. A spanning star forest SF_G of a graph G is a spanning subgraph of G in which each connected component is a star. The size of SF_G is defined to be the number of edges in SF_G; if G is weighted, the size of SF_G is defined to be the sum of the weights of all edges in SF_G. Even in a graph, spanning star forests may have different sizes. As in any forest, the size of a spanning star forest of a graph G is equal to the number of vertices of G minus the number of stars in the star forest. We call the vertex that is incident to all edges the center of a star. A subset of vertices dominates G if every other vertex is adjacent to at least one vertex in the subset. It is not hard to see that the centers of the stars in a spanning star forest form a dominating set for G. Hence, finding a maximum spanning star forest of a graph is equivalent to finding a smallest dominating set. The latter is a fundamental problem in algorithmic graph theory.

Although the size of a spanning star forest of a graph and its relationship to the dominating number have been observed [13], the problem of finding a maximum

*Received by the editors February 8, 2007; accepted for publication (in revised form) February 25, 2008; published electronically June 6, 2008. The extended abstract of this work appeared in the Proceedings of the 18th Annual SIAM–ACM Symposium on Discrete Algorithms (SODA'07).
http://www.siam.org/journals/sicomp/38-3/68215.html

†Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350 (ncthach@cs.washington.edu). This work was done when this author was at the National University of Singapore. This author was supported by NUS ARF grant R-146-000-068-112.

‡Department of Mathematics, Texas State University, San Marcos, TX 78666 (js48@txstate.edu).

§Department of Computer Science, Northern Illinois University, DeKalb, IL 60115 (mhou@cs.niu.edu). This work was done when this author studied at Penn State University. This author was supported by NIH grant HG002238.

¶Department of Mathematics, Drexel University, Philadelphia, PA 19104 (lsheng@math.drexel.edu). This author was supported by NSF grant CCR-0311413.

‖Department of Biology and Department of Comput. Sci. & Eng., Penn State University, University Park, PA 16802 (webb@bx.psu.edu). This author was supported by NIH grant HG002238.

**Department of Mathematics, National University of Singapore, Singapore 117543 (matzlx@nus.edu.sg). This author was supported by NUS ARF grant R-146-000-068-112.

spanning star forest of a graph has yet to be well studied. Our work focuses on this algorithmic problem.

Our motivation for studying this problem comes from aligning multiple genomic sequences, a basic bioinformatics task in comparative genomics. A speciation is an evolutionary process that creates new species. Two genes from different species are orthologous if they diverged as the result of a speciation event. A duplication is an evolutionary event in which a genomic segment is copied and inserted at a different position. Two related genes could have diverged due to an intraspecies duplication event. To perform comparative genome analysis, it is desirable to produce a multispecies orthologous alignment, in which there is at most one row of sequence from a given species in each alignment block. The many-to-many orthologous relationships among duplicated genes causes a dramatic explosion of the alignment size when the multispecies alignment contains all combinations of pairwise orthologous relationships [16].

Currently, there are no good solutions to aligning duplication-rich genomic regions. The existing threaded blockset aligner (TBA) program screens out duplicated alignments and thus is not able to capture all pairwise orthologous relationships in a duplicated-gene cluster [7]. For example, using TBA, each human alpha-globin gene is aligned to only one rat alpha-globin gene, despite the fact that a human alpha-globin gene is actually orthologous to several rat alpha-globin genes, and those other alignments are lost. To control the size of the computed alignment blocks while guaranteeing the alignment quality, we propose to define a so-called alignment graph using the pairwise similarity of the given sequences and then utilize a maximum spanning star forest of the resulting alignment graph as a guide for building alignment blocks (see section 5 for details).

In addition, the spanning star forest problem has potential application in comparison of phylogenetic trees [6]. A directed star is defined to be outward if all arcs are from the center to a leaf. The directed version of the spanning star forest problem is, given a directed graph, to find a maximum spanning subgraph in which each connected component is an outward star. Such a directed version of the spanning star forest problem arises from the diversity problem in the automobile industry [1].

The dominating set problem is a well-known NP-hard problem. Because of this, the problem of finding a maximum spanning star forest is NP-hard in general but is polynomial-time solvable for trees and has a polynomial-time approximation scheme (PTAS) for planar graphs, as indicated in section 3.1. In section 3.2, we also present a polynomial-time algorithm of approximation ratio $\frac{3}{5}$ for any graph. On the other hand, we prove in section 3.3 that it is NP-hard to approximate the problem within ratio $\frac{259}{260} + \epsilon$ for any small $\epsilon > 0$.

For weighted graphs, the spanning star forest problem is not equivalent to the dominating set problem. In section 4, we present a dynamic programming algorithm for weighted trees; we also give a simple polynomial-time algorithm of approximation ratio $\frac{1}{2}$ for arbitrary weighted graphs.

Finally, in section 5, we show how to apply this spanning star forest model to aligning multiple genomic sequences over a duplicated region.

2. The spanning star forest problem. In this paper, we consider simple graphs that are undirected and connected, weighted or unweighted. We simply say that G is a graph if it is unweighted. A *star* is a tree having a vertex (called the center) incident to each of all edges in the graph. The center of a star S has the maximum degree. If a star has only two vertices, either vertex can be its center. A

star forest is a graph in which each component is a star.

Star forests have previously appeared in the literature on star arboricity [2]. The star arboricity of a graph G is the minimum number of star forests whose union contains all edges of G. Bounds on star arboricity have been established for several classes of graphs including planar graphs [3, 4, 11, 14].

The *size* of a graph G is the number of edges in G, and it is the sum of the edge weights if G is weighted. A *spanning star forest* of a graph G is a star forest that contains all the vertices of G. Different spanning star forests of G may have different sizes. In the rest of paper, we study the following algorithmic problem:

Spanning Star Forest.

INSTANCE: A (unweighted or weighted) graph $G = (V, E)$.

OBJECTIVE: Find a spanning star forest of G that has the largest size.

As we shall prove in the next section, this problem is NP-hard. Hence, we shall focus on developing approximation algorithms for it. An approximation algorithm for the spanning star forest problem has approximation ratio $r < 1$ if it always outputs a spanning star forest of size at least $r \cdot \mathrm{OPT}_{sf}(G)$ given a graph G, where $\mathrm{OPT}_{sf}(G)$ is the maximum size of a spanning star forest of G. We call such an algorithm an r-approximation algorithm.

3. Algorithms for unweighted graphs.

3.1. Spanning star forest and dominating set. The dominating set is one of the most important concepts in graph theory [15]. Given a graph $G = (V, E)$, a subset of vertices $D \subseteq V$ is called a *dominating set* if, for every $v \in V - D$, there is at least one vertex $u \in D$ that is adjacent to it, i.e., $(u, v) \in E$. Assume $D = \{v_1, v_2, \ldots, v_k\}$ is a k-vertex dominating set of G. For each $u \in V - D$, we select a unique vertex $v_u \in D$ such that $(u, v_u) \in E$ and associate the edge (u, v_u) to v_u. For each $d \in D$, all the associated edges of the form (v, d) give rise to a star S_d centered at d. Trivially, $\cup_{d \in D} S_d$ is a spanning star forest of G. As in any spanning forest of d components, there are $n - d$ edges in $\cup_{d \in D} S_d$, where n is the number of vertices in G. Note that from a dominating set, one may construct different spanning star forests with the same size.

Conversely, given a spanning star forest F of size k of G, the centers of the stars in F form a dominating set that contains $n - k$ vertices. In summary, we have the following simple fact.

LEMMA 3.1. *Let the largest size of a spanning star forest of a graph G be denoted by $\alpha(G)$ and the domination number of G be denoted by $\gamma(G)$. Then, $\alpha(G) = n - \gamma(G)$, where n is the order of G.*

This implies that finding a maximum spanning star forest of a graph is equivalent to finding a minimum dominating set. Therefore, as the dominating set problem [10], the spanning star forest problem can be solved in linear time for trees. The lemma also implies the following result.

THEOREM 3.2. *The spanning star forest problem has a PTAS for planar graphs.*

Proof. Recall that $\gamma(G)$ denotes the dominating number of a graph G. Since the dominating set problem has a PTAS for planar graphs [5], for any small $\epsilon > 0$, there is a polynomial-time algorithm $\mathcal{A}_\epsilon$ of approximation ratio $(1+\epsilon)$ for the problem. We first obtain a dominating set D_ϵ of at most $(1+\epsilon)\gamma(G)$ vertices by applying $\mathcal{A}_\epsilon$ to G. Then, we construct a spanning star forest F_ϵ from D_ϵ as described before Lemma 3.1. Since $\gamma(G) \leq \frac{n}{2}$ (Ore theorem; see [15], for example), by Lemma 3.1, F_ϵ has size

$$n - |D_\epsilon| \geq n - (1+\epsilon)\gamma(G) \geq (1-\epsilon)(n - \gamma(G)) = (1-\epsilon)\alpha(G).$$

Thus, F_ϵ is a $(1-\epsilon)$-approximation solution of the spanning star forest problem for G. ∎

3.2. A $\frac{3}{5}$-approximation algorithm. In this subsection, we mainly present a polynomial-time $\frac{3}{5}$-approximation algorithm using a known upper bound on the size of a dominating set in a graph.

THEOREM 3.3. *If there is a polynomial-time algorithm that finds a dominating set of at most $(\frac{1}{2}-\epsilon)n$ vertices given an n-vertex graph of minimum degree at least 2, then there is a polynomial-time $(\frac{1}{2}+\epsilon)$-approximation algorithm for the spanning star forest problem.*

Proof. Let $\mathcal{A}$ be the algorithm that outputs a dominating set of at most $(\frac{1}{2}-\epsilon)n$ vertices given an n-vertex graph of minimum degree 2. Consider a graph G. A vertex is called a support vertex if it is adjacent to some degree-1 vertices. Suppose G contains l degree-1 vertices. For simplicity, we first assume that G contains an even number of support vertices $u_1, u_2, \ldots, u_{2k}$, $k \geq 0$. Obviously, $l \geq 2k$. We first construct a graph H from G by removing all the degree-1 vertices and adding k vertices $v_1, v_2, \ldots, v_k$ and the following $2k$ edges

$$(u_{2i-1}, v_i), (u_{2i}, v_i), \qquad 1 \leq i \leq k.$$

Then, H has $n-l+k$ vertices of degree at least 2. Applying $\mathcal{A}$ to H, we obtain a dominating set D_H of at most $(\frac{1}{2}-\epsilon)(n-l+k)$ vertices. Now we construct a dominating set D_G of G as follows: for each i, if $v_i \in D_H$, we remove v_i from D_H and add u_{2i-1} and u_{2i} into the set; if $v_i \notin D_H$, then at least one of u_{2i-1} and u_{2i} is in D_H, and we add the other into the set. In other words, $D_G = (D_H - \{v_1, v_2, \ldots, v_k\}) \cup \{u_1, u_2, \ldots, u_{2k}\}$. It is easy to verify that D_G is a dominating set. By the construction of D_G, its size is at most

$$|D_H| + k \leq \left(\frac{1}{2}-\epsilon\right)(n-l+k) + k \leq \left(\frac{1}{2}-\epsilon\right)n + \left(\frac{1}{2}+\epsilon\right)k \tag{3.1}$$

since $l \geq 2k$. This implies that a spanning star forest F of G can be obtained from D_G with at least

$$n - |D_G| \geq n - \left[\left(\frac{1}{2}-\epsilon\right)n + \left(\frac{1}{2}+\epsilon\right)k\right] > \left(\frac{1}{2}+\epsilon\right)(n-2k) \tag{3.2}$$

edges. Since any dominating set of G must contain each support vertex or its degree-1 neighbors, $\gamma(G) \geq 2k$, and hence $\alpha(G) \leq n-2k$. Therefore, F contains at least $(\frac{1}{2}+\epsilon)\alpha(G)$ edges. This has proved that $\mathcal{A}$ can be extended into a ratio-$(\frac{1}{2}+\epsilon)$ approximation algorithm for the spanning star forest problem.

When G contains an odd number of support vertices $u_1, u_2, \ldots, u_{2k+1}$, $k \geq 1$, we modify the construction of H presented above by adding two new vertices x_{k+1} and x_{k+2} and three edges (u_{2k+1}, x_{k+1}), (x_{k+1}, x_{k+2}), (x_{k+2}, u_{2k+1}). Now, H has $n-l+k+2$ vertices. We derive dominating set D_G of G from the output dominating set D_H from the algorithm as

$$D_G = (D_H - \{v_1, v_2, \ldots, v_k, x_{k+1}, x_{k+2}\}) \cup \{u_1, u_2, \ldots, u_{2k+1}\}.$$

The size of D_G is at most $|D_H| + k$ and the inequalities (3.1) and (3.2) become

$$|D_H| + k \leq \left(\frac{1}{2}-\epsilon\right)(n-l+k+2) + k \leq \left(\frac{1}{2}-\epsilon\right)(n+1) + \left(\frac{1}{2}+\epsilon\right)k$$

and

$$n - |D_G| \geq n - \left[\left(\frac{1}{2} - \epsilon\right)(n+1) + \left(\frac{1}{2} + \epsilon\right)k\right] > \left(\frac{1}{2} + \epsilon\right)(n - 2k - 1),$$

respectively. Since $\alpha(G) \leq n - 2k - 1$, we have that the corresponding spanning star forest F contains at least $(\frac{1}{2} + \epsilon)\alpha(G)$ edges. Hence, the theorem also holds in this case. □

THEOREM 3.4. *There is a $\frac{3}{5}$-approximation algorithm for the spanning star forest problem.*

Proof. McCuaig and Shepherd proved that any n-vertex graph of minimum degree 2 has a dominating set of size at most $\frac{2}{5}n$ for any $n \geq 8$ (see [18]). We demonstrate that such a dominating set can be found in polynomial time by giving a different constructive proof of their theorem (see Appendix A). Hence, by Theorem 3.3, there is a polynomial-time algorithm of approximation ratio $\frac{1}{2} + \frac{1}{10} = \frac{3}{5}$ for the problem. □

The above result could be improved by using a better upper bound on $\gamma(G)$ for graphs G with large minimum degree $\delta(G)$. If $\delta(G) \geq 7$, $\gamma(G) \leq |V_G|[1 - \delta(G)(\frac{1}{\delta(G)+1})^{1+1/\delta(G)}]$ (Theorem 2.8 in [15]). Although such a bound is not true for $\delta(G) = 3$, $\gamma(G) \leq \frac{3}{8}|V_G|$ if $\delta(G) \geq 3$ (see [19]). If there is a polynomial-time algorithm that, given a graph of minimum degree at least 3, always outputs a dominating set of at most $\alpha|V_G|$, $\frac{3}{8} \leq \alpha < \frac{2}{5}$, the following theorem implies a better approximation algorithm.

THEOREM 3.5. *Let $\frac{3}{8} \leq \alpha < \frac{2}{5}$. If there is a polynomial-time algorithm that finds a dominating set of at most αn vertices given an n-vertex graph of minimum degree at least 3, then there is a polynomial-time $(\frac{4}{5} - \frac{1}{2}\alpha)$-approximation algorithm for the spanning star forest problem.*

Proof. Let $G = (V, E)$ be a graph with n vertices, where $n \geq 8$. We denote the subset of vertices of degree i by V_i, and hence $V = \cup_i V_i$. First, we construct a maximal subset $A_1 \subseteq V_1$ such that the distance between any two vertices in A_1 is at least 3. Recall that a vertex is called a support vertex if it is adjacent to some degree-1 vertices. It is easy to see that each support vertex is adjacent to a unique vertex in A_1. We next construct an $A_2 \subseteq V_2$ having the following property:

(i) Any pair of vertices in $A_1 \cup A_2$ has distance at least 3; and

(ii) A_2 is maximal; i.e., $A_2 \cup \{u\}$ does not satisfy condition (i) for any $u \in V_2 - A_2$.

Such a subset can be constructed recursively in polynomial time. Initially, we set $A_2 = \phi$ and add the vertices in V_2 one by one to A_2 subject to condition (i).

Let a_j denote the number of vertices in A_j, $j = 1, 2$. By condition (i), no vertex in G can dominate more than one vertex in $A_1 \cup A_2$. Thus,

$$\gamma(G) \geq a_1 + a_2. \tag{3.3}$$

For each vertex u, we use $N(u)$ to denote the set of its neighbors. Define $B_j = \cup_{u \in A_j} N(u)$, $j = 1, 2$. By the above observation, B_1 is identical to the set of the support vertices and $|B_1| = |A_1| = a_1$. Hence, for each $u \in V_1 - A_1$, there is a unique $v \in A_1$ such that the distance $d(u, v)$ between u and v is 2.

Since no two vertices in A_2 have a common neighbor, $|B_2| = 2a_2$. By the maximality property of A_2, for each $u \in V_2 - A_2$, there is some $v \in A_1 \cup A_2$ such that $d(u, v) \leq 2$. If $d(u, v)$=1, then $u \in B_1 \cup B_2$. If $d(u, v) = 2$, then u is adjacent to some vertex in $B_1 \cup B_2$. This implies that the disjoint union $B_1 \cup B_2$ dominates $V_1 \cup V_2$.

Now, we construct from G a graph G' such that $\delta(G') \geq 3$ as follows.

Step 1. Delete all the degree-1 vertices. For simplicity, we assume that 3 divides $|B_1|$. Partition B_1 into groups of 3 vertices each. For each group of vertices x, y, z, we add a new vertex u_{xyz} and three edges $u_{xyz}x$, $u_{xyz}y$, and $u_{xyz}z$. We also add edges xy, yz, and zx if any of them is not in G. In this way, x, y, x, u_{xyz} all have degree at least 3. After the completion of step 1, the number of vertices in the resulting graph is

$$n - |V_1| + \frac{|B_1|}{3} \leq n - |B_1| + \frac{|B_1|}{3} = n - \frac{2a_1}{3}.$$

Step 2. Add edge uv if it is not in G for every $u, v \in V_2 - A_2$. This makes each vertex in $V_2 - A$ have degree at least three if $|V_2 - A_2| \geq 4$. (In case $0 \leq |V_2 - A_2| \leq 3$, in the argument below $D \cup B_1 \cup V_2$ would be a dominating set of G, and so the right-hand side of inequality (3.4) would be replaced by $\alpha n + \frac{2}{3}(1-\alpha)a_1 + \frac{1}{3}(1-\alpha)a_2 + 3$.)

Step 3. Delete all vertices in A_2. For simplicity, again, we assume that 3 divides $|B_2|$. Partition B_2 into groups of 3 vertices each. For each group of three vertices x, y, z, add a new vertex u_{xyz} and three edges $u_{xyz}x$, $u_{xyz}y$, and $u_{xyz}z$. We also add edges xy, yz, and zx if any of them is not in G. As a result, x, y, z, u_{xyz} all have degree at least 3. After this step is done, the resulting graph G' has at most

$$n - \frac{2a_1}{3} - a_2 + \frac{|B_2|}{3} = n - \frac{2a_1}{3} - \frac{a_2}{3}$$

vertices and $\delta(G') \geq 3$.

By applying the polynomial-time algorithm to G', we obtain a dominating set D' of size at most $(n - \frac{2a_1}{3} - \frac{a_2}{3})\alpha$. Since $B_1 \cup B_2$ dominates $V_1 \cup V_2$, $D' \cup B_1 \cup B_2$ induces a dominating set D_1 for G: For each group x, y, z in steps 1 and 3, if u_{xyz} is in D', we replace it with x, y, z. The resulting dominating set D_1 is of size at most

$$|D'| + \frac{2(|B_1| + |B_2|)}{3} \leq \alpha n + \frac{2(1-\alpha)}{3}a_1 + \frac{4-\alpha}{3}a_2. \tag{3.4}$$

Notice that B_1 is the set of support vertices and $|B_1| = a_1$. Applying inequality (3.1) with the McCuaig–Shepherd bound ($\frac{1}{2} - \epsilon = \frac{2}{5}$, $k = \frac{a_1}{2}$), we can also obtain a dominating set D_2 of size at most $\frac{2}{5}n + \frac{3}{10}a_1$. By selecting the smaller one between D_1 and D_2, we obtain a dominating set D having size at most

$$\frac{1}{2}(|D_1| + |D_2|) \leq \frac{5\alpha + 2}{10}n + \frac{29 - 20\alpha}{60}a_1 + \frac{4-\alpha}{6}a_2.$$

From D, by inequality (3.2), we construct a desired spanning star forest that has at least

$$\begin{aligned} & n - |D| \\ \geq\ & (\tfrac{4}{5} - \tfrac{1}{2}\alpha)n - \tfrac{29-20\alpha}{60}a_1 - \tfrac{4-\alpha}{6}a_2 \\ \geq\ & (\tfrac{4}{5} - \tfrac{1}{2}\alpha)(n - a_1 - a_2) \\ \geq\ & (\tfrac{4}{5} - \tfrac{1}{2}\alpha)(n - \gamma(G)) \end{aligned} \tag{3.5}$$

edges since $\frac{4}{5} - \frac{1}{2}\alpha \geq \frac{29-20\alpha}{60}$, $\frac{4}{5} - \frac{1}{2}\alpha \geq \frac{4-\alpha}{6}$, and $\gamma(G) \geq a_1 + a_2$. This shows the fact stated in the theorem.

When 3 does not divide $|B_1|$ and $|B_2|$, we can modify the above argument in the same way as we have done in the proof of Theorem 3.4. For $j = 1, 2$, if $|B_j| \equiv 2 \pmod 3$, we add two more vertices to form a 4-vertex clique with the last two

vertices in B_j; if $|B_j| \equiv 1 \pmod 3$, we add three more vertices to form a 4-vertex clique with the last vertex. By this modification, the inequality (3.5) becomes

$$n - |D| \geq \left(\frac{4}{5} - \frac{1}{2}\alpha\right)(n - \gamma(G)) - \frac{6 - r_1 - r_2}{6}(4\alpha - 1),$$

where $r_j \equiv |B_j| \pmod 3$, $j = 1, 2$. Since $n - \gamma(G) \geq n/2$, the fact stated in the theorem remains true. □

3.3. Hardness of approximation. We have shown that the spanning star forest problem can be approximated within a constant ratio in polynomial time. On the other hand, the dominating set problem cannot be approximated within $(1-\epsilon)\ln n$ for any ϵ unless $NP \subset DTIME(n^{\log\log n})$ [17, 12]. Hence, intuitively, the spanning star forest problem should not be approximated within a large constant ratio in polynomial time. Now, we prove this fact rigorously.

THEOREM 3.6. *The spanning star forest problem cannot be approximated within a ratio* $\frac{259}{260} + \epsilon$ *in polynomial time for any* $\epsilon > 0$ *unless* $P = NP$.

Proof. We prove this fact using a reduction from the VERTEX COVER problem. Recall that the VERTEX COVER problem is to find the smallest subset $U \subset V$ such that every edge has at least one endpoint in U given a graph $G = (V, E)$. It is known that this problem cannot be approximated within a ratio $\frac{53}{52} - \epsilon$ for 4 regular graphs unless $P = NP$ [9]. Given a 4-regular graph $G = (V_G, E_G)$, we construct a graph $H = (V_H, E_H)$ from G by adding a length-2 path parallel to each edge in G. Without loss of generality, we assume that G is connected and not a tree. Formally,

$$V_H = V_G \cup \{v_e |\ e \in E_G\},$$
$$E_H = E_G \cup \{(x, v_e), (v_e, y) |\ e = (x, y) \in E_G\}.$$

It is easy to see that the following facts hold:

> ($\star$) For any $V \subset V_G$, if it is a vertex cover set of G, then it is a dominating set of H. Conversely, for each subset $V \subset V_H$, if it is a dominating set of H, then $(V \cap V_G) \cup \{x \in V_G \mid v_e \in V$ and $e = (x, y) \in E$ and $x < y\} \subseteq V_G$ is a vertex cover of G.

Let $\mathrm{OPT}_{vc}(G)$, $\mathrm{OPT}_d(H)$, and $\mathrm{OPT}_{sf}(H)$ denote the minimum size of a vertex cover of G, the minimum size of a dominating set of H, and the maximum size of a spanning star forest of H, respectively. Then, the above facts imply that $\mathrm{OPT}_d(H) = \mathrm{OPT}_{vc}(G)$. Since G is 4-regular and connected,

$$|E_G| \leq 4\mathrm{OPT}_{vc}(G).$$

By the handshaking theorem,

$$|V_G| = \frac{1}{2}|E_G| \leq 2\mathrm{OPT}_{vc}(G).$$

Therefore,

$$\begin{aligned} & \mathrm{OPT}_{sf}(H) \\ = \ & |V_H| - \mathrm{OPT}_d(H) = (|V_G| + |E_G|) - \mathrm{OPT}_d(H) \\ = \ & (|V_G| + |E_G|) - \mathrm{OPT}_{vc}(G) \\ \leq \ & 5\mathrm{OPT}_{vc}(G). \end{aligned}$$

Assume S is a spanning star forest of H. If S contains k stars, then the size $|S|$ of S is $|V_H| - k$. In addition, these k centers of S form a dominating set of H and

hence induce a vertex cover V_S of size at most k of G by the fact $(\star)$. Hence, since $\mathrm{OPT}_d(H) = \mathrm{OPT}_{vc}(G)$,

$$|V_S| - \mathrm{OPT}_{vc}(G) \le k - \mathrm{OPT}_d(H) = k - (|V_H| - \mathrm{OPT}_{sf}(H)) = \mathrm{OPT}_{sf}(H) - |S|.$$

Thus, if $|S|$ is a ratio-$(1 - \frac{1}{5} \cdot \frac{1}{52} + \epsilon)$ approximation of the spanning star forest problem for H, where $\epsilon > 0$, i.e., $|S| \ge (1 - \frac{1}{5} \cdot \frac{1}{52} + \epsilon)\mathrm{OPT}_{sf}(H)$, then the resulting vertex cover set V_S satisfies

$$|V_S| - \mathrm{OPT}_{vc}(G) \le \left(\frac{1}{5} \cdot \frac{1}{53} - \epsilon\right) \mathrm{OPT}_{sf}(H) \le \left(\frac{1}{52} - 5\epsilon\right) \mathrm{OPT}_{vc}(G).$$

In other words, V_S is a ratio-$(\frac{53}{52} - 5\epsilon)$ approximation of the VERTEX COVER problem for G. This implies that the spanning star forest problem cannot be approximated within a ratio $1 - \frac{1}{5} \cdot \frac{1}{52} + \epsilon = \frac{259}{260} + \epsilon$ in polynomial time unless $P = NP$. □

4. Algorithms for weighted graphs.

4.1. A remark on the maximum spanning star forests. In a connected weighted graph, every maximum spanning star forest may contain some isolated vertices. For example, the graph shown in Figure 4.1 has a unique spanning star graph with the maximum weight 7. Therefore, when we design an algorithm for the spanning star forest problem for connected weighted graphs, we have to consider the start forests with isolated vertices.

4.2. A linear-time algorithm for weighted trees. Trees are the simplest connected graphs. In this subsection, we present a linear-time algorithm for finding the maximum spanning star forest of a tree.

Given a tree T, we first root T at an arbitrary vertex r and consider each edge (u, v) as a directed edge from the endpoint closer to the root to the other endpoint. In the rest of this subsection, when we mention that (u, v) is an edge, we mean that u is closer to the root; we say that u is the parent of v or v is a child of u if (u, v) is an edge in the rooted tree. Obviously, in a star forest of T, the root can be a center of a star, a leaf of a star centered at one of its children, or an isolated vertex. For each vertex u of T, we let $T(u)$ be the subtree rooted at u and define the following three numbers:

$\Phi(u)$: The maximum weight of a spanning star forest of $T(u)$ in which u is a center of a multiple-vertex star.
$\Psi(u)$: The maximum weight of a spanning star forest of $T(u)$ in which u is a leaf of a multiple-vertex star.
$\Omega(u)$: The maximum weight of a spanning star forest of $T(u)$ in which u is an isolated node.

First, these three numbers can be computed through recurrence formulas as shown below.

LEMMA 4.1. *Let $C(u)$ be the set of the children of u. Then,*

$$\begin{aligned}
\Phi(u) &= \textstyle\sum_{v \in C(u)} \Delta(v) \\
&\quad - \min_{v \in C(u)} (\Delta(v) - \Omega(v) - w(uv)), \\
\Psi(u) &= \max_{v \in C(u)} [w(uv) + \max\{\Phi(v), \Omega(v)\} \\
&\quad + \textstyle\sum_{x \in C(u) - \{v\}} \max\{\Phi(x), \Psi(x), \Omega(x)\}], \\
\Omega(u) &= \textstyle\sum_{v \in C(u)} \max\{\Phi(v), \Psi(v), \Omega(v)\},
\end{aligned}$$

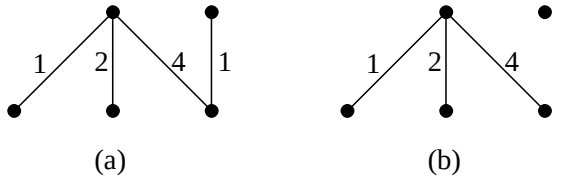


FIG. 4.1. (a) *A weighted graph;* (b) *The unique maximum spanning star forest with an isolated vertex.*

where $\Delta(v) = \max\{\Phi(v), \Psi(v), \Omega(v) + w(uv)\}$.

Proof. For any vertex u of T, let $SFR(u)$ be a star forest of $T(u)$ that has a maximum weight $\Phi(u)$, over all the star forests in which u is a center of a star S_u. For any $v \in C(u)$, we use $SFR(u)|_v$ to denote the restriction of $SFR(u)$ in the subtree $T(v)$ rooted at v and define $\Delta(v) = \max\{\Phi(v), \Psi(v), \Omega(v) + w(uv)\}$. We consider the following cases.

Case 1. *The star S_u centered at u contains at least two leaves.* If v is a leaf of the star S_u centered at u in $SFR(u)$, then $SFR(u)|_v$ must be a star forest of $T(v)$ that has the maximum weight $\Omega(v)$, over all the star forests in which v is an isolated vertex. Moreover, $w(uv) + \Omega(v) \geq \Phi(v), \Psi(v)$. Equivalently, $\Delta(v) = w(uv) + \Omega(v)$. Otherwise, we could obtain a star forest with larger weight from $SFR(u)$ by removing edge uv and replacing $SFR(u)|_v$ by a star forest (of $T(v)$) in which v is not isolated, contradicting our assumption that $SFR(u)$ has the maximum weight.

If v is a not a leaf of the star S_u, then $SFR(u)|_v$ is a star forest (of $T(v)$) with weight $\Delta(v)$, which is $\max\{\Psi(v), \Phi(v)\}$. Otherwise, we can obtain a star forest of $T(u)$ with a larger weight by replacing $SFR(u)|_v$ by another star forest of weight $\Omega(v)$ and adding the edge uv into $T(u)$.

This implies that $\Phi(u) = \sum_{v \in C(u)} \Delta(v) = \sum_{v \in C(u)} \max\{\Phi(v), \Psi(v), \Omega(v) + w(uv)\}$. For any $v \in C(u)$, by definition, $\Delta(v) - w(uv) - \Omega(v) \geq 0$. For any v that is a leaf of the star S_u, by the above argument, $\Delta(v) = w(uv) + \Omega(v)$. Therefore, in this case, $\min_{v \in C(u)}(\Delta(v) - w(uv) - \Omega(v)) = 0$, and hence

$$\begin{aligned} \Phi(u) &= \sum_{v \in C(u)} \max\{\Phi(v), \Psi(v), \Omega(v) + w(uv)\} \\ &\quad - \min_{v \in C(u)}(\Delta(v) - w(uv) - \Omega(v)). \end{aligned}$$

Case 2. *The star S_u centered at u contains only one leaf.* Let v' be the unique leaf in S_u. Then, for any $v \in C(u)$, we have $\Delta(v) - w(uv) - \Omega(v) \geq \Delta(v') - w(uv') - \Omega(v')$. Otherwise, we could obtain a star forest of a larger weight from $SRF(u)$ by the following operations:

(a) add edge uv,
(b) delete uv',
(c) replace $SFR(u)|_v$ by a star forest (of $T(v)$) in which v is isolated, and
(d) replace $SFR(u)|_{v'}$ by the star forest (of $T(v')$ with the maximum weight $\Delta(v')$.

Hence, the weight $\Phi(u)$ of the star forest $SFR(u)$ is equal to

$$\begin{aligned} & \sum_{v \in C(u) - \{v'\}} \Delta(v) + (w(uv') + \Omega(v')) \\ = \; & \sum_{v \in C(u)} \Delta(v) - (\Delta(v') - w(uv') - \Omega(v')) \\ = \; & \sum_{v \in C(u)} \Delta(v) - \min_{v \in C(u)}(\Delta(v) - w(uv) - \Omega(v)). \end{aligned}$$

This proves the recurrence formula for $\Phi(u)$.

Let SFL be a star forest (of $T(u)$) in which u is a leaf of a star S_u centered at one of its children, $v \in C(u)$. Then, $SFL|_v$ is a star forest (of $T(v)$) in which v is isolated or the center of a star. Hence, $SFL|_v$ has weight $\max\{\Phi(v), \Omega(v)\}$. For any other $v' \neq v$ in $C(u)$, $SFL|_{v'}$ is a star forest (of $T(v')$) in which v' can be a center of a star, a leaf of a star, or an isolated vertex, and so $SFL|_{v'}$ has weight $\Delta(v')$. Hence,

$$\begin{aligned}\Psi(u) &= \max_{v\in C(u)}[w(uv) + \max\{\Phi(v), \Omega(v)\} \\ &\quad + \textstyle\sum_{x\in C(u)-\{v\}} \max\{\Phi(x), \Psi(x), \Omega(x)\}].\end{aligned}$$

This proves the recurrence formula for $\Psi(u)$.

Let $SFI(u)$ be the star forest (of $T(u)$) with the maximum weight $\Omega(u)$ in which u is isolated. Then, for each $v \in C(u)$, $SFI(u)|_v$ has to be a star forest of $T(v)$ with the maximum weight $\Delta(v)$. Otherwise, $SFI(u)$ does not have the maximum weight $\Omega(u)$. Therefore, the recurrence formula for $\Omega(u)$ is correct. This finishes the proof of the lemma. □

THEOREM 4.2. *There is a linear-time algorithm that outputs a star forest with the maximum weight given a weighted tree.*

Proof. Lemma 4.1 implies a dynamic programming approach for computing the maximum star forest of a weight tree rooted at r. For each u in the tree, compute $\Psi(u)$, $\Phi(u)$, $\Omega(u)$ from the corresponding numbers at its children according to the recurrence formulas given in the lemma in linear time.

After $\Phi(r)$, $\Psi(r)$, $\Omega(r)$ are computed, the maximum star forest can be found by backtracking along the computation path. We call a star forest $SF(u)$ in which u is a center of a star, a leaf of star, or an isolated vertex a type-1, type-2, or type-3 star forest. The goal of finding the maximum star forest can be achieved by introducing backtracking pointers r-ptr, l-ptr, i-ptr, which are used to construct the maximum star forest of type-1, type-2, and type-3 at each vertex. At each vertex u, r-ptr(u) specifies (a) which child of u is in the star S_u centered at u, and (b) if $v \in C(u)$ is not in the star S_u, the type of the restriction star forest on $T(v)$; l-ptr(u) specifies (a) which child of u is the center of the star containing u, and (b) the type of the restriction star forest on $T(v)$ for each $v \in C(u)$; i-ptr(u) specifies the type of the restriction star forest on $T(v)$ for each $v \in C(u)$. Obviously, with these backtracking pointers, the maximum star forest can be found in linear time. This finishes the proof. □

4.3. A $\frac{1}{2}$-approximation algorithm. For an arbitrary weighted graph, we find a spanning star forest using the following algorithm:

1. Find a maximum spanning tree T_G of the given graph G.
2. Compute a maximum spanning star forest SF_G of T_G.

Given a positively weighted connected graph, its maximum spanning tree can be computed in polynomial time. For example, we can use Kruskal's algorithm for this purpose. Since Step 2 also takes polynomial time, the above method can be executed in polynomial time.

Let $\mathrm{OPT}_{sf}(G)$ and $\mathrm{OPT}_t(G)$ be the maximum weight of a spanning star forest and a spanning tree of G, respectively. Since G is connected, any spanning star forest can be extended into a spanning tree by adding some "bridge" edges between the stars. Hence, $\mathrm{OPT}_{sf}(G) \leq \mathrm{OPT}_t(G)$.

Consider a rooted weighted tree T. We call a node an odd node if the unique path from the root to it has an odd number of edges and an even node otherwise. Deleting from T all edges from an odd node to an even node gives a spanning star forest of T; deleting from T all edges from an even node to an odd node gives another

spanning star forest of T. In addition, these two resulting spanning star forests are edge-disjoint and the weight of one of these two spanning star forests is at least half of the weight of T. Hence,

$$w(SF_G) \geq \frac{1}{2}w(T_G) = \frac{1}{2}\mathrm{OPT}_t(G) \geq \frac{1}{2}\mathrm{OPT}_{sf}(G).$$

Therefore, the above algorithm has approximation ratio $\frac{1}{2}$. This has proved the following theorem.

THEOREM 4.3. *The spanning star forest problem can be approximated within ratio $\frac{1}{2}$ in polynomial time for arbitrary weighted graphs.*

5. Application to aligning genomic sequences. To align multiple genomic sequences from different species, the TBA program [7] works on the phylogenetic tree T over the species in a bottom-up fashion. At each internal node v of T, the program outputs a set of alignments of multiple segments (called blocks) in the given sequences by merging the blocks generated at the left and right children of v through pairwise alignments between sequences contained in left and right children. The program stops and outputs a set of blocks (called a blockset) at the root of T. The output blockset can be considered as a packing of the pairwise alignments between sequence segments generated at each internal node.

Tandem duplication is an evolutionary event in which a genomic segment is duplicated into several adjacent copies. It is believed to be a major mechanism for producing large gene clusters. In the human and mouse genomes, a gene family may have dozens or even hundreds of members due to tandem duplication. Assume we align a set of genomes. Consider a gene family that has multiple homologous genes in each species. At an internal node v of T, in the worst case, an alignment program will generate a pairwise alignment between every pair of the orthologous gene sequences contained in the left and right subtrees, respectively. As a result, every pair of blocks that contain the orthologous gene sequences and that are generated at the left and right children of v, respectively, will be merged into a larger block. Hence, the number of blocks that contain the gene sequences is equal to the product of the numbers of blocks generated in the left and right children of v. When the program stops at the root of T, it will output a blockset of an exponentially large size.

Currently, there are no good solutions to aligning duplication-rich genomic regions. The existing TBA program screens out duplicated alignments and thus is not able to capture all pairwise orthologous relationships in a duplicated-gene cluster. An example of this deficiency, involving the human and rat alpha-globin genes, is described in the introduction.

One way to avoid exponential growth of the number of blocks is to generate a linear number of blocks at each internal node by carefully selecting the blocks to be merged. More specifically, for the node v, we use B_v to denote the blockset generated at v and $|B_v|$ to denote the number of blocks contained in B_v. Obviously, when v is a leaf in T, $B_v = \phi$. Let v' and v'' be the left and right children of v, respectively. A size explosion can be avoided by arranging that the blockset B_v contains at most $|B_{v'}| + |B_{v''}| + c$ blocks, where c is a constant independent of $|B_{v'}|$ and $|B_{v''}|$. In this way, for a gene family having g_i copies in species i of N species, the final output blockset will contain at most $\sum_{i=1}^{N} g_i + cN$ blocks over the gene family.

Let P be the set of pairwise alignments generated at v. We denote an alignment in P with rows a and b by $a.b$, where a and b are segments of sequences in the left

and right subtree, respectively. Define

$$L_P = \{a \mid a.b \in P \text{ for some } b\}$$

and

$$R_P = \{b \mid a.b \in P \text{ for some } a\}.$$

Our method selects blocks in $B_{v'}$ and $B_{v''}$ to merge according to the following theory.

We first define a weighted bipartite graph G_v with vertex bipartition (V', V''). The vertices in V' correspond one-to-one to the blocks in $B_{v'}$ and the rows in L_p that are not contained in any blocks in $B_{v'}$; the vertices in V'' correspond to the blocks in $B_{v''}$ and the rows in R_P in the same way. For simplicity, we considered the rows in L_P and R_P as trivial blocks. For each $x \in V'$, $y \in V''$, there is an edge between x and y if and only if there is a pairwise alignment $a.b \in P$ such that a and b appear in the blocks corresponding to x and y, respectively, called a *reference alignment* of the blocks. In general, there are multiple reference alignments for each pair of blocks. Hence, the weight of the edge (x, y) is then defined to be the maximum alignment score over all the reference alignments of the blocks corresponding to x and y. In practice, the rows in P can partially overlap with some rows of a block in $B_{v'} \cup B_{v''}$. Here, however, we assume a row in P is either contained in or disjoint from any row in $B_{v'} \cup B_{v''}$. Since each genomic sequence may contain many different orthologous sequences, the resulting bipartite graph G_v has more than one connected component in general. By construction, none of the connected components in the graph are singletons.

To control the size of the output blockset, we make use of a maximum spanning star forest of G_v to merge the blocks $B_{v'}$ and $B_{v''}$ as shown in the *Block-Merging Algorithm*. One desired property of a spanning star forest SF of a graph G is that each edge has at least one degree-1 endpoint in SF. Such a property is critical for controlling the size of the output blockset as shown below.

BLOCK-MERGING ALGORITHM

Input: $B_{v'}$, $B_{v''}$, and P.

0. $B_v = \phi$;
1. Construct the graph G_v by using $B_{v'}$, $B_{v''}$, and P;
2. Heuristically compute a maximum spanning star forest SF of G_v;
3. For each edge (x, y) in SF, merge the blocks correspondent to x and y, and add the resulting block into B_v;
4. Output B_v.

THEOREM 5.1. *The Block-Merging Algorithm outputs a blockset satisfying the following properties:*

Full coverage property. *Any position covered by a pairwise alignment produced during the aligning process appears in some block in the output blockset.*

Nonredundancy property. *Each block in the output blockset contains a unique row that does not appear in any other blocks.*

Proof. The full coverage property is obvious. We prove the nonredundancy property by induction on the depth of an internal node. Let v be an internal node. If the children v' and v'' of v are leaves, then $B_{v'}$ and $B_{v''}$ are empty; hence each vertex of the bipartite graph G_v corresponds to a segment in some given sequence. For each edge $e = (x, y)$ in the spanning star forest of G_v computed in step 3 of the algorithm, one of x and y, say, x, is of degree 1 and hence incident only to e. Then, the segment corresponding to x is covered only by the block derived from e through merging the two segments corresponding to x and y, respectively. Therefore, each block in B_v has a unique row.

If at least one of v' and v'' is not a leaf, by induction, we assume that both $B_{v'}$ and $B_{v''}$ satisfy the nonredundancy property if they are not empty. Consider a block $Z \in B_v$ obtained through an edge (x, y) of the spanning star forest found in step 3 of the algorithm. Without loss of generality, we assume that x is of degree 1 in the spanning star forest. If x corresponds to a block in $B_{v'}$, then the unique row in the block corresponding to x appears only in Z among all the blocks in B_v. If x corresponds to a row R_x of a pairwise alignment in P, then, by the definition of G_v, R_x is not contained in any block in $B_{v'}$ and $B_{v''}$ and hence appears uniquely in Z among all the blocks in B_v. This concludes the proof. □

Theorem 5.1 can be used to estimate the base-pair size $||B||$ of the blockset B generated by the algorithm. Let N be the number of sequences in the input genomic sequence set S. Since any pair of orthologous sequence segments s and s' can be assumed to have more than 50% identical bases, they have roughly equal length; i.e., $|s| = \Theta(|s'|)$. For each block $X \in B$, we use $r_{unique}(X)$ to denote the unique row in X. By assumption, the base-pair size $|X|$ of X is at most $N\Theta(|r_{unique}(X)|)$ since X has at most N rows. Therefore, using the fact that $r_{unique}(X)$'s are disjoint segments of the input sequences in S, we have

$$||B|| = \sum_{X \in B} |X| \leq \sum_{X \in B} N\Theta(|r_{unique}(X)|) \leq N\Theta\left(\sum_{s \in S} |s|\right).$$

In other words, the base-pair size of the output blockset is at most N times the base-pair size of the input sequence set.

An initial test shows that our proposed approach is quite promising. For example, we consider the alpha-globin gene cluster that has as many as 91 genes in 20 mammals. Both alpha-related and theta-related genes have clear many-to-many homologous relationships. When aligning the alpha-globin gene cluster region (of total length 3.9 Mbytes) in 20 mammals, a straightforward strategy meeting the two requirements described in the last theorem produced a blockset of size over 900 Mbytes, but the preliminary implementation of our approach can reduce the alignment size to less than 10 Mbytes with little sacrifice of the alignment quality.

6. Conclusions. In this paper, we initiate the algorithmic study of the spanning star forest problem. In particular, we present a $\frac{3}{5}$-approximation algorithm for graphs and a $\frac{1}{2}$-approximation algorithm for weighted graphs. In contrast, we also prove that it is NP-hard to approximate the problem within a ratio $\frac{259}{260}+\epsilon$. This study raises several research questions for future investigation. For example, how to obtain an approximation algorithm with better ratio and how to improve the inapproximability result are two interesting open problems. After this work was submitted, a polynomial-time algorithm with approximation ratio 0.71 was presented in [8]. Another interesting question is how to design approximation algorithms for the spanning star forest problem for directed and weighted graphs.

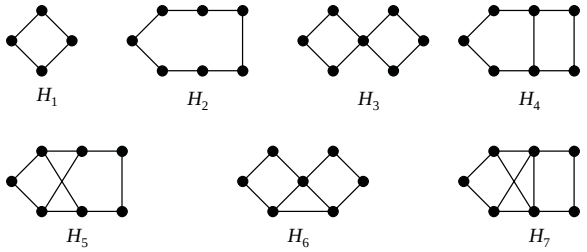


FIG. A.1. *Seven bad graphs whose minimum dominating sets have size larger than* $2n/5$ *(reproduced from* [18]*; reprinted with permission of John Wiley & Sons, Inc.).*

Appendix A. Finding a dominating set of size at most $2n/5$.

THEOREM A.1. *Let* $G = (V, E)$ *be a connected graph such that each vertex has degree at least* 2*. If* G *is not one of the* 7 *graphs in Figure* A.1 *(called* bad graphs*), then* $\gamma(G) \leq \frac{2}{5}|V|$*. Moreover, such a dominating set can be found in* $O(|E|^2|V|)$ *time.*

Proof. We use $\delta(G)$ to denote the minimum degree of a vertex in $G = (V, E)$. We prove by induction on $|V|$. It is easy to check that, for $|V| \leq 15$, we can always find a desired dominating set. Now assume that $|V| > 15$ and that the theorem holds for all graphs G' that have a smaller number of vertices than G and that are not listed in Figure A.1. We show how to construct a dominating set of G whose size is at most $\frac{2}{5}|V|$.

If G is not edge-minimal with respect to connectedness and with a minimum degree of 2, we keep deleting edges. Therefore, in the following, we assume that G is edge-minimal with respect to these conditions.

Let $B(G) = \{u \in V \mid d(u) > 2\}$. If $|B(G)| = 0$, then G is a cycle. In this case, a dominating set of G can be found by numbering its vertices starting from 0 and choosing all the vertices whose indices is divisible by 3. It is readily verified that such a dominating set contains at most $\frac{2}{5}|V|$ vertices for any cycles that are different from H_1 and H_2.

If $|B(G)| = 1$, then G is a union of cycles that intersect at the unique vertex $u \in B(G)$. Let $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ be the sets of cycles of length 4, 7, and another length, respectively. Furthermore, let $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$. A dominating set of G can be constructed by the following steps: (i) Construct a dominating set of size 2 that contains u for each cycle in $\mathcal{A}$; (ii) construct a dominating set of size 3 that contains u for each cycle in $\mathcal{B}$; (iii) construct a dominating set that contains u and is of size at most $\frac{2}{5}|C_i|$ for each i; (iv) merge all the dominating sets. The size of this dominating set is

$$1 + |\mathcal{A}| + 2|\mathcal{B}| + \sum_{i=1}^{t} \left(\frac{2}{5} n_i - 1 \right) = 1 + |\mathcal{A}| + 2|\mathcal{B}| + \frac{2}{5} \sum_{i=1}^{t} n_i - t,$$

in which n_i denotes the number of vertices in C_i. Noting that $|V| = 3|\mathcal{A}| + 6|\mathcal{B}| + \sum_{i=1}^{t} n_i - t + 1$, we have that the size of the constructed dominating set is larger than $\frac{2}{5}|V|$ only if $G = H_3$, which contains exactly two length-4 cycles.

Now assume that $|B(G)| \geq 2$. In the following, we call G a good graph if it is not listed in Figure A.1. We consider the following cases.

Case 1. *There is an edge* uv *between two vertices* u *and* v *in* $B(G)$. Due to the minimality of G, this edge must be a bridge of G. Let G_u and G_v be the components

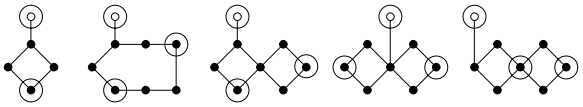


FIG. A.2. *A graph formed by adding a vertex to a bad graph contains one of these graphs. The white vertex is the additional vertex. The circles mark the vertices in a dominating set of each graph.*

of $G - uv$ containing u and v, respectively. Then both G_u and G_v are connected and $\delta(G_u) \geq 2$ and $\delta(G_v) \geq 2$.

Case 1.1. *Both G_u and G_v are good.* By hypothesis, we can find dominating sets D_u and D_v of G_u and G_v such that $D_u \leq \frac{2}{5}|V_u|$ and $D_v \leq \frac{2}{5}|V_v|$. $D_u \cup D_v$ contains at most $\frac{2}{5}|V|$ vertices and dominates G.

Case 1.2. *Only one, say, G_v is a good graph.* Then G_u is a bad graph. Let $G_u = (V_u, E_u)$, and let G_u^* be the graph with vertex set $V_u^* = V_u \cup \{v\}$ and edge set $E_u^* = E_u \cup \{uv\}$. We also let G_v^* be the graph constructed by (i) choosing a neighbor v' of v in G_v; (ii) joining v' with all other neighbors of v in G_v; and (iii) deleting v. Then, G_v^* is connected and $\delta(G_v^*) \geq 2$. Since G_u is a bad graph, G_u^* contains one of the graphs in Figure A.2. Note that H_4, H_5, H_7 in Figure A.1 contain H_2 and H_6 contains H_3, and so we need only to consider the cases in Figure A.2. From this figure, we can see that G_u^* has a dominating set D_u^* that contains v and satisfies that $|D_u^*| \leq \frac{2}{5}|V_u^*|$. Since G_u is bad and $|V| \geq 16$, G_v^* contains at least eight vertices, and so it is good. By induction, it has a dominating set D_v^* of size at most $\frac{2}{5}|V_v^*|$. Then $D_u^* \cup D_v^*$ form a dominating set of G whose size is at most $\frac{2}{5}|V|$.

Case 2. *There is no edge between any two vertices in $B(G)$.* In the following, we will use 2-*cycles* to refer to the cycles that contain exactly one vertex in $B(G)$, and this unique vertex is called the *endpoint* of the cycle. We will also use 2-*paths* to refer to the paths whose endpoints are in $B(G)$ and in which other vertices are of degree 2.

If there is a path $vv_1v_2v_3v'$ in G such that $v \neq v'$ and $d(v_1) = d(v_2) = d(v_3) = 2$, we construct a graph G' by removing v_1, v_2, v_3 and adding the edge vv' if $vv' \notin E(G)$. Clearly G' is connected and $\delta(G') \geq 2$. Since $|V| > 16$, G' must be good. Hence, G' has a dominating set D' of size at most $\frac{2}{5}|V'|$. We construct a dominating set D for G as follows: (i) if $v \in D'$, then $D = D' \cup \{v_3\}$; (ii) if $v' \in D$, then $D = D' \cup \{v_1\}$; and (iii) if neither v nor v' is in D', then $D = D' \cup \{v_2\}$. Then $|D| = |D'| + 1 \leq \frac{2}{5}(|V'| + 3) = \frac{2}{5}|V|$. Now assume that all 2-cycles in G are of length at most 4 and all 2-paths in G are of length at most 3.

Case 2.1. *G contains a 2-cycle of length 4 whose endpoint's degree is larger than 3.* Let this cycle be $vv_1v_2v_3v$, where v is the endpoint. A graph G' is formed by deleting v_1, v_2, and v_3 from G. Since G' contains at least thirteen vertices, it is good and has a dominating set D' of size at most $\frac{2}{5}|V'|$. Then $D' \cup \{v_2\}$ is a dominating set of G whose size is smaller than $\frac{2}{5}(|V'| + 3) = \frac{2}{5}|V|$.

Case 2.2. *Every 2-cycle of length 4 in G has an endpoint of degree 3.* Consider such a cycle $vv_1v_2v_3v$, where v is the endpoint. The other neighbor v' of v has degree 2, since there is no edge between two vertices in $B(G)$. The subgraph of G induced by $\{v, v_1, v_2, v_3, v'\}$ is called a 4-cluster and v' is called the *anchor* of this cluster.

In this case, we assume that G contains a 4-cluster, say, $C = \{v, v_1, v_2, v_3, v'\}$, with anchor v' such that the neighbor u of v' not in C has degree larger than 2 in G, and we define $G' = G - C$. Then G' is a connected graph with $\delta(G) \geq 2$. Since G' contains at least eleven vertices, G' is good, and it has a dominating set D' such that

$|D'| \leq \frac{2}{5}|V'|$. Furthermore, $D = D' \cup \{v, v_2\}$ is a dominating set of G and $|D| \leq \frac{2}{5}|V|$.

Case 2.3. *For every* 4*-cluster, its anchor's neighbor that is not in the cluster is of degree* 2. We define a 2-cycle of length 3 in G to be a 3-cluster, and the anchor of a 3-cluster is the endpoint of the cluster. Note that we define the anchor of a 3-cluster differently. Finally, each vertex u such that $d(u) > 2$ and u is not an endpoint of any 2-cycles is considered as a 1-cluster with u as the anchor.

If there is an edge joining two anchors, the two anchors must be that of two 4-clusters since other cases have been considered in Case 2.2. In this case, this graph has only ten vertices. But we assume G has at least sixteen vertices. Hence, there is no edge joining two anchors in G. A dominating set D of G is formed by collecting all anchors and one vertex in each 4-cluster. Now we prove that $|D| \leq \frac{2}{5}|V|$.

Let m_4, m_3, m_1 be the number of 4-clusters, 3-clusters, and 1-clusters, respectively. Then, there are also m_4, m_3, and m_1 anchors of 4-clusters, 3-clusters, and 1-clusters, respectively. Clearly, $|D| = 2m_4 + m_3 + m_1$.

Now we count the nonanchor vertices. Each 4-cluster, 3-cluster, and 1-cluster contains 4, 2, and 0 nonanchor vertices, respectively. Furthermore, on each 2-path connecting two anchors, there are one or two nonanchor vertices depending on whether it is of length two or three. Therefore, the number of nonanchor vertices on these paths is at least $\frac{1}{2}(m_4 + m_3 + 3m_1)$ since there is at least one nonanchor vertex in the 2-path between two anchors and the degree of the unique vertex in a 1-cluster is at least 3. Hence, the number of nonanchor vertices is at least $\frac{9}{2}m_4 + \frac{5}{2}m_3 + \frac{3}{2}m_1$. Recall that the dominating set D contains one nonanchor vertex in each 4-cluster. This implies $|V| \geq \frac{11}{2}m_4 + \frac{7}{2}m_3 + \frac{5}{2}m_1$. Simple computation shows that $|D| \leq \frac{2}{5}|V|$. □

A recursive algorithm to compute a desired dominating set follows from the above induction proof. To implement this algorithm, we have to compute $d(v)$ for each vertex in G and find (a) an edge between two vertices in $B(G)$, (b) a length-4 path that contains three consecutive degree-2 vertices in the middle, and (c) a 4-cluster with an endpoint of degree larger than 3. Given the adjacent matrix of a graph G as input, we can find the degree of a vertex in $O(|V|)$ time and thus find $B(G)$ in $O(|V|^2)$ time. With $B(G)$, we can find an edge between two vertices in $B(G)$ if any exist in $O(|V|^2)$ time. To compute (b), we just need to check if there is a degree-2 vertex having two degree-2 neighbors and these two neighbors have different neighbors in $O(|E|^2)$ time. The task (c) can be done similarly. Overall, the algorithm finds a dominating set iteratively. The number of vertices decreases by at least 1 after each iteration step, and each iteration step takes $O(|E|^2)$ time and removes three vertices from the graph. Therefore, the running time of the algorithm is $O(|E|^2|V|)$.

Acknowledgment. The authors would like to thank the anonymous reviewers for various useful suggestions for improving the presentation of this work.

REFERENCES

[1] A. Agra, D. Cardoso, O. Cerdeira, and E. Rocha, *A Spanning Star Forest Model for the Diversity Problem in Automobile Industry*, manuscript, 2005.

[2] J. Akiyama and M. Kano, *Path factors of a graph*, in Graph Theory and Its Applications, F. Haray and J. Maybee, eds., Wiley, New York, 1984, pp. 1–21.

[3] Y. Aoki, *The star-arboricity of the complete regular multipartite graphs*, Discrete Math., 81 (1990), pp. 115–122.

[4] I. Algor and N. Alon, *The star arboricity of graphs*, Discrete Math., 75 (1989), pp. 11–22.

[5] B. S. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. ACM, 41 (1994), pp. 153–180.

[6] V. Berry, S. Guillemot, F. Nicolas, and C. Paul, *On the approximation of computing evolutionary trees*, in Proceedings of the 11th Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 3595, Springer-Verlag, New York, 2005, pp. 115–125.

[7] W. J. Kent, C. Riemer, L. Elnitski, A. F. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, D. Haussler, and W. Miller, *Aligning multiple genomic sequences with the threaded blockset aligner*, Genome Res., 14 (2004), pp. 708–715.

[8] N. Chen, R. Engelberg, C. T. Nguyen, P. Raghavendra, A. Rudra, and G. Singh, *Improved approximation algorithms for the spanning star forest problem*, in Proceedings of APPROX 2007, Lecture Notes in Comput. Sci. 4627, Springer-Verlag, New York, pp. 44–58.

[9] M. Chlebík and J. Chlebíková, *Inapproximability results for bounded variants of optimization problems*, Theoret. Comput. Sci., 354 (2006), pp. 320–338.

[10] E. J. Cockayne, S. E. Goodman, and S. T. Hedetniemi, *A linear algorithm for the domination number of a tree*, Inform. Process. Lett., 4 (1975), pp. 41–44.

[11] Y. Egawa, T. Fukada, S. Nagoya, and M. Urabe, *A decomposition of complete bipartite graphs into edge-disjoint subgraphs with star components*, Discrete Math., 58 (1986), pp. 93–95.

[12] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[13] S. Ferneyhough, R. Haas, D. Hannson, and G. MacGillivray, *Star forests, dominating sets and Ramsey-type problems*, Discrete Math., 245 (2002), pp. 255–262.

[14] S. I. Hakimi, J. Mitchem, and E. Schmeichel, *Star arboricity of graphs*, Discrete Math., 149 (1996), pp. 93–98.

[15] T. Haynes, S. T. Hedetniemi, and P. J. Slater, *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.

[16] M. M. Hou, P. Berman, L. X. Zhang, and W. Miller, *Controlling size when aligning multiple genomic sequences with duplications*, in Proceedings of WABI (Zurich, 2006), Springer-Verlag, Berlin, 2006, pp. 138–149.

[17] C. Lund and M. Yannakakis, *On the hardness of approximation minimization problems*, J. ACM, 41 (1994), pp. 961–981.

[18] W. McCuaig and B. Shepherd, *Domination in graphs with minimum degree two*, J. Graph Theory, 13 (1989), pp. 749–762.

[19] B. Reed, *Paths, stars and the number three*, Combin. Probab. Comput., 5 (1996), pp. 277–295.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 963–981

SIMULATING QUANTUM COMPUTATION BY CONTRACTING TENSOR NETWORKS*

IGOR L. MARKOV† AND YAOYUN SHI†

Abstract. The treewidth of a graph is a useful combinatorial measure of how close the graph is to a tree. We prove that a quantum circuit with T gates whose underlying graph has a treewidth d can be simulated deterministically in $T^{O(1)}\exp[O(d)]$ time, which, in particular, is polynomial in T if $d = O(\log T)$. Among many implications, we show efficient simulations for log-depth circuits whose gates apply to nearby qubits only, a natural constraint satisfied by most physical implementations. We also show that *one-way quantum computation* of Raussendorf and Briegel (*Phys. Rev. Lett.*, 86 (2001), pp. 5188–5191), a universal quantum computation scheme with promising physical implementations, can be efficiently simulated by a randomized algorithm if its quantum resource is derived from a small-treewidth graph with a constant maximum degree. (The requirement on the maximum degree was removed in [I. L. Markov and Y. Shi, preprint:quant-ph/0511069].)

Key words. quantum computation, computational complexity, treewidth, tensor network, classical simulation, one-way quantum computation

AMS subject classifications. 81P68, 68Q05, 68Q10, 05C83, 68R10

DOI. 10.1137/050644756

1. Introduction.

The recent interest in quantum circuits is motivated by several complementary considerations. Quantum information processing is rapidly becoming a reality as it allows the manipulation matter on an unprecedented scale. Such manipulations may create particular entangled states or implement specific quantum evolutions—they find uses in atomic clocks, ultra-precise metrology, high-resolution lithography, optical communication, etc. On the other hand, engineers traditionally simulate new designs before implementing them. Such simulation may identify subtle design flaws and save both costs and effort. It typically uses well-understood host hardware, e.g., one can simulate a quantum circuit on a commonly-used conventional computer.

More ambitiously, quantum circuits compete with conventional computing and communication. Quantum-mechanical effects may potentially lead to computational speed-ups, more secure or more efficient communication, better keeping of secrets, etc. To this end, one seeks new circuits and algorithms with revolutionary behavior as in Shor's work on number-factoring, or provable limits on possible behaviors. While proving abstract limitations on the success of unknown algorithms appears more difficult, a common line of reasoning for such results is based on simulation. For example, if the behavior of a quantum circuit can be faithfully simulated on a conventional computer, then the possible speed-up achieved by the quantum circuit is limited by the cost of simulation. Thus, aside from sanity-checking new designs for quantum information-processing hardware, more efficient simulation can lead to sharper bounds on all possible algorithms.

*Received by the editors November 10, 2005; accepted for publication (in revised form) November 27, 2007; published electronically June 25, 2008.
http://www.siam.org/journals/sicomp/38-3/64475.html

†Department of Electrical Engineering and Computer Science, University of Michigan, 2260 Hayward Street, Ann Arbor, MI 48109-2121 (imarkov@eecs.umich.edu, shiyy@eecs.umich.edu). The first author's research was supported in part by NSF 0208959, the DARPA QuIST program, and the Air Force Research Laboratory. The second author's research was supported in part by NSF 0323555, 0347078, and 0622033.

Since the outcome of a quantum computation is probabilistic, we shall clarify our notion of simulation. By a randomized simulation, we mean a classical randomized algorithm whose output distribution on an input is identical to that of the simulated quantum computation. By a deterministic simulation, we mean a classical deterministic algorithm which, on a given pair of input x and output y of the quantum computation, outputs the probability that y is observed at the end of the quantum computation on x.

To simulate a quantum circuit, one may use a naïve brute-force calculation of quantum amplitudes that has exponential overhead. Achieving significantly smaller overhead in the generic case appears hopeless—in fact, this observation lead Feynman to suggest that quantum computers may outperform conventional ones in some tasks. Therefore, only certain restricted classes of quantum circuits were studied in existing literature on simulation.

Classes of quantum circuits that admit efficient simulation are often distinguished by a restricted "gate library," but do not impose additional restrictions on how gates are interconnected or sequenced. A case in point is the seminal Gottesman–Knill theorem [12] and its recent improvement by Aaronson and Gottesman [1]. These results apply only to circuits with stabilizer gates—controlled-NOT, Hadamard, phase, and single-qubit measurements in the so-called Clifford group. Another example is given by *match gates* defined and studied by Valiant [33], and extended by Terhal and DiVincenzo [31].

A different way to impose a restriction on a class of quantum circuits is to limit the amount of entanglement in intermediate states. Jozsa and Linden [16], as well as Vidal [36], demonstrate efficient classical simulation of such circuits and conclude that achieving quantum speed-ups requires more than a bounded amount of entanglement.

In this work we pursue a different approach to efficient simulation and allow the use of arbitrary gates. More specifically, we assume a general quantum circuit model in which a gate is a general quantum operation (so-called *physically realizable operators*) on a constant number of qubits. This model, proposed and studied by Aharonov, Kitaev, and Nisan [2], generalizes the standard quantum circuit model, defined by Yao [40], where each gate is unitary and measurements are applied at the end of the computation. We also assume that (i) the computation starts with a fixed unentangled state in the computational basis, and (ii) at the end each qubit is either measured or *traced-out*.

Our simulation builds upon the framework of *tensor network contraction*. Being a direct generalization of matrices, tensors capture a wide range of linear phenomena including vectors, operators, multilinear forms, etc. They facilitate convenient and fundamental mathematical tools in many branches of physics such as fluid and solid mechanics, and general relativity [14]. More recently, several methods have been developed to simulate quantum evolution by contracting variants of tensor networks, under the names of *matrix product states (MPS)*, *projected entangled pairs states (PEPS)*, etc. [36, 37, 34, 41, 35, 24]. Under this framework, a quantum circuit is regarded as a network of tensors. The simulation contracts edges one by one and performs the convolution of the corresponding tensors, until there is only one vertex left. Having degree 0, this vertex must be labeled by a single number, which gives the final measurement probability sought by simulation. In contrast with other simulation techniques, we do not necessarily simulate individual gates in their original order—in fact, a given gate may even be simulated *partially* at several stages of the simulation.

While tensor network contraction has been used in previous work, little was known

about optimal contraction orders. We prove that the minimal cost of contraction is determined by the *treewidth* $\mathrm{tw}(G_C)$ of the circuit graph G_C. Moreover, existing constructions that approximate optimal *tree-decompositions* (e.g., [28]) produce near-optimal contraction sequences. We shall define the concepts of treewidth and tree decompositions in section 2. Intuitively, the smaller a graph's treewidth is, the closer it is to a tree, and a tree decomposition is a drawing of the graph to make it look like a tree as much as possible. Our result allows us to leverage the extensive graph-theoretical literature dealing with the properties and computation of treewidth.

THEOREM 1.1. *Let C be a quantum circuit with T gates and whose underlying circuit graph is G_C. Then C can be simulated deterministically in time $T^{O(1)} \exp[O(\mathrm{tw}(G_C))]$.*

A rigorous restatement of the above theorem is Theorem 4.6. By this theorem, given a function computable in polynomial time by a quantum algorithm but not classically, any polynomial-size quantum circuit computing the function must have super-logarithmic treewidth. The following corollary is an immediate consequence.

COROLLARY 1.2. *Any polynomial-size quantum circuit of a logarithmic treewidth can be simulated deterministically in polynomial time.*

Quantum formulas defined and studied by Yao [40] are quantum circuits whose underlying graphs are trees. Roychowdhury and Vatan [30] showed that quantum formulas can be efficiently simulated deterministically. Since every quantum formula has treewidth 1, Corollary 1.2 gives an alternative efficient simulation.

Our focus on the *topology* of the quantum circuit allows us to accommodate arbitrary gates, as long as their qubit-width (number of inputs) is limited by a constant. In particular, Corollary 1.2 implies efficient simulation of some circuits that create the maximum amount of entanglement in a partition of the qubits, e.g., a layer of two-qubit gates. Therefore, our results are not implied by previously published techniques.

We now articulate some implications of our main result to classes of quantum circuits, in terms of properties of their underlying graphs. The following two classes of graphs are well-studied, and their treewidths are known. The class of *series parallel graphs* arises in electric circuits, and such circuits have treewidth ≤ 2. Planar graphs G with n vertices are known to have treewidth $\mathrm{tw}(G) = O(\sqrt{|V(G)|})$ [4].

COROLLARY 1.3. *Any polynomial size parallel serial quantum circuit can be simulated deterministically in polynomial time.*

COROLLARY 1.4. *A size T planar quantum circuit can be simulated deterministically in $\exp[O(\sqrt{T})]$ time.*

Another corollary deals with a topological restriction representative of many physical realizations of quantum circuits. Let $q \geq 1$ be an integer. A circuit is said to be q-local-interacting if under a linear ordering of its qubits, each gate acts only on qubits that are at most q distance apart. A circuit is said to be local-interacting if it is q-local interacting with a constant q independent of the circuit size. Such *local-interaction* circuits generalize the restriction of qubit couplings to nearest-neighbor qubits (e.g., in a spin-chain) commonly appearing in proposals for building quantum computers, where qubits may be stationary and cannot be coupled arbitrarily. To this end, we observe that the treewidth of any local-interaction circuit of logarithmic depth is at most logarithmic.

COROLLARY 1.5. *Let C be a quantum circuit of size T and depth D and q-local-interacting. Then C can be simulated deterministically in $T^{O(1)} \exp[O(qD)]$ time. In particular, if C is a polynomial-size local-interacting circuit with a logarithmic depth,*

then it can be simulated deterministically in polynomial time.

Yet another important application of our approach is the simulation of *one-way quantum computation*. In two influential papers [7, 25], Briegel and Raussendorf introduced the concept of *graph states*—quantum states derived from graphs,—and show that an arbitrary quantum circuit can be simulated by *adaptive*, *single-qubit measurements* on the *graph state* derived from the grid graph. Note that the graph state for a one-way quantum computation does not depend on the quantum circuit to be simulated (except that its size should be large enough) and that for most physical implementations single-qubit measurements are much easier to implement than multiqubit operations. Hence it is conceivable that graph states would be manufactured by a technologically more advanced party, then used by other parties with lesser quantum-computational power in order to facilitate universal quantum computing. This makes one-way quantum computation an attractive scheme for physical implementations of universal quantum computation. An experimental demonstration of one-way quantum computation appeared in a recent Nature article [38].

A natural question about one-way computation is to characterize the class of graphs whose graph states are universal for quantum computation. We call a family of quantum states $\phi = \{|\phi_1\rangle, |\phi_2\rangle, \ldots, |\phi_n\rangle, \ldots\}$ *universal for one-way quantum computation* if (a) the number of qubits in $|\phi_n\rangle$ is bounded by a fixed polynomial in n; (b) any quantum circuit of size n can be simulated by a one-way quantum computation on $|\phi_n\rangle$. On the other hand, ϕ is said to be efficiently simulatable if any one-way quantum computation on $|\phi_n\rangle$ can be efficiently simulated classically for all sufficiently large n. Note that the class of universal families and that of efficiently simulatable families are disjoint if and only if efficient quantum computation is indeed strictly more powerful than efficient classical computation. We show that it is necessary for graphs with a constant maximum degree to have high treewidth so that the corresponding graph states are not efficiently simulatable.

THEOREM 1.6. *Let G be a simple undirected graph with the maximum degree $\Delta(G)$. Then a one-way quantum computation on the respective graph state can be simulated by a randomized algorithm in time $|V(G)|^{O(1)} \exp[O(\Delta(G)\mathrm{tw}(G))]$.*

The above result was improved in [19] so that the simulation time does not depend exponentially in $\Delta(G)$. Our simulation can be made deterministic with a better upper bound on time complexity if the one-way computation satisfies additional constraints, such as those in [25]. We shall elaborate on this improvement in section 6.

An important limitation of our techniques is that a circuit family with sufficiently fast-growing treewidth may require super-polynomial resources for simulation. In particular, this seems to be the case with known circuits for modular exponentiation. Therefore, there is little hope to efficiently simulate number-factoring algorithms using tree decompositions. As an extreme example to illustrate the limitation of our technique, we give a depth-4 circuit—including the final measurement as the fourth layer—that has large treewidth.

THEOREM 1.7. *There exists a depth-4 quantum circuit on n qubits using only one- and two-qubit gates such that its treewidth is $\Omega(n)$.*

Note that a circuit satisfying the assumption in the above theorem must have $O(n)$ size. Our construction is based on expander graphs, whose treewidth must be linear in the number of vertices (Lemma 5.2).

This finding is consistent with the obstacles to efficient simulation that are evident in the results of Terhal and DiVincenzo [32], later extended by Fenner et al. [13]. In contrast, we are able to efficiently simulate any depth-3 circuit *deterministically* while

the simulation in [32] is probabilistic.

THEOREM 1.8. *Assuming that only one- and two-qubit gates are allowed, any polynomial-size depth-3 quantum circuit can be simulated deterministically in polynomial time.*

Our simulation algorithm is related to algorithms for other tasks in that its runtime depends on the treewidth of a graph derived from the input. Bodlaender wrote an excellent survey [8] on this subject. Particularly relevant are algorithms based on "vertex eliminations," e.g., the *bucket elimination* algorithm for Bayesian inference [11]. Another parallel can be made with the work by Broering and Lokam [10], which solves circuit-SAT in time exponential in the treewidth of the graph of the given circuit. However, to the best of our best knowledge, we are the first to relate the treewidth of a quantum circuit to its classical simulation.

Our results are applicable to the simulation of classical probabilistic circuits, which can be modeled by matrices, similarly to quantum circuits. Such simulation has recently gained prominence in the literature on the reliability of digital logic [17], and is particularly relevant to satellite-based and airborne electronics which experience unpredictable particle strikes at higher rates.

The rest of this paper is organized as follows. After introducing notation, we describe how quantum circuits and their simulation can be modeled by tensor networks. The runtime of such simulation depends on the graph parameter that we call the *contraction complexity*. We then relate the contraction complexity to treewidth, and apply the simulation to restricted classes of graphs and to one-way quantum computation. Finally, we discuss possible directions for future investigations with a brief survey on the subsequent development since the announcement of our results.

2. Notation and definitions. For integer $n \geq 1$, define $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. An ordering π of an n-element set is denoted by $\pi(1)$, $\pi(2)$, ..., $\pi(n)$. Unless otherwise stated, graphs in this paper are undirected and may have multiple edges or loops. Edges connecting the same pair of vertices are called parallel edges. If G is a graph, then its vertex set is denoted by $V(G)$ and its edge set by $E(G)$. When it is clear in this context, we use $V = V(G)$ and $E = E(G)$. The *degree* of a vertex v, denoted by $d(v)$, is the number of edges incident to it. In particular, a loop counts as 1 edge. The maximum degree of a vertex in G is denoted by $\Delta(G)$.

Treewidth of a graph. Let G be a graph. A *tree decomposition* of G [27] is a tree $\mathcal{T}$, together with a function that maps each vertex $w \in V(\mathcal{T})$ to a subset $B_w \subseteq V(G)$. These subsets B_w are called *bags* (of vertices). In addition, the following conditions must hold.

(T1) $\bigcup_{v \in V(\mathcal{T})} B_v = V(G)$, i.e., each vertex must appear in at least one bag.
(T2) $\forall\ \{u, v\} \in E(G)$, $\exists w \in V(\mathcal{T})$, $\{u, v\} \subseteq B_w$, i.e., for each edge, at least one bag must contain both of its endvertices.
(T3) $\forall\ u \in V(G)$, the set of vertices $w \in V(\mathcal{T})$ with $u \in B_w$ form a connected subtree, i.e., all bags containing a given vertex must be connected in $\mathcal{T}$.

The *width* of a tree decomposition is defined by $\max_{w \in V(\mathcal{T})} |B_w| - 1$. The *treewidth* of G is the minimum width over its tree decompositions. For example, all trees have treewidth 1 and single cycles of length at least 3 have treewidth 2. Figure 1 shows an example of tree decomposition. Intuitively, a tree decomposition $\mathcal{T}$ is a way of drawing a graph to look like a tree, which may require viewing sets of vertices (bags) as single vertices. The less a graph looks like a tree, the larger the bags become. The notion of tree decomposition has been useful in capturing the complexity of constraint

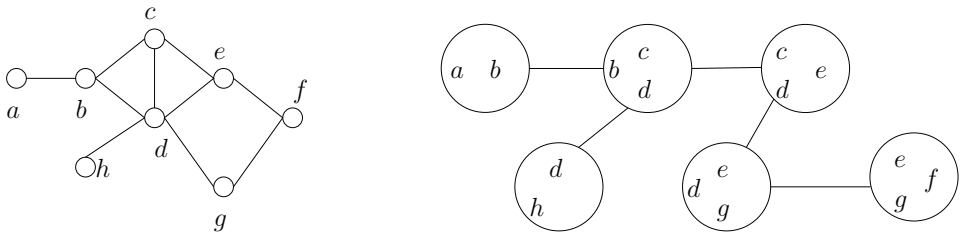


FIG. 1. *A graph and its decomposition of width* 2 *with* 6 *bags.*

satisfaction problems, Bayesian networks, and other combinatorial phenomena represented by graphs. In further writing, we may refer to a vertex in $\mathcal{T}$ by its bag when the context is clear.

Treewidth can be defined in several seemingly unrelated ways, e.g., as the minimum k for which a given graph is a *partial k-tree*, as the *induced width* (also called the *dimension*), or as the *elimination width* [29, 5]. An *elimination ordering* π of a graph G is an ordering of $V(G)$. The *induced width of a vertex* $v \in V(G)$ in the ordering is the number of its neighbors at the time it is being removed in the following process: Start with $\pi(1)$, add an edge for each pair of its neighbors that were previously not adjacent, remove $\pi(1)$, then repeat this procedure with the next vertex in the ordering. The *width of* π is the maximum induced width of a vertex, and the *induced width of* G is the minimum width of an elimination ordering. It is known that the induced width of a graph is precisely its treewidth [5].

It follows straightforwardly from the definition of treewidth that if G is obtained from G' by removing a degree 1 vertex, then $\mathrm{tw}(G) = \mathrm{tw}(G')$, unless G' has only 1 edge, in which case $\mathrm{tw}(G) = 0$ and $\mathrm{tw}(G') = 1$. We will also use the following well-known and simple fact, a proof for which is provided in the appendix.

PROPOSITION 2.1. *Let G be a simple undirected graph, and w be a degree* 2 *vertex. Then removing w and connecting its two adjacent vertices does not change the treewidth.*

Quantum circuits. We review some basic concepts of quantum mechanics and quantum computation. For a more detailed treatment, we refer the readers to the book by Nielsen and Chuang [23].

The state space of one qubit is denoted by $\mathcal{H} \stackrel{\text{def}}{=} \mathbb{C}^2$. We fix an orthonormal basis for $\mathcal{H}$ and label the basis vectors with $|0\rangle$ and $|1\rangle$. The space of operators on a vector space V is denoted by $\mathbf{L}(V)$. The identity operator on V is denoted by I_V, or by I if V is implicit from the context. A density operator, or a mixed state, of n qubits is a positive semidefinite operator $\rho \in \mathbf{L}(\mathcal{H}^{\otimes n})$ with $\mathrm{trace}\rho = 1$. For a binary string $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$, let $\rho_x \stackrel{\text{def}}{=} \bigotimes_{i=1}^n |x_i\rangle\langle x_i|$ be the density operator of the state $|x\rangle \stackrel{\text{def}}{=} \otimes_{i=1}^n |x_i\rangle$.

In this paper, a quantum gate with a input qubits and b output qubits is a superoperator $Q : \mathbf{L}(\mathcal{H}^{\otimes a}) \to \mathbf{L}(\mathcal{H}^{\otimes b})$. There are certain constraints that Q must satisfy in order to represent a physically realizable quantum operation. We need not be concerned about those constraints as our simulation method does not depend on them. In existing applications one typically has $a \geq b$ and often $a = b$, though a density operator can also be regarded as a gate with $a = 0$. The ordering of inputs and outputs is, in general, significant. If Q is a *traced out* operator, then $b = 0$, and $Q(|x\rangle\langle y|) = \langle x|y\rangle$ for all $x, y \in \{0,1\}^a$. We denote by $Q[A]$ the application of Q to an

ordered set A of a qubits.

The information in a quantum state is retrieved through the application of measurements. A POVM (positive operator-valued measure) $\mathcal{M}$ on n qubits is a set $\mathcal{M} = \{M_1, M_2, \ldots, M_k\}$, where each M_i is called a POVM element, and is a positive semidefinite operator in $\mathbf{L}(\mathcal{H}^{\otimes n})$ such that $\sum_{i=1}^{k} M_i = I$. The single-qubit measurement in the computational basis is $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$.

We assume that the maximum number of qubits on which a quantum gate can act is bounded by a constant (often two or three). A quantum circuit of size T with n input-qubits and m output-qubits consists of the following:

(1) A sequence of n input-wires, each of which represents one input-qubit, i.e., a qubit which is not the output qubit of any gate.
(2) A sequence of T quantum gates g_1, g_2, ..., g_T, each of which is applied to some subset of the wires.
(3) A sequence of m output-wires, each of which represents an output-qubit, i.e., a qubit which is not the input qubit of any gate.

Note that by the above definition, a quantum circuit C defines a function $C : \mathbf{L}(\mathcal{H}^{\otimes n}) \to \mathbf{L}(\mathcal{H}^{\otimes m})$. In most applications, a circuit C is applied to an input state $\rho_x \stackrel{\text{def}}{=} \otimes_{i=1}^{n} |x_i\rangle\langle x_i|$, for some binary string $x = x_1 \cdots x_n \in \{0,1\}^n$, and at the end of the computation, measurements in the computational basis are applied to a subset of the qubits. We shall restrict our discussions to such a case, though our results can be extended to more general cases.

The graph of a quantum circuit C, denoted by G_C, is obtained from C as follows. Regard each gate as a vertex, and for each input/output wire add a new vertex to the open edge of the wire.[1] Each wire segment can now be represented by an edge in the graph.

3. Tensors and tensor networks. Tensors, commonly used in physics, are multidimensional matrices that generalize more traditional tools from linear algebra, such as matrix products. Here we focus on features of tensors that are relevant to our work.

DEFINITION 3.1. *A rank-k tensor in an m-dimension space $g = [g_{i_1,i_2,\ldots,i_k}]_{i_1,i_2,\ldots,i_k}$ is an m^k-dimensional array of complex numbers $g_{i_1,i_2,\ldots,i_k}$, indexed by k indices, i_1, i_2, ..., i_k, each of which takes m values. When the indices are clear we omit them outside the bracket.*

For example, a rank-0 tensor is simply a complex number, and a rank-1 tensor is a dimension-m complex vector. We focus on dimension-4 tensors, and set the range of each index to be $\Pi \stackrel{\text{def}}{=} \{|b_1\rangle\langle b_2| : b_1, b_2 \in \{0,1\}\}$. We fix the following tensor representation of a density operator and a superoperator.

DEFINITION 3.2. *Let ρ be a density operator on a qubits. The tensor of ρ is $[\rho_{\sigma_1,\sigma_2,\ldots,\sigma_a}]_{\sigma_1,\sigma_2,\ldots,\sigma_a \in \Pi}$, where*

$$\rho_{\sigma_1,\ldots,\sigma_a} \stackrel{\text{def}}{=} tr(\rho \cdot (\otimes_{i=1}^{a} \sigma_i)^\dagger).$$

Let Q be a superoperator acting on a input qubits and b output qubits. The tensor of Q is

$$Q_{\sigma_1,\sigma_2,\ldots,\sigma_a,\tau_1,\tau_2,\ldots,\tau_b}]_{\sigma_1,\ldots,\sigma_a,\tau_1,\ldots,\tau_b \in \Pi},$$

[1] These vertices are going to represent input states, as well as measurements and trace-out operators at the end of the computation.

where

$$Q_{\sigma_1,\sigma_2,\ldots,\sigma_a,\tau_1,\tau_2,\ldots,\tau_b} \stackrel{\text{def}}{=} tr(Q(\otimes_{i=1}^{a}\sigma_i)\cdot(\otimes_{j=1}^{b}\tau_j)^\dagger).$$

We shall use the same notation for a density operator (or a superoperator) and its tensor. We now define the central object of this paper.

DEFINITION 3.3. *A* tensor network *is a collection tensors, each index of which may be used by either one or two tensors.*

A rank-k tensor g can be graphically represented as a vertex labeled with g, and connected to k open wires, each of which is labeled with a distinct index. We may represent a tensor network by starting with such graphical representations of its tensors, and then connecting wires corresponding to the same index. Note that now each wire corresponds to a distinct index. Also, an index that appears in one tensor corresponds to an open wire, and an index that appears in two tensors corresponds to an edge connecting two vertices. Parts (a) and (b) in Figure 2 give an example of the graphical representation of a tensor and a tensor network. In the tensor g_Q, we call the σ_i wires, $1 \le i \le a$, input wires, and the τ_j wires, $1 \le j \le b$, the output wires.

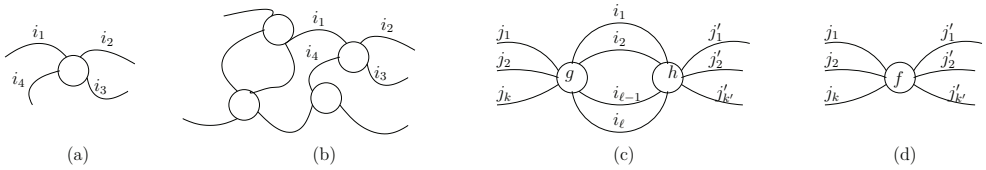


FIG. 2. *A rank-4 tensor is illustrated in* (a)*, and a tensor network with four tensors is shown in* (b)*. Contraction of two tensors is illustrated in* (c) *and* (d)*.*

Suppose in a tensor network there are ℓ parallel edges i_1, i_2, …, i_ℓ between two vertices $g = [g_{i_1,\ldots,i_\ell,j_1,\ldots,j_k}]$ and $h = [h_{i_1,\ldots,i_\ell,j'_1,\ldots,j'_{k'}}]$. We may contract those edges by first removing them, then merging v_g and v_h into a new vertex v_f, whose tensor is $f = [f_{j_1,\ldots,j_k,j'_1,\ldots,j'_{k'}}]$, and

$$f_{j_1,\ldots,j_k,j'_1,\ldots,j'_{k'}} \stackrel{\text{def}}{=} \sum_{i_1,i_2,\ldots,i_\ell} g_{i_1,\ldots,i_\ell,j_1,\ldots,j_k} \cdot h_{i_1,\ldots,i_\ell,j'_1,\ldots,j'_{k'}}. \tag{1}$$

Parts (c) and (d) in Figure 2 illustrate the above contraction. Note that a tensor network with k open wires can be contracted to a single tensor of rank k, and the result does not depend on the order of contractions. The following example is instructive.

Example 1. Let ρ be an a-qubit density operator and Q be a superoperator with a input qubits and b output qubits. Consider the tensor network that connects all wires of the tensor ρ to the input wires of the tensor Q. Then contracting this tensor network gives the tensor of the density operator $Q(\rho)$. Figure 3 illustrates this example.

A quantum circuit C can be naturally regarded as a tensor network $N(C)$: each gate is regarded as the corresponding tensor. The qubit lines are wires connecting the tensors, or open wires that correspond to the input and output qubits. Figure 4 illustrates the concept.

Let C be a quantum circuit with n input qubits and m output qubits. Suppose that C is applied to the initial state ρ_x, for some $x \in \{0,1\}^n$. We are interested in

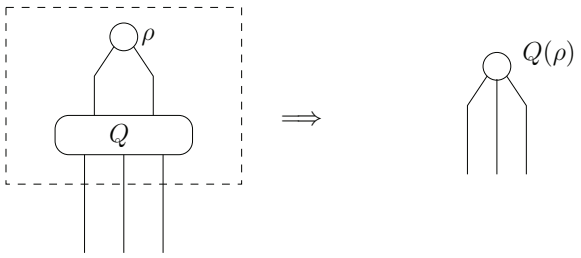


FIG. 3. *Contracting the wires connecting the tensors for a density operator ρ and a gate Q results in the tensor for $Q(\rho)$.*

knowing the probability of observing some particular outcome when some single-qubit measurements are applied to a subset of the qubits. The setting can be described by a measurement scenario defined as follows.

DEFINITION 3.4. *Let $m \geq 1$ be an integer. A* measurement scenario *on m qubits is a function $\tau : [m] \to \mathbf{L}(\mathbb{C}^2)$, such that $\tau(i)$ is a single-qubit POVM measurement element.*

Note that if a qubit i is not to be measured, then we can set $\tau(i) = I$.

To compute the probability that τ is realized on $C(\rho_x)$, we build a tensor network $N(C; x, \tau)$ from $N(C)$ by attaching to each input open wire i the tensor for $|x_i\rangle\langle x_i|$, and attaching to each open wire for the output qubit i the tensor for $\tau(i)$. When $x = 0^n$, we abbreviate $N(C; x, \tau)$ as $N(C; \tau)$. Figure 4 illustrates the concept of $N(C)$ and $N(C; \tau)$.

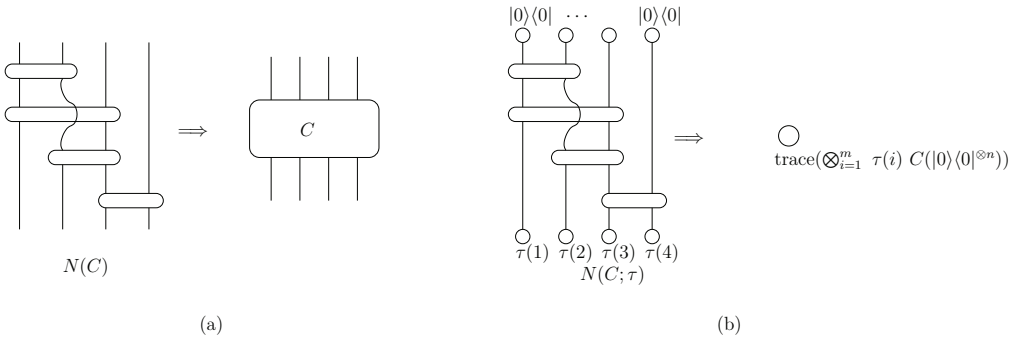


FIG. 4. *In* (a)*, a circuit C can be naturally regarded as a tensor network $N(C)$. Contracting $N(C)$ gives the tensor for the operator that C realizes. Part* (b) *illustrates the tensor network $N(C; \tau)$, contracting which gives the rank-*0 *tensor whose value is precisely the probability that the measurement scenario τ is realized on $C(|0\rangle\langle 0|^{\otimes n})$.*

PROPOSITION 3.5. *Let C be a quantum circuit, x be a binary string, and τ be a measurement scenario. Contracting the tensor network $N(C; x, \tau)$ to a single vertex gives the rank-*0 *tensor which is the probability that τ is realized on $C(\rho_x)$.*

Proof. Let $\rho^t \stackrel{\text{def}}{=} g_t g_{t-1} \cdots g_1(\rho_x)$, $1 \leq t \leq T$, and $\rho^0 = \rho_x$. By the definitions of tensors for density operators and superoperators and tensor contraction, contracting wires connecting the tensor of a superoperator Q and the tensors for a density operator ρ gives the tensor of $Q(\rho)$. Thus sequentially contracting input wires of $g_1, \ldots, g_t$ gives the tensor for ρ^t, and contracting the remaining wires gives the tensor for $\tau(\rho^T)$, which is the probability of realizing τ on $\rho^T = C(\rho_x)$. ∎

We remark that $N(C; x, \tau)$ is not the only tensor network for which the previous proposition holds.

Although the ordering of the edges in the contraction process does not affect the final tensor, it may significantly affect space and time requirements.

PROPOSITION 3.6. *Given a tensor network N of a size T quantum circuit, and a contraction process specified by an ordering of wires in N, let d be the maximum rank of all the tensors that appear in the process. Then the contraction takes $O(T \exp[O(d)])$ time.*

Proof. Note that the size of N is $\Theta(T)$. The algorithm stores the tensors of each vertex. When contracting an edge, it computes the new tensor according to (1) and updates the tensor accordingly. This takes $\exp[O(d)]$ time. Hence the total runtime is $O(T \exp[O(d)])$. □

In the next section we will investigate near-optimal orderings for simulation and ways to find them. While traditional simulation of quantum circuits proceeds in the same order in which the gates are applied, it appears that an optimal ordering may not have any physical meaning. Therefore, we formalize this optimization using abstract graph contractions.

4. Contraction complexity and treewidth. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. Recall that the contraction process discussed in the previous section removes parallel edges in one step because contracting one edge at a time can create multiple loops. However, for future convenience we prefer the latter simulation and therefore allow loops to remain uncontracted, counting toward the degree of a vertex. Note that if a "parallel" contraction contracts ℓ edges between two vertices u and v of degrees $\ell + k$ and $\ell + k'$, respectively, the corresponding "one-edge-at-a-time" contraction would create vertices of degrees $k+k'+\ell-1$, $k+k'+\ell-2$, ..., $k+k'$, each of which is $\leq d(u) + d(v)$. Thus the one-edge-at-a-time contraction process can emulate the parallel contraction, while increasing the maximum vertex degree observed by no more than twofold. We make the definition of this new contraction process precise below.

DEFINITION 4.1. *The* contraction of an edge *e removes e and replaces its end vertices (or vertex) with a single vertex. A* contraction ordering *π is an ordering of all the edges of G, $\pi(1)$, $\pi(2)$, ..., $\pi(|E(G)|)$. The complexity of π is the maximum degree of a merged vertex during the contraction process. The* contraction complexity *of G, denoted by $\mathrm{cc}(G)$, is the minimum complexity of a contraction ordering.*

Since only the degrees of the merged vertices are considered in defining the contraction complexity, $\mathrm{cc}(G)$ could be strictly larger than $\Delta(G)$. For example, if G is a path, then $\mathrm{cc}(G) = 1$ and $\Delta(G) = 2$.

Note that sequentially contracting all $\pi(i)$, $1 \leq i \leq |E(G)|$, reduces G to a single vertex (or an empty graph of several vertices). Also, for any graph G, $\mathrm{cc}(G) \leq |E(G)| - 1$, since any merged vertex would be incident to no more than $|E(G)| - 1$ number of edges. Furthermore, $\mathrm{cc}(G) \geq \Delta(G) - 1$, since when an edge incident to a vertex of degree $\Delta(G)$ is removed, the resulting merged vertex is incident to at least $\Delta(G) - 1$ edges.

The nature of $\mathrm{cc}(G)$ becomes clearer once we consider the *line graph* of G, denoted by G^*. That is, the vertex set of G^* is $V(G^*) \stackrel{\text{def}}{=} E(G)$, and the edge set is

$$E(G^*) \stackrel{\text{def}}{=} \{e_1, e_2\} \subseteq E(G) : e_1 \neq e_2, \exists v \in V(G)$$

such that e_1 and e_2 are both incident to v.

PROPOSITION 4.2. *For any graph $G = (V, E)$, $\mathrm{cc}(G) = \mathrm{tw}(G^*)$. Furthermore, given a tree decomposition of G^* of width d, there is a deterministic algorithm that outputs a contraction ordering π with $\mathrm{cc}(\pi) \leq d$ in polynomial time.*

Computing the treewidth of an arbitrary graph is NP-hard [6], but we do not know if this remains true for the special class of graphs G^*. Nevertheless, this is not critical in our work since the constant-factor approximation due to Robertson and Seymour [28] suffices for us to prove our key results.

THEOREM 4.3 (see Robertson and Seymour [28]). *There is a deterministic algorithm that, given a graph G, outputs a tree decomposition of G of width $O(\mathrm{tw}(G))$ in time $|V(G)|^{O(1)} \exp[O(\mathrm{tw}(G))]$.*

Proof of Proposition 4.2. There is a one-to-one correspondence of the contraction of an edge in G and the elimination of a vertex in G^*, and the degree of the merged vertex resulting from contracting an edge e in G is the same as the degree of e being eliminated in G^*. Thus $\mathrm{cc}(G) = \mathrm{tw}(G^*)$.

To prove the second part of the statement, denote the tree decomposition by $\mathcal{T}$. Repeat the following until the tree decomposition becomes an empty graph. Choose a leaf ℓ in $\mathcal{T}$. If ℓ is the single vertex of $\mathcal{T}$, then output vertices (of G^*) in B_ℓ in any order. Otherwise, let ℓ' be its parent. If $B_\ell \subseteq B_{\ell'}$, then remove ℓ and repeat this process. Otherwise, let $e \in B_\ell - B_{\ell'}$. Then, output e, remove it from the tree decomposition, and continue the process until all vertices of the tree decomposition are removed. The number of steps in this process is polynomial in the size of the tree decomposition.

Note that each output e appears in only one bag in the tree decomposition. Therefore, all (current) neighbors of e must appear in the same bag. Hence its induced width is at most d, by the one-to-one correspondence of the vertex elimination in G^* and the contraction process in G, $\mathrm{cc}(\pi) \leq d$. □

Before we complete the description of our simulation algorithm, we relate the treewidth of G to that of G^*. This is useful for reasoning about quantum circuits C when the graph G_C is easier to analyze than its line graph G_C^*. In such cases one hopes to bound the runtime of the simulation algorithm in terms of parameters of G rather than G^*. Fortunately, since G_C is of bounded degree, the treewidths of G_C and G_C^* are asymptotically the same.

LEMMA 4.4. *For any graph G of maximum degree $\Delta(G)$,*

$$(\mathrm{tw}(G) - 1)/2 \leq \mathrm{tw}(G^*) \leq \Delta(G)(\mathrm{tw}(G) + 1) - 1.$$

Proof. From a tree decomposition $\mathcal{T}$ of G of width d we obtain a tree decomposition $\mathcal{T}^*$ of G^* of width $(d+1) \cdot \Delta(G) - 1$ by replacing each vertex $v \in V(G)$ with all edges e incident to v. This guarantees that every edge of G^* is in some bag, i.e., (T1) is true. Item (T2) is true since if e_1 and e_2 are both incident to a vertex u in G, then any bag in $\mathcal{T}$ containing u contains both e_1 and e_2 in $\mathcal{T}^*$. To verify item (T3), suppose that e connects u and v in $V(G)$. Take two bags a and b that both contain e. Then, in $\mathcal{T}$, both bags a and b must have either u or v. If they contain the same vertex, then a and b are connected, by (T3). Otherwise, there must be a bag c that contains both u and v, by (T2). So a and b are connected through c. Therefore we have proved that $\mathrm{tw}(G^*) \leq \Delta(G)(\mathrm{tw}(G) + 1) - 1$.

Now to prove $\mathrm{tw}(G) \leq 2\mathrm{tw}(G^*) + 1$, we start with a tree decomposition $\mathcal{T}^*$ of G^* of width d, and replace every e by its two endvertices in $V(G)$. The verification of (T1) through (T3) can be accomplished in a similar way. □

Note that the above bounds are asymptotically tight, since for an m-ary tree (of which each nonroot internal vertex has degree $m + 1$), the treewidth is 1 and the

contraction complexity is m. We summarize the previous finding in the following theorem.

THEOREM 4.5. *Let $d \geq 1$ be an integer. For any family of graphs G_n, $n \in \mathbb{N}$, such that $\Delta(G_n) \leq d$ for all n. Then*

$$(\mathrm{tw}(G_n) - 1)/2 \leq \mathrm{cc}(G_n) = \mathrm{tw}(G_n^*) \leq d(\mathrm{tw}(G_n) + 1) - 1 \quad \forall n \in \mathbb{N}.$$

We are now ready to put everything together to prove the following restatement of Theorem 1.1.

THEOREM 4.6. *Let C be a quantum circuit of size T and with n input and m output qubits, $x \in \{0,1\}^n$ be an input, and $\tau : [m] \to \mathbf{L}(\mathbb{C}^2)$ be a measurement scenario. Denote by G_C the underlying circuit graph of C. Then the probability that τ is realized on $C(\rho_x)$ can be computed deterministically in time $T^{O(1)} \exp[O(\mathrm{cc}(G_C))] = T^{(1)} \exp[O(\mathrm{tw}(G_C))]$.*

Proof. The following algorithm computes the desired probability.

(1) Construct $N = N(C; x, \tau)$.
(2) Apply the Robertson–Seymour algorithm to compute a tree decomposition $\mathcal{T}$ of N^* of width $w = O(\mathrm{tw}(N^*))$ (Theorem 4.3).
(3) Find a contraction ordering π from $\mathcal{T}$ (Proposition 4.2) of width w.
(4) Contract N using π, and output the desired probability from the final (rank-0) tensor (Proposition 3.5).

The runtime bottlenecks are steps (2) and (4), which can be combined, and both take time $T^{O(1)} \exp(O[\mathrm{tw}(N^*)])$, which by Theorem 4.5 is $T^{O(1)} \exp[O(\mathrm{cc}(G_C))] = T^{O(1)} \exp[O(\mathrm{tw}(G_C))]$; for the sake of clarity we separate both steps . ∎

5. Treewidth and quantum circuits. In this section we prove the implications of Theorem 1.1 stated in the introduction. A number of tight bounds for the treewidth of specific families of graphs have been published, including those for planar and series-parallel graphs. However, similar results for graphs derived from quantum circuits are lacking. To this end, we strengthen Corollary 1.5 as follows.

PROPOSITION 5.1. *Let C be a quantum circuit in which each gate has an equal number of input and output qubits, and whose qubits are index by $[n]$, for an integer $n \geq 1$. Suppose that the size of C is T, and r is the minimum integer so that for any i, $1 \leq i \leq n-1$, no more than r gates act on some qubits j and j' with $j \leq i < j'$. Then C can be simulated deterministically in time $T^{O(1)} \exp[O(r)]$.*

Corollary 1.5 follows since $r = O(qD)$ under its assumption.

Proof of Proposition 5.1. Assume without loss of generality that $\mathrm{tw}(G_C) \geq 2$. Let G be the graph obtained from G_C by removing degree 1 vertices and contracting edges incident to degree 2 vertices. Then $\mathrm{tw}(G) = \mathrm{tw}(G_C)$, by Proposition 2.1 and the observation stated before it. Then each vertex in G corresponds to a multiqubit gate in C.

We now construct a tree decomposition $\mathcal{T}$ for G that forms a path of $n-1$ vertices B_1–B_2–$\cdots$–B_{n-1}. The bag B_i of the ith vertex ($1 \leq i \leq n-1$) consists of multiqubit gates (vertices) that act on some qubits j and j' with $j \leq i < j'$. Hence $|B_i| \leq r$ by the assumption. If u acts on qubits $i_1, i_2, \ldots, i_k$, $i_1 < i_2 < \cdots < i_k$, then $u \in B_i$ for all i, $i_1 \leq i \leq i_k$. Thus (T1) and (T3) are true. If a wire segment corresponding to the qubit i connects two gates u and v, then the bag B_i contains both u and v. Thus (T2) is true. Therefore $\mathcal{T}$is a tree decomposition for G with width $r-1$. Hence $\mathrm{tw}(G_C) = \mathrm{tw}(G) = O(r)$, which by Theorem 1.1 implies that C can be simulated in $T^{O(1)} \exp[O(r)]$ time. ∎

We now turn to quantum circuits of bounded depth. To prove Theorem 1.7 we will make use of the following observation that relates expander graphs to contraction complexity. Let d be a constant and $\{G_n\}_{n\in\mathbb{N}}$ be a family of d-regular graphs, and $\epsilon > 0$ be a universal constant. Recall that $\{G_n\}$ is called a family of expander graphs with expansion parameter ϵ if, for any subset $S \subseteq V(G_n)$ with $|S| \le |V(G_n)|/2$, there are no less than $\epsilon|S|$ edges connecting vertices in S with vertices in $V(G) - S$.

LEMMA 5.2. *For an expander graph G_n with the expansion parameter ϵ,* $\mathrm{cc}(G_n) \ge \epsilon|V(G_n)|/4$.

Proof. Fix a contraction ordering of G_n. Let v be the first merged vertex so that k_v, the number of vertices in $V(G_n)$ that were eventually merged to v, is at least $|V(G_n)|/4$. Then $k_v \le |V(G_n)|/2$, and v must have a degree $\epsilon|V(G_n)|/4$. ∎

The following graph is shown to be an expander by Lubotzky, Phillips, and Sarnak [18]. Let $p > 2$ be a prime, and G_n be the graph with $V(G_p) \stackrel{\mathrm{def}}{=} \mathbb{Z}_p \cup \{\infty\}$. Every vertex x is connected to $x+1$, $x-1$, and x^{-1} ($\infty \pm 1$ are defined to be ∞). Note that G_p is a 3-regular graph.

Proof of Theorem 1.7. By Lemma 5.2, $\mathrm{cc}(G_p) = \Omega(p)$. Since G_p is a 3 regular graph, $\mathrm{tw}(G_p) = \Theta(\mathrm{cc}(G_p)) = \Omega(p)$, by Theorem 4.5. Let G'_p be the graph obtained from G_p by removing the vertex ∞ and the edge $\{0, p-1\}$. This would only decrease $\mathrm{tw}(G_p)$ by at most constant. Hence $\mathrm{tw}(G'_p) = \Omega(p)$. Therefore to prove the theorem, it suffices to construct a quantum circuit C on p qubits so that G'_p is a minor of G^*_C.

Each qubit of C corresponds to a distinct vertex in $V(G'_p)$. Observe that edges in $E(G'_p)$ can be partitioned into three vertex-disjoint subsets: (1) $\{x, x^{-1}\}$; (2) $\{x, x+1\}$ for even x, $0 \le x \le p-3$; (3) the remaining edges. Each subset gives a layer of two-qubit gates in C. In G^*_C, contracting all of the vertices that correspond to the same qubit gives a graph of which G'_p is a minor. Hence $\mathrm{tw}(C) = \Theta(\mathrm{tw}(G^*_C)) = \Omega(p)$. ∎

Proof of Theorem 1.8. By Theorem 4.5, it suffices to prove that $\mathrm{cc}(G_C) = O(1)$ for any depth-2 circuit. Observe that for any such circuit, after contracting the input and output vertices (those are of degree 1, hence contracting them will not increase the contraction complexity), every vertex in G_C has degree either 1 or 2. Hence the edges can be decomposed into disjoint paths and cycles, which can be contracted without increasing the degree. Hence $\mathrm{cc}(G_C) \le 2$. ∎

6. Simulating one-way quantum computation. This section revisits the notions of *graph states* and *one-way quantum computation*. We simulate one-way computation with an algorithm whose complexity grows exponentially with the contraction complexity of the underlying graph.

Let $G = (V, E)$ be a simple undirected graph with $|V| = n$. For a subset $V' \subseteq V$, denote by $e(V')$ the number of edges in the subgraph induced by V'. We associate a qubit with each vertex $v \in V$ and refer to it by qubit v. For a subset $V' \subseteq V$, we identify the notation $|V'\rangle$ with the computational basis $|x\rangle$, for $x \in \{0,1\}^n$ being the characteristic vector of V' (i.e., the ith bit of x is 1 if and only if the ith vertex under some fixed ordering is in V'). The graph state $|G\rangle$ is the following n-qubit quantum state [7]

$$|G\rangle \stackrel{\mathrm{def}}{=} \frac{1}{\sqrt{2^n}} \sum_{V' \subseteq V} (-1)^{e(V')} |V'\rangle.$$

Note that $|G\rangle$ can be created from $|0^n\rangle$ by first applying Hadamard gates to all qubits, followed by the controlled-phase gate $\Lambda(\sigma^z) = \sum_{b_1,b_2\in\{0,1\}} (-1)^{b_1 \cdot b_2} |b_1, b_2\rangle\langle b_1, b_2|$ on each pair of qubits u and v with $\{u, v\} \in E$. Since all of the $\Lambda(\sigma^z)$ operators commute, the order of applying them does not affect the result.

A basic building block of our simulation algorithm is the following.

LEMMA 6.1. *Let $G = (V, E)$ be a graph with n vertices, and let τ be a measuring scenario (defined in Definition 3.4) on n qubits. Then the probability p that τ is realized on $|G\rangle$ can be computed deterministically in time $O(|V|^{O(1)} \exp[O(\mathrm{cc}(G))])$.*

Proof. Fix a circuit C_G that creates $|G\rangle$ from $|0\rangle\langle 0|^{\otimes n}$. Let $\{u, v\} \in E$ and $g = g_{u^+,u^-,v^+,v^-}$ be a tensor in $N(C_G; \tau)$ corresponding to $\Lambda(\sigma^z)[u, v]$. The wires representing the qubit u (or v) before and after the gate are labeled u^+ (or v^+) and u^- (or v^-), respectively. We replace g by two tensors $g^u = g^u_{u^+,u^-,t^+,t^-}$ and $g^v = g^v_{v^+,v^-,t^+,t^-}$, which share two labels t^+ and t^- and are defined as follows. For a wire segment with a label a, denote by $\mathbf{L}_a$ the 4-dimensional space of linear operators associated with this wire segment. Set g^u to be the identity superoperator that maps $\mathbf{L}_{u^+} \otimes \mathbf{L}_{t^-} \to \mathbf{L}_{t^+} \otimes \mathbf{L}_{u^-}$, and g^v to be the tensor for a $\Lambda(\sigma^z)$ that maps $\mathbf{L}_{t^+} \otimes \mathbf{L}_{v^+} \to \mathbf{L}_{t^-} \otimes \mathbf{L}_{v^-}$. By their definitions, contracting g^u and g^v gives precisely g. We call the inserted wires labeled with t^+ and t^- *transition wires*. See Figure 5 for an illustration.

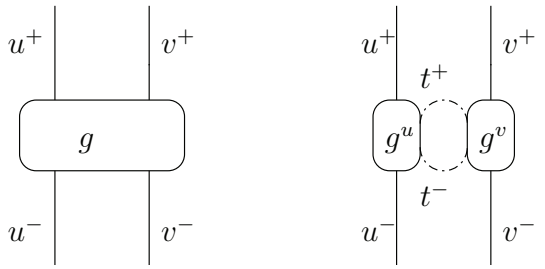


FIG. 5. *Replacing a tensor g corresponding to $\sigma_z[u, v]$ by two tensors g^u and g^v.*

Denote by $N'(C_G; \tau)$ the tensor network obtained from $N(C_G; \tau)$ by applying the above replacement procedure for each edge in E. Let G' be the underlying graph of $N'(C_G; \tau)$. Note that G' has the maximum degree 4 and the number of vertices is $O(|E|)$. See Figure 6 for an illustration. Thus p can be computed by contracting $N'(C_G; \tau)$ in time $O(|V|^{O(1)} \exp[O(\mathrm{cc}(G'))])$, according to Theorem 4.6.

We now prove that $\mathrm{cc}(G') = O(\mathrm{cc}(G))$. This can be seen by contracting all wire segments corresponding to the same qubit in G', while leaving the transition wires untouched. Since contracting the edge incident to an input or output vertex results in a new vertex of degree 3, and contracting the rest of the wires for a qubit v results in a new vertex of degree $2d(v)$, the maximum degree of a merged vertex in this process is $\max\{3, 2\Delta(G)\}$. The one-to-one correspondence between the resulting vertex set and V induces naturally a one-to-one correspondence between the pairs of transition wires and E. Thus a contraction ordering of G gives a contraction ordering of G' (of this stage) with at most twice the contraction complexity. Therefore

$$\mathrm{cc}(G') \le \max\{3, 2\Delta(G), 2\mathrm{cc}(G)\} = O(\mathrm{cc}(G) + 1).$$

Thus p can be computed deterministically in time $O(|V|^{O(1)} \exp[O(\mathrm{cc}(G'))])$ $= O(|V|^{O(1)} \exp[O(\mathrm{cc}(G))])$. ∎

A *one-way* computation on a quantum state $|\phi\rangle$ consists of a sequence of adaptive single-qubit measurements and single-qubit unitary operations applied to $|\phi\rangle$. The description of each measurement or unitary operation, including the index of the qubit that it acts on, can be computed by a deterministic and efficient (polynomial

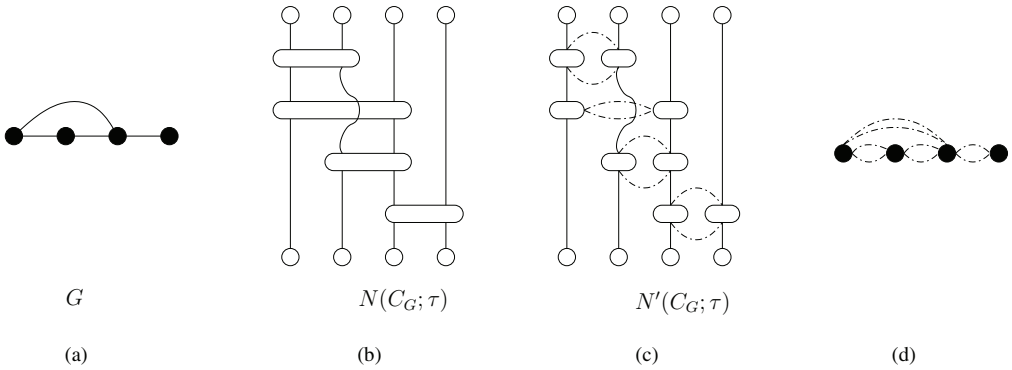


FIG. 6. *For a graph G in* (a)*, the tensor network $N(C_G;\tau)$ is shown in* (b)*. Input vertices are at the top, and output vertices are at the bottom. Each box is a tensor corresponding to a $\Lambda(\sigma_z)$ applied to qubits adjacent in G. In* (c)*, each $\Lambda(\sigma_z)$ tensor is replaced by two tensors and two wires connecting them, as described in Figure* 5*. Contracting all solid lines in* (c) *produces the graph in* (d)*, which is precisely G with each edge doubled.*

time) algorithm from previous operations and their measurement outcomes. In our discussion we treat this computation time as a constant. We call a one-way quantum computation *oblivious* if before the last measurement (which produces the outcome of the computation), different computational paths involve the same number of measurements, take place with the same probability, and result in an identical state. Note that the one-way computation of Raussendorf and Briegel [25] is oblivious.

We point out that allowing single-qubit unitary operations in the definition is for the convenience of discussion only, since each single-qubit unitary can be combined with a future measurement on the same qubit (should there be one). To see this fact, let us call two quantum states *LU-equivalent* (where LU stands for local unitary), if there exists a set of single-qubit unitary operations applying which maps one state to the other. A one-way computation with unitary operators always has an almost identical one-way computation without unitary operations: the measurements are in one-to-one correspondence with identical outcome distributions, and the states after corresponding measurements are LU-equivalent. Therefore, when we are only interested in the distribution of the measurement outcomes, we may assume without loss of generality that a one-way computation does not involve any unitary operation.

We now derive a simulation algorithm whose complexity depends on the contraction complexity.

THEOREM 6.2. *A one-way quantum computation on a graph $G = (V, E)$ can be simulated by a randomized algorithm in time $O(|V|^{O(1)} \exp[O(\mathrm{cc}(G))])$. If the one-way computation is oblivious, then the simulation can be made deterministic.*

Theorem 1.6 follows immediately from the combination of the above theorem with Theorem 4.5.

Proof of Theorem 6.2. Let T be the number of measurements during the one-way computation. Assume without loss of generality that no single-qubit unitary operation is applied. The simulation consists of T steps, one for each single-qubit measurement. It maintains a data structure $r = (\tau, p)$, where τ is a measurement scenario, and p is the probability that τ is realized on $|G\rangle$. Denote by $r_t = (\tau_t, p_t)$ the value of r when t measurements have been simulated. Initially $\tau_0(i) = I$ for all i, $1 \le i \le n$, and $p_0 = 1$.

Suppose we have simulated the first $t-1$ measurements, $1 \le t \le T-1$.

(1) Based on the one-way algorithm, compute from τ_{t-1} the description of the tth measurement $P_t = \{P_t^0, P_t^1\}$ and the qubit a_t that it acts on. Denote by τ_t^0 the measurement scenario identical to τ_{t-1}, except that $\tau_t^0(a_t) = P_t^0$.
(2) Compute p_t^0, the probability of realizing τ_t^0. By Lemma 6.1, this step takes $O(|V|^{O(1)} \exp[O(\mathrm{cc}(G))])$ time.
(3) Flip a coin that produces 0 with probability p_t^0/p_{t-1}, resulting in an outcome $b_t \in \{0,1\}$. Set τ_t to be identical to τ_{t-1}, except that $\tau(a_t) = p_t^{b_t}$. Set $p_t = (1-b_t)p_t^0 + b_t(p_{t-1} - p_t^0)$. Continue the simulation until $t = T$.

By construction, the output distribution is identical to that of the one-way computation. The complexity of the algorithm is $O(|V|^{O(1)} \exp[\mathrm{cc}(G)])$.

If the one-way computation is oblivious, then there is no need to adaptively simulate the first $T-1$ measurements, as all of them lead to the same state with the same probability p_{T-1}. Let τ_{T-1} (τ_T) be the measurement scenario corresponding to the first $T-1$ (T, respectively) measurements giving the outcome 0. We compute the probabilities p_{T-1} and p_T that τ_{T-1} and τ_T are realized. Then the probability that the one-way computation produces 0 is precisely p_T/p_{T-1}. The computation is deterministic and takes $|V|^{O(1)} \exp[O(\mathrm{cc}(G))]$ time by Lemma 6.1. □

7. Discussion. In this work we studied quantum circuits regardless of the types of gates they use, but with a focus on how the gates are connected. We have shown that quantum circuits that look too similar to trees do not offer significant advantage over classical computation. More generally, when solving a difficult classical problem on a quantum computer, one encounters an *inherent* trade-off between the treewidth and the size of quantum circuits for solving it—the smaller the quantum circuit, the more topologically sophisticated it must be. Investigating such trade-offs for specific problems of interest is an entirely open and very attractive avenue for future research. Similar considerations may apply to classical circuits. We conjecture that there are simple functions, such as modular exponentiation, whose circuit realizations require large treewidth.

Furthermore, our work raises an intriguing possibility that the treewidth of some quantum circuits may be systematically reduced by restructuring the circuit, while preserving the final result of the entire computation. Perhaps future research in this direction can clarify the limits to efficient quantum computation, while the tools developed in this context will be useful for practical tasks.

The preprint of this paper [19] has lead to several follow-up results. Jozsa [15] and Aharonov, Landau, and Makowsky [3] gave alternative proofs for some of our theorems. Furthermore, Aharonov, Landau, and Makowsky [3], and Yoran and Short [39] pointed out that quantum Fourier transform (QFT) over $\mathbb{Z}_n$ admits approximate circuit realizations that, viewed as tensor networks, have small treewidth. Given the central role of QFT in known quantum algorithms, their results are somewhat unexpected and their implications are yet to be fully explored. For example, what type of circuits would remain efficiently simulatable when interleaved with QFT circuits? In general, as implied by Theorems 1.7 and 1.8, the treewidth of a circuit may increase dramatically under composition. Yoran and Short [39] have shown that this drawback may be avoided in some cases. Extending their result would deepen our understanding of quantum speed-ups.

As mentioned earlier, Theorem 1.6 was improved in [19] so that any one-way quantum computation on a graph state $|G\rangle$ can be simulated deterministically in $|V(G)|^{O(1)} \exp(\mathrm{tw}(G))$ time. The proof relies on a graph-theoretical result from [20]:

for any graph G there is a graph G' such that $\Delta(G') \leq 3$, $\mathrm{tw}(G') \leq \mathrm{tw}(G) + 1$, and that G can be restored from G' by contracting a subset of edges forming a forest. A small modification turns G' into a graph G'' with $\Delta(G'') \leq 3$, $\mathrm{tw}(G'') \leq \mathrm{tw}(G) + 1$, and $|G\rangle$ can be constructed from $|G''\rangle$ through an efficient one-way computation. Since $\mathrm{cc}(G'') = O(\mathrm{tw}(G'')) = O(\mathrm{tw}(G))$ (Theorem 4.5), this implies, by Theorem 6.2, that any one-way computation on $|G\rangle$ can be simulated deterministically in time $|V(G)|^{O(1)} \exp(O(\mathrm{tw}(G)))$. The interested reader is referred to [19, 20] for details.

The important question of characterizing quantum states that are universal (or efficiently simulatable) for one-way quantum computation remains unsolved. In another follow-up thread, van den Nest et al. [22, 21] defined additional width-based parameters of quantum states and demonstrated results for those parameters similar to Theorem 1.6. It is unlikely that the set of quantum states with small width-based parameters includes all efficiently simulatable states because a set of simulatable states of high widths was identified recently by Bravyi and Raussendorf [9]. Nevertheless, it remains plausible that those width-based results and their further extensions may be part of a classification theorem that gives a complete characterization of efficiently simulatable states.

Appendix. Proof of Proposition 2.1. Recall that a minor of a graph G is a graph obtained from a subgraph of G by contracting edges. A basic property of treewidth is that it does not increase under taking minors [26].

Proof of Proposition 2.1. Let G' be the graph resulting from the contractions. Since G' is a minor of G, $\mathrm{tw}(G') \leq \mathrm{tw}(G)$ [26]. If $\mathrm{tw}(G') = 1$, then G' is a nonempty forest (otherwise, G has a triangle minor, thus $\mathrm{tw}(G) \geq 2$). Thus G is also a nonempty forest and $\mathrm{tw}(G) = 1 = \mathrm{tw}(G')$. Suppose $\mathrm{tw}(G') \geq 2$. Let $\mathcal{T}$ be a tree decomposition for G'. We obtain a tree decomposition $\mathcal{T}'$ for G by inserting a bag containing $\{u, w, v\}$, and connecting it to a bag that contains $\{u, v\}$. One can verify directly that the three conditions $(T_1 - T_3)$ that define tree decompositions hold for $\mathcal{T}'$. Since the width of $\mathcal{T}'$ is no more than that of $\mathcal{T}$, we have $\mathrm{tw}(G) \leq \mathrm{tw}(G')$. Therefore, $\mathrm{tw}(G) = \mathrm{tw}(G')$. □

Acknowledgments. We thank George Viamontes and John Hayes for motivating this study and helpful discussions. We are grateful to Guifré Vidal, Frank Verstraete, Ashwin Nayak, Tzu-Chieh Wei, and the anonymous reviewers for many valuable comments, including pointing out relevant previous works. Yaoyun Shi is grateful to Alexei Kitaev for introducing him to the general concept of tensor networks, and to Peng-Jun Wan for useful comments. Igor Markov is grateful to Ike Chuang for useful discussions.

REFERENCES

[1] S. Aaronson and D. Gottesman, *Improved simulation of stabilizer circuits*, Physical Rev. A, 70 (2004), p. 052328-1–052328-14.

[2] D. Aharonov, A. Kitaev, and N. Nisan, *Quantum circuits with mixed states*, in Proceedings of the 30th Annual ACM Symposium on the Theory of Computation (STOC), Dallas, TX, ACM Press, New York, 1998, pp. 20–30.

[3] D. Aharonov, Z. Landau, and J. Makowsky, *The quantum FFT can be classically simulated*, preprint: quant-ph/0611156.

[4] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier, *Fixed parameter algorithms for dominating set and related problems on planar graphs*, Algorithmica, 33 (2002), pp. 461–493.

[5] S. Arnborg, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey*, BIT, 25 (1985), pp. 2–23.

[6] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Algebraic and Discrete Methods, 8 (1987), pp. 277–284.
[7] H. J. Briegel and R. Raussendorf, *Persistent entanglement in arrays of interacting particles*, Phys. Rev. Lett., 86 (2001), pp. 910–913.
[8] H. L. Bodlaender, *Treewidth: Characterizations, Applications, and Computations*, Technical report UU-CS-2006-041, Universiteit Utrecht, Utrecht, The Netherlands, 2006.
[9] S. Bravyi and R. Raussendorf, *On measurement-based quantum computation with the toric code states*, preprint: quant-ph/0610102.
[10] E. Broering and S. Lokam, *Width-based algorithms for SAT and Circuit-SAT (extended abstract)*, in Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), Santa Margherita, Italy, Lecture Notes in Comput. Sci. 2919, Springer-Verlag, 2004, pp. 162–171.
[11] R. Dechter, *Bucket elimination: A unifying framework for reasoning*, Artificial Intelligence, 113 (1999), pp. 41–85.
[12] D. Gottesman, *The Heisenberg representation of quantum computers*, in Proceedings of the 22nd International Colloquium on Group Theoretical Methods in Physics, Group 22: S. P. Corney, R. Delbourgo, and P. D. Jarvis, eds., International Press, Cambridge, MA, 1999, pp. 32–43. Long version: quant-ph/9807006.
[13] S. A. Fenner, F. Green, S. Homer, and Y. Zhang, *Bounds on the power of constant-depth quantum circuits*, in Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT), Lübeck, Germany, 2005, Lecture Notes in Comput. Sci. 3623, M. Liskiewicz and R. Reischuk, eds., Springer, Berlin, 2005, pp. 44–55.
[14] A. W. Joshi, *Matrices and Tensors in Physics*, Halsted Press [John Wiley & Sons], New York-London-Sydney, 1975.
[15] R. Jozsa, *On the simulation of quantum circuits*, preprint: quant-ph/0603163.
[16] R. Jozsa and N. Linden, *On the role of entanglement in quantum computational speed-up*, Lond. Proc. R. Soc. Ser. A Math. Phys. Eng. Sci., 459, 2003, pp. 2011–2032.
[17] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, *Accurate reliability evaluation and enhancement via probabilistic transfer matrices*, in Proceedings of the Design Automation and Test in Europe (DATE'05), Munich, Germany, 2005, pp. 282–287.
[18] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[19] I. L. Markov and Y. Shi, *Simulating quantum computation by contracting tensor networks*, preprint: quant-ph/0511069.
[20] I. L. Markov and Y. Shi, *Constant Degree Graph Expansions and Treewidth*, manuscript.
[21] M. van den Nest, W. Dür, G Vidal, and H J. Briegel, *Classical simulation versus universality in measurement based quantum computation*, Phys. Rev. A, 75 (2007), p. 012337.
[22] M. van den Nest, A. Miyake, W. Dür, and H. J Briegel, *Universal resources for measurement-based quantum computation*, Phys. Rev. Lett., 97 (2006), p. 150504.
[23] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[24] D. Porras, F. Verstraete, and J. I. Cirac, *Renormalization algorithm for the calculation of spectra of interacting quantum systems*, preprint: cond-mat/0504717.
[25] R. Raussendorf and H. J. Briegel, *A one-way quantum computer*, Phys. Rev. Lett., 86 (2001), pp. 5188–5191.
[26] N. Robertson and P. D. Seymour, *Graph minors.* II. *Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.
[27] N. Robertson and P. D. Seymour, *Graph minors.* III. *Planar tree-width*, J. Comb. Theory Series B, 36 (1984), pp. 49–64.
[28] N. Robertson and P. D. Seymour, *Graph minors.* X. *Obstructions to tree-decomposition*, Journal of Comb. Theory Ser. B, 52 (1991), pp. 153–190.
[29] D. J. Rose, *Triangulated graphs and the elimination process*, J. Math. Anal. Appl., 32 (1970), pp. 597–609.
[30] V. P. Roychowdhury and F. Vatan, *Quantum formulas: A lower bound and simulation*, SIAM J. Comput., 31 (2001), pp. 460–476.
[31] B. M. Terhal and D. P. DiVincenzo, *Classical simulation of noninteracting-fermion quantum circuits*, Phys. Rev. A, 65 (2002), pp. 32325–32334.
[32] B. M. Terhal and D. P. DiVincenzo, *Adaptive quantum computation, constant depth quantum circuits and Arthur-Merlin games*, Quantum Inf. Comput., 4 (2004), pp. 134–145.
[33] L. G. Valiant, *Quantum circuits that can be simulated classically in polynomial time*, SIAM J. Comput., 31 (2002), pp. 1229–1254.

[34] F. Verstraete and J. I. Cirac, *Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions*, preprint: cond-mat/0407066.

[35] F. Verstraete, J. J. Garcia-Ripoll, and J. I. Cirac, *Matrix product density operators: Simulation of finite-temperature and dissipative systems*, Phys. Rev. Lett., 93 (2004), p. 207204.

[36] G. Vidal, *Efficient classical simulation of slightly entangled quantum computations*, Phys. Rev. Lett., 91 (2003), p. 147902.

[37] G. Vidal, *Efficient simulation of one-dimensional quantum many-body systems*, Phys. Rev. Lett., 93 (2004), p. 040502.

[38] P. Walther, K. J. Resch, T. Rudolph, E. Schenck, H. Weinfurter, V. Vedral, M. Aspelmeyer, and A. Zeilinger, *Experimental one-way quantum computing*, Nature, 434 (2005), pp. 169–176.

[39] N. Yoran and A. Short, *Efficient classical simulation of the approximate quantum Fourier transform*, Phys. Rev. A, 76 (2007), p. 042321.

[40] A. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, IEEE Comput. Soc. Press, Los Alamitos, CA, 1993, pp. 352–361.

[41] M. Zwolak and G. Vidal, *Mixed-State dynamics in one-dimensional quantum lattice systems: A time-dependent superoperator renormalization algorithm*, Phys. Rev. Lett., 93 (2004), p. 207205.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 982–1011

EFFICIENT COLORED ORTHOGONAL RANGE COUNTING*

HAIM KAPLAN†, NATAN RUBIN†, MICHA SHARIR‡, AND ELAD VERBIN†

Abstract. Let P be a set of n points in $\mathbb{R}^d$, so that each point is colored by one of C given colors. We present algorithms for preprocessing P into a data structure that efficiently supports queries of the following form: Given an axis-parallel box Q, count the number of distinct colors of the points of $P \cap Q$. We present a general and relatively simple solution that has a polylogarithmic query time and worst-case storage about $O(n^d)$. It is based on several interesting structural properties of the problem, which we establish here. We also show that for random inputs, the data structure requires almost linear expected storage. We then present several techniques for achieving space-time tradeoff. In $\mathbb{R}^2$, the most efficient solution uses fast matrix multiplication in the preprocessing stage. In higher dimensions we use simpler tradeoff mechanisms, which behave just as well. We give a reduction from matrix multiplication to the off-line version of problem, which shows that in $\mathbb{R}^2$ our time-space tradeoffs are reasonably sharp, in the sense that improving them substantially would improve the best exponent of matrix multiplication. Finally, we present a generalized matrix multiplication problem and show its intimate relation to counting colors in boxes in higher dimension.

Key words. colored orthogonal range counting, generalized range searching, matrix multiplication, union of orthants, output-sensitive decomposition, time-space tradeoff

AMS subject classifications. 68Q25, 68P05, 68U05, 65D18, 52C45, 52C35

DOI. 10.1137/070684483

1. Introduction.

We consider the following range counting problem. Let P be an input set of n points in $\mathbb{R}^d$, each colored in one of C different colors. Our goal is to preprocess P into a data structure that, for any query axis-parallel box $Q \subset \mathbb{R}^d$, can efficiently count the number of *distinct colors* of points in $Q \cap P$. We call this problem *colored orthogonal range counting*. In this work we deal only with the static setting of the problem, not allowing insertions of points into, or deletions, from the data structure.

The problem arises in many applications. For example, in database applications, a person may be stored in the database as an item with attributes (e.g., the city where the person has been born, the college that she has attended, or even numerical attributes such as her age), and the query may ask for the number of different *attribute values* of all the items in a query box (e.g., How many different cities of birth do the persons in a query box have?). In geometric contexts, the problem arises, e.g., in the following scenario: We are given C rectilinear polygons in the plane with a total of n edges and wish to count the number of distinct polygons that intersect a given query box (which represents, say, a window on the screen). Similar problems arise in higher

*Received by the editors March 7, 2007; accepted for publication (in revised form) February 15, 2008; published electronically June 25, 2008. This work is part of the second author's M.Sc. Dissertation, prepared under the supervision of the first and third authors. A preliminary version of this paper appeared in [21].

http://www.siam.org/journals/sicomp/38-3/68448.html

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (haimk@post.tau.ac.il, rubinnat@post.tau.ac.il, eladv@post.tau.ac.il). The first author's research was supported by grant 975/06 from the Israel Science Fund.

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il). This author's research was partially supported by NSF grant CCF-05-14079, by a grant from the U.S.-Israeli Binational Science Foundation, by grant 155/05 from the Israel Science Fund, Israeli Academy of Sciences, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

dimensions.

1.1. Related work. *Colored intersection range searching.* In colored intersection range searching we are given n colored objects of constant description complexity. We wish to construct a data structure that can report or count the colors of objects in a query range. Colored range searching problems[1] have been extensively studied by Gupta, Janardan, Smid, and others (see a recent survey by Gupta, Janardan, and Smid [12] for a comprehensive review). As has been observed, colored variants of range searching are in general much harder than the standard variants. The reason for this contrast is that colored problems are not *decomposable*. For example, partitioning the query box into two (disjoint) subranges and counting the number of colors in each subrange tells us practically nothing about the number of colors in the full range. Halfspace colored range searching (with halfspaces as queries and points as colored objects) was studied in [15]. Orthogonal colored range searching problems (where queries are axis-parallel boxes) were studied in [4, 5, 13, 16, 19]. Additional colored range searching problems were studied in [4, 5, 14, 15]. Batched colored range intersection problems, where we wish to report all pairs of colors (c_1, c_2) such that an object of color c_1 intersects an object of color c_2, were studied in [6, 20].

Orthogonal colored range reporting. In [16], Gupta, Janardan, and Smid describe solutions for colored range *reporting* in $\mathbb{R}^1$, $\mathbb{R}^2$, and $\mathbb{R}^3$, with polylogarithmic query time and near-linear storage. For $\mathbb{R}^1$, the authors perform orthogonal colored range searching (counting and reporting) in both static and dynamic settings, by a reduction to standard orthogonal range searching in $\mathbb{R}^2$. The authors obtain a dynamic data structure of size $O(n)$, such that the i distinct colors of the points in a query interval can be reported in $O(\log n + i)$ time, while supporting updates (insertions and deletions of points) in $O(\log n)$ time. Specifically, if the points of some color are $\{p_1, p_2, \ldots, p_n\}$, such that $p_1 < p_2 < \cdots < p_n$, then they are mapped to the points $(p_1, -\infty), (p_2, p_1), \ldots, (p_n, p_{n-1})$ in $\mathbb{R}^2$. A query interval $[a, b]$ is mapped to the semiunbounded rectangle $[a, b] \times (-\infty, a]$. It is an easy observation that $[a, b]$ contains at least one (resp., no) point of color c iff the corresponding rectangle contains exactly one (resp., no) transformed point of color c. Hence, counting or reporting colors in intervals in $\mathbb{R}^1$ is equivalent to counting or reporting points in the above kind of semiunbounded rectangles in the transformed set in $\mathbb{R}^2$. See Figure 1.1 for an illustration.

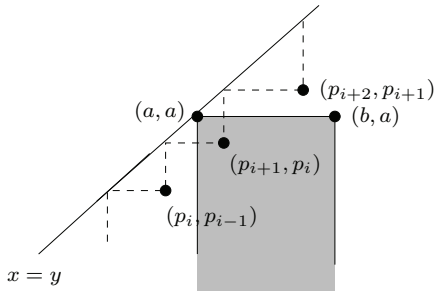


FIG. 1.1. *The transformed point set of color c (points below the line $x = y$). The highlighted semiunbounded rectangle $[a, b] \times (-\infty, a]$ contains exactly one transformed point of color c; the interval $[a, b]$ contains two original points of color c.*

[1] These problems are also referred to in the literature as *generalized range searching* problems; see [12].

In $\mathbb{R}^2$, the static data structure of Janardan and Lopez [19] uses $O(n \log^2 n)$ space and answers reporting queries in time $O(\log n + i)$, where as before, i is the output size. Gupta, Janardan, and Smid [16] obtain a semidynamic solution for the reporting problem by first dealing with the special case, when the queries are quadrants of the form $[a, \infty) \times [b, \infty)$. They reduce this special case to standard ray-segment intersection searching in $\mathbb{R}^2$. In order to process arbitrary rectangular queries, the authors decompose a rectangular query into four quadrant queries, each answered over an appropriate subset of points. The resulting data structure requires $O(n \log^2 n)$ space and allows the reporting of the i distinct colors of points contained in a query rectangle in $O(\log^2 n + i)$ time. A point can be inserted into this data structure in $O(\log^3 n)$ amortized time. Gupta, Janardan, and Smid [16] then give a fully dynamic solution to the reporting problem in $\mathbb{R}^2$. Their solution is based on decomposing the two-dimensional query into $O(\log n)$ one-dimensional queries, each answered over a proper subset of points. This data structure uses $O(n \log n)$ space, supports queries in $O(\log^2 n + i \log n)$ time, and allows insertions and deletions in $O(\log^2 n)$ time.

In $\mathbb{R}^3$, Gupta, Janardan, and Smid [16] extend their two-dimensional semidynamic solution and describe a static data structure of size $O(n \log^4 n)$ with $O(\log^2 n + i)$ query time, where i is the number of reported colors. In [13], Gupta, Janardan, and Smid describe a static data structure for orthogonal colored range reporting in any dimension, which requires storage $O(n^{1+\epsilon})$, such that for any query box in $\mathbb{R}^d$, the i distinct colors of points contained in it are reported in $O(\log n + i)$ time.[2]

These results use the fact that if we decompose the problem into a small (constant or logarithmic) number of subproblems and solve each subproblem separately, then each color is discovered only a small number of times. This still keeps the query time close to $O(\text{polylog}(n) + i)$.

Orthogonal colored range counting. Orthogonal colored range *counting* turns out to be harder than orthogonal colored range reporting. In [16], Gupta, Janardan, and Smid describe solutions for colored range counting in $\mathbb{R}^1$ and $\mathbb{R}^2$. Similarly to the case of reporting, they reduce[3] the one-dimensional problem to standard orthogonal range searching in $\mathbb{R}^2$. The resulting solution has query time $O(\log n)$ and storage and preprocessing cost $O(n \log n)$. Its *dynamic* version also requires $O(n \log n)$ space and supports queries and updates in $O(\log^2 n)$ time. (The storage of the static and the dynamic data structures can be further reduced by a logarithmic factor, as noted in [4].) Using persistence, Gupta, Janardan, and Smid [16] extend their one-dimensional data structure into a static two-dimensional structure which supports queries that involve 3-*sided boxes*, that is, boxes of the form, say, $[a, b] \times (-\infty, c]$. Using a linear number of copies of this structure, they obtain a complete solution in $\mathbb{R}^2$, with query time $O(\log^2 n)$ and storage and preprocessing cost $O(n^2 \log^2 n)$.

Halfspace colored range searching. Efficient static data structures for halfspace colored range searching were described by Gupta, Janardan, and Smid [15]. Their solution in $\mathbb{R}^2$ and $\mathbb{R}^3$ for counting and reporting is based, through duality, on a straightforward reduction to an instance of the *ray-envelope intersection problem*. In this problem, we are given a set of upper envelopes of linear functions (the lines/planes dual to the input points), such that each envelope is computed for the lines/planes of

[2] Bounds of the form $O(f(n, \epsilon))$ hold for any $\epsilon > 0$; the constant of proportionality depends on ϵ and generally tends to ∞ as $\epsilon > 0$ approaches to 0.

[3] In a sense, their reduction is the forefather of the decomposition scheme that we develop for the higher-dimensional cases; see section 3.

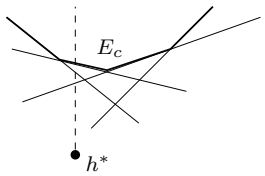


FIG. 1.2. *E_c is the upper envelope of the dual hyperplanes of the input points having color c. h^* is the point dual to the hyperplane h bounding the query halfspace below h. There is an input point below h having color c iff the upward vertical ray emanating from h^* intersects E_c (that is, h^* lies below E_c).*

some fixed color. The overall complexity of the envelopes is $O(n)$. The objective is to preprocess the envelopes into a data structure such that, given an upward vertical ray, we can report (resp., count) the envelopes that it intersects. See Figure 1.2. The solution in $\mathbb{R}^2$ uses $O(n \log n)$ space and answers reporting (resp., counting) queries in $O(\log^2 n + i)$ (resp., $O(n^{1/2})$) time. The solution in $\mathbb{R}^3$ is based on partition trees (see [23]) and uses $O(n \log^2 n)$ space and answers reporting (resp., counting) queries in $O(n^{1/2+\epsilon} + i)$ (resp., $O(n^{2/3+\epsilon})$) time. Using cutting trees instead [24, 25], reporting in $\mathbb{R}^3$ can be solved with $O(n^{2+\epsilon})$ storage, so that a query takes $O(\log^2 n + i)$ time. Note that the time and space bounds for the colored range counting in $\mathbb{R}^2$ and $\mathbb{R}^3$ are close to those achieved for the standard range counting problems in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively, [23, 24, 25].

This approach does not efficiently generalize to $\mathbb{R}^d$, for $d > 3$. Gupta, Janardan, and Smid [15] solve the reporting problem in $\mathbb{R}^d$, for $d > 3$, using a balanced binary tree CT built over the colors. Each node v in CT points to a halfspace range-emptiness data structure (described in [22]), built over the points, whose colors belong to the subtree of v. The query algorithm starts at the root of CT and continues to the children of a node v only if the query range contains at least one point, whose color belongs to the subtree of v. The resulting data structure requires $O(n^{\lfloor d/2 \rfloor} / \log^{\lfloor d/2 \rfloor - 1 - \epsilon} n)$ storage and can report the i distinct colors of points contained in a query halfspace in time $O(\log n + i \log^2 n)$. Other applications of this approach can be found in [14].

Batched colored intersection searching. In a *batched colored intersection searching problem* we are given a set of colored geometric objects and wish to compute all the pairs of colors (c_1, c_2) such that there are two intersecting input objects, one of color c_1 and one of color c_2. Often a *bipartite* version of the problem is considered, where we are given two sets of objects of two different classes and wish to report all pairs (c_1, c_2) such that an object of the first set having color c_1 intersects an object of the second set having color c_2. Efficient solutions of the colored batched intersection searching problem for line segments in $\mathbb{R}^1$, line segments in $\mathbb{R}^2$, axis-parallel boxes in $\mathbb{R}^d$, points and triangles in $\mathbb{R}^2$, and points and halfspaces in $\mathbb{R}^d$ are given in [6, 20]. For example, in the case of segments in $\mathbb{R}^1$ the problem can be solved in time $O(nc^{0.69})$ if $n \geq c^{1.69}$, and $O(n^{0.7}c^{1.21} + c^2)$ if $n \leq c^{1.69}$, using matrix multiplication techniques [20]. Variants of these techniques will also be used in this paper. See section 4.

1.2. Our contribution. We present a different approach to orthogonal colored range counting, based on a reduction from this problem to standard orthogonal range counting, which works in any dimension. We transform P into a higher-dimensional space, where each query box corresponds to a point, and for each color $c \in C$, the space $Q(c)^+$ of all (points representing) query boxes containing a point with color c is the union of positive orthants.

For each color c, we show how to decompose $Q(c)^+$ into pairwise disjoint boxes. As a consequence, a query box contains a point of color c iff the point corresponding to the query in the transformed space is contained in exactly one of the boxes in the decomposition of $Q(c)^+$. Our algorithm thus collects the decomposition boxes, over all colors c, and stores them in a data structure that supports efficient containment counting queries of the following form: Given a query point x, count the number of boxes that contain x.

A straightforward implementation of this technique yields a solution for the colored orthogonal range counting problem in $\mathbb{R}^d$, having query time $O(\log^{2d-1} n)$, storage complexity $O(n^d \log^{2d-1} n)$, and worst-case deterministic preprocessing time $O(n^d \log^{2d-1} n)$. A simple enhancement of the technique reduces all three performance parameters by a factor of $\log n$, thus matching the performance parameters of the algorithms of Gupta, Janardan, and Smid [12] for $d = 2$. If our queries are orthants, that is, boxes that are semiunbounded in each (say, negative) coordinate direction, then we can do better, obtaining a data structure with a query time of $O(\log^{d-1} n)$ and storage and preprocessing cost $O(n^{\lfloor d/2 \rfloor} \log^{d-1} n)$. For $d \geq 4$ and even, the same enhancement trick mentioned above reduces all three performance parameters by a $\log n$ factor.

In fact, the storage size and preprocessing time of our algorithm depends on the overall number of boxes in the decomposition of $Q(c)^+$, over all colors c. We show that for some sets of points the number of boxes in any such decomposition is $\Omega(n^d)$, but in practice we expect it to be much smaller. To support this statement, we show that, for random point sets (drawn independently and uniformly at random from $[0,1]^d$), the expected number of boxes in our decomposition is only $O(n \log^{d-1} n)$, which leads to algorithms with polylogarithmic query time and near-linear expected storage and preprocessing cost.

Time-space tradeoff. We also consider techniques for reducing the storage (and preprocessing) at the cost of increasing the query time. In $\mathbb{R}^2$, we use a technique of Gupta, Janardan, and Smid [12] to decompose a query $[a,b] \times [c,d]$ into two 3-sided queries $[a,b] \times (-\infty, d]$ and $[a,b] \times [c,\infty)$ on secondary structures stored at the nodes of a binary search tree over the points sorted by their y-coordinates. (This leads to significant gains, because, as we show, the structure for answering 3-sided queries requires only near-linear storage and preprocessing, whereas the structure for 4-sided queries may require near-quadratic storage.) We obtain the solution for each 3-sided query as a collection of pairwise disjoint canonical sets of colors. The difficulty is that a single color may appear in the solutions of the two 3-sided queries but we need to count it only once. To efficiently compute the union of the answers of the two queries, we use the principle of inclusion-exclusion to reduce the problem to computing the sizes of all pairwise intersections of the output canonical subsets. We use sparse matrix multiplication techniques [7, 20, 27] to precompute efficiently some of these intersection sizes and handle others on the fly when processing a query.

We obtain a solution that has query time $O(X \log^7 n)$ and $O\big(\big(\frac{n}{X}\big)^2 \log^6 n + n \log^4 n\big)$ storage, for any tradeoff parameter $1 \leq X \leq n$. The construction time of the data structure is (the $O^*(\cdot)$ notation hides polylogarithmic factors)

$$\begin{cases} O^*\left(\frac{n^{(\omega+1)/2}}{X^{(\omega-1)/2}}\right) = O\left(\frac{n^{1.688}}{X^{0.688}}\right) & \text{when } X \geq n^{\frac{\omega-1}{\omega+1}} \approx n^{0.408}, \\ O^*\left(\frac{n^{\frac{2-\alpha\beta+2\beta}{\beta+1}}}{X^{\frac{2-\alpha\beta}{\beta+1}}}\right) = O\left(\frac{n^{1.898}}{X^{1.203}}\right) & \text{when } n^{\frac{\alpha/2}{\alpha/2+1}} \approx n^{0.128} \leq X \leq n^{\frac{\omega-1}{\omega+1}} \approx n^{0.408}, \\ O^*(\frac{n^2}{X^2}) & \text{when } X \leq n^{\frac{\alpha/2}{\alpha/2+1}} \approx n^{0.128}. \end{cases}$$

Here ω is the smallest number such that two $t \times t$ matrices can be multiplied in time $O(t^\omega)$ (the best known upper bound on ω is roughly 2.376), $\alpha > 0.294$ is another parameter related to matrix multiplication, and $\beta = \frac{\omega-2}{1-\alpha}$; see [10, 11] and section 4. In particular, it follows from these bounds that, for $m \leq n$, we can answer m queries in overall time $O^*(nm^{\frac{\omega-1}{\omega+1}}) = O(nm^{0.408})$ (including preprocessing).

Interestingly, we also show a reduction of a version of sparse matrix multiplication to an off-line version of colored orthogonal range counting in $\mathbb{R}^2$. This reduction implies that if we can answer m queries on a set of n points, for $m^{\frac{1+\omega}{4}} \leq n$, in $o(nm^{\frac{\omega-1}{4}}) = o(n^{1.34})$ time, then we can obtain a better algorithm for sparse multiplication of rectangular zero-one matrices than the best known to date. Furthermore, if we can answer n such queries in $o(n^{\frac{2.376}{2}}) = o(n^{1.188})$ time, we improve the best known bound on ω (restricted to zero-one matrices). This suggests that our bounds, while not claimed to be near optimal, are significant, and that substantial improvements are likely to be quite difficult.

A simple bucketing technique allows us to trade time for space also in dimension $d > 2$. Specifically, for any threshold parameter $1 \leq X \leq n$, we obtain a data structure, having query time $O(X \log^d n + \log^{2d-1} n)$ and preprocessing and storage cost $O(\frac{n^d}{X^{d-1}} \log^{2d-1} n)$. We suggest two additional techniques to improve this tradeoff for $X = \Omega^*(n^{\frac{d-2}{d-1}})$.

Finally, we show that colored orthogonal range counting in dimension $d > 2$ is related to the following generalization of sparse matrix multiplication, which we believe to be of independent interest. One is given a 0-1 matrix A with N nonzero entries in sparse representation and a list O of M d-tuples of indices of rows of A. Let t be the number of columns in A. The goal is to compute for each tuple $(i_1, \ldots, i_d) \in O$ the sum $\sum_{j=1}^{t} \prod_{k=1}^{d} A_{i_k,j}$. We call this problem *$d$-dimensional output restricted sparse matrix multiplication* ($ORSMM_d$). (Note that the case $d = 2$ asks for computing t specified entries in AA^T.) We give a reduction from this problem to colored range counting in boxes in dimension d, showing that the off-line version of the latter problem is at least as hard as this generalized matrix multiplication problem. Finally, we describe reasonably efficient algorithms for $ORSMM_d$.

This paper is organized as follows.

In section 2 we describe the solution to colored orthogonal range counting in $\mathbb{R}^d$, which requires polylogarithmic query time and uses $\tilde{O}(n^d)$ space. The solution is based on a decomposition of the c-positive region $Q^+(c)$ (the union of the positive orthants dual to points with color c), for each of the colors c, into pairwise disjoint boxes.

In section 3 we describe the decomposition of the c-positive region into pairwise disjoint boxes and analyze its complexity and construction time. Next, we analyze, for random point sets (according to a natural model that we detail in section 3.3), the expected number of boxes in our decomposition. This leads to algorithms with polylogarithmic query time and $\tilde{O}(n)$ expected storage and preprocessing cost.

In section 4 we describe efficient methods for achieving time-space tradeoff. We also consider the time required to answer m queries over an input set of n colored points.

In section 5 we consider the relation between colored orthogonal range counting in $\mathbb{R}^d$ and sparse matrix multiplication, or its generalized version $ORSMM_d$, as defined above. In addition, we provide efficient algorithms for solving the generalized $ORSMM_d$.

We conclude this paper in section 6, with a brief discussion and a few open

problems.

2. Reducing to standard orthogonal range counting. We first solve the *semiunbounded colored range counting* problem, in which the query boxes are orthants of the form $\prod_{i=1}^{d}(-\infty, a_i]$. Then we show how to reduce the colored counting problem in general boxes to the semiunbounded case.

Let us represent each query orthant $\prod_{i=1}^{d}(-\infty, a_i]$ by its apex $(a_1, \ldots, a_d) \in \mathbb{R}^d$. Fix a color c, $1 \le c \le C$, and let P_c denote the subset of points of P with color c. For a point $p \in P_c$, denote by $Q_p^+ \subseteq \mathbb{R}^d$ the locus of all points that represent (closed) query orthants containing p. Clearly, if $p = (p_1, \ldots, p_d)$, then Q_p^+ is the positive orthant $\prod_{i=1}^{d}[p_i, \infty) \subseteq \mathbb{R}^d$. We refer to Q_p^+ as the *dual orthant* of p.

The *c-positive region* of color c, denoted $Q(c)^+$, is the region in $\mathbb{R}^d$ of all points representing queries that contain at least one point of P_c. Clearly, $Q(c)^+ = \bigcup_{p \in P_c} Q_p^+$. For any point set $A \subset \mathbb{R}^d$, define $U(A) := \bigcup_{p \in A} Q_p^+$. According to this definition, $Q(c)^+ = U(P_c)$. In section 3 we establish the following theorem.

THEOREM 2.1. *Let A be a set of n points in $\mathbb{R}^d$. We can decompose $U(A)$ into B pairwise disjoint boxes, where B is proportional to the complexity of $U(A)$. Furthermore, we can generate these boxes in $O((B+n)\log^{d-1} n)$ time. Our construction implies that $B = O(n^{\lfloor d/2 \rfloor})$, and we prove that this bound is tight in the worst case.*

In particular, when A is a set of n points in $\mathbb{R}$, $U(A)$ is a (positively oriented) closed orthant, which can be computed in $O(n)$ time.

We remark that Theorem 2.1 can be regarded as a refinement of the result of Boissonnat et al. [3] concerning the complexity of the union of n congruent axis-parallel cubes in $\mathbb{R}^d$ (think of the orthants as very large congruent cubes). As a matter of fact, our constructive proof of the theorem follows similar footsteps to those in the proof of the bound in [3]; see also [8]. In the rest of this paper, unless stated otherwise, decomposition of $U(A)$ means the disjoint decomposition asserted in Theorem 2.1.

We use Theorem 2.1 to solve the semiunbounded colored range counting problem as follows. For each color $1 \le c \le C$, we generate the boxes in the decomposition of $U(P_c) = Q(c)^+$. We build a standard orthogonal range searching data structure for counting the number of boxes containing a query point[4] $q \in \mathbb{R}^d$. By construction (using the fact that the boxes corresponding to any fixed color are pairwise disjoint), this number is equal to the number of distinct colors that appear in the original query orthant.

To implement the data structure, we use a d-dimensional segment tree with fractional cascading at its deepest level. Let B_c be the number of boxes in the decomposition of $Q(c)^+$. Then this structure has a query time $O(\log^{d-1} n)$ and requires $O(\sum_{1 \le c \le C} B_c \log^{d-1} n)$ space and preprocessing time. These improved bounds on the storage and preprocessing time (by a logarithmic factor, as compared with standard d-dimensional segment trees) are based on the fact, established in section 3, that all the decomposition boxes are unbounded in the positive x_1-direction. This allows us to use sorted lists instead of segment trees at the bottom level of the data structure, thereby saving a logarithmic factor in both storage and preprocessing time. Fractional cascading takes care of the corresponding saving in the query time.

Specifically, at the top level we have a segment tree over the x_d-coordinates of the boxes. The boxes assigned to a node v in this top-level tree are stored in a secondary segment tree, associated with v, defined over their x_{d-1}-coordinates. The structure

[4]The problem is called the "stabbing query problem" in [2].

continues this way where each node of a segment tree over the x_j-coordinates of the boxes stores the boxes assigned to it in a segment tree over their x_{j-1}-coordinates for every $3 \le j \le d$. At the bottom-level we have a segment tree over the x_2-coordinate of the boxes. Each node of such a segment tree stores the boxes assigned to it in a list sorted by the left endpoints of their x_1-projections. The sorted lists at the bottom-level of each segment tree are linked together in a fractional cascading data structure (see, e.g., [2]) so once we find the location of the query in the list associated with a node v we can search in the lists associated with the children of v in $O(1)$ time. (We also need a search tree over the list associated with the root of the tree to efficiently search in the first list.) Each node of a bottom-level list stores the number of boxes covering it, that is, the number of elements preceding it in the list. We obtain the following theorem.

THEOREM 2.2. *Let P be a set of n colored points in dimension $d \ge 2$, let B_c be the number of boxes in the decomposition of $Q(c)^+$, for each color $1 \le c \le C$, and let $B = \sum_{1 \le c \le C} B_c$. Then there exists a data structure supporting semiunbounded (orthant) colored range counting queries in $O(\log^{d-1} n)$ time, whose storage is $O(B \log^{d-1} n)$ and which can be constructed in $O((B+n)\log^{d-1} n)$ time.*

Using the bounds of Theorem 2.1, we immediately obtain the following theorem.

THEOREM 2.3. *There exists a data structure for colored semiunbounded (orthant) range counting queries on n colored points in dimension $d \ge 2$, which answers a query in $O(\log^{d-1} n)$ time, requires $O(n^{\lfloor d/2 \rfloor} \log^{d-1} n)$ space, and can be constructed in $O(n^{\lfloor d/2 \rfloor} \log^{d-1} n)$ time.*

General orthogonal range counting. The general colored orthogonal range counting problem, in which queries are arbitrary bounded axis-parallel boxes in $\mathbb{R}^d$, can be reduced to the semiunbounded case in $\mathbb{R}^{2d}$, as follows. We denote the x_i-coordinate of a point p by $x_i(p)$. Double all of the coordinates of each point $p = (x_1(p), \ldots, x_i(p), \ldots, x_d(p)) \in P$, to obtain the point $(x_1(p), -x_1(p), \ldots, x_i(p), -x_i(p), \ldots, x_d(p), -x_d(p))$ in $\mathbb{R}^{2d}$, which is given the same color as the original point p. Now, answering a colored range counting query $\prod_{i=1}^{d} [a_i, b_i]$ on the original point set is equivalent to answering the range counting query $\prod_{i=1}^{d} [a_i, \infty) \times [-b_i, \infty)$ on the transformed point set. Thus we obtain the following theorems.

THEOREM 2.4. *Let P be a set of n colored points in $\mathbb{R}^d$, let $\tilde{P} \subset \mathbb{R}^{2d}$ be the transformed point set of P as defined above, let B_c be the number of boxes in the decomposition of $U(\tilde{P}_c)$, for color $1 \le c \le C$, and let $B = \sum_{1 \le c \le C} B_c$. Then there exists a data structure supporting colored range counting queries in $O(\log^{2d-1} n)$ time, whose storage is $O(B \log^{2d-1} n)$ and which can be constructed in $O((B+n)\log^{2d-1} n)$ time.*

THEOREM 2.5. *There exists a data structure for colored orthogonal range counting queries on n colored points in $\mathbb{R}^d$, which answers a query in $O(\log^{2d-1} n)$ time, requires $O(n^d \log^{2d-1} n)$ space, and can be constructed in $O(n^d \log^{2d-1} n)$ time.*

We can generalize this approach to colored orthogonal range counting queries for boxes with bounded projections on k specific coordinates and semiunbounded projections on the remaining $d-k$ coordinates. In this case we can reduce the problem to a semiunbounded problem in $\mathbb{R}^{d+k}$, by duplicating the k "bounded" coordinates of each point in our input set. A query then takes $O(\log^{d+k-1} n)$ time, and the storage and preprocessing costs are both $O(n^{\lfloor (d+k)/2 \rfloor} \log^{d+k-1} n)$.

As a special case, consider the colored orthogonal range counting problem on n points in the plane, where the queries are 3-sided boxes of the form $[a, b] \times (-\infty, c]$. Here $d = 2$, $k = 1$, and we obtain an algorithm with $O(\log^2 n)$ query time and space

and preprocessing cost $O(n \log^2 n)$. These performance parameters are the same as those obtained by Gupta, Janardan, and Smid [12], using a different approach based on persistence (which, as already noted, does not seem to extend to higher dimensions).

We can then use the same paradigm as in [12] to extend this solution to the general case of bounded box queries. That is, let $(p_1, \ldots, p_n)$ be the sorted sequence of the points of P in their increasing y-order. For each $i = 1, \ldots, n$, construct the above data structure (for 3-sided queries) for the set $P_i^+ = \{p_j \mid j \geq i\}$. Now, given a query box $[a, b] \times [c, d]$, we find, by binary search, the index i satisfying $y(p_{i-1}) < c \leq y(p_i)$ and search with the 3-sided box $[a, b] \times (-\infty, d]$ in the structure of P_i^+. This yields a slightly improved algorithm, in which a query takes $O(\log^2 n)$ time, and the storage and the preprocessing cost are both $O(n^2 \log^2 n)$ (saving a logarithmic factor over the bounds in Theorem 2.5). The same enhancement can be applied in arbitrary dimension $d \geq 2$, leading to the following result.

THEOREM 2.6. *There exists a data structure for colored orthogonal range counting queries on n colored points in dimension $d \geq 2$, which answers a query in $O(\log^{2d-2} n)$ time, requires $O(n^d \log^{2d-2} n)$ space, and can be constructed in $O(n^d \log^{2d-2} n)$ time.*

The same enhancement can save a logarithmic factor for any d and k for which $d + k$ is even and ≥ 4. In particular, for even $d \geq 4$ and $k = 0$, we can improve all three performance parameters in Theorem 2.3 by a logarithmic factor.

In section 4 we present a more sophisticated approach that reduces colored box range counting to colored range counting with boxes unbounded in one direction, so as to obtain a much more significant reduction in storage and preprocessing (at the cost of increasing query time).

3. Decomposing the union of orthants into disjoint boxes. Let A be a set of n points in $\mathbb{R}^d$ in general position, meaning that no two points have the same x_i-coordinate,[5] for any $i = 1, \ldots, d$. In this section we prove Theorem 2.1. We start proving the first part of Theorem 2.1 and show how to decompose $U(A)$ into $B = O(n^{\lfloor d/2 \rfloor})$ pairwise disjoint boxes, where B is proportional to the complexity of $U(A)$. We then show, in section 3.2, that this bound is tight in the worst case.

An open *maximal empty orthant* O (with respect to A) is a region of the form $\prod_{i=1}^d (-\infty, a_i)$, where $a_i \in \mathbb{R} \cup \{+\infty\}$ for each i, which does not contain any point of A, and is maximal with this property under inclusion. That is, any open orthant O' that strictly contains O must also contain a point of A. It follows that each facet of O must contain a distinct point of A in its *relative interior*. Let s_i be the point in the relative interior of the facet of O orthogonal to the x_i-axis, for $1 \leq i \leq d$. Thus $O = \prod_{i=1}^d (-\infty, x_i(s_i))$ (see Figure 3.1). We also include, under this definition, orthants O that are unbounded in the positive direction of some coordinate axes; for each such direction, we replace the corresponding value $x_i(s_i)$ by $+\infty$. As an extreme example, the halfspace $x_1 < \min_{s \in A} x_1(s)$ is such a degenerate maximal empty orthant. Alternatively, for each coordinate x_i in which O is unbounded in both directions, we can define s_i to be the point at infinity whose x_i-coordinate is $+\infty$ and all other coordinates are $-\infty$. We follow the latter convention and say that O is defined by the tuple of points $\langle s_1, \ldots, s_d \rangle$.

[5] We can avoid this assumption by symbolically perturbing the x_i-coordinates of all points for $1 \leq i \leq d$ and constructing the decomposition with respect to the perturbed point set. In terms of the old coordinates, we obtain a legal decomposition of $U(A)$ into pairwise disjoint boxes. The cardinality of this decomposition, however, might be larger than the boundary complexity of $U(A)$, and some of its boxes are possibly empty (redundant). Alternatively, we can apply standard but tedious care of degenerate configurations which we omit here; see [2] for an example of such a treatment.

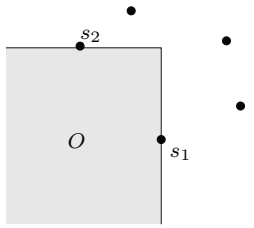


FIG. 3.1. *A maximal empty orthant.*

Our decomposition of $U(A)$ is constructed so that there is a bijection between its boxes and the maximal empty orthants defined by tuples $\langle s_1, \ldots, s_d \rangle$ such that $x_1(s_1) < \infty$. Specifically, let O be such a maximal empty orthant defined by $\langle s_1, \ldots, s_d \rangle$. The box in our decomposition corresponding to O is

$$B(O) = [x_1(s_1), \infty) \times \prod_{i=2}^{d} \Big[\max_{j<i} \{x_i(s_j)\}, x_i(s_i) \Big).$$

Note that each interval in the product is nonempty, which follows from the above general position assumption. See Figure 3.2 for an illustration.

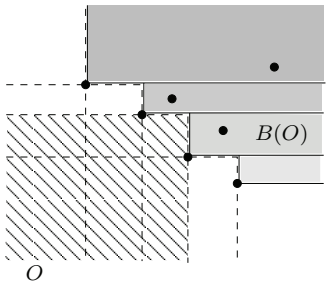


FIG. 3.2. *A maximal empty orthant O and its corresponding box $B(O)$ in the decomposition of $U(A)$. Altogether there are four maximal empty orthants bounded in the $(+x_1)$-direction and four respective boxes in the decomposition of $U(A)$.*

LEMMA 3.1. *Let A be a finite point set in $\mathbb{R}^d$, let $\mathcal{O}$ be the set of all maximal empty orthants with respect to A, which are defined by tuples $\langle s_1, \ldots, s_d \rangle$ satisfying $x_1(s_1) < \infty$, and let $\mathcal{B}$ be the collection of boxes of the form $B(O)$, for $O \in \mathcal{O}$. Then the boxes of $\mathcal{B}$ are pairwise disjoint, and their union is $U(A)$.*

Proof. We establish the lemma by induction on the dimension of the space containing A.

For the induction basis, consider a one-dimensional set of points $A = \{p_1, \ldots, p_n\} \subset \mathbb{R}$, where $p_1 < p_2 < \cdots < p_n$. The orthant $(-\infty, p_1)$, defined by $\langle p_1 \rangle$, is the only maximal empty orthant with respect to A (which clearly satisfies $x_1(p_1) < \infty$). The corresponding box $B = [x_1(s_1), \infty) = [p_1, \infty)$ is indeed equal to $U(A)$, as required.

Assume now that the lemma is true for any finite set of points in $\mathbb{R}^{d-1}$. Sweep $U(A)$ with a hyperplane h orthogonal to the x_d-axis. Let $(p_1, \ldots, p_n)$ be the sequence of the points of A sorted in increasing order of their x_d-coordinates. For $1 \le j \le n$, put $q_j := x_d(p_j)$, let h_j be the hyperplane $x_d = q_j$, and let H_j be the half-closed slab bounded between h_j and h_{j+1} (containing h_j but not h_{j+1}); let H_n be the infinite slab "beyond" h_n, i.e., the slab $x_d \ge x_d(p_n)$.

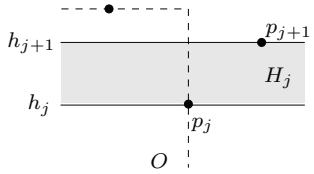


FIG. 3.3. *The proof of Lemma* 3.1.

By definition, the boxes in our decomposition that intersect H_j correspond to maximal empty orthants defined by tuples $\langle s_1, \ldots, s_d \rangle$, such that $q_{j+1} \leq x_d(s_d)$ (or $x_d(s_d) = \infty$ for $j = n$), $\max_{j<d}\{x_d(s_j)\} \leq q_j$, and $x_1(s_1) < \infty$. See Figure 3.3 for an illustration. Denote by $\mathcal{O}_j$ the set of these orthants, and denote by $\mathcal{B}(\mathcal{O}_j)$ the corresponding set of boxes. We claim that the intersections of the boxes in $\mathcal{B}(\mathcal{O}_j)$ with H_j form a pairwise disjoint decomposition of $H_j \cap U(A)$.

Indeed, let $D_j = \{p_i \mid i \leq j\}$, and let D'_j be the orthogonal projection of D_j on the hyperplane h_j. Let $\mathcal{O}'_j$ be the set of maximal empty orthants with respect to D'_j in the $(d-1)$-dimensional space h_j, defined by tuples $\langle s_1, \ldots, s_{d-1} \rangle$ satisfying $x_1(s_1) < \infty$. Let $\mathcal{B}(\mathcal{O}'_j)$ be the corresponding set of boxes within h_j. The following facts are easy to verify.

1. The intersection of $U(A)$ with h_j is equal to the intersection of $U(D_j)$ with h_j, which in turn is equal to $U(D'_j)$.
2. Each maximal empty orthant $O' \in \mathcal{O}'_j$ is the intersection of an orthant $O \in \mathcal{O}_j$ with h_j. Conversely, for each orthant $O \in \mathcal{O}_j$, the $(d-1)$-orthant $O' = O \cap h_j$ is a maximal empty orthant with respect to D'_j.
3. Let $O \in \mathcal{O}_j$, and let $B(O)$ be the corresponding box. The intersection of $B(O)$ with h_j is equal to $B(O')$, where O' is the $(d-1)$-orthant $O \cap h_j$.

See Figure 3.3 for an illustration. By the induction hypothesis, the boxes of $\mathcal{B}(\mathcal{O}'_j)$ form a pairwise disjoint decomposition of $U(D'_j)$. Furthermore, by 2. and 3., each box $B(O)$ in $\mathcal{B}(\mathcal{O}_j)$ corresponds to a box $B(O')$ in $\mathcal{B}(\mathcal{O}'_j)$, where $O' = O \cap h_j$, so that $B(O') \times [q_j, q_{j+1}) \subseteq B(O)$, and vice versa. Thus the boxes in $\mathcal{B}(\mathcal{O}_j)$ are pairwise disjoint within H_j, and by 1. their union is equal to $U(A) \cap H_j$. Repeating this argument for each slab H_j completes the proof. ∎

The number of boxes. Let $A \subset \mathbb{R}^d$ be an arbitrary set of points. As we have seen, the number of cells in the decomposition of $U(A)$ is bounded by the number of maximal empty orthants with respect to A. Hence, it suffices to bound the latter quantity.

Here is a straightforward constructive derivation of the bound $O(n^{\lfloor d/2 \rfloor})$ for this latter quantity, which resembles the analysis of Boissonnat et al. [3]. We have already seen, for dimension $d = 1$, that the decomposition of $U(A)$ consists of a single orthant. Now assume that $d \geq 2$. Note that the number of maximal empty orthants that are halfspaces, and contain only one point of A on their boundary, is $O(n)$. So it remains to count maximal empty orthants with at least two points of A on their boundary. Given any empty orthant O, with at least two points of A on its boundary (possibly on lower-dimensional faces of the boundary), a *shift* of O is the operation of shrinking O by translating a facet of O in the negative direction of the orthogonal coordinate axis. By the general position assumption, when a shift starts, exactly one point leaves the boundary of O. The shift is *legal* if such a point lies in the relative interior of a facet. A legal shift terminates as soon as one of the points of A on ∂O reaches the relative boundary of the face it is currently on, so that it now lies on a lower-dimensional face.

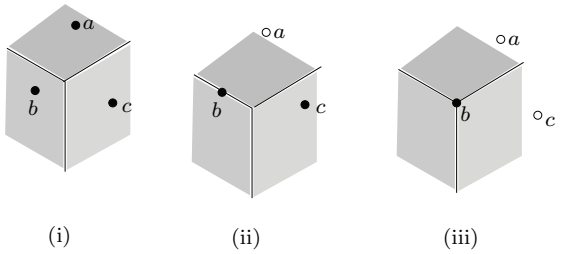


FIG. 3.4. *Three legal shifts that turn a maximal empty orthant into the one that has point b as an apex.*

Note that the orthant reached at the end of a legal shift is contained in the original orthant and is thus also empty.

We start with a maximal empty orthant O, with at least two points of A on its boundary, and apply to it any sequence of legal shifts, until we reach an orthant O' to which no further legal shifts can be applied. See Figure 3.4. Upon termination, no point of A lies in the relative interior of any facet of O'. Hence each point of A on $\partial O'$ determines at least two of the coordinate values that define facets of O', and, since we assume general position, no such coordinate value is determined by more than one point. Hence, the number of points on $\partial O'$ is at most $\lfloor d/2 \rfloor$, which implies that the number of such empty orthants O', which can be obtained from some maximal empty orthant in the manner described above, is $O(n^{\lfloor d/2 \rfloor})$. Moreover, knowing the sequence of directions of the legal shifts that have transformed an original maximal empty orthant O into O', the orthant O can be uniquely reconstructed from O', using a sequence of reverse legal shifts, each stopping when the facet that moves outwards hits a new point. Clearly, the total number of such sequences of shifts is at most $d!$. The bound of $O(n^{\lfloor d/2 \rfloor})$ on the number of maximal empty orthants with respect to A, and therefore on the number of boxes in our decomposition, follows.

We claim that each maximal empty orthant corresponds to a distinct face on the boundary of $U(A)$. This implies that the number of boxes B in our decomposition is at most the number of distinct faces on the boundary of $U(A)$. (Note that not every maximal empty orthant induces a decomposition cell $B(O)$, because O may be unbounded in the positive direction of the x_1-axis, having $x_1(s_1) = \infty$.) On the other hand, it is clear that the number of boxes cannot be smaller by more than a constant factor than the complexity of $U(A)$. So it follows that number of boxes B is proportional to the complexity of $U(A)$ (and to the number of maximal empty orthants) and thereby asymptotically optimal.

To establish the claim, let O be a maximal empty orthant, defined by the d-tuple $\langle s_1, \ldots, s_d \rangle$. Let J denote the (nonempty) set of all indices $1 \leq i \leq d$, such that $x_i(s_i) < \infty$. For each $i \in J$, let F_i denote the (closed) facet of the (positively oriented) dual orthant, $Q^+_{s_i}$, of s_i, that is orthogonal to the x_i-axis. Let F be the set of all points o, such that for $1 \leq i \leq d$, $x_i(o) = x_i(s_i)$ if $i \in J$, and $x_i(o) \geq \max_{j \in J}\{x_i(s_j)\}$, otherwise. We claim that $F = \bigcap_{i \in J} F_i$, and therefore $F \subseteq U(A)$. It is immediate that $\bigcap_{i \in J} F_i \subseteq F$. To show that $F \subseteq \bigcap_{i \in J} F_i$, note that for each $o \in F$ and $i \in J$, the negatively oriented orthant with apex o contains s_i on its facet orthogonal to the x_i-axis, and therefore o is contained in F_i. We next show that F does not intersect the interior of $U(A)$. To see that, note that any negatively oriented orthant O', whose apex belongs to F, does not contain any point of A in its interior. Let k be the number

of indices $1 \leq i \leq d$ such that $x_i(s_i) = \infty$. Then F is a k-dimensional (closed) face, on the boundary of $U(A)$, having nonzero (and unbounded) extent in exactly the k coordinates just defined. Moreover, as is easy to check, O is uniquely defined by F.

3.1. Output-sensitive construction. In order to construct the desired decomposition of $U(A)$, it suffices to enumerate all the maximal empty orthants with respect to A, each with the d-tuple defining it. As mentioned previously, when $d = 1$, $U(A)$ is an orthant which we can compute in $O(n)$ time. Hence, we can assume $d \geq 2$. We have seen that each such maximal empty orthant can be defined (up to a constant number of possibilities) by a t-tuple of points of A, such that $t \leq \lfloor d/2 \rfloor$. The constructive proof of the bound on the number of boxes (or, rather, of maximal open orthants) can be trivially converted into an algorithm that generates $O(n^{\lfloor d/2 \rfloor})$ candidate empty orthants, prunes away those that are not empty (using orthogonal emptiness range queries on the set A), and extends each of them, by some sequence (out of $O(1)$ possible ones) of reverse legal shifts, into a maximal empty orthant.

The running time of this straightforward algorithm is close to $O(n^{\lfloor d/2 \rfloor})$, and is always $\Omega(n^{\lfloor d/2 \rfloor})$, which might be much larger than the actual complexity of the decomposition of $U(A)$. In this section, we present an alternative *output-sensitive* algorithm for constructing all maximal empty orthants with respect to A. This immediately leads to an *output-sensitive* construction of our decomposition of $U(A)$.

We can enumerate all maximal empty orthants with respect to A by implementing the sweep that was used to establish Lemma 3.1. Let $p_1, \ldots, p_n$ be the points in the increasing order of their x_d-coordinates. We sweep $\mathbb{R}^d$ with a hyperplane π orthogonal to the x_d-direction. As above, let $D_j = \{p_i \mid i \leq j\}$, let p'_j be the projection of p_j on π, and let D'_j be the projection of D_j onto π, i.e., $D'_j = \{p'_i \mid i \leq j\}$.

After π passes through p_j, we update the maximal empty ($(d-1)$-dimensional) orthants with respect to D'_j on π. Specifically, we find the set Q of all maximal empty orthants (on π) with respect to D'_{j-1} which are not maximal empty orthants with respect to D'_j (because the projection p'_j of p_j, or rather, p_j itself at this moment, lies in their relative interior). Each such orthant, together with p_j, defines a maximal empty orthant with respect to A, which we output. We delete all orthants in Q and generate a new set N of maximal empty orthants with respect to D'_j which were not maximal empty $(d-1)$-orthants with respect to D'_{j-1} (those have a $(d-2)$-facet containing p_j in its relative interior). See Figure 3.5.

To efficiently identify the set Q when processing p_j, we maintain the maximal empty $(d-1)$-orthants on π in a dynamic $(d-1)$-dimensional range tree, where each orthant is represented by its apex (a point on π). Using this structure, we can find all k orthants containing p'_j in $O(\log^{d-1} n + k)$ time. We can also delete from, or insert into, the dynamic data structure an orthant, in $O(\log^{d-1} n)$ time (see [26] for more details).

To efficiently identify the set N, we make the following observation. Let O be an orthant in N, containing p_j on its facet which is orthogonal to the x_i-axis, for some $i < d$. Ignoring p_j, and shifting the facet containing p_j away from O, we either hit a point of D'_{j-1} or have this facet of O reach infinity. In both cases, we end up with an orthant O' of Q. Hence, each orthant in N can be obtained by taking an orthant O' of Q and by replacing a facet of O' by a parallel facet through p_j. (Not all orthants obtained in this manner are valid: One also needs to ensure that each facet of the shrunk orthant, other than the one through p_j, still contains a point of D'_j in its relative interior.)

It is easily verified that the total number of updates to the dynamic range tree

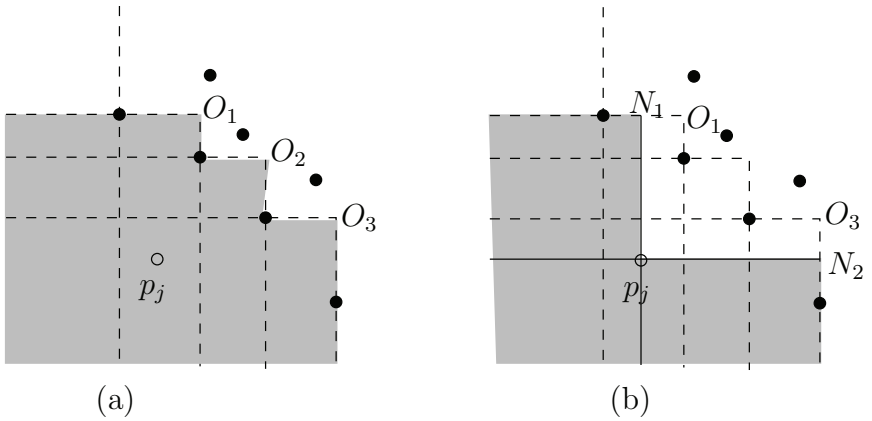


FIG. 3.5. *The sweep procedure in* $\mathbb{R}^3$*:* (a) *The structure on the sweep plane* π *just before reaching* p_j*; the set* Q *of* $(d-1)$*-orthants that are eliminated by* p_j *is shaded.* (b) *The structure on* π *immediately after reaching* p_j*; the set* N *of* $(d-1)$*-orthants that are generated by* p_j *is shaded. In the example,* $|Q| = 3$ *and* $|N| = 2$*. The new orthant* N_1 *is obtained by shrinking* O_1 *to the left, and* N_2 *is obtained by shrinking* O_3 *downwards.*

is bounded by the number H of maximal empty orthants with respect to A. Indeed, this is clear for orthants in Q: each such orthant O corresponds to a unique maximal empty d-orthant that is "completed" when the sweep hyperplane reaches its present position. Each orthant O in N will either be completed into a maximal empty d-orthant when it hits a new point at some future position of the sweep hyperplane, or else survive until the sweep is completed, and then become a degenerate d-orthant, unbounded in both directions of the x_d-axis. Since we argued before that the overall number H of maximal empty orthants with respect to A is proportional to the number B of boxes in our decomposition of $U(A)$, it follows that the decomposition of $U(A)$ can be constructed in $O((B+n)\log^{d-1} n)$ time.

3.2. Lower bounds. We now show that the upper bound in Theorem 2.1 is tight in the worst case. Specifically, we construct, for every dimension $d \geq 2$ and n, a set of n points A in $\mathbb{R}^d$ such that there are $\Omega(n^{\lfloor d/2 \rfloor})$ maximal empty orthants with respect to A. Since each of these orthants corresponds to a distinct face on the boundary of $U(A)$, we obtain that the number of boxes in any decomposition of $U(A)$ is $\Omega(n^{\lfloor d/2 \rfloor})$.

In fact, we prove a more general lower bound on the number of maximal empty boxes with respect to a finite set of points $A \subseteq \mathbb{R}^d$. A *maximal empty box* with respect to a point set A is an open axis-parallel box, which does not contain any point of A in its interior, and is maximal with this property under inclusion (so, as in the case of orthants, each facet of the box contains a point of P in its relative interior). We allow a maximal empty box to be unbounded in certain directions, so every maximal empty orthant with respect to P is also a maximal empty box.

Worst-case tightness of the upper bound in Theorem 2.1 is an easy consequence of the following two lemmas.

LEMMA 3.2. *There exists a set of points* $A = \{p_i^j \mid 1 \leq i \leq n, 1 \leq j \leq d\}$ *in general position in dimension* $d \geq 2$ *such that, for any choice of indices* $1 \leq i_d \leq i_{d-1} \leq \cdots \leq i_1 \leq n$*, there is an (open) empty box* R *whose boundary contains only the points* $p_{i_j}^j$*, for* $1 \leq j \leq d$*.*

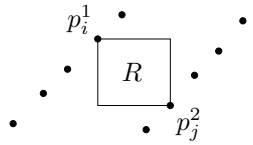


FIG. 3.6. *Illustrating Lemma* 3.2 *in dimension* $d = 2$. *We have* $p_i^1 = (i, i)$ *for* $1 \le i \le n$, *and* $p_j^2 = (n + j, j)$ *for* $1 \le j \le n$. *For any choice of* $1 \le j \le i \le n$, *the rectangle* $(i, j + n) \times (j, i)$ *contains only* p_i^1 *and* p_j^2 *on its boundary.*

Proof. Let $\vec{1} \in \mathbb{R}^d$ be the vector with all components equal to 1. Let $\vec{1}_{<j} \in \mathbb{R}^d$ be the vector whose ℓth component is 1, for $\ell < j$, and 0 for $\ell \ge j$. Let $p_i^j = n\vec{1}_{<j} + i\vec{1}$, for $1 \le i \le n$ and $1 \le j \le d$. We claim that the box $R = (i_1, i_2 + n) \times (i_2, i_3 + n) \times \cdots \times (i_{d-2}, i_{d-1} + n) \times (i_{d-1}, i_d + n) \times (i_d, i_1)$ contains only the points $p_{i_j}^j$. See Figure 3.6 for an illustration in dimension $d = 2$.

To prove the claim, consider a fixed point $p_{i_j}^j$. The first $j - 1$ coordinates of this point are equal to $i_j + n$, and the rest are equal to i_j. It easily follows that this point is on the boundary of R. On the other hand, let p_a^j be a point such that $a \ne i_j$. We argue that p_a^j does not belong to the closure of R.

Case 1a. If $a < i_j$ and $j \ne d$, then the projection of R on the jth coordinate is the interval $(i_j, i_{j+1} + n)$, but the jth coordinate of p_a^j is a, and since $a < i_j$, p_a^j does not belong to the closure of R.

Case 1b. If $a < i_j$ and $j = d$, then the projection of R on the dth coordinate is the interval (i_d, i_1), but the dth coordinate of p_a^j is a, and since $a < i_d$, p_a^j does not belong to the closure of R.

Case 2a. If $a > i_j$ and $j \ne 1$, then the projection of R on the $(j-1)$th coordinate is the interval $(i_{j-1}, i_j + n)$, but the $(j - 1)$th coordinate of p_a^j is $a + n$, since $a > i_j$, p_a^j does not belong to the closure of R.

Case 2b. If $a > i_j$ and $j = 1$, then the projection of R on the dth coordinate is the interval (i_d, i_1), but the dth coordinate of p_a^j is a, since $a > i_1$, p_a^j does not belong to the closure of R.

By symbolically perturbing the points of A, we can ensure that A is a set of points in general position, and the lemma still holds for the perturbed set. □

LEMMA 3.3. *For any dimension* d, *there exists a set* A *of* n *points in* $\mathbb{R}^d$ *in general position, such that there is* $\Omega(n^d)$ *maximal empty boxes with respect to* A.

Proof. If $d = 1$, we can choose A to be any set of points in $\mathbb{R}$ in general position. We thus assume that $d > 1$.

Let A be the point set of Lemma 3.2. Then, for any of the $\Omega(n^d)$ possible choices of indices $1 \le i_d \le i_{d-1} \le \cdots \le i_1 \le n$, there is an empty box R whose boundary contains p_{i_j} for all $1 \le j \le d$. Let $\mathcal{R}$ be this set of empty boxes. For each box $R \in \mathcal{R}$, let $B(R)$ be any maximal empty box containing R. (By the definition of a maximal empty box, $B(R)$ always exists.) Clearly, the boundary of $B(R)$ also contains p_{i_j} for all $1 \le j \le d$. Since A is a set of points in general position, $B(R)$ contains at most $2d$ points on its boundary. It follows that $B(R)$ may be equal to $B(R')$ for at most a constant number of boxes $R' \in \mathcal{R}$. So we obtain $\Omega(n^d)$ distinct maximal empty boxes in this way. □

To see the tightness of the upper bound in Theorem 2.1 for an even $d \ge 2$, take a point set B of n points in $\mathbb{R}^{d/2}$ such that there are $\Omega(n^{d/2})$ maximal empty boxes with respect to B. Such a set exists by Lemma 3.3. Let A be the point set in dimension d obtained from B by applying the transformation of Theorem 2.4. It is

easy to verify that if $\prod_{i=1}^{d}(a_i, b_i)$ is a maximal empty box with respect to B, then $\prod_{i=1}^{d}(-\infty, a_i) \times (-\infty, -b_i)$ is a maximal empty orthant with respect to A and vice versa. So it follows that there are $\Omega(n^{d/2})$ maximal empty orthants with respect to A.

If we embed A in $\mathbb{R}^{d+1}$ by setting coordinate x_{d+1} of all points to 0 (and then symbolically perturb it), we establish the tightness of the bound in Theorem 2.1 also for odd $d \geq 3$.

This, at long last, completes the proof of Theorem 2.1. □

Remark. The storage required by the data-structure of Theorem 2.3 depends on the number of maximal empty orthants with respect to each of the sets of points P_c of color c, for $1 \leq c \leq C$. Since this quantity can be as large as $\Omega(n^{\lfloor d/2 \rfloor})$, we have a lower bound of $\Omega(n^{\lfloor d/2 \rfloor})$ for the storage required by the data structure of Theorem 2.3 in a worst-case scenario.

The storage required by the data-structure of Theorem 2.5 depends on the number of maximal empty orthants with respect to each of the sets $\tilde{P}_c$, for $1 \leq c \leq C$, obtained from P_c by doubling each coordinate in the manner described above. As just noted, the number of maximal empty orthants with respect to each transformed set $\tilde{P}_c$ is equal to the number of maximal empty boxes with respect to the initial set P_c. By Lemma 3.3, a point set of cardinality n in dimension $d \geq 1$ can define $\Omega(n^d)$ maximal empty boxes in the worst-case. So we obtain a lower bound of $\Omega(n^d)$ on the worst-case storage required by the data-structure of Theorem 2.5. Similarly, we can prove a lower bound of $\Omega(n^d)$ for the worst-case storage required by the data-structure of Theorem 2.6.

3.3. Random sets of points. The decomposition-based data structures of Theorems 2.5 and 2.3 are especially efficient, when the number of maximal empty boxes with respect to each of the sets $P_c \subset \mathbb{R}^d$ is small for all $1 \leq c \leq C$. In this subsection we prove that the expected number of maximal empty boxes for a random point-set with n points in $\mathbb{R}^d$ is only $O(n \log^{d-1} n)$. It would be interesting to explore the connection between this result and the known bound of $O(\log^{d-1} n)$ on the expected number of *maximal* points in a set of n random points in $\mathbb{R}^d$; see [1, 17].

We assume that the set P is constructed so that each point is chosen independently and uniformly at random from the uniform distribution on $[0, 1]^d$. Thus, with probability 1, the points of P are in general position, and, for each i, the x_i-coordinates of the sampled points form a random permutation, which are independent of each other. We define a *t-box* to be an axis-parallel box B, that may be unbounded in certain directions, containing t points on its boundary, such that each one of the finite facets of B contains a point of P (possibly on its relative boundary).[6] Let $B_{t,k}(P)$ (resp., $B_{t,\leq k}(P)$), for $1 \leq t \leq 2d$, denote the set of t-boxes containing exactly (resp., at most) k points of P in their interior, and put $N_{t,k}(P) = |B_{t,k}(P)|$ (resp., $N_{t,\leq k}(P) = |B_{t,\leq k}(P)|$). Note that we may have degenerate boxes with identical opposite facets. However, the number of such boxes is at most $O(n)$, due to the general position assumption. Finally, we let $N^{(d)}_{t,k}(n)$ (resp., $N^{(d)}_{t,\leq k}(n)$) denote the expected value of $|B_{t,k}(P)|$ (resp., $|B_{t,\leq k}(P)|$), over the random choice of a set P of n points in $\mathbb{R}^d$. In order to obtain the asserted bound on the expected number of maximal empty boxes with respect to P, it suffices to show that $N^{(d)}_{t,0}(n) = O(n \log^{d-1} n)$ for all $1 \leq t \leq 2d$.

[6] With some abuse of notation, we allow degenerate 1-boxes contained in orthogonal planes. Clearly, there are $O(n)$ such boxes.

We prove the bound by induction on d. The case $d = 1$ is trivial: We clearly have $N^{(1)}_{1,0}(n)$ is 2, since any empty nondegenerate 1-box in $\mathbb{R}^1$ is an empty ray emanating from a point of P. Similarly, $N^{(1)}_{2,0}(n) = n - 1$, since any empty 2-box is a segment bounded by a pair of consecutive points of P.

Assume now that $d > 1$, and that the bound holds for $d - 1$ (and for all $t \leq 2(d-1)$). Assume also (without really changing the model) that we draw each point $p \in P$ by first drawing a point $p' \in [0,1]^{d-1}$, thereby fixing the projection of p on the hyperplane $x_d = 0$ to be p', and then drawing x_d uniformly from $[0,1]$ to be the x_d-coordinate of p.

Let $P' = \{p' \mid p \in P\}$ be the set of projections of all points in P on the hyperplane $x_d = 0$. Let $\hat{B}_{t,0}(P)$ denote the set of empty t-boxes of P whose both facets orthogonal to the x_d-axis contain a point of P on their relative boundary. Set $\hat{N}_{t,0}(P) = |\hat{B}_{t,0}(P)|$ and $\hat{N}_{t,0}(n) = \max_{|P|=n} \hat{N}_{t,0}(P)$. Every t-box B in $\hat{B}_{t,0}(P)$ corresponds to the t-box B' of P', obtained by projecting B onto $x_d = 0$; note, however, that B' is not necessarily empty.

Fix P', and consider the random drawings of the x_d-coordinates of the points of P. What is the probability that a t-box B' in $B_{t,k}(P')$ corresponds to an empty t-box B in $\hat{B}^d_{t,0}(P)$? For this to happen, it is necessary and sufficient that the t boundary points of B' be consecutive in the permutation defined by the $t + k$ boundary and interior points of B' along the x_d-axis (see Figure 3.7). This happens with probability $\frac{t!(k+1)!}{(k+t)!} = \frac{t!}{(k+2)(k+3)\cdots(k+t)}$. So we obtain the following inequality:

$$\mathbb{E}(\hat{N}_{t,0}(P)) \leq \sum_{k=0}^{n-t} \frac{t!}{\prod_{i=2}^{t}(k+i)} |B_{t,k}(P')|.$$

Rearranging this sum, we get

$$\mathbb{E}(\hat{N}_{t,0}(P)) \leq \sum_{k=0}^{n-t-1} \frac{t!(t-1)}{\prod_{i=2}^{t+1}(k+i)} |B_{t,\leq k}(P')| + \frac{t!}{\prod_{i=2}^{t}(n-t+i)} |B_{t,\leq n-t}(P')|.$$

This holds for every choice of P', so if we average over the choices of P', we obtain the following recurrence:

$$\hat{N}^{(d)}_{t,0}(n) \leq \sum_{k=0}^{n-t-1} \frac{t!(t-1)}{\prod_{i=2}^{t+1}(k+i)} N^{(d-1)}_{t,\leq k}(n) + \frac{t!}{\prod_{i=2}^{t}(n-t+i)} N^{(d-1)}_{t,\leq n-t}(n). \tag{3.1}$$

To estimate $N^{(d-1)}_{t,\leq k}(n)$, for $k \geq 1$, we note that if we sample a random subset of P of size n/k, we obtain a random set of n/k points drawn independently from the same uniform distribution as P. Hence, we can apply the Clarkson–Shor probabilistic technique [9] to conclude that

$$N^{(d-1)}_{t,\leq k}(n) = O(k^t N^{(d-1)}_{t,0}(n/k)). \tag{3.2}$$

By the induction hypothesis we have $N^{(d-1)}_{t,0}(n) = O(n \log^{d-2} n)$, for any $1 \leq t \leq 2d$, and for any n. Hence, by (3.2), $N^{(d-1)}_{t,\leq k}(n) = O(nk^{t-1} \log^{d-2} n)$, for any $1 \leq t \leq 2d$. Plugging this into inequality (3.1), we obtain the bound $\hat{N}^{(d)}_{t,0}(n) = O(n \log^{d-1} n)$, for $1 \leq t \leq 2d$.

We next consider empty t-boxes, for which neither of their facets orthogonal to the x_d-axis contains points of P in their relative boundaries. We consider only boxes that are not strips bounded by one or two hyperplanes orthogonal to the x_d-direction (there are $O(n)$ such strips). Any such box B can be charged to a box in $\hat{B}_{t',0}(P)$, for some $t-2 \le t' \le t$, by applying two legal shifts to B (as defined at the beginning of this section), the first of which shifts the top facet down, and the second shifts the bottom facet up. With some care, this also applies to boxes that are unbounded in the x_d-direction. The resulting box $\hat{B}$ is uniquely charged in this manner. Similarly, an empty t-box B, such that one of its facets orthogonal to the x_d-axis does not contain a point in the relative boundary, can be charged to a box $\hat{B}$ in $\hat{B}_{t',0}(P)$, for some $t-1 \le t' \le t$. This can be done by applying just one legal shift that shifts that facet. Such a $\hat{B}$ is charged at most twice. Hence we have

$$N_{t,0}^{(d)}(n) = O(\hat{N}_{t,0}^{(d)}(n) + \hat{N}_{t-1,0}^{(d)}(n) + \hat{N}_{t-2,0}^{(d)}(n) + n). \tag{3.3}$$

Since we have already proved that $\hat{N}_{t,0}^{(d)}(n) = O(n \log^{d-1} n)$, for $1 \le t \le 2d$, the induction step follows from (3.3). The following theorem summarizes what we have shown.

THEOREM 3.4. *Let P be a set of n points in $\mathbb{R}^d$, drawn independently from the uniform distribution on $[0,1]^d$. Then the expected number of maximal empty boxes with respect to P is $O(n \log^{d-1} n)$, where the constant of proportionality depends on d.*

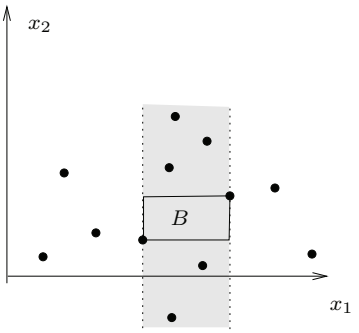


FIG. 3.7. *Bounding the expected number of empty boxes in the plane. The highlighted box B belongs to $\hat{B}_{2,0}(P)$, and its x_1-projection belongs to $B_{2,5}(P')$. The two points defining B must be consecutive in the x_2-order of the points in the shaded strip.*

Semiunbounded colored range counting for random sets. In particular, the bound of Theorem 3.4 also applies to the expected number of maximal empty orthants. Since the storage required by the data-structure of Theorem 2.2 depends on the number of maximal empty orthants with respect to each of the sets P_c, for $1 \le c \le C$, we obtain the following result.

THEOREM 3.5. *Let P be a set of n colored points in dimension $d \ge 2$, such that each point is drawn independently from the uniform distribution on $[0,1]^d$. Then there exists a data structure supporting semiunbounded colored range counting queries in $O(\log^{d-1} n)$ time, whose expected storage is $O(n \log^{2d-2} n)$ and which can be constructed in expected $O(n \log^{2d-2} n)$ time.*

General colored range counting for random sets. Recall that we solve the problem when queries are general bounded boxes by a reduction to the semibounded problem.

However, since this reduction transforms the points by doubling each coordinate, we can no longer assume that the transformed set satisfies our randomness assumption.

We overcome this difficulty using the following observation from section 3.2. Let A be a random point set in $\mathbb{R}^d$ and $\tilde{A}$ the point set in $\mathbb{R}^{2d}$ into which A is mapped; then every maximal empty orthant in $\mathbb{R}^{2d}$ with respect to $\tilde{A}$ corresponds to a maximal empty box in $\mathbb{R}^d$ with respect to A. Hence, we can still apply Theorem 3.4 to the original set A in order to upper-bound the number of maximal empty orthants with respect to $\tilde{A}$. Using the data structure of Theorem 2.4 we obtain the following result.

THEOREM 3.6. *Let P be a random colored point-set of cardinality n in $\mathbb{R}^d$, such that the points of each color are selected as above. Then there exists a data structure supporting colored range counting queries in $O(\log^{2d-1} n)$ time, whose expected storage is $O(n \log^{3d-2} n)$ and which can be constructed in expected $O(n \log^{3d-2} n)$ time.*

Using the same technique, we can build a data structure to answer colored range counting queries for boxes with bounded projections on k specific coordinates, for a random point set, by reducing the problem to a semiunbounded problem in $\mathbb{R}^{d+k}$ and using Theorem 3.4.

Remark. The enhancement that shaves off a logarithmic factor, as provided in Theorem 2.6, does not apply in this case, because it is based on n copies of the structure, and the expected number of maximal empty orthants may be linear in each subproblem, giving rise to an overall data structure of superquadratic size.

4. Achieving time-space tradeoff. In this section we present several techniques for reducing storage at the expense of increasing query time.

4.1. The planar case: Splitting boxes. Our tradeoff technique for the planar case uses the more space efficient data structure of Theorem 2.3 that supports colored range counting queries with 3-sided boxes of the form $[a, b] \times (-\infty, d]$ or $[a, b] \times [c, \infty)$. Recall that we transform this problem into a standard orthogonal containment searching in $\mathbb{R}^3$, and that the resulting data structure requires only $O(n \log^2 n)$ storage and preprocessing cost. (See the discussion preceding Theorem 2.2.) This data structure is a three-level segment tree where each node of a segment tree at the second level stores a linked list of all the boxes associated with it sorted according to their x_1-coordinate.

To obtain Theorem 2.2 we stored at each node of a bottom-level list the number of boxes preceding it in its list. For the present solution, in addition to counting colors, we also want to represent the set of all colors in the query 3-sided box as a disjoint union of canonical sets of colors. This is easy to achieve with our three-level segment tree. We associate a *canonical set of colors* with each node w of a bottom-level list in the three-level segment tree. This is the set of colors of boxes preceding w in the list. (The prefix of the list up to w in fact represents this set.) The representation of the set of all colors in the query 3-sided box consists of the canonical sets of the bottom-level nodes reached by the query. Since the data structure has three levels, the number of such sets in a query output is $O(\log^3 n)$, and each color appears in at most one of these sets (because of the disjointness of the boxes in the decomposition of any single $U(P_c)$).

The full data structure. In order to handle general queries of the form $[a, b] \times [c, d]$, we use the following technique, which has already been used in [12], for solving orthogonal colored range *reporting* problems. We store the points of P, in the increasing order of their y-coordinates, at the leaves of a balanced binary tree T. At each internal node v, we store a pair of auxiliary data structures, one for answering queries of the form $[a, b] \times [c, \infty)$, and one for queries of the form $[a, b] \times (-\infty, d]$. The first (resp., second) structure is built on the points stored at the left (resp., right) subtree of v.

We also store at v a y-coordinate $Y(v)$ that separates the y-coordinates of the points stored at the left subtree from those stored at the right subtree of v. If v is a leaf, we let $Y(v)$ be the y-coordinate of the singleton point stored at v.

For each node w of a bottom-level list of an auxiliary structure we denote by $c(w)$ the canonical set of colors associated with w.

Let $q = [a, b] \times [c, d]$ be a query box. We search down the tree T with $[c, d]$ and denote by v_q the highest node for which $[c, d]$ contains $Y(v_q)$. If v_q is a leaf, we check whether the single point that it stores lies in q, and return 1 if it does and 0 otherwise. If v_q is an internal node, we query the two structures at v_q with $[a, b] \times [c, \infty)$ and with $[a, b] \times (-\infty, d]$, respectively. Let D_q (resp., U_q) denote the set of $O(\log^3 n)$ bottom-level nodes reached by the former (resp., latter) subquery. Observe that the set of colors in q is exactly the union of the canonical sets of colors associated with the nodes in $U_q \cup D_q$. Another crucial property (which follows from the fact that, for each color c, the boxes that represent $U(P_c)$ are pairwise disjoint) is that each color in the output appears in the canonical set of at most one node of U_q and at most one node of D_q (see [12] for more details). Hence, using the exclusion-inclusion principle, the number of colors that appear in q is equal to

$$\sum_{s \in U_q} |c(s)| + \sum_{t \in D_q} |c(t)| - \sum_{s \in U_q,\, t \in D_q} |c(s) \cap c(t)|. \tag{4.1}$$

Ideally, we would like to have for every $s \in U_q$ and $t \in D_q$ the value of $|c(s) \cap c(t)|$ prestored. However, doing this for every possible pair of canonical sets would be too expensive and may result in superquadratic space complexity. Nevertheless, if we did have these values available, answering a query could then be done in $O(\log^6 n)$ time, using (4.1).

Instead, we derive a tradeoff between storage and query time, determined by a threshold parameter X in the range $1 \le X \le n$. Let $\mathcal{D}(v)$ (resp., $\mathcal{U}(v)$) be the set of all bottom-level nodes in the auxiliary structure of a node $v \in T$ used for answering queries of type $[a, b] \times [c, \infty)$ (resp., $[a, b] \times (-\infty, d]$). A node $t \in \mathcal{D}(v) \cup \mathcal{U}(v)$ is called *X-heavy* if $|c(t)| > X$; otherwise it is called *X-light*. For every node $v \in T$ we construct and store, as part of the preprocessing stage, a matrix $M(v)$, whose rows and columns correspond to the X-heavy nodes in $\mathcal{D}(v)$ and $\mathcal{U}(v)$, respectively. For each pair of X-heavy nodes $s \in \mathcal{U}(v)$ and $t \in \mathcal{D}(v)$, we store in $M_{s,t}(v)$ the value of $|c(t) \cap c(s)|$. Let n_v be the number of points stored at the subtree of T rooted at v. The overall size of all the canonical sets associated with the nodes of $\mathcal{U}(v) \cup \mathcal{D}(v)$ is $O(n_v \log^3 n)$. Hence, the number of X-heavy nodes in $\mathcal{D}(v) \cup \mathcal{U}(v)$ is $O\big(\frac{n_v}{X} \log^3 n\big)$, so $M(v)$ has size $O\big(\big(\frac{n_v}{X}\big)^2 \log^6 n\big)$. (The time needed to construct $M(v)$, for all nodes $v \in T$, is discussed later.) Summing this bound over all nodes v of T, we get an overall bound of $O(\big(\frac{n}{X}\big)^2 \log^6 n)$.

For each node $t \in \mathcal{D}(v) \cup \mathcal{U}(v)$, in addition to storing the size $|c(t)|$ of its canonical set, we also store a dictionary data structure on $c(t)$ (implemented as a binary search tree), supporting logarithmic-time searches. Clearly, since $\sum_{t \in \mathcal{D}(v) \cup \mathcal{U}(v)} |c(t)| = O(n_v \log^3 n)$, these additional dictionary structures take a total of $O(n \log^4 n)$ extra storage.

The total space complexity of our solution is thus $O\big(\big(\frac{n}{X}\big)^2 \log^6 n + n \log^4 n\big)$.

Answering queries. It suffices to describe how to compute $|c(s) \cap c(t)|$ efficiently, for each $s \in U_q$ and $t \in D_q$. If both s and t are X-heavy, then this value is stored in $M(v_q)$ and we simply retrieve it. Otherwise, we can assume, without loss of generality,

that $|c(s)| \le X$. In this case we check, for each $c \in c(s)$, whether $c \in c(t)$ and count the number of such colors. Using the dictionary structure over $c(t)$ takes a total of $O(|c(s)| \log n) = O(X \log n)$ time. We repeat this for each pair $(s,t) \in U_q \times D_q$, for a total of $O(X \log^7 n)$ query time.

Preprocessing. It is easy to see that the primary tree T with all its auxiliary data structures can be constructed in $O(n \log^3 n)$ time and the dictionary structures at the nodes of $|\mathcal{U} \cup \mathcal{D}|$ can be constructed in overall time $O(n \log^4 n)$. It remains to describe the construction of matrices M_v.

Denote by $\mathcal{D}$ (resp., $\mathcal{U}$) the set of bottom-level nodes in all of the auxiliary structures used for answering queries of type $[a,b] \times [c,\infty)$ (resp., $[a,b] \times (-\infty,d]$). Let M_D (resp., M_U) denote the matrix whose rows correspond to the X-heavy nodes of $\mathcal{D}$ (resp., of $\mathcal{U}$), and whose columns correspond to the colors $1,\dots,C$, such that, for $t \in \mathcal{D}$ (resp., $t \in \mathcal{U}$) and color c, the (t,c)-entry of the matrix is 1 if $c \in c(t)$ and is 0 otherwise. It follows that $M = M_D {M_U}^T$ contains all the matrices M_v as submatrices.[7]

Using the fact that $\sum_v n_v = O(n \log n)$, we get that each of the matrices M_D and M_U has $t = O\left(\frac{n}{X} \log^4 n\right)$ rows and $N = O(n \log^4 n)$ nonzero entries. Hence, we can construct M using the sparse rectangular matrix multiplication technique of Kaplan, Sharir, and Verbin [20] (which extends a technique of Yuster and Zwick [27] and of Chan [7]). Specifically, for two matrices A, B, each having t rows and at most N nonzero items, these methods construct AB^T in time

$$\begin{cases} O(Nt^{\frac{\omega-1}{2}}) & \text{if } N \ge t^{\frac{\omega+1}{2}}, \\ O(N^{\frac{2\beta}{\beta+1}} t^{\frac{2-\alpha\beta}{\beta+1}}) & \text{if } t^{1+\frac{\alpha}{2}} \le N \le t^{\frac{\omega+1}{2}}, \\ O(t^2) & \text{if } N \le t^{1+\frac{\alpha}{2}}. \end{cases} \tag{4.2}$$

Here (i) ω is the exponent of matrix multiplication; i.e., ω is the smallest number such that two $t \times t$ matrices can be multiplied in time $O(t^\omega)$, (ii) α is the largest value of r for which a $t \times t^r$ matrix and a $t^r \times t$ matrix can be multiplied in time $O(t^2)$, and (iii) $\beta = \frac{\omega-2}{1-\alpha}$. It is known (see, e.g., [18]) that $\omega < 2.376$ and $\alpha > 0.294$; thus we can use $\beta \approx 0.533$.

Hence, the construction time of the data structure is

$$\begin{cases} O^*\left(\frac{n^{(\omega+1)/2}}{X^{(\omega-1)/2}}\right) = O\left(\frac{n^{1.688}}{X^{0.688}}\right) & \text{when } X \ge n^{\frac{\omega-1}{\omega+1}} \approx n^{0.408}, \\ O^*\left(\frac{n^{\frac{2-\alpha\beta+2\beta}{\beta+1}}}{X^{\frac{2-\alpha\beta}{\beta+1}}}\right) = O\left(\frac{n^{1.898}}{X^{1.203}}\right) & \text{when } n^{\frac{\alpha/2}{\alpha/2+1}} \approx n^{0.128} \le X \le n^{\frac{\omega-1}{\omega+1}} \approx n^{0.408}, \\ O^*(\frac{n^2}{X^2}) & \text{when } X \le n^{\frac{\alpha/2}{\alpha/2+1}} \approx n^{0.128}. \end{cases} \tag{4.3}$$

We thus have the following result.

THEOREM 4.1. *Let P be a set of n colored points in the plane, and let $1 \le X \le n$ be a given tradeoff parameter. Then we can preprocess P into a data structure of size $O\left(\left(\frac{n}{X}\right)^2 \log^6 n + n \log^4 n\right)$ so that a colored range counting query can be answered in $O(X \log^7 n)$ time. The preprocessing cost is as given in* (4.3).

[7] We combine all of the matrices M_v into one matrix M to simplify the presentation. In doing so, we lose a polylogarithmic factor in the time bound.

Optimizing for a fixed number of queries. Suppose next that we know (or guess) in advance the number m of queries. We can then optimize the choice of X, so as to minimize the overall time for answering m colored range counting queries, including the time spent at the preprocessing stage. A simple calculation yields the following corollary.

COROLLARY 4.2. *Let P be a set of n colored points in the plane.*

(a) *$m \leq n$ colored range counting queries can be answered in overall time $O^*(nm^{\frac{\omega-1}{\omega+1}}) = O(nm^{0.408})$.*

(b) *$n \leq m \leq n^{\frac{4-\alpha}{\alpha+2}} \approx n^{1.616}$ colored range counting queries can be answered in overall time $O^*(n^{\frac{2-\alpha\beta+2\beta}{3-\alpha\beta+\beta}} m^{\frac{2-\alpha\beta}{3-\alpha\beta+\beta}}) = O(n^{0.862}m^{0.546})$.*

(c) *$m \geq n^{\frac{4-\alpha}{\alpha+2}} \approx n^{1.616}$ colored range counting queries can be answered in overall time $O^*(m^{2/3}n^{2/3})$.*

In particular, n colored range counting queries can be answered in $O(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.408})$ time, including time spent at the preprocessing stage.

4.2. Time-space tradeoff in dimension $d > 2$. In higher dimensions, there are other approaches to achieving a tradeoff between query time and storage.

Bucketing. Partition the set of colors into $O(\log n)$ "buckets" $\mathcal{C}_i$, such that $c \in \mathcal{C}_i$ iff $2^{i-1} \leq |P_c| < 2^i$. Put $C_i := |\mathcal{C}_i|$, and let n_i be the number of points having colors in $\mathcal{C}_i$. Clearly, $n_i = \Theta(2^i C_i)$. We solve the colored range counting problem separately within each $\mathcal{C}_i$ and output the sum of the counts obtained in each of these $O(\log n)$ subproblems.

We use a threshold parameter $1 \leq X \leq n$. Answering a colored range counting query with a box q within $\mathcal{C}_i$ depends on the relationship between X and C_i. If $X \geq C_i$, we simply test, for each color c in $\mathcal{C}_i$, whether $q \cap P_c \neq \emptyset$ and count the number of colors with this property. Using the standard orthogonal range searching machinery [2], this takes $O(\log^{d-1}|P_c|) = O(i^{d-1})$ time per color c, for a total of $O(i^{d-1}C_i) = O(i^{d-1}X)$ time. Summing these bounds over all buckets with $X \geq C_i$, we obtain a total of $O(X \log^d n)$ time.

If $X < C_i$, we use the technique of Theorem 2.5 for answering the query. We use a single colored range counting data structure for all the buckets with $C_i > X$. Let $N = \sum_{\{i|X<C_i\}} n_i \leq n$ be the overall number of points in buckets with $X < C_i$. The query time is $O(\log^{2d-1} N) = O(\log^{2d-1} n)$, and the space and preprocessing cost are both

$$O\left(\sum_i C_i \cdot \left(\frac{n_i}{C_i}\right)^d \log^{2d-1} n\right) = O\left(\sum_i \frac{n_i^d}{X^{d-1}} \log^{2d-1} n\right) = O\left(\sum_i \frac{n^d}{X^{d-1}} \log^{2d-1} n\right),$$

where the summation is over all those "heavy" buckets (we use here the facts that $n_i/C_i = \Theta(2^i)$ and that $X < C_i$). Hence, summing over the buckets, and adding the costs for buckets with $X \geq C_i$, the overall query time is $O(X \log^d n + \log^{2d-1} n)$. The overall preprocessing and space complexity is $O\big(\frac{n^d}{X^{d-1}} \log^{2d-1} n\big)$. Optimizing the choice of X, we can answer m colored range counting queries in time $O(nm^{1-1/d}\log^{d+1-1/d} n + m \log^{2d-1} n)$, including time spent at the preprocessing stage (for this we take $X = \frac{n}{m^{1/d}} \log^{\frac{d-1}{d}}(n)$). We thus have the following result.

THEOREM 4.3. *Let P be a set of n colored points in $\mathbb{R}^d$, $d > 2$, and let $1 \leq X \leq n$ be a tradeoff parameter. We can preprocess P, in time $O\big(\frac{n^d}{X^{d-1}} \log^{2d-1} n\big)$, into a data structure of size $O\big(\frac{n^d}{X^{d-1}} \log^{2d-1} n\big)$, which supports colored range counting queries in time $O(X \log^d n)$. In particular, m such queries can be answered in*

$O(nm^{1-1/d}\log^{d+1-1/d} n + m\log^{2d-1} n)$ *time, including the cost of preprocessing. The storage required in this case is* $O(nm^{1-1/d}\log^{d+1-1/d} n)$.

Two additional methods are presented in section 4.3. They partially improve the upper bounds in the time-space tradeoff for $X = \Omega^*(n^{\frac{d-2}{d-1}})$, but they are more expensive at the preprocessing stage. Ignoring storage, the bucketing method described previously provides the best upper bound so far on the time required to answer any fixed number of queries, if time spent at the preprocessing stage is also included.

4.3. Additional time-space tradeoffs in dimension $d > 2$.

Bucketing with box-splitting. We further improve the tradeoff achieved by bucketing for values of $X = \Omega^*(n^{\frac{d-2}{d-1}})$, by using the box-splitting technique within each bucket, as in the planar case. Specifically, apply bucketing with some threshold value $1 \le X \le n$. Again, if the number of colors within a fixed bucket $\mathcal{C}_i$ is smaller than X, test each of the C_i colors in the bucket for intersection with the query range. This costs $O(X\log^{d-1} n)$ time per bucket, for a total of $O(X\log^d n)$.

Consider then all the "heavy" buckets with $C_i > X$. Apply the box splitting technique for the colors in the corresponding union $\cup_i \mathcal{C}_i$ (using the same threshold parameter X to distinguish between "heavy" and "light" canonical sets; see section 4.1). That is, construct a binary tree on the points sorted by their x_1-coordinates. For each node of the tree we maintain two auxiliary structures for querying with boxes that are semiunbounded in the x_1-direction. As above, the data structure is implemented so that processing a query takes $O(\log^{2d-1} n)$ time, and returns the output as the disjoint union of $O(\log^{2d-1} n)$ canonical sets, each associated with some bottom-level node of the structure. The preprocessing and space complexity of the structure are both

$$O\left(\sum_i C_i \cdot \left(\frac{n_i}{C_i}\right)^{d-1} \log^{2d-1} n_i\right) = O\left(\sum_i \frac{n_i^{d-1}}{X^{d-2}} \log^{2d-1} n\right) = O\left(\frac{n^{d-1}}{X^{d-2}} \log^{2d-1} n\right),$$

where the summation is over all "heavy buckets." Note that this bound holds also if we store a dictionary over each canonical set.

To process a bounded-box query, we proceed as above, finding the highest node in the tree that splits the x_1-span of the query box, performing appropriate queries in the auxiliary data structures with the two corresponding semiunbounded "half-boxes," and combining the solutions, using an appropriate variant of (4.1). The time of the query is then $O(X\log^{4d-1} n)$ by a straightforward extension of the analysis of the planar case.

Since the total size of all the canonical sets of nodes of the respective collections $\mathcal{U}(v)$, $\mathcal{D}(v)$ is now $O\big(\frac{n_v^{d-1}}{X^{d-2}}\log^{2d-1} n\big)$, for a given vertex $v \in T$, the size of the matrix $M(v)$ is

$$O\left(\left(\frac{\frac{n_v^{d-1}}{X^{d-2}}}{X}\right)^2 \log^{4d-2} n\right) = O\left(\left(\frac{n_v}{X}\right)^{2d-2} \log^{4d-2} n\right),$$

and the total storage, over all nodes v and buckets i, is thus

$$O\left(\frac{n^{d-1}}{X^{d-2}}\log^{2d-1} n + \left(\frac{n}{X}\right)^{2d-2} \log^{4d-2} n\right).$$

Comparing this with the bound for the bucketing technique alone, we get an improved tradeoff for $X = \Omega^*(n^{\frac{d-2}{d-1}})$, as asserted.

It remains to bound the preprocessing time, which is dominated by the cost of computing the matrices $M(v)$. As in section 4.1, denote by $\mathcal{D}$ (resp., $\mathcal{U}$) the set of bottom-level nodes in all of the auxiliary structures used for answering queries of type $[a,b]\times[c,\infty)$ (resp., $[a,b]\times(-\infty,d]$). Let M_D (resp., M_U) denote the matrix whose rows correspond to the X-heavy nodes of $\mathcal{D}$ (resp., of $\mathcal{U}$), and whose columns correspond to the colors $1,\dots,C$, such that, for $t\in\mathcal{D}$ (resp., $t\in\mathcal{U}$) and color c, the (t,c)-entry of the matrix is 1 if $c\in c(t)$, the canonical set of t, and is 0 otherwise. We compute the matrix $M=M_D{M_U}^T$ which contains all the matrices M_v as submatrices. Here each of the matrices M_D and M_U has $O\big(\frac{n^{d-1}}{X^{d-2}}\log^{2d-1}n\big)$ nonzero entries, which is the total size of all canonical set, and $O\big(\frac{n^{d-1}}{X^{d-1}}\log^{2d-1}n\big)$ rows, each corresponding to an X-heavy canonical set. Using (4.2) we obtain the following bound on the time to compute M:

$$\text{(4.4)}\qquad \begin{cases} O^*\left(\frac{n^{(d-1)(\omega+1)/2}}{X^{(d-1)(\omega+1)/2-1}}\right) & \text{when } n^{\frac{(d-1)(\omega-1)}{(d-1)(\omega-1)+2}}\le X,\\ O^*\left(\frac{n^{(d-1)\frac{2\beta+2-\alpha\beta}{\beta+1}}}{X^{(d-2)\frac{2\beta}{\beta+1}+(d-1)\frac{2-\alpha\beta}{\beta+1}}}\right) & \text{when } n^{\frac{(d-1)\alpha}{(d-1)\alpha+2}}\le X\le n^{\frac{(d-1)(\omega-1)}{(d-1)(\omega-1)+2}},\\ O^*\left(\frac{n^{2d-2}}{X^{2d-2}}\right) & \text{when } X\le n^{\frac{(d-1)\alpha}{(d-1)\alpha+2}}. \end{cases}$$

That is, we have the following theorem.

THEOREM 4.4. *Let P be a set of n colored points in $\mathbb{R}^d$, $d>2$, and let $1\le X\le n$ be a tradeoff parameter. We can preprocess P, in time as in* (4.4)*, into a data structure of size*

$$O\left(\frac{n^{d-1}}{X^{d-2}}\log^{2d-1}n+\left(\frac{n}{X}\right)^{2d-2}\log^{4d-2}n\right),$$

which supports colored range counting queries in time $O(X\log^{4d-1}n)$.

Grid. Using this method, we achieve additional improvement of the time-space tradeoff for $X=\Omega^*(n^{\frac{d+1}{d+2}})$. Recall that we assume general position of the points of P, in the sense that no two points have the same x_i-coordinate for any $1\le i\le d$. We fix a parameter t, and partition $\mathbb{R}^d$, for each $1\le i\le d$, into t slabs, each containing n/t points of P and bounded by two hyperplanes orthogonal to the x_i-axis. These partitions, when superimposed on each other, induce a nonuniform grid G with t^d cells. There are $O(t^{2d})$ "canonical" boxes, each bounded by $2d$ grid hyperplanes. In the preprocessing stage we compute for each such box the number of colors that it contains, using the bucketing method in $O(nt^{2d-2}\log^{d+1-1/d}n+t^{2d}\log^{2d-1}n)$ time. This bound follows from Theorem 4.3 (with $m=t^{2d}$).

To perform an actual query with some box Q, we find the maximal canonical box Q_0 that it contains and retrieve the number C_0 of colors in Q_0. We then retrieve the $n'\le 2dn/t$ points of P that lie in the "fringe" $Q\setminus Q_0$ of Q, using a standard orthogonal range *reporting* data structure [2]. See Figure 4.1. For each such point p, we check whether its color $c(p)$ is a color that appears in Q_0. For this, we preprocess each of the monochromatic subsets P_c of P, for $c=1,\dots,C$, for d-dimensional orthogonal emptiness range queries. For each $p\in Q\setminus Q_0$, with color $c=c(p)$, such that this is the first fringe point of that color, we test whether $Q_0\cap P_{c(p)}=\emptyset$ using the corresponding structure in $O(\log^{d-1}n)$ time. If this is the case, we add 1 to the color count. In all other cases p is ignored. The overall query time is $O\big(\frac{n}{t}\log^{d-1}n\big)$. The data structure

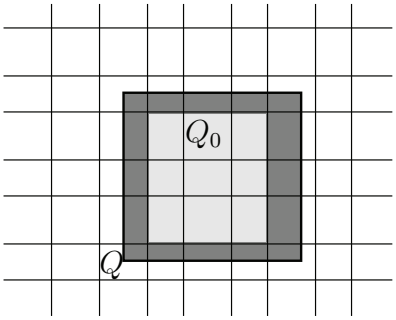


FIG. 4.1. *The grid construction in the plane, a query rectangle Q, its (lightly-shaded) "core" Q_0, and its (darkly-shaded) fringe.*

uses $O(n \log^{d-1} n + t^{2d})$ storage and can be constructed in $O(t^{2d-2} n \log^{d+1-1/d} n + t^{2d} \log^{2d-1} n)$ time. Alternatively, substituting $X := \frac{n}{t}$, we obtain tradeoff bounds that are similar to the previous ones (we leave it to the reader to verify that we get an improvement for $X = \Omega^*(n^{\frac{d+1}{d+2}})$).

THEOREM 4.5. *Let P be a set of n colored points in $\mathbb{R}^d$, $d > 2$, and let $1 \le X \le n$ be a tradeoff parameter. We can preprocess P, in $O\big(\big(\frac{n}{X}\big)^{2d-2} n \log^{d+1-1/d} n + \big(\frac{n}{X}\big)^{2d} \log^{2d-1} n\big)$ time, into a data structure of size $O((\frac{n}{X})^{2d} + n \log^{d-1} n)$, which supports colored range counting queries in $O(X \log^{d-1} n)$ time.*

5. Colored range counting and sparse matrix multiplication. In this section we further elaborate on the relation between sparse matrix multiplication and colored orthogonal range counting. We use this relation to derive lower bounds for the off-line version of the problem in $\mathbb{R}^2$. We also define and analyze a generalized version of sparse matrix multiplication that is related to colored orthogonal range counting in higher dimensions.

5.1. Hardness of orthogonal colored range counting. Consider the following *output restricted sparse matrix multiplication* problem ($ORSMM$). The input is a sparse matrix A with N nonzero entries (and therefore at most N rows and columns), and a set O of M pairs, (i, j), where i and j are indices of two rows of A. The goal is to compute, for each pair $(i, j) \in O$, the (i, j)th entry of the product AA^T. We further assume here that A is zero-one, although AA^T is computed over the integers (or reals).[8]

The off-line version of the colored orthogonal range counting problem with n points and m queries is closely related to the $ORSMM$ problem, as follows from our solution to colored range counting in $\mathbb{R}^2$ in section 4.

THEOREM 5.1. *The 2-dimensional orthogonal colored range counting problem on n points and m query rectangles can be reduced to an $ORSMM$ problem, where the matrix A has $N = O(n \log^4 n)$ nonzero entries and we ask for $M = O(m \log^6 n)$ entries of the matrix AA^T. The reduction takes $O(n \log^4 n)$ time.*

The following theorem shows that a reverse reduction also exists.

THEOREM 5.2. *The $ORSMM$ problem, for a zero-one matrix A with N nonzero entries, where we need to compute M output pairs, can be reduced in linear time to*

[8] We can instead ask for M entries in the product of two arbitrary zero-one matrices A and B with N nonzero items in both. Our results carry over to this generalized version.

a 2-dimensional colored orthogonal range counting problem on $O(N)$ points and M query rectangles.

Proof. We can restate the $ORSMM$ problem as follows. Let Z be the set of columns in the matrix A. For each row i, let $S_i \subseteq Z$ denote the set of columns where row i has ones. Let O be the set of M output pairs to be computed. For each pair $(i,j) \in O$ we have to compute $|S_i \cap S_j|$, which is the (i,j)th entry of AA^T. Since $|S_i \cap S_j| = |S_i| + |S_j| - |S_i \cup S_j|$, this is equivalent to computing, for each pair $(i,j) \in O$, the quantity $|S_i \cup S_j|$.

Let k denote the number of nonempty rows of A. We construct a colored range counting instance with the point set defined in Lemma 3.2; see Figure 3.6. That is, we choose $p_1^1 = (1,1)$, $p_2^1 = (2,2)$, ..., $p_k^1 = (k,k)$ and $p_1^2 = (k+1,1)$, $p_2^2 = (k+2,2)$, ..., $p_k^2 = (2k,k)$. We assign a distinct color to each column of A. We next replace each point p_i^1 by $|S_i|$ points, colored by the colors of the columns in S_i. We place the $|S_i|$ colored points, which correspond to p_i^1, close to each other within distance $\epsilon \ll 1$ of p_i. We do the same for each of the points p_i^2. Let P denote the resulting point set; we have $|P| = 2N$. Clearly, with an appropriate representation of the input, this construction takes $O(N)$ time.

Now, in order to calculate $|S_i \cup S_j|$, for $j \leq i$, we query P with the rectangle $R = [i - \epsilon, k + j + \epsilon] \times [j - \epsilon, i + \epsilon]$ which is slightly larger than the rectangle in the proof of Lemma 3.2 and contains it. Since R contains only the points p_i^1 and p_j^2, it is clear that the number of distinct colors in R is $|S_i \cup S_j|$. ∎

We next describe some observations regarding the complexity of $ORSMM$ and the implications of Theorem 5.2. For any $t \times t$ zero-one matrix A, computing AA^T (over the reals) is a special case of $ORSMM$, with $N = M = t^2$ [11]. The best known algorithm for computing AA^T runs in $O(t^\omega) = O(N^{\omega/2})$ time, for $\omega \simeq 2.376$. So any algorithm for $ORSMM$ whose running time is faster than $O(\min(N,M)^{\omega/2})$ would immediately imply a better algorithm for multiplying zero-one matrices (over the reals), thereby solving a long-standing open problem. Using Theorem 5.2, we obtain that an algorithm for planar colored orthogonal range counting that can answer m box queries with respect to a set of $O(n)$ colored points in $o(\min(n,m)^{\omega/2})$ time would imply an algorithm that can compute AA^T in $o(t^\omega)$ time, for any $t \times t$ zero-one matrix A.

Similarly, consider the problem of computing AA^T, where A is a sparse rectangular zero-one matrix with t rows and N ones such that $N \geq t^{\frac{\omega+1}{2}}$. The best known algorithm for performing this computation runs in $O(Nt^{\frac{\omega-1}{2}})$ time (see section 4). For any such matrix A, computing AA^T is also an instance of $ORSMM$, with N ones and $M = t^2$ pairs. So an algorithm for $ORSMM$ that runs in $o(NM^{\frac{\omega-1}{4}})$ time, for $N \geq M^{\frac{\omega+1}{4}}$, would imply a faster algorithm for sparse rectangular matrix multiplication than the best known to date. Using Theorem 5.2, we obtain that the existence of an algorithm for planar colored orthogonal range counting that can answer m box queries with respect to a set of n colored points in $o(nm^{\frac{\omega-1}{4}}) \approx o(nm^{0.344})$ time, for $n \geq m^{\frac{1+\omega}{4}} \approx m^{0.844}$, would also imply an algorithm that can compute AA^T for any rectangular zero-one matrix A with t rows and N ones, such that $N \geq t^{\frac{\omega+1}{2}}$, faster than what is known to date.

Generalized matrix multiplication and higher-dimensional colored range counting. For $d > 2$, we can show a similar relation between colored orthogonal range counting and a generalization of $ORSMM$. In this generalization, one is given a zero-one matrix A with N nonzero entries in sparse representation and a list O of M d-tuples of indices of rows of A. Let t be the number of columns in A. The goal is to compute,

for each tuple $(i_1, \ldots, i_d) \in O$, the sum $\sum_{j=1}^{t} \prod_{k=1}^{d} A_{i_k,j}$, which is the $ORSMM_d$ problem stated in subsection 1.2, and denote it by $ORSMM_d$. The following theorem generalizes Theorem 5.2 to dimension $d > 2$.

THEOREM 5.3. *Any instance of the $ORSMM_d$ problem, of a zero-one matrix A with N nonzero entries and M output tuples, can be reduced, in linear time, to $O(1)$ instances of d'-dimensional colored orthogonal range counting, for $d' \leq d$, each on $O(N)$ points and M query boxes.*

Proof. Let Z be the set of columns of A. The $ORSMM_d$ problem is equivalent to the following problem if we take S_i to be the set of columns with nonzero entries in row i.

We are given a family of sets $\mathcal{F} = \{S_1, \ldots, S_k\}$, such that $S_i \subseteq Z$ and $\sum_{i=1}^{k} |S_i| = N$, and M d-tuples $\{(i_1, \ldots, i_d)\}_{i=1}^{M}$; the goal is to compute, for each of the tuples, the corresponding quantity $|\bigcap_{j=1}^{d} S_{i_j}|$.

By using the inclusion-exclusion principle, one can easily verify that the above problem is equivalent to computing, for each output tuple $(i_1, \ldots, i_d)$, the quantities $|\bigcup_{j \in J} S_{i_j}|$, for every $J \subseteq \{1, \ldots, d\}$. We first show how to do it for the case where $J = \{1, \ldots, d\}$.

Let k denote the number of nonempty rows of A. We assign to each column of A a distinct color. Let $\{p_i^j \mid 1 \leq i \leq k, 1 \leq j \leq d\}$ be the set of points defined in Lemma 3.2. We define a colored range counting instance composed of dk "point clusters" P_i^j, for $1 \leq i \leq k$ and $1 \leq j \leq d$, as follows. For each i and j, we replace p_i^j by a set P_i^j of $|S_i|$ distinct points placed at distance less than $\epsilon \ll 1$ from p_i^j, each colored by one of the colors of the columns in S_i. Let P be the resulting set of points.

Now, in order to compute $|\bigcup_{j=1}^{d} S_{i_j}|$, where $1 \leq i_d < i_{d-1} < \cdots < i_1 \leq k$, we query P with the box $R = [i_1 - \epsilon, i_2 + k + \epsilon] \times [i_2 - \epsilon, i_3 + k + \epsilon] \times \cdots \times [i_{d-2} - \epsilon, i_{d-1} + k + \epsilon] \times [i_{d-1} - \epsilon, i_d + k + \epsilon] \times [i_d - \epsilon, i_1 + \epsilon]$. (This box is slightly larger than the corresponding box used in Lemma 3.2 and contains it.)

By Lemma 3.2, $R \cap P = \bigcup_{j=1}^{d} P_{i_j}^j$. The number of colors in R is then $|\bigcup_{i=1}^{d} S_{i_j}|$. We have $|P| = dN$, and P can be constructed in $O(N)$ time, assuming an appropriate representation of A.

We find $|\bigcup_{j \in J} S_{i_j}|$, for each tuple $(i_1, \ldots, i_d)$ and some fixed $J \subset \{1, \ldots, d\}$ by a similar reduction obtaining a $|J|$-dimensional colored orthogonal range counting instance with $O(N)$ points and M query boxes. ∎

5.2. Efficient algorithms for $ORSMM_d$. We obtain an efficient algorithm for $ORSMM_d$ in three stages, where each stage uses the previous algorithm as a subroutine.

The case of nonsparse matrices with no output restriction. We start by considering the nonsparse version of the problem where A is any $t \times t$ matrix and we want to compute $\sum_{j=1}^{t} \prod_{k=1}^{d} A_{i_k,j}$ for all d-tuples of rows of A. We refer to this problem as *d-dimensional matrix multiplication.*

Recall that we denote by ω the smallest constant such that standard matrix multiplication of $t \times t$ matrices takes $O(t^\omega)$ time. Let ω_d be a constant such that d-dimensional matrix multiplication takes $O(t^{\omega_d})$ time (in particular, $\omega_2 = \omega$). The output size of the d-dimensional matrix multiplication problem is $\Theta(t^d)$, so $\omega_d \geq d$. On the other hand we can compute each of the $O(t^d)$ sums $\sum_{j=1}^{t} \prod_{k=1}^{d} A_{i_k,j}$ in $O(t)$ time, so $\omega_d \leq d + 1$. In fact it is not hard to obtain a better upper bound on ω_d.

LEMMA 5.4. *For any $d \geq 2$, $\omega_d \leq \omega + d - 2$. Furthermore, if $\omega > 2$ and $d > 2$, then $\omega_d < \omega + d - 2$.*

Proof. For $d = 2$ the lemma obviously holds, so we assume $d \geq 3$. Let A be the input matrix. First compute two matrices B and C, so that B (resp., C) is a $t^{\lceil d/2 \rceil} \times t$ matrix (resp., $t \times t^{\lfloor d/2 \rfloor}$ matrix) whose rows (resp., columns) contain the elementwise products of all sets of $\lceil d/2 \rceil$ rows (resp., $\lfloor d/2 \rfloor$ columns) of A. Both B and C can be computed in time $O(\binom{t}{\lceil d/2 \rceil} \cdot t) = O(t^d)$. By construction, the (usual, two-dimensional) product BC provides a solution to the d-dimensional matrix multiplication for A. By partitioning BC into blocks of size $t \times t$, computing BC can be done in $O(t^{\omega+d-2})$ time. In fact, if $\omega > 2$, then the results of Huang and Pan [18] about rectangular matrix multiplication allows us to improve this, thus establishing the second part of the lemma. □

LEMMA 5.5. *Let A be an $s \times t$ rectangular matrix. Then one can compute $\sum_{j=1}^{t} \prod_{k=1}^{d} A_{i_k,j}$, for all d-tuples of rows of A, in time*

$$\begin{cases} O(s^{\omega+d-3}t) & if\ s \leq t, \\ O(s^{d-\alpha\beta}t^{\beta}) & if\ s^{\alpha} < t < s, \\ O(s^d) & if\ t \leq s^{\alpha}. \end{cases}$$

Proof. As in the proof of Lemma 5.4, we reduce the problem, in $O(\binom{s}{\lceil d/2 \rceil} \cdot t)$ time, to the multiplication of two rectangular matrices B and C of size $s^{\lceil d/2 \rceil} \times t$ and $t \times s^{\lfloor d/2 \rfloor}$, respectively. The multiplication can be done by partitioning B and C into blocks of size $s \times t$ and $t \times s$, respectively. The lemma follows using the upper bound

$$\begin{cases} O(s^{\omega-1}t) & \text{if } s \leq t, \\ O(s^{2-\alpha\beta}t^{\beta}) & \text{if } s^{\alpha} < t < s, \\ O(s^2) & \text{if } t \leq s^{\alpha} \end{cases}$$

on the complexity of multiplication of two rectangular matrices of size $s \times t$ and $t \times s$, respectively (Coppersmith [10], Huang and Pan [18]; see also [7, 20, 27]). □

The case of sparse matrices with no output restriction. We next consider the d-dimensional sparse matrix multiplication problem, where the zero-one input matrix has N ones and s rows, and the goal is to compute $\sum_{j=1}^{t} \prod_{k=1}^{d} A_{i_k,j}$ for all d-tuples of the rows. We already considered this problem for $d = 2$ in section 4.

We solve this problem, for $d > 2$, using the same approach as in [20] (which is built upon the earlier technique of [27]; see also [7]). Let x be a parameter to be determined shortly. We consider separately columns with fewer than x nonzero entries (*light columns*) and columns with at least x nonzero entries (*heavy columns*). A light column with $z \leq x$ nonzero entries contributes a nonzero term to z^d products. So the total time to handle the light columns is at most $O(\sum_i x_i^d)$, where x_i is the number of nonzero items in the ith light column. This expression is maximized and equals $\frac{N}{x}x^d$, if we split the at most N nonzero entries among at most N/x light columns, placing x nonzero entries in each column. We have at most N/x heavy columns and we compute their contribution to the output by the previous algorithm for d-dimensional multiplication of the corresponding rectangular (nonsparse) matrix with s rows and N/x columns, as provided by Lemma 5.5. Optimizing over x, we obtain the following overall bound on the running time:

$$\begin{cases} \left(Ns^{(\omega+d-3)\frac{d-1}{d}}\right) & \text{if } N \geq s^{1+(\omega+d-3)/d}, \\ O\left(N^{\frac{\beta-1}{d+\beta-1}(d-1)+1}s^{\frac{d-\alpha\beta}{d+\beta-1}(d-1)}\right) & \text{if } s^{1+\alpha-\frac{\alpha}{d}} \leq N \leq s^{1+(\omega+d-3)/d}, \\ O(s^d) & \text{if } N \leq s^{1+\alpha-\frac{\alpha}{d}}. \end{cases}$$

The general case: An efficient algorithm for $ORSMM_d$. We fix another threshold parameter y and consider separately the rows with at most y nonzero entries (the *light rows*) and the rows with at least y nonzero entries (the *heavy rows*). We can compute each product in which a light row takes part in $O^*(y)$ time, so all such products can be computed in $O^*(My)$ time. We are left with products that include only heavy rows. Since there are at most N/y heavy rows, we can compute all remaining products by solving a d-dimensional sparse multiplication problem (with no output restriction) with $s = \frac{N}{y}$ rows and N nonzero entries, using the algorithm just described. Optimizing over y in order to minimize the total time complexity of the solution, we obtain the following theorem.

THEOREM 5.6. *For any $d \geq 2$, $ORSMM_d$ with N nonzero entries and M output tuples can be solved in time:*

$$\begin{cases} O^*\left(NM^{\frac{(d-1)(\omega+d-3)}{d+(d-1)(\omega+d-3)}}\right) & if\ M \leq N^{\frac{d+(d-1)(\omega+d-3)}{\omega+2d-3}}, \\ O^*\left(N^{\frac{(d-\alpha\beta+\beta-1)(d-1)+d+\beta-1}{(d-\alpha\beta)(d-1)+d+\beta-1}} M^{\frac{(d-1)(d-\alpha\beta)}{(d-1)(d-\alpha\beta)+d+\beta-1}}\right) & if\ N^{\frac{d+(d-1)(\omega+d-3)}{\omega+2d-3}} \leq M \leq N^{\frac{d-\alpha+\alpha/d}{1+\alpha-\alpha/d}}, \\ O^*\left(N^{\frac{d}{d+1}} M^{\frac{d}{d+1}}\right) & if\ M \geq N^{\frac{d-\alpha+\alpha/d}{1+\alpha-\alpha/d}}. \end{cases}$$

In particular, we get for $d = 2$ a solution for $ORSMM_2$ that takes $O^*\left(NM^{\frac{\omega-1}{\omega+1}}\right)$ time, for $M \leq N$. Note that this solution is inferior to the one given in subsection 4.1 for the case $M = \binom{t}{2}$ (no output restriction): The previous algorithm takes $O\left(Nt^{\frac{\omega-1}{2}}\right)$ time, whereas the new one takes $O\left(Nt^{\frac{2(\omega-1)}{\omega+1}}\right)$ time, which is indeed much larger. It would be interesting to improve the new algorithm for all possible values of M.

6. Conclusion. In this paper we obtain the first nontrivial solution for colored orthogonal range counting in dimension $d \geq 3$. In dimension $d = 2$, we provide an efficient tradeoff scheme between query time and storage. In particular, we show that the off-line version of the problem is nearly equivalent to an output-restricted version of sparse (zero-one) matrix multiplication ($ORSMM$). This suggests that our tradeoff scheme, in dimension $d = 2$, which takes $O^*(n^{1.408})$ time to answer n queries of n points, is fairly efficient. Moreover, its substantial improvement (to $o(n^{1.188})$ for the above off-line version) would immediately reduce the best known bound on the exponent ω of matrix multiplication (for zero-one matrices).

The challenge, however, is to provide a uniform (and efficient) tradeoff scheme in dimension $d \geq 3$. It would be interesting to find a reduction from the off-line version of the problem in dimension $d \geq 3$ to the $ORSMM_d$ problem (which is a straightforward generalization of $ORSMM$), presented in section 5. The reverse reduction is stated (and proven) in Theorem 5.3.

Acknowledgment. We thank three anonymous referees for valuable suggestions that helped us to improve the presentation.

REFERENCES

[1] J. L. BENTLEY, H. T. KUNG, M. SCHKOLNICK, AND C. D. THOMPSON, *On the average number of maxima in a set of vectors and applications*, J. ACM, 25 (1978), pp. 536–543.

[2] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Heidelberg, 2000.

[3] J.-D. BOISSONNAT, M. SHARIR, B. TAGANSKY, AND M. YVINEC, *Voronoi diagrams in higher dimensions under certain polyhedral distance functions*, Discrete Comput. Geom., 19 (1998), pp. 485–519.

[4] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis, *New upper bounds for generalized intersection searching problems*, in Proceedings of the 22nd Annual International Colloquium on Automata, Languages, and Programming, Szeged, Hungary, 1995, Lecture Notes in Comput. Sci. 944, Springer-Verlag, Berlin, 1995, pp. 464–474.

[5] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis, *Red-blue intersection reporting for objects of non-constant size*, Comput. J., 39 (1996), pp. 541–546.

[6] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis, *New results on intersection query problems,* Comput. J., 40 (1997), pp. 22–29.

[7] T. M. Chan, *Dynamic subgraph connectivity with geometric applications*, SIAM J. Comput., 36 (2006), pp. 681–694.

[8] L. P. Chew, D. Dor, A. Efrat, and K. Kedem, *Geometric pattern matching in d-dimensional space*, Discrete Comput. Geom., 21 (1999), pp. 257–274.

[9] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry,* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[10] D. Coppersmith, *Rectangular matrix multiplication revisited*, J. Complexity, 13 (1997), pp. 42–49.

[11] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[12] D. Gupta, R. Janardan, and M. Smid, *Computational geometry: Generalized intersection searching*, in Handbook of Data Structures and Applications, D. Mehta and S. Sahni, eds., Chapman & Hall/CRC, Boca Raton, FL, 2005, pp. 64-1–64-17.

[13] D. Gupta, R. Janardan, and M. Smid, *A technique for adding range restrictions to generalized searching problems*, Inform. Process. Lett., 64 (1997), pp. 263–269.

[14] D. Gupta, R. Janardan, and M. Smid, *Algorithms for some intersection searching problems involving circular objects*, Internat. J. Math. Algorithms, 1 (1999), pp. 35–52.

[15] D. Gupta, R. Janardan, and M. Smid, *Algorithms for generalized halfspace range searching and other intersection searching problems*, Comput. Geom., 5 (1996), pp. 321–340.

[16] D. Gupta, R. Janardan, and M. Smid, *Further results on generalized intersection problems: Counting, reporting, and dynamization*, J. Algorithms, 19 (1995), pp. 282–317.

[17] S. Har-Peled, *On the Expected Complexity of Random Convex Hulls*, Technical report 330/98, School Math. Sci., Tel-Aviv University, Tel-Aviv, Israel, 1998.

[18] X. Huang and V. Y. Pan, *Fast rectangular matrix multiplication and applications*, J. Complexity, 14 (1998), pp. 257–299.

[19] R. Janardan and M. Lopez, *Generalized intersection searching problems*, Internat. J. Comput. Geom. Appl., 3 (1993), pp. 39–69.

[20] H. Kaplan, M. Sharir, and E. Verbin, *Colored intersection searching via sparse rectangular matrix multiplication*, in Proceedings of the 22nd Annual ACM Symposium on Computational Geometry, Sedona, AZ, 2006, pp. 52–60.

[21] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin, *Counting colors in boxes*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 2007, pp. 785–794.

[22] J. Matoušek and O. Schwarzkopf, *On ray shooting in convex polytopes*, Discrete Comput. Geom., 10 (1993), pp. 215–232.

[23] J. Matoušek, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.

[24] J. Matoušek, *Cutting hyperplane arrangements*, Discrete Comput. Geom., 6 (1991), pp. 385–406.

[25] J. Matoušek, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.

[26] D. E. Willard and G. S. Lueker, *Adding range restriction capability to dynamic data structures*, J. ACM, 32 (1985), pp. 597–617.

[27] R. Yuster and U. Zwick, *Fast sparse matrix multiplication*, ACM Trans. Algorithms, 1 (2005), pp. 2–13.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1012–1032

FINDING BRANCH-DECOMPOSITIONS AND RANK-DECOMPOSITIONS*

PETR HLINĚNÝ† AND SANG-IL OUM‡

Abstract. We present a new algorithm that can output the rank-decomposition of width at most k of a graph if such exists. For that we use an algorithm that, for an input matroid represented over a fixed finite field, outputs its branch-decomposition of width at most k if such exists. This algorithm works also for partitioned matroids. Both of these algorithms are fixed-parameter tractable, that is, they run in time $O(n^3)$ where n is the number of vertices / elements of the input, for each constant value of k and any fixed finite field. The previous best algorithm for construction of a branch-decomposition or a rank-decomposition of optimal width due to Oum and Seymour [*J. Combin. Theory Ser. B*, 97 (2007), pp. 385–393] is not fixed-parameter tractable.

Key words. rank-width, clique-width, branch-width, fixed-parameter tractable algorithm, graph, matroid

AMS subject classifications. 05C85, 68R10

DOI. 10.1137/070685920

1. Introduction. Many graph problems are known to be NP-hard in general; however, for practical application we still need to solve them. One method to solve them is to restrict the input graph to have a certain structure. Clique-width, defined by Courcelle and Olariu [4], is very useful for that purpose. Many hard graph problems (in particular all those expressible in monadic second-order logic (MSOL) of adjacency graphs) are solvable in polynomial time as long as the input graph has bounded clique-width and is given in the form of the decomposition for clique-width, called a *k-expression* [3, 24, 6, 15, 10]. A k-expression is an algebraic expression with the following four operations on a vertex-labeled graph with k labels: create a new vertex with label i, take the disjoint union of two labeled graphs, add all edges between vertices of label i and label j, and relabel all vertices with label i to have label j. However, for fixed $k > 3$, it is not known how to find a k-expression of an input graph having clique-width at most k. (If $k \leq 3$, then it has been shown in [2, 1].)

Rank-width is another graph structural invariant introduced by Oum and Seymour [19], aiming at the construction of an $f(k)$-expression of the input graph having clique-width k for some fixed function f in polynomial time. Rank-width is defined (section 7) as the branch-width (see section 2) of the *cut-rank* function of graphs. Rank-width turns out to be very useful for algorithms on graphs of bounded clique-width, since a class of graphs has bounded rank-width if and only if it has bounded clique-width. In fact, if rank-width of a graph is k, then its clique-width lies between

*Received by the editors March 21, 2007; accepted for publication (in revised form) March 3, 2008; published electronically June 25, 2008.
http://www.siam.org/journals/sicomp/38-3/68592.html

†Faculty of Informatics, Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic (hlineny@fi.muni.cz). This author's research was supported by Czech research grant GAČR 201/08/0308 and by Research intent MSM0021622419 of the Czech Ministry of Education.

‡Department of Mathematical Sciences, KAIST, Daejeon, 305-701, Republic of Korea (sangil@kaist.edu). This author's research was done while at the Georgia Institute of Technology and University of Waterloo and partially supported by NSF grant DMS 0354742 and the SRC Program of Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (R11-2007-035-01002-0).

k and $2^{k+1}-1$ [19] and an expression can be constructed from a rank-decomposition of width k.

In this paper, we are mainly interested in the following problem.

> Find a fixed-parameter tractable algorithm that outputs a rank-decomposition of width at most k if the rank-width of an input graph (with more than one vertex) is at most k.

The first rank-width algorithm by Oum and Seymour [19] finds only a rank-decomposition of width at most $3k+1$ for n-vertex graphs of rank-width at most k in time $O(n^9 \log n)$. This algorithm has been improved by Oum [18] to output a rank-decomposition of width at most $3k$ in time $O(n^3)$. Using this approximation algorithm and finiteness of excluded vertex-minors [17], Courcelle and Oum [5] have constructed an $O(n^3)$-time algorithm to decide whether a graph has rank-width at most k. However, this is only a decision algorithm; if the rank-width is at most k, then this algorithm verifies that the input graph contains none of the excluded graphs for rank-width at most k as a vertex-minor. It does *not* output a rank-decomposition showing that the graph indeed has rank-width at most k.

In another paper, Oum and Seymour [20] have constructed a polynomial-time algorithm that can output a rank-decomposition of width at most k for graphs of rank-width at most k. However, it is not fixed-parameter tractable; its running time is $O(n^{8k+12} \log n)$. Obviously, it is very desirable to have a fixed-parameter tractable algorithm to output such an "optimal" rank-decomposition, because most algorithms on graphs of bounded clique-width require a k-expression on their input. So far, the only known efficient way of constructing an expression with bounded number of labels for a given graph of bounded clique-width uses rank-decompositions.

In this paper, we present an affirmative answer to the above problem (Theorem 7.3). An amusing aspect of our solution is that we deeply use submodular functions and matroids to solve the rank-decomposition problem, which shows (somehow unexpectedly) a "truly geometrical" nature of this graph-theoretical problem. In fact we solve the following related problem on matroids, too (Theorem 6.7).

> Find a fixed-parameter tractable algorithm that, given a matroid represented by a matrix over a fixed finite field, outputs a branch-decomposition of width at most k if the branch-width of the input matroid (with more than one element) is at most k.

So to give the final solution of our first problem, Theorem 7.3, we are going to bring together two previously separate lines of research: We will combine Oum and Seymour's above sketched work on rank-width and on branch-width of submodular functions with Hliněný's recent works [13, 14] on parametrized algorithms for matroids over finite fields.

Namely, Hliněný [13] has given a parametrized algorithm which in time $O(n^3)$ either outputs a branch-decomposition of width $\leq 3k+1$ of an input matroid M represented over a fixed finite field or confirms that the branch-width of M is more than $k+1$ (Algorithm 6.1). Using the ideas of [14] and minor-monotonicity of the branch-width parameter, he has concluded with an $O(n^3)$ fixed-parameter tractable algorithm [13] for the (exact value of) branch-width k of an input matroid M represented over a fixed finite field (Theorem 5.1). Similarly as above, this algorithm is only a decision algorithm and does not output a branch-decomposition of width k.

We last remark that the following (indeed widely expected) hardness result has been given only recently by Fellows et al. [7]: It is NP-hard to find graph clique-width. To argue that it is NP-hard to find rank-width, we combine some known

results: Hicks and McMurray Jr. [11] and Mazoit and Thomassé [16] independently proved that the branch-width of the cycle matroid of a graph is equal to the branch-width of the graph if it is 2-connected. Hence, we can reduce (section 7) the problem of finding branch-width of a graph to finding rank-width of a certain bipartite graph, and finding graph branch-width is NP-hard as shown by Seymour and Thomas [23]. Still, the main advantage of rank-width over clique-width is the fact that we currently have a fixed-parameter tractable algorithm for rank-width but not for clique-width.

Our paper is structured as follows: The next section briefly introduces definitions of branch-width, partitions, matroids, and the amalgam operation on matroids. In section 3, we explain the notion of so-called titanic partitions, which we further use in section 4 to "model" partitioned matroids in ordinary matroids. At this point it is worth noting that partitioned matroids present the key tool that allows us to shift from a branch-width-testing algorithm [13] to a construction of an "optimal" branch-decomposition (see Theorem 5.7) and of a rank-decomposition. In section 5, we will discuss a simple but slow algorithm for matroid branch-decompositions. In section 6, we will present a faster algorithm. As the main application, we then use our result to give an algorithm for constructing a rank-decomposition of optimal width of a graph in section 7.

2. Definitions. *Branch-width.* Let $\mathbb{Z}$ be the set of integers. For a finite set V, a function $f : 2^V \to \mathbb{Z}$ is called *symmetric* if $f(X) = f(V \setminus X)$ for all $X \subseteq V$, and is called *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ for all subsets X, Y of V. A tree is *subcubic* if all vertices have degree 1 or 3. For a symmetric submodular function $f : 2^V \to \mathbb{Z}$ on a finite set V, the branch-width is defined as follows (see Figure 1).

A *branch-decomposition* of the symmetric submodular function f is a pair (T, μ) of a subcubic tree T and a bijective function $\mu : V \to \{t : t \text{ is a leaf of } T\}$. (If $|V| \leq 1$, then f admits no branch-decomposition.) For an edge e of T, the connected components of $T \setminus e$ induce a partition (X, Y) of the set of leaves of T. (In such a case, we say that $\mu^{-1}(X)$ (or $\mu^{-1}(Y)$) is *displayed* by e in the branch-decomposition (T, μ). We also say that V and $\emptyset$ are displayed by the branch-decomposition.) The *width* of an edge e of a branch-decomposition (T, μ) is $f(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width of all edges of T. The *branch-width* of f, denoted by $\mathrm{bw}(f)$, is the minimum of the width of all branch-decompositions of f. (If $|V| \leq 1$, we define $\mathrm{bw}(f) = f(\emptyset)$.)

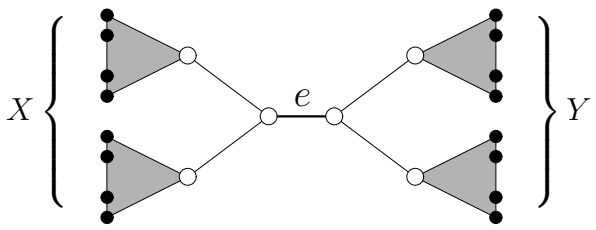


FIG. 1. *An illustration of the definition of a branch-decomposition (T, μ) of f: An edge e of the tree T displays the sets $\mu^{-1}(X)$ and $\mu^{-1}(Y)$, and the width of e is $f(\mu^{-1}(X))$.*

A natural application of this definition is the branch-width of a graph, as introduced by Robertson and Seymour [22] along with better known tree-width, and its direct matroidal counterpart below in this section. We also refer to further formal definition of rank-width in section 7.

Partitions. A *partition* $\mathcal{P}$ of V is a collection of nonempty pairwise disjoint subsets of V whose union is equal to V. Each element of $\mathcal{P}$ is called a *part*. For a symmetric submodular function f on 2^V and a partition $\mathcal{P}$ of V, let $f^{\mathcal{P}}$ be a function on $2^{\mathcal{P}}$ (also symmetric and submodular) such that $f^{\mathcal{P}}(X) = f(\cup_{Y\in X} Y)$. The *width* of a partition $\mathcal{P}$ is $f(\mathcal{P}) = \max\{f(Y) : Y \in \mathcal{P}\}$.

We will often denote a partition by a function as follows. For a function $\pi : V \to W$, let $\pi_y = \{x : \pi(x) = \pi(y)\}$ for $y \in V$, and let $[\pi] = \{\pi_x : x \in V\}$ be the partition of V induced by π.

Matroids. We refer to Oxley [21] in our matroid terminology. A *matroid* is a pair $M = (E, \mathcal{B})$, where $E = E(M)$ is the ground set of M (elements of M), and $\mathcal{B} \subseteq 2^E$ is a nonempty collection of *bases* of M, no two of which are in an inclusion. Moreover, matroid bases satisfy the "exchange axiom": if $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 \setminus B_2$, then there is $y \in B_2 \setminus B_1$ such that $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. We consider only finite matroids. A typical example of a matroid is given by a set of vectors (forming the columns of a matrix $\boldsymbol{A}$) with usual linear independence. The matrix $\boldsymbol{A}$ is then called a *representation* of the matroid.

All matroid bases have the same cardinality called the *rank* $r(M)$ of the matroid. Subsets of bases are called *independent*, and sets that are not independent are *dependent*. A matroid M is *uniform* if all subsets of $E(M)$ of certain size are the bases, and M is *free* if $E(M)$ is a basis. The *rank function* $r_M(X)$ in M is the maximum cardinality of an independent subset of a set $X \subseteq E(M)$. The *dual matroid* M^* is defined on the same ground set with the bases as set-complements of the bases of M. For a subset X of E, the *deletion* $M \setminus X$ of X from M, or the *restriction* $M \restriction (E \setminus X)$ of M to $E \setminus X$, is the matroid on $E \setminus X$ in which $Y \subseteq E \setminus X$ is independent in $M \setminus X$ if and only if Y is an independent set of M. The *contraction* M/X of X in M is the matroid $(M^* \setminus X)^*$. Matroids of the form $M/X \setminus Y$ are called *minors* of M.

To define the branch-width of a matroid, we consider its (symmetric and submodular) connectivity function

$$\lambda_M(X) = r_M(X) + r_M(E \setminus X) - r_M(E) + 1$$

defined for all subsets $X \subseteq E = E(M)$. A "*geometric*" meaning is that the subspaces spanned by X and $E\setminus X$ intersect in a subspace of dimension $\lambda_M(X)-1$. *Branch-width* $\mathrm{bw}(M)$ and *branch-decompositions* of a matroid M are defined as the branch-width and branch-decompositions of λ_M. Notice that $\lambda_{M^*} \equiv \lambda_M$.

Partitioned matroids. A pair $(M, \mathcal{P})$ is called a *partitioned matroid* if M is a matroid and $\mathcal{P}$ is a partition of $E(M)$. A partitioned matroid $(M, \mathcal{P})$ is *representable* if M is representable. A connectivity function of a partitioned matroid $(M, \mathcal{P})$ is defined as $\lambda^{\mathcal{P}}_M$. We note that $\lambda^{\mathcal{P}}_M$ is *symmetric* and *submodular*; in other words,

$$\lambda^{\mathcal{P}}_M(X) = \lambda^{\mathcal{P}}_M(\mathcal{P} \setminus X),$$
$$\lambda^{\mathcal{P}}_M(X) + \lambda^{\mathcal{P}}_M(Y) \geq \lambda^{\mathcal{P}}_M(X \cap Y) + \lambda^{\mathcal{P}}_M(X \cup Y).$$

Branch-width $\mathrm{bw}(M, \mathcal{P})$ and *branch-decompositions* of a partitioned matroid $(M, \mathcal{P})$ are defined as branch-width, branch-decompositions of $\lambda^{\mathcal{P}}_M$.

Amalgams of matroids. Let M_1, M_2 be matroids on E_1, E_2, respectively, and $T = E_1 \cap E_2$. Moreover, let us assume that $M_1 \restriction T = M_2 \restriction T$. If M is a matroid on $E_1 \cup E_2$ such that $M \restriction E_1 = M_1$ and $M \restriction E_2 = M_2$, then M is called an *amalgam* of M_1 and M_2 (see Figure 2).

It is known that an amalgam of two matroids need neither exist nor be unique. However, there are certain interesting cases when an amalgam is known to exist.

We show one such example here, and we use another one in Proposition 5.4 with representable matroids. Let r_1, r_2 be the rank function of M_1, M_2, respectively. Let r be the rank function of $M_1 \restriction T$. Let

$$\eta(X) = r_1(X \cap E_1) + r_2(X \cap E_2) - r(X \cap T)$$

and

$$\zeta(X) = \min\{\eta(Y) : Y \supseteq X\}.$$

PROPOSITION 2.1 (see [21, Proposition 12.4.2]). *If ζ is submodular, then ζ is the rank function of a matroid that is an amalgam of M_1 and M_2.*

If ζ is submodular, then the matroid on $E_1 \cup E_2$ having ζ as its rank function is called the *proper amalgam* of M_1 and M_2.

LEMMA 2.2. *If $M_1 \restriction T$ is free, then ζ is submodular and therefore the proper amalgam of M_1 and M_2 exists.*

Proof. Since $M_1 \restriction T$ is a free matroid, we have $r(X \cap T) = |X \cap T|$ and therefore η is submodular. We will show that this implies that ζ is submodular, that is to show that $\zeta(X_1) + \zeta(X_2) \geq \zeta(X_1 \cap X_2) + \zeta(X_1 \cup X_2)$. For $i \in \{1, 2\}$, let Y_i be a set such that $Y_i \supseteq X_i$ and $\zeta(X_i) = \eta(Y_i)$. Then $\zeta(X_1) + \zeta(X_2) = \eta(Y_1) + \eta(Y_2) \geq \eta(Y_1 \cap Y_2) + \eta(Y_1 \cup Y_2) \geq \zeta(X_1 \cap X_2) + \zeta(X_1 \cup X_2)$. By Proposition 2.1, the proper amalgam of M_1 and M_2 exists. □

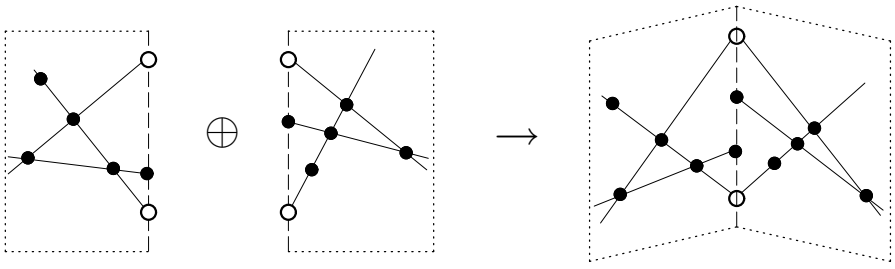


FIG. 2. *A "geometrical" illustration of an amalgam of two matroids, in which hollow points are the shared elements T.*

3. Titanic partitions. This technical section is about general symmetric submodular functions. Let V be a finite set and f be a symmetric submodular function on 2^V.

A set $\mathcal{T} \subset 2^V$ of subsets of V is called an *f-tangle* of order $k+1$ if it satisfies the following three axioms.

(T1) For all $A \subseteq V$, if $f(A) \leq k$, then either $A \in \mathcal{T}$ or $V \setminus A \in \mathcal{T}$.

(T2) If $A, B, C \in \mathcal{T}$, then $A \cup B \cup C \neq V$.

(T3) For all $v \in V$, we have $V \setminus \{v\} \notin \mathcal{T}$. Robertson and Seymour [22] showed that tangles are related to branch-width.

THEOREM 3.1 (Robertson and Seymour [22]). *There is no f-tangle of order $k+1$ if and only if the branch-width of f is at most k.*

A subset X of V is called *titanic* with respect to f if whenever A_1, A_2, A_3 are pairwise disjoint subsets of X such that $A_1 \cup A_2 \cup A_3 = X$, there is $i \in \{1, 2, 3\}$ such that $f(A_i) \geq f(X)$.

LEMMA 3.2. *Let V be a finite set and f be a symmetric submodular function on 2^V. Let X be a titanic set with respect to f. If $X_1 \cup X_2 \cup X_3 = X$, then there exists*

$i \in \{1,2,3\}$ such that $f(X_i) \geq f(X)$. (Note that X_1, X_2, X_3 are not required to be pairwise disjoint.)

Proof. Suppose not. We pick a counterexample with minimum $|X_1|+|X_2|+|X_3|$. If X_1, X_2, X_3 are pairwise disjoint, then by definition the lemma is true.

We may assume that $X_1 \cap X_2 \neq \emptyset$. Let Y_1 be a set minimizing $f(Y_1)$ subject to the condition $X_1 \setminus X_2 \subseteq Y_1 \subseteq X_1$. Then $f(X_1) \geq f(Y_1)$ and $f(X_1 \setminus X_2) \geq f(Y_1)$. Let $Y_2 = X_2 \setminus Y_1$. By the submodular inequality,

$$f(Y_1) + f(X_2) \geq f(X_1 \setminus X_2) + f(Y_2) \geq f(Y_1) + f(Y_2),$$

and therefore $f(X_2) \geq f(Y_2)$. Since $Y_1 \cup Y_2 \cup X_3 = X$, $|Y_1|+|Y_2|+|X_3| < |X_1|+|X_2|+|X_3|$, and $f(X_3) < f(X)$, we conclude that either $f(Y_1) \geq f(X)$ or $f(Y_2) \geq f(X)$. But both cases lead to the conclusion that either $f(X_1) \geq f(X)$ or $f(X_2) \geq f(X)$, a contradiction. □

A partition $\mathcal{P}$ of V is called *titanic* with respect to f if every part of $\mathcal{P}$ is titanic with respect to f. The following lemmas are equivalent to a lemma by Geelen, Gerards, and Whittle [9, Proposition 4.4], which generalizes a result of Robertson and Seymour [22, Proposition (8.3)].

LEMMA 3.3. *Let V be a finite set and f be a symmetric submodular function on 2^V of branch-width k. Let Q be a nonempty titanic set with respect to f. Let $y \in Q$. Let $\pi(x) = x$ if $x \notin Q$ and $\pi(x) = y$ if $x \in Q$. If $f(Q) \leq k$, then the branch-width of $f^{[\pi]}$ is at most k.*

Proof. Suppose that the branch-width of $f^{[\pi]}$ is larger than k. Then by Theorem 3.1, there is an $f^{[\pi]}$-tangle $\mathcal{T}^{[\pi]}$ of order $k+1$. Let $\mathcal{T}' = \{\cup_{Z \in Y} Z : Y \in \mathcal{T}^{[\pi]}\}$.

We would like to construct an f-tangle $\mathcal{T}$ of order $k+1$ as follows:

$$\mathcal{T} = \big\{X \subseteq V : f(X) \leq k \text{ and either } X \cup Q \in \mathcal{T}' \text{ or } X \setminus Q \in \mathcal{T}'\big\}.$$

To show that $\mathcal{T}$ is an f-tangle of order $k+1$, it is enough to verify the three axioms (T1)–(T3).

For (T1), suppose that $f(X) \leq k$ and $X, V \setminus X \notin \mathcal{T}$. Since Q is titanic, either $f(X \cap Q) \geq f(Q)$ or $f(Q \setminus X) \geq f(Q)$. We may assume that $f(X \cap Q) \geq f(Q)$ by replacing X with $V \setminus X$ if necessary. By the submodular inequality,

$$f(X) + f(Q) \geq f(X \cup Q) + f(X \cap Q) \geq f(X \cup Q) + f(Q),$$

and therefore $f(X \cup Q) \leq f(X) \leq k$. Since $\mathcal{T}^{[\pi]}$ is an $f^{[\pi]}$-tangle, we know that either $X \cup Q \in \mathcal{T}'$ or $V \setminus (X \cup Q) \in \mathcal{T}'$. If $X \cup Q \in \mathcal{T}'$, then $X \in \mathcal{T}$. If $V \setminus (X \cup Q) = (V \setminus X) \setminus Q \in \mathcal{T}'$, then $V \setminus X \in \mathcal{T}$. So, (T1) is proved.

To show (T2), suppose that there are $X_1, X_2, X_3 \in \mathcal{T}$ such that $X_1 \cup X_2 \cup X_3 = V$. By Lemma 3.2, there exists $i \in \{1,2,3\}$ such that $f(X_i \cap Q) \geq f(Q)$. We may assume that $i = 1$. By the submodular inequality, we deduce that $f(X_1 \cup Q) \leq f(X_1) \leq k$. Since $\mathcal{T}'$ has no three sets whose union is V, we have $X_1 \cup Q \notin \mathcal{T}'$. Therefore, $X_1 \setminus Q \in \mathcal{T}'$ and $V \setminus (X_1 \cup Q) \in \mathcal{T}'$. By (T3) of $\mathcal{T}^{[\pi]}$, we have $V \setminus Q \notin \mathcal{T}'$. Since $f(Q) \leq k$, we have $Q \in \mathcal{T}'$ by (T1) of $\mathcal{T}^{[\pi]}$. However, $(V \setminus (X_1 \cup Q)) \cup (X_1 \setminus Q) \cup Q = V$, contradictory to the fact that $\mathcal{T}^{[\pi]}$ is an $f^{[\pi]}$-tangle.

To show (T3), suppose that $X = V \setminus \{v\} \in \mathcal{T}$ for some $v \in V$. Since $V, V \setminus Q \notin \mathcal{T}'$ and $V \setminus \{v\} \notin \mathcal{T}'$ when $v \notin Q$, we deduce that $v \notin Q$ and $V \setminus \{v\} \setminus Q \in \mathcal{T}'$. We know that $\{v\}, Q \in \mathcal{T}'$. Then the union of three sets $\{v\}, Q, V \setminus \{v\} \setminus Q$ is equal to V, a contradiction.

Now we conclude that $\mathcal{T}$ is an f-tangle of order $k+1$. However, this is a contradiction to Theorem 3.1, because we assumed that the branch-width of f is k. □

LEMMA 3.4. *Let V be a finite set and f be a symmetric submodular function on 2^V of branch-width at most k. If $\mathcal{P}$ is a titanic partition of width at most k with respect to f, then the branch-width of $f^{\mathcal{P}}$ is at most k.*

Proof. Suppose that there is a counterexample. We pick a counterexample with a minimum number of parts having at least two elements. If all parts have exactly one element, then it is trivial.

Choose one member from each part of $\mathcal{P}$ and consider a function $\pi : V \to V$ that maps each element x of V to a representative of the part containing x. Then $[\pi] = \mathcal{P}$.

Let y be an element of V such that $|\pi_y| \geq 2$. Then let $\pi' : V \to V$ be a function such that

$$\pi'(x) = \begin{cases} \pi(x) & \text{if } x \notin \pi_y, \\ x & \text{if } x \in \pi_y. \end{cases}$$

By definition, $[\pi'] = \{\pi_x : x \notin \pi_y\} \cup \{\{x\} : x \in \pi_y\}$. Since the number of parts in $[\pi']$ having at least two elements is strictly smaller than that of $[\pi] = \mathcal{P}$ and $[\pi']$ is a titanic partition of width at most k, we know that the branch-width of $f^{[\pi']}$ is at most k.

Then $Q = \{\{x\} : x \in \pi_y\}$ is titanic with respect to $f^{[\pi']}$, because $\mathcal{P}$ is a titanic partition and $\pi_y \in \mathcal{P}$. In addition, $f^{[\pi']}(Q) = f(\pi_y) \leq k$. Therefore, by Lemma 3.3, the branch-width of $f^{[\pi']}$ is at most k. This contradicts the assumption that $\mathcal{P}$ is chosen as a counterexample with a minimum number of parts with more than one element. ∎

4. Replacing each part by a gadget. The purpose of this section is to show how a partitioned matroid may be "modeled" by an ordinary matroid having the same branch-width.

LEMMA 4.1. *Let M be a matroid and T be a subset of $E(M)$. If $|T|+1 > \lambda_M(T)$, then there is $e \in T$ such that one of the following is true:*

1. *$\lambda_{M/e}(X) = \lambda_M(X)$ for all $X \subseteq E(M) \setminus T$, or*
2. *$\lambda_{M\setminus e}(X) = \lambda_M(X)$ for all $X \subseteq E(M) \setminus T$.*

Proof. Let X be a subset of $E(M) \setminus T$. If there is an element $e \in T$ that is not spanned by $E(M) \setminus T$, then $\mathrm{r}_{M/e}(X) = \mathrm{r}_M(X)$. Therefore, $\lambda_{M/e}(X) = \mathrm{r}_{M/e}(X) + \mathrm{r}_{M/e}\big(E(M)\setminus(\{e\}\cup X)\big) - \mathrm{r}(M/e) + 1 = \mathrm{r}_M(X) + \mathrm{r}_M\big(E(M)\setminus X\big) - \mathrm{r}_M(\{e\}) - \big(\mathrm{r}(M) - \mathrm{r}_M(\{e\})\big) + 1 = \lambda_M(X)$.

So, we may assume that $E(M) \setminus T$ spans T. Since $|T| + 1 > \lambda_M(T) = \mathrm{r}_M(T) + 1$, T is dependent in M, and hence in the dual matroid M^* the set T is not spanned by $E(M^*) \setminus T$. We apply the previous argument to M^*. (Note that $(M \setminus e)^* = M^*/e$ and $\lambda_{M^*} \equiv \lambda_M$.) ∎

COROLLARY 4.2. *Let M be a matroid, and let T be a subset of $E(M)$. Then there exist disjoint subsets C, D of T such that $\lambda_{M/C\setminus D}(T \setminus (C \cup D)) = |T \setminus (C \cup D)| + 1$, and $\lambda_{M/C\setminus D}(X) = \lambda_M(X)$ for all $X \subseteq E(M) \setminus T$.*

We aim to transform a partitioned matroid $(M, \mathcal{P})$ to another partitioned matroid $(M^\#, \mathcal{P}^\#)$, such that they have the same branch-width and $\mathcal{P}^\#$ is a titanic partition with respect to $\lambda_{M^\#}$. To do so, we use an amalgam operation on matroids, described in section 2.

Let $(M, \mathcal{P})$ be a partitioned matroid. We may assume each part T of $\mathcal{P}$ satisfies $\lambda_M(T) = |T| + 1$ if $|T| > 1$, because otherwise we can contract or delete elements in T while preserving $\mathrm{bw}(M, \mathcal{P})$ by Corollary 4.2. This means that $M \restriction T$ is a free matroid. For each part T of $(M, \mathcal{P})$, if $|T| > 1$, then we define a matroid U_T (a *titanic*

gadget of T) as a rank-$|T|$ uniform matroid on the ground set $E_T = E(U_T)$ such that $|E_T| = 3|T| - 2$, $E(M) \cap E_T = T$, and $E_T \cap E_{T'} = \emptyset$ if $T' \neq T$ is a part of $\mathcal{P}$ and $|T'| > 1$. Since $M \restriction T = U_T \restriction T$ is a free matroid, an amalgam of M and U_T exists by Lemma 2.2.

LEMMA 4.3. *Let M be a matroid and T be a subset of $E(M)$ such that $\lambda_M(T) = |T| + 1$. Let M' be an amalgam of M and U_T. Then the following are true:*

1. *If $T \subseteq X \subseteq E(M')$, then $\mathrm{r}_M(X \cap E(M)) = \mathrm{r}_{M'}(X)$.*
2. *$\lambda_M(X) = \lambda_{M'}(X)$ for all $X \subseteq E(M) \setminus T$.*
3. *The set $E(U_T)$ is titanic in the matroid M'.*

Proof. 1. Because $M' \restriction E(M) = M$, we have $\mathrm{r}_M(X \cap E(M)) = \mathrm{r}_{M'}(X \cap E(M)) \leq \mathrm{r}_{M'}(X)$.

Since T is independent in U_T, we can pick a maximally independent subset I of X in M' such that $T \subseteq I$. Since $M' \restriction E(U_T) = U_T$, the set $I \cap E(U_T)$ is independent in U_T, and therefore $I \cap E(U_T) = T$. So, $I \subseteq E(M)$. Therefore, $\mathrm{r}_M(X \cap E(M)) \geq |I| = \mathrm{r}_{M'}(X)$.

2. Let $Y = E(M') \setminus X$. We note that $E(U_T)$ is a subset of Y. By definition,

$$\lambda_M(X) = \mathrm{r}_M(X) + \mathrm{r}_M(Y \cap E(M)) - \mathrm{r}(M) + 1,$$
$$\lambda_{M'}(X) = \mathrm{r}_{M'}(X) + \mathrm{r}_{M'}(Y) - \mathrm{r}(M') + 1.$$

Since $M' \restriction E(M) = M$, we have $\mathrm{r}_M(X) = \mathrm{r}_{M'}(X)$. By 1, $\mathrm{r}_M(Y \cap E(M)) = \mathrm{r}_{M'}(Y)$ and $\mathrm{r}(M') = \mathrm{r}(M)$. Thus $\lambda_M(X) = \lambda_{M'}(X)$.

3. We claim that if X is a subset of $E(U_T)$ and $|X| \geq |T|$, then $\lambda_{M'}(X) \geq \lambda_{M'}(E(U_T))$. Since U_T is a uniform matroid of rank $|T|$, we have

$$\mathrm{r}_{M'}(X) = |T| = \mathrm{r}_{M'}(E(U_T)),$$
$$\mathrm{r}_{M'}(E(M') \setminus X) \geq \mathrm{r}_{M'}(E(M') \setminus E(U_T)).$$

Therefore, $\lambda_{M'}(X) \geq \lambda_{M'}(E(U_T))$.

Now suppose that X_1, X_2, X_3 are pairwise disjoint subsets of $E(U_T)$. Then there is $i \in \{1, 2, 3\}$ such that $|X_i| \geq \lceil |E(U_T)|/3 \rceil = |T|$ and therefore $\lambda_{M'}(X_i) \geq \lambda_{M'}(E(U_T))$. Therefore, $E(U_T)$ is titanic in M'. □

Using Corollary 4.2, we first obtain a minor M_0 of M such that $\lambda_{M_0}(T \cap E(M_0)) = |T \cap E(M_0)| + 1$ for all parts $T \in \mathcal{P}$, and if a subset X of $E(M)$ satisfies that $X \cap T \in \{\emptyset, T\}$ for all parts $T \in \mathcal{P}$, then $\lambda_{M_0}(X \cap E(M_0)) = \lambda_M(X)$. Let $\mathcal{P}_0$ be the partition of $E(M_0)$ induced by $\mathcal{P}$. Then we deduce from Corollary 4.2 that the branch-width of $(M, \mathcal{P})$ is equal to the branch-width of $(M_0, \mathcal{P}_0)$. However, the branch-width of the matroid M_0 may still be different from the branch-width of the partitioned matroid $(M_0, \mathcal{P}_0)$.

In the following theorem, we will extend $(M_0, \mathcal{P}_0)$ by amalgamating uniform matroids in the fashion of Lemma 4.3 so that the obtained partitioned matroid $(M^\#, \mathcal{P}^\#)$ has the same branch-width as the matroid $M^\#$ itself.

THEOREM 4.4. *Let $(M_0, \mathcal{P}_0)$ be a partitioned matroid, and let $T_1, T_2, \ldots, T_m$ be the parts of $\mathcal{P}_0$ having at least two elements. Assume that $\lambda_{M_0}(T_i) = |T_i| + 1$ for every $i \in \{1, 2, \ldots, m\}$. For all $i = 1, 2, \ldots, m$, let M_i be an amalgam of M_{i-1} and U_{T_i}. Then the branch-width of M_m is equal to the branch-width of the partitioned matroid $(M_0, \mathcal{P}_0)$.*

We call the resulting $M^\# = M_m$ the *normalized matroid* of the partitioned matroid $(M_0, \mathcal{P}_0)$.

Proof. Let $\mathcal{P}_i = (\mathcal{P}_{i-1} \setminus \{T_i\}) \cup \{E(U_{T_i})\}$. By 2. of Lemma 4.3, the branch-width of $(M_i, \mathcal{P}_i)$ is equal to that of $(M_{i-1}, \mathcal{P}_{i-1})$, and therefore the branch-width

of $(M_m, \mathcal{P}_m)$ is equal to the branch-width of $(M_0, \mathcal{P}_0)$. By 3. of Lemma 4.3, $\mathcal{P}_m$ is a titanic partition. Let k be the branch-width of M_m. It is easy to see that the branch-width of the uniform matroid U_{T_i} is $|T_i| + 1 = \lambda_{M_m}(E(U_{T_i}))$. Since U_{T_i} is a minor of M_m, the branch-width of M_m is at least $|T_i| + 1$ for all i, and therefore the width of $\mathcal{P}_m$ is at most k. We conclude that the branch-width of $(M_m, \mathcal{P}_m)$ is at most k by Lemma 3.4.

To finish the proof, we need to show that the branch-width of $(M_m, \mathcal{P}_m)$ is at least k. Let (T, μ) be the branch-decomposition of $(M_m, \mathcal{P}_m)$ of width at most w. From (T, μ), we would like to obtain a branch-decomposition (T', μ') of M_m whose width is at most w as follows. Let v_i be the leaf of T corresponding to $E(U_{T_i})$. We prepare a rooted binary tree with a bijection from its leaves to $E(U_{T_i})$ and then identify the root with v_i. Let T' be the new tree obtained by the above process for all i. A bijection μ' from $E(M_m)$ to leaves of T' is easily obtained from the above process. Since $\lambda_{M_m}(X) = |X| + 1 \leq \lambda_{M_m}(E(U_{T_i})) \leq w$ for all $X \subseteq E(U_{T_i})$, the width of (T', μ') is at most w. □

5. Branch-decompositions of represented partitioned matroids. We now specialize the above ideas to the case of representable matroids. We aim to provide an efficient algorithm for testing small branch-width on such partitioned matroids. For the rest of our paper, a *represented matroid* is the vector matroid of a (given) matrix over a *fixed* finite field. We also write an $\mathbb{F}$-*represented* matroid to explicitly refer to the field $\mathbb{F}$. In other words, an $\mathbb{F}$-represented matroid is a set of points (a *point configuration*) in a (finite) projective geometry over $\mathbb{F}$. To begin, we restate a previous result of Hliněný.

THEOREM 5.1 (see [13, Theorems 4.14 and 5.5]). *Let $k > 1$ be a constant, and let $\mathbb{F}$ be a fixed finite field. There is a parametrized algorithm that, for a given matroid M represented by an $r \times n$ matrix over $\mathbb{F}$ for some $r \leq n$, tests whether the branch-width of M is at most k in time $O(n^3)$.*

We remark that the algorithm for Theorem 5.1 in [13] is purely of theoretical importance. First, it uses the result of Geelen et al. [8] stating that minor-minimal matroids of branch-width larger than k have size at most $(6^k - 1)/5$. Second, it tests whether the input matroid contains such a minor-minimal matroid as a minor by encoding the question in a monadic second-order logic formula on matroids and using a generic algorithm to solve an MSOL formula for $\mathbb{F}$-represented matroids of branch-width at most k. Since our algorithm will use Theorem 5.1, it will be purely theoretical and difficult to implement.

Not all matroids are representable over $\mathbb{F}$. Particularly, in the construction of the normalized matroid (Theorem 4.4) we apply amalgams with uniform matroids which need not be $\mathbb{F}$-representable. To achieve their representability, we extend the field $\mathbb{F}$ in a standard algebraic way.

Remark 5.2. Let $\mathbb{F}$ be a fixed finite field with q elements and d be a fixed positive integer. We assume that one can perform arithmetic operations over $\mathbb{F}$ in time depending only on q. Then, one can construct by brute force an extension (finite) field $\mathbb{F}' = \mathbb{F}(\alpha)$ with q^d elements by searching for a polynomial root α of degree d over $\mathbb{F}$. This can be done by searching through all polynomials in $\mathbb{F}[x]$ of degree d for the irreducible ones.

LEMMA 5.3. *The n-element rank-r uniform matroid $U_{r,n}$ is representable over a (finite) field $\mathbb{F}$ if $|\mathbb{F}| \geq n - 1$.*

Proof. Let $|\mathbb{F}| = q$. It is trivial to represent $U_{0,n}, U_{1,n}, U_{n-1,n}$, or $U_{n,n}$ over every field. Furthermore, standard arguments of projective geometry show that a

so-called normal rational curve in a projective geometry over $\mathbb{F}$ is a representation of the uniform matroid $U_{r,\,q+1}$, for every $1 < r < q$; see, for instance, [12, section 3]. Although it is not useful in our context, it is worth noting that the size bound $q+1$ is almost optimal in most cases. Finally, if $q+1 > n$, then we delete arbitrary $q+1-n$ points from the representation to get $U_{r,n}$. □

Recall the notion of a matroid amalgam from section 2 from the perspective of represented matroids. We shall use the following proposition.

PROPOSITION 5.4. *Let M_1, M_2 be two matroids such that $E(M_1) \cap E(M_2) = T$ and $M_1 \restriction T = M_2 \restriction T$. If both M_1, M_2 are $\mathbb{F}$-represented, and the matroid $M_1 \restriction T$ has a unique $\mathbb{F}$-representation up to linear transformations, then there exists an amalgam of M_1 and M_2 which is also $\mathbb{F}$-represented.*

Proof. We denote by $[\boldsymbol{A}_1 \mid \boldsymbol{A}_T]$ the matrix over $\mathbb{F}$ representing M_1, where the columns of $\boldsymbol{A}_T$ represent the set T. Analogously we denote by $[\boldsymbol{A}_2 \mid \boldsymbol{A}'_T]$ the matrix representing M_2. Since $M_1 \restriction T = M_2 \restriction T$ has a unique $\mathbb{F}$-representation, there is a linear transformation carrying $\boldsymbol{A}'_T$ onto $\boldsymbol{A}_T$. This transformation takes a whole $[\boldsymbol{A}_2 \mid \boldsymbol{A}'_T]$ to an equivalent matrix $[\boldsymbol{A}'_2 \mid \boldsymbol{A}_T]$ representing the same matroid M_2. It is now trivial to verify that the matroid M which is $\mathbb{F}$-represented by the composed matrix $[\boldsymbol{A}_1 \mid \boldsymbol{A}_T \mid \boldsymbol{A}'_2]$ is an amalgam of M_1 and M_2. □

We remark that representable matroids typically do have inequivalent vector representations, but we are using Proposition 5.4 only in the case when $M_1 \restriction T$ is a free matroid which clearly has a unique $\mathbb{F}$-representation for every field $\mathbb{F}$.

We now outline a simple fixed-parameter tractable algorithm for testing branch-width $\leq k$ on $\mathbb{F}$-represented partitioned matroids.

ALGORITHM 5.5. Testing branch-width of a represented partitioned matroid.

Parameters: A finite field $\mathbb{F}$, and a positive integer k.

Input: A rank-r matrix $\boldsymbol{A} \in \mathbb{F}^{r \times n}$ and a partition $\mathcal{P}$ of the columns of $\boldsymbol{A}$. (Assume $n \geq 2$.)

Output: For the vector matroid $M = M(\boldsymbol{A})$ on the columns of $\boldsymbol{A}$ partitioned by $\mathcal{P}$, a correct answer whether the branch-width of $(M, \mathcal{P})$ is at most k.

1. First, we extend $\mathbb{F}$ to a nearest field $\mathbb{F}'$ such that $|\mathbb{F}'| \geq 3k-6$ (Remark 5.2).
2. If the width of the partition $\mathcal{P}$ in given $(M, \mathcal{P})$ is more than k, then we answer NO.
3. Otherwise, we directly construct the normalized matroid $M^{\#}$ (Theorem 4.4), together with its vector representation over $\mathbb{F}'$ (Lemma 5.3 and Proposition 5.4).
4. Finally, we use Theorem 5.1 to test whether the branch-width of $M^{\#}$ is at most k.

Hence, we can conclude with the following theorem.

THEOREM 5.6. *Let $k > 1$ be a constant, and let $\mathbb{F}$ be a fixed finite field. There is a parametrized algorithm that, for a partitioned matroid $(M, \mathcal{P})$ represented over $\mathbb{F}$, tests in time $O(|E(M)|^3)$ whether the branch-width of $(M, \mathcal{P})$ is at most k.*

Proof. We refer to Algorithm 5.5. Denote $n = |E(M)|$. In the first step we find the extension field $\mathbb{F}'$ which takes only constant time by Remark 5.2. Since $\mathbb{F}' \supseteq \mathbb{F}$, we do not need to touch the input vector representation of M. Step 2. of Algorithm 5.5 is trivial.

The technical part comes in step 3. of Algorithm 5.5. For each part $X \in \mathcal{P}$ of more than one element, we compute the intersection of the spans of X and of $E(M) \setminus X$, called the *guts* of the separation $(X, E(M) \setminus X)$, in the representation of M. The reader should understand that we deal with represented matroids, that means we

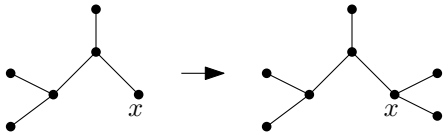


FIG. 3. *Splitting x.*

compute with actual vectors and subspaces in a projective geometry over $\mathbb{F}'$. If the dimension $\lambda_M(X)-1$ of these guts was at least k, then each branch-decomposition of $(M,\mathcal{P})$ should have width at least $\lambda_M(X) > k$, and we have answered NO in step 2. of Algorithm 5.5 in such a case. Therefore, the dimension of the guts of $(X, E(M)\setminus X)$ is bounded by a constant k, and a set T of its independent generator vectors can be easily computed in time $O(n^2)$ (see, e.g., [13, Algorithm 4.4]), per each part of $\mathcal{P}$.

In order to get the situation anticipated in Theorem 4.4—each part representing a free matroid—we replace the vectors of each nonsingleton part $X \in \mathcal{P}$ by the respective vectors T computed in the previous paragraph. Let the resulting represented partitioned matroid be denoted by $(M_0,\mathcal{P}_0)$, and note that $\mathrm{bw}(M_0,\mathcal{P}_0) = \mathrm{bw}(M,\mathcal{P})$. For each $T \in \mathcal{P}_0$ we construct an $\mathbb{F}'$-representation of the titanic gadget (uniform matroid) U_T from section 4 using Lemma 5.3, and then construct an amalgam of M_0 with U_T according to Proposition 5.4. Since U_T is of bounded size, this last step can be computed in time proportional to the vector length $O(n)$, per each part of $\mathcal{P}_0$.

After processing all $O(n)$ nonsingleton parts in $\mathcal{P}_0$ by the previous procedure, we get a vector $\mathbb{F}'$-representation of the normalized matroid $M^\#$ for $(M_0,\mathcal{P}_0)$. By Theorem 4.4, $\mathrm{bw}(M^\#) = \mathrm{bw}(M_0,\mathcal{P}_0) = \mathrm{bw}(M,\mathcal{P})$. So, we call the algorithm of Theorem 5.1 to determine whether the branch-width of $M^\#$ (as well as the branch-width of $(M,\mathcal{P})$) is at most k. This takes only $O(n^3)$ time since both k, $\mathbb{F}'$ are of bounded size here. □

We are now able to test branch-width of partitioned matroids. We show how this result can be extended to finding an appropriate branch-decomposition.

THEOREM 5.7. *Let $\mathcal{K}$ be a class of matroids, and let k be an integer. If there is an $f(|E(M)|,k)$-time algorithm to decide whether a partitioned matroid $(M,\mathcal{P})$ has branch-width at most k for every pair of a matroid $M \in \mathcal{K}$ and a partition $\mathcal{P}$ of $E(M)$, then a branch-decomposition of the partitioned matroid $(M,\mathcal{P})$ of width at most k, if it exists, can be found in time $O\big(|\mathcal{P}|^3 \cdot f(|E(M)|,k)\big)$.*

The idea of the proof is due to Jim Geelen, published by Oum and Seymour in [20]. Details of the algorithm follow. Clearly, we may assume that the branch-width of the partitioned matroid $(M,\mathcal{P})$ in question is at most k, since otherwise there is nothing to compute. A *splitting* of a leaf x of a subcubic tree is an operation that creates two new leaves and makes them adjacent to x (see Figure 3).

ALGORITHM 5.8. Computing a branch-decomposition of a partitioned matroid.
Oracle: A subroutine for testing the branch-width of a partitioned matroid $(M,\mathcal{P})$, where M belongs to a given class $\mathcal{K}$, and $\mathcal{P}$ is a partition of $E(M)$.
Input: A partitioned matroid $(M,\mathcal{P})$ of branch-width at most k, where $M \in \mathcal{K}$.
Output: A branch-decomposition of $(M,\mathcal{P})$ of width at most k.

1. If $|\mathcal{P}| \le 2$, then it is trivial to output a branch-decomposition.
2. We find a pair X,Y of disjoint parts of $\mathcal{P}$ such that a partitioned matroid $(M,(\mathcal{P}\setminus\{X,Y\})\cup\{X\cup Y\})$ has branch-width at most k. Let $\mathcal{P}' = (\mathcal{P}\setminus\{X,Y\})\cup\{X\cup Y\}$.
3. Let (T',μ') be the branch-decomposition of $(M,\mathcal{P}')$ of width at most k ob-

tained by calling this algorithm recursively.

4. Let T be a tree obtained from T' by splitting the leaf $\mu'(X \cup Y)$ into two leaves which we denote by $\mu(X)$ and $\mu(Y)$. Let $\mu(Z) = \mu'(Z)$ for all $Z \in \mathcal{P} \setminus \{X, Y\}$. We output (T, μ) as the resulting branch-decomposition of (M, P) of width at most k.

Proof of Theorem 5.7. We start by estimating the running time of the algorithm. At each level of recursion, we call our oracle (the decision algorithm) at most $\binom{|\mathcal{P}|}{2} = O(|\mathcal{P}|^2)$ times. The depth of recursion is $|\mathcal{P}| - 1$, and therefore the number of calls to the decision algorithm is at most $O(|\mathcal{P}|^3)$. Thus, the running time of the algorithm is $O\big(|\mathcal{P}|^3 \cdot f(|E(M)|, k)\big)$.

It remains to show correctness of the algorithm. It is obvious that in every subcubic tree with at least three leaves, there are two leaves that have a common neighbor. Suppose that (T, μ) is a branch-decomposition of $(M, \mathcal{P})$ of width at most k. Then there are two leaves $\mu(X)$ and $\mu(Y)$ having a common neighbor z in T. It is easy to see that if we remove $\mu(X)$ and $\mu(Y)$ from T and map $X \cup Y$ to z by μ, then (T, μ) is a branch-decomposition of $(M, \mathcal{P}')$ of width at most k. Therefore, the branch-width of $(M, \mathcal{P}')$ is at most k.

Conversely, suppose that (T', μ') is the branch-decomposition of $(M, \mathcal{P}')$. Since $(M, \mathcal{P})$ has a branch-width of at most k, we know that $\lambda_M(X) \leq k$ and $\lambda_M(Y) \leq k$. Thus (T, μ) is a branch-decomposition of $(M, \mathcal{P})$ of width at most k. □

COROLLARY 5.9. *For a constant k and a fixed finite field $\mathbb{F}$, we can find a branch-decomposition of a given $\mathbb{F}$-represented matroid M of branch-width at most k, if it exists, in time $O(|E(M)|^6)$.*

Proof. Let $\mathcal{P} = \{\{x\} : x \in E(M)\}$. Then the branch-decomposition of M is one-to-one correspondent to the branch-decomposition of a partitioned matroid $(M, \mathcal{P})$. By Theorem 5.6, the problem of deciding whether branch-width is at most k can be done in time $O(|E(M)|^3)$, and therefore we can construct the branch-decomposition of width at most k in time $O(|\mathcal{P}|^3 \cdot |E(M)|^3) = O(|E(M)|^6)$ by Theorem 5.7. □

Remark 5.10. One can actually improve the bound in Theorem 5.7 to $O\big(|\mathcal{P}|^2 \cdot f(|E(M)|, k)\big)$ time. The basic idea is the following: At the first level of recursion we find not only one pair of parts but a maximal set of disjoint pairs of parts from $\mathcal{P}$ that can be joined (pairwise) while keeping the branch-width at most k. This again requires $O(|\mathcal{P}|^2)$ calls to the decision algorithm. At the deeper levels of recursion we then use the same approach but process only such pairs of parts that contain one joined at the previous level. An amortized complexity analysis shows that only additional $O(|\mathcal{P}|^2)$ calls to the decision algorithm are necessary at all subsequent levels of recursion together. Detailed arguments of this approach can be found further in Theorem 6.7, part (III) of the proof.

6. Faster algorithm for branch-decompositions. Even with Remark 5.10 in account, the approach of section 5 results in an $O(n^5)$ parametrized algorithm for constructing a branch-decomposition of an n-element matroid represented over a fixed finite field. That is still far from the cubic running time of the decision algorithm in Theorem 5.1. Although not straightforwardly, we are able to improve the running time of our constructive algorithm to asymptotically match cubic time of that and [5].

It is the purpose of this section to present a detailed analysis of such a faster implementation of Algorithm 5.8 using Remark 5.10. For that we have to dive into fine details of the ideas and algorithms in [13]. To be consistent, we also adopt the writing style of [13] for this section and recall a few necessary technical definitions

here. More technical details are given along with a formal setting of Algorithm 6.6. We first give a brief informal outline of our faster algorithm, which seems necessary since Algorithm 6.6 itself is quite long and complex. We also collect formal statements of useful subroutines from [13], and then we conclude with the main algorithm (Algorithm 6.6) and a proof of its correctness.

One key point in the approach of [13, 14] is that a matroid M, which is $\mathbb{F}$-represented by a matrix $\boldsymbol{A}$, is equivalent to a projective point configuration over $\mathbb{F}$, and therefore we can speak about M in schematic geometric terms. Briefly speaking, a *parse tree* [14] of an $\mathbb{F}$-represented matroid M is a rooted tree $\mathcal{T}$, with at most two children per node, such that the leaves of $\mathcal{T}$ hold nonloop elements of M represented by points of a projective geometry over $\mathbb{F}$, or loops of M represented by the empty set. The internal nodes of $\mathcal{T}$, on the other hand, hold suitable "composition operators over $\mathbb{F}$."

Such a *composition operator* $\odot$ is a configuration in the projective geometry over $\mathbb{F}$ such that $\odot$ has three subspaces (possibly empty) distinguished as its *boundaries*; two of which are used to "glue" the matroid elements represented in the left and right subtrees, respectively, together. The third one, *upper boundary*, is then used to "glue" this node further up in the parse tree $\mathcal{T}$. Our "glue" operation, called the boundary sum by Hliněný [14], is analogous to the amalgam of matroids in Proposition 5.4. The ranks of adjacent boundaries of two composition operators in $\mathcal{T}$ must be equal for "gluing." A parse tree $\mathcal{T}$ is $\leq t$*-boundaried* if all composition operators in $\mathcal{T}$ have boundaries of rank at most t. Such a parse tree actually gives a branch-decomposition of width at most $t+1$ and vice versa, by [14, Theorem 3.8]. See [14] for additional details.

We will use the following algorithm shown by Hliněný [13].

ALGORITHM 6.1 (see [13, Algorithm 4.1]). Computing a parse tree of a represented matroid.

Parameters: A finite field $\mathbb{F}$, and a positive integer k.
Input: A rank-r matrix $\boldsymbol{A} \in \mathbb{F}^{r\times n}$ representing a matroid M over $\mathbb{F}$. (Assume $n \geq 2$.)
Output: Computed in time $O(n^3)$; either a $\leq 3k$-boundaried parse tree $\mathcal{T}$ of the matroid M, or a proof that the branch-width of M is more than $k+1$.

The basic idea of Algorithm 6.1 is as follows: We start with a basis $\boldsymbol{I}_r$ of the input matrix $\boldsymbol{A} = [\boldsymbol{I}_r \mid \boldsymbol{A}'] \in \mathbb{F}^{r\times n}$, and assign an arbitrary parse tree $\mathcal{T}$ to $\boldsymbol{I}_r$. Then we are adding, one by one, the remaining elements of $\boldsymbol{A}'$ arbitrarily to $\mathcal{T}$. Whenever the largest boundary rank (the *width*) of $\mathcal{T}$ exceeds certain constant threshold, say $10k$, we "compress" $\mathcal{T}$ into a new parse tree $\mathcal{T}'$ of width at most $3k$ again. However, if the compression step fails, then we have a certificate that the branch-width of $M(\boldsymbol{A})$ is more than $k+1$. The compression routine, [13, Algorithm 4.1, step 3] and [13, Lemma 4.13], is crucial also in our new algorithm, and thus we restate it explicitly here.

ALGORITHM 6.2 (see [13, Algorithm 4.1]). "Compressing" a parse tree of bounded width.

Parameters: A finite field $\mathbb{F}$, and a positive integer k.
Input: A $\leq ck$-boundaried parse tree $\mathcal{T}$ (of an n-element matroid M represented over $\mathbb{F}$), where $c > 3$ is a fixed constant, say $c = 10$.
Output: Computed in time $O(n^2)$; either a $\leq 3k$-boundaried parse tree $\mathcal{T}'$ of the same matroid M, or a proof that the branch-width of M is more than $k+1$.

Outline of the faster algorithm. Before giving full details of our new Algorithm 6.6 for computing a branch-decomposition of a represented partitioned matroid, we sketch

its main ideas with respect to the previous Algorithms 6.2 and 6.1.

Parameters: A finite field $\mathbb{F}$, and a positive integer k.

Input: A rank-r matrix $\boldsymbol{A} \in \mathbb{F}^{r\times n}$ and a partition $\mathcal{P}$ of the columns of $\boldsymbol{A}$. (Assume $n \geq 2$.)

Initial phase. Let $M = M(\boldsymbol{A})$ be the vector matroid on the columns of $\boldsymbol{A}$. We run Algorithm 5.5 to obtain the represented normalized matroid $M^{\#}$ for our M and $\mathcal{P}$, and to decide whether $\mathrm{bw}(M^{\#}) \leq k$. In the positive case, we also call Algorithm 6.1 to obtain a $\leq 3(k-1)$-boundaried parse tree $\mathcal{T}$ for the matroid $M^{\#}$.

Construction phase. We construct a branch-decomposition of $(M, \mathcal{P})$ as a rooted forest D which is initialized to the set of disconnected nodes $\mathcal{P}_1 := \mathcal{P}$. A *rooted forest* is a forest in which every connected component has a specified vertex called a *root*.

In the first iteration, we find an inclusion-wise maximal collection of pairwise disjoint pairs $\{X_i, Y_i\}$, $i = 1, 2, \ldots, c$, of parts of $\mathcal{P}_1$ such that the branch-width of $(M, \mathcal{P}_1')$ is at most k, where $\mathcal{P}_1'$ is obtained from $\mathcal{P}_1$ via replacing parts X_i, Y_i with $X_i \cup Y_i$ for $i = 1, 2, \ldots, c$. The meaning is that these pairs $\{X_i, Y_i\}$ simultaneously correspond to pairs of leaves of distance two in some branch-decomposition of width $\leq k$. We let $\mathcal{Q}_1 = \{X_i \cup Y_i : i = 1, 2, \ldots, c\}$, and add the new nodes from $\mathcal{Q}_1$ to our forest D connected to the appropriate X_i, Y_i's. Then we set $\mathcal{P}_1 := \mathcal{P}_1'$.

In each of the subsequent iterations, we again find an inclusion-wise maximal collection of pairwise disjoint pairs $\{X_i, Y_i\}$, $i = 1, 2, \ldots, c$, of parts of $\mathcal{P}_1$ such that the branch-width of $(M, \mathcal{P}_1')$ is at most k, but now we *restrict* $Y_i \in \mathcal{Q}_1$ (whereas $X_i \in \mathcal{P}_1$). Then we continue analogously to the first iteration, until D becomes a tree.

Output: Either a branch-decomposition D of $(M, \mathcal{P})$ of width at most k, or the answer NO if $\mathrm{bw}(M, \mathcal{P}) > k$.

There are two important points to notice in the above outline, which make the whole algorithm run in time $O(n^3)$. First, we only consider altogether $O(n^2)$ pairs $\{X_i, Y_i\}$ of parts for possible merging, throughout all iterations of the algorithm. A formal proof of this fact in included as part (III) of the proof of Theorem 6.7. Second, to be able to run a quick test whether the branch-width of $(M, \mathcal{P}_1')$ exceeds k or not, we need to maintain a certain "working" parse tree $\mathcal{T}_1$ of bounded width. Then, as noted already after Theorem 5.1, such a test can be done by looking for the excluded minors for branch-width at most k because each excluded minor has size at most $(6^k - 1)/5$, shown by Geelen et al. [8].

THEOREM 6.3 (see [14, Theorem 6.1] and [13, Corollary 5.4]). *Let $t > 1$ be a constant, and let $\mathbb{F}$ be a fixed finite field. There is a parametrized algorithm that, for every $k \leq t$ and for a given $\leq t$-boundaried parse tree $\mathcal{T}$ of an n-element matroid M, decides whether the branch-width of M is at most k in time $O(n)$.*

We have skipped, for simplicity, an explicit reference to the "working" parse tree $\mathcal{T}_1$ in the above outline; however, one can roughly say that $\mathcal{T}_1$ is maintained as a parse tree of the normalization of the current partitioned matroid $(M, \mathcal{P}_1)$. This will be precise in Algorithm 6.6. It is essential that we keep the width of $\mathcal{T}_1$ bounded throughout the computation, for which we use to call Algorithm 6.2 after each of the $O(n)$ major updates to $\mathcal{T}_1$.

Therefore, to quickly test whether merging a pair of parts $X_i, Y_i \in \mathcal{P}_1$ increases the branch-width of $(M, \mathcal{P}_1)$ above k or not, we temporarily modify the parse tree $\mathcal{T}_1$

each time by replacing $W = X_i \cup Y_i$ with the titanic gadget (amalgamated according to section 4). As this (Algorithm 6.4) does not increase the width of $\mathcal{T}_1$ much, we then solve the task in time $O(n)$ using Theorem 6.3.

To make a precise statement of this procedure, we introduce an additional technical definition inspired by section 4: Let M be a matroid and $X \subseteq E(M)$. Let $F = E(M) \setminus X$ and Y be disjoint from $E(M)$. Assume M' is a matroid on $E(M) \cup Y$ such that $M' \restriction F = M \restriction F$, $\mathrm{r}_{M'}(X \cup Y) = \mathrm{r}_M(X)$ (i.e., Y is spanned by X), and $\lambda_{M' \setminus X}(Y) = |Y| + 1 = \lambda_M(X) > 1$. If a matroid N is an amalgam of $M' \setminus X$ and the titanic gadget U_Y, then we say that N is obtained from M by a (titanic) *normalization* of the set X. If, on the other hand, $\lambda_M(X) = 1$, then a normalization of the set X in M results in $M \setminus X$. The point is that, by Lemmas 3.3 and 4.3, part 3., the branch-width of N equals the branch-width of $(M, \mathcal{P}_X)$, where $\mathcal{P}_X = \big\{\{X\}\big\} \cup \big\{\{y\} : y \in E(M) \setminus X\big\}$.

Algorithm 6.4. Computing a titanic normalization of a point set on the parse tree.

Parameters: A finite field $\mathbb{F}$, and an integer $k \geq 1$. (We may assume that $|\mathbb{F}| \geq 3k-6$ as in Remark 5.2.)

Input: A $\leq(3k-1)$-boundaried parse tree $\mathcal{T}_1$ representing a matroid M_1 with n elements, and a set $W \subseteq E(M_1)$ such that $\lambda_{M_1}(W) = \ell \leq k$.

Output: A $\leq(3k+\ell-2)$-boundaried parse tree $\mathcal{T}_2$ of an $\mathbb{F}$-represented matroid M_2 such that M_2 can be obtained from M_1 by the normalization of W.

Algorithm 6.4 is an immediate extension of [13, Algorithm 4.9] for computing $\lambda_{M_1}(W)$. We describe it in terms of a projective geometry and the point configuration representing a matroid M_1 via the parse tree $\mathcal{T}_1$. If $\ell = 1$, then we return $\mathcal{T}_1$ without W, immediately.

At the beginning we make $\mathcal{T}_2$ a copy of $\mathcal{T}_1$. The idea is to "enlarge" all of the composition operators in $\mathcal{T}_2$ to fully contain the guts Γ (a projective subspace of rank $\ell-1$ with a basis Y) of the separation $\big(W, E(M_1) \setminus W\big)$, and then to "glue" or amalgamate a decomposition of the titanic gadget $U_Y \simeq U_{\ell-1,3\ell-5}$ to the root of $\mathcal{T}_2$ so that it matches Y in Γ. For that we apply leaf-to-root dynamic programming on $\mathcal{T}_2$ with constant-time operations at each node.

At a node $x \in V(\mathcal{T}_2)$, we compute the subspace Σ_x of Γ spanned by the elements of W held in the leaves below x. Knowing $\Sigma_{x'}$ and $\Sigma_{x''}$ for the children x', x'' of x in $\mathcal{T}_2$, it is a constant-time manipulation to determine Σ_x using the composition operator $\odot$ at x. Notice that as our algorithm is set up, Σ_x is spanned by $\odot$. If the upper boundary of $\odot$ does not fully contain Σ_x, we enlarge it accordingly and also freely extend the matching boundary at the parent node of x. Note that $\Sigma_r = \Gamma$ will become the upper boundary of the root node r.

After finishing that computation, we take an arbitrary parse tree $\mathcal{T}_3$ of the titanic gadget (i.e., uniform matroid) $U_Y \simeq U_{\ell-1,3\ell-5}$, and add to $\mathcal{T}_2$ a new root node r' adjacent to the former root r of $\mathcal{T}_2$ and to the root of $\mathcal{T}_3$. The composition operator at r' "glues" U_Y directly to Σ_r. Finally, we strip from $\mathcal{T}_2$ all leaves holding the points of W. This is trivial since our definition of a parse tree allows nodes with only one descendant.

Since we use only constant-time operations at each node of $\mathcal{T}_2$, we conclude with the following lemma.

Lemma 6.5. *Algorithm* 6.4 *computes correctly in time* $O(n)$.

We are now ready to restate the above algorithmic outline in a formal setting. Our notation of variables in Algorithm 6.6 essentially follows the outline, but we need

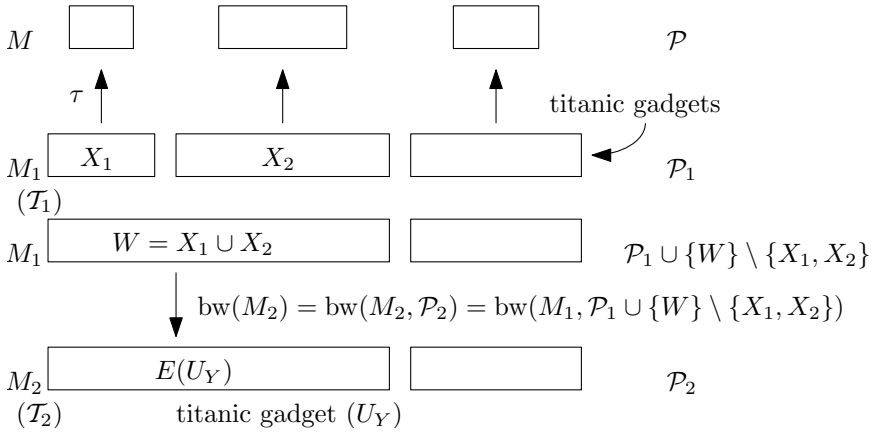


FIG. 4. *An illustration of Algorithm* 6.6.

a few more of them. For instance, $\mathcal{Q}_2$ at each round holds the set of all pairs of parts among which we are looking for the admissible ones. See also an informal hint in Figure 4.

ALGORITHM 6.6. Computing a branch-decomposition of a represented partitioned matroid.

Parameters: A finite field $\mathbb{F}$, and a positive integer k.

Input: A rank-r matrix $\boldsymbol{A} \in \mathbb{F}^{r \times n}$ and a partition $\mathcal{P}$ of the columns of $\boldsymbol{A}$. (Assume $n \geq 2$.)

Output: For the vector matroid $M = M(\boldsymbol{A})$ on the columns of $\boldsymbol{A}$, either a branch-decomposition of the partitioned matroid $(M, \mathcal{P})$ of width at most k, or the answer NO if $\mathrm{bw}(M, \mathcal{P}) > k$.

1. Using brute force, we extend the field $\mathbb{F}$ to a (nearest) finite field $\mathbb{F}'$ such that $|\mathbb{F}'| \geq 3k - 6$ (Remark 5.2 and Lemma 5.3).
2. We check whether $\mathrm{bw}(M, \mathcal{P}) \leq k$, using Algorithm 5.5. If $\mathrm{bw}(M, \mathcal{P}) > k$, then we answer NO. Otherwise we keep the *normalized matroid* $M^{\#}$ and its $\mathbb{F}'$-representation $\boldsymbol{A}^{\#}$ obtained at this step. We denote by $\mathcal{P}_1$ the (titanic) partition of $E(M^{\#})$ corresponding to $\mathcal{P}$, and by $\tau(X) \in \mathcal{P}$ for $X \in \mathcal{P}_1$ the corresponding parts.
3. Calling Algorithm 6.1, we compute a $\leq 3(k-1)$-boundaried parse tree $\mathcal{T}$ for the matroid $M^{\#}$ which is $\mathbb{F}'$-represented by $\boldsymbol{A}^{\#}$ (regardless of $\mathcal{P}_1$).
4. We initially set $\mathcal{T}_1 := \mathcal{T}$, $\mathcal{Q}_1 := \emptyset$, $\mathcal{Q}_2 := \big\{\{X_1, X_2\} : X_1 \neq X_2, X_1, X_2 \in \mathcal{P}_1\big\}$, and create a new rooted forest D consisting so far of the set of disconnected nodes $\mathcal{P}_1$.
   Let M_1 (M_2) denote the matroid represented by $\mathcal{T}_1$ ($\mathcal{T}_2$, respectively) at each step. Then we repeat the following steps (a), (b), until $\mathcal{P}_1$ contains at most two parts:
   (a) While there is $\{X_1, X_2\} \in \mathcal{Q}_2$ such that $X_1, X_2 \in \mathcal{P}_1$, we perform the following steps:
      i. Let $\mathcal{Q}_2 := \mathcal{Q}_2 \setminus \big\{\{X_1, X_2\}\big\}$. Calling [13, Algorithm 4.9] in linear time, we compute connectivity value $\ell = \lambda_{M_1}(X_1 \cup X_2)$ over the parse tree $\mathcal{T}_1$. If $\ell > k$, then we continue this cycle again from (a).
      ii. We call Algorithm 6.4 on $\mathcal{T}_1$ and $W = X_1 \cup X_2$ to compute a $\leq (3k + \ell - 2)$-boundaried parse tree $\mathcal{T}_2$ of a matroid M_2 which is obtained

by a titanic normalization of the part W.
By Lemmas 3.4 and 4.3 we have $\mathrm{bw}\big(M_1,\ \mathcal{P}_1 \cup \{W\} \setminus \{X_1, X_2\}\big) = \mathrm{bw}(M_2)$.

iii. We check whether branch-width $\mathrm{bw}(M_2) \leq k$ by applying Theorem 6.3. If $\mathrm{bw}(M_2) > k$, then we continue this cycle again from (a).

iv. If $\mathrm{bw}\big(M_1,\ \mathcal{P}_1 \cup \{W\} \setminus \{X_1, X_2\}\big) = \mathrm{bw}(M_2) \leq k$, then we add a *new node* $Z = E(U_Y)$ (U_Y given by the normalization of W in Algorithm 6.4) adjacent to X_1 and X_2 in the rooted forest D, and make Z the root for its component. We update $\mathcal{P}_1 := \mathcal{P}_2 = \mathcal{P}_1 \cup \{Z\} \setminus \{X_1, X_2\}$, and $\mathcal{Q}_1 := \mathcal{Q}_1 \cup \{Z\}$.

v. Last, by calling Algorithm 6.2 on $\mathcal{T}_2$, we compute in a *new* $\leq 3(k-1)$-*boundaried parse tree* $\mathcal{T}_3$ for the matroid M_2, and set $\mathcal{T}_1 := \mathcal{T}_3$.

(b) When the "while" cycle (4.a) is finished, we set $\mathcal{Q}_2 := \big\{\{X_1, X_2\} : X_1 \neq X_2,\ X_1 \in \mathcal{P}_1,\ X_2 \in \mathcal{Q}_1\big\}$ and $\mathcal{Q}_1 := \emptyset$, and continue from (4.a).

5. Finally, if $|\mathcal{P}_1| = 2$, then we connect by an edge in D the two nodes $X_1, X_2 \in \mathcal{P}_1$. We output (D, τ) as the branch-decomposition of $(M, \mathcal{P})$.

THEOREM 6.7. *Let k be a fixed integer and $\mathbb{F}$ be a fixed finite field. We assume that a vector matroid $M = M(\boldsymbol{A})$ is given as an input together with a partition $\mathcal{P}$ of $E(M)$, where $n = |E(M)|$ and $|\mathcal{P}| \geq 2$. Algorithm* 6.6 *outputs in time $O(n^3)$ (parametrized by k and $\mathbb{F}$) a branch-decomposition of the partitioned matroid $(M, \mathcal{P})$ of width at most k, or confirms that $\mathrm{bw}(M, \mathcal{P}) > k$.*

Proof. We refer to the above outline. Our proof of the theorem constitutes the following three claims holding true if $\mathrm{bw}(M, \mathcal{P}) \leq k$.

(I) The computation of Algorithm 6.6 maintains invariants, with respect to the actual matroid M_2 of $\mathcal{T}_2$, the rooted forest D, and the current value $\mathcal{P}_2$ of the partition variable $\mathcal{P}_1$ after each call to step (4.a.iv), such that
- $\mathcal{P}_2$ is the set of roots of D, and a titanic partition of M_2 such that $\mathrm{bw}(M_2, \mathcal{P}_2) = \mathrm{bw}(M_2) \leq k$,
- $\lambda_M\big(\tau^D(\mathcal{S})\big) = \lambda_{M_2}^{\mathcal{P}_2}(\mathcal{S})$ for each $\mathcal{S} \subseteq \mathcal{P}_2$, where $\tau^D(\mathcal{S})$ is a shortcut for the union of $\tau(X)$ with X running over all leaves of the connected components of D whose root is in $\mathcal{S}$ (see Algorithm 6.6, step 2. for τ).

(II) Each iteration of the main cycle in Algorithm 6.6 (4.) succeeds to step (4.a.iv) at least once.

(III) The main cycle in Algorithm 6.6 step 4. is repeated $O(n)$ times. Moreover, the total number of calls to the steps in (4.a) is $O(n^2)$ for steps i, ii, iii, and $O(n)$ for steps iv, v.

Having all of these facts at hand, it is now easy to finish the proof. It is immediate from (I) that the resulting (D, τ) is a branch-decomposition of width at most k of $(M, \mathcal{P})$. Note that all parse trees involved in the algorithm have constant width less than $4k$ (see in steps (4.a.ii,v)). The starting steps (1.), (2.), (3.) of the algorithm are already known to run in time $O(n^3)$ (Theorem 5.6 and Algorithm 6.1), and the particular steps in (4.a) need time (III) $O(n^2) \cdot O(n) + O(n) \cdot O(n^2) = O(n^3)$ by Lemma 6.5 and Algorithm 6.2. The size of the matroid M_1 clearly stays linear in n after $O(n)$ constant-size updates. Hence, our Algorithm 6.6 runs correctly in parametrized time $O(n^3)$, provided that (I)–(III) hold true.

The proof of (I) essentially extends the arguments of Theorem 5.7. Initially, with M_1 and $\mathcal{P}_1$ in place of M_2, $\mathcal{P}_2$, all the claims of (I) obviously hold true, analogously to Theorem 5.6. Each call to step (4.a.iv) then adds a new titanic set $E(U_Y)$ to $\mathcal{P}_2$ (see Lemma 4.3 (3)), and hence the partition $\mathcal{P}_2$ remains titanic for M_2 and, subsequently,

$\mathrm{bw}(M_2, \mathcal{P}_2) = \mathrm{bw}\big(M_1, \mathcal{P}_1 \cup \{W\} \setminus \{X_1, X_2\}\big) = \mathrm{bw}(M_2) \le k$ follows from Lemma 3.4. The most complex claim of (I) is the last assertion, that $\lambda_M(\tau^D(\mathcal{S})) = \lambda_{M_2}^{\mathcal{P}_2}(\mathcal{S})$ for each $\mathcal{S} \subseteq \mathcal{P}_2$. By induction, we may assume that $\lambda_M(\tau^D(\mathcal{S}_1)) = \lambda_{M_1}^{\mathcal{P}_1}(\mathcal{S}_1)$ holds for all $\mathcal{S}_1 \subseteq \mathcal{P}_1$ just before this call to (4.a.iv). Now, by Algorithm 6.4, the titanic gadget $E(U_Y)$ in the representation spans exactly the same subspace because it is the guts of the separation $(X_1 \cup X_2,\ E(M_1) \setminus (X_1 \cup X_2))$ in M_1. Therefore, for all $\mathcal{S}_1 \subseteq \mathcal{P}_1$ such that $|\mathcal{S}_1 \cap \{X_1, X_2\}| \neq 1$, the corresponding $\mathcal{S} \subseteq \mathcal{P}_2$ satisfies $\lambda_{M_2}^{\mathcal{P}_2}(\mathcal{S}) = \lambda_{M_1}^{\mathcal{P}_1}(\mathcal{S}_1)$. This proves the assertion.

To prove (II), we use that $\mathrm{bw}(M_1, \mathcal{P}_1) \le k$ at each iteration of the main cycle (4.), which directly follows from above $\mathrm{bw}(M_2, \mathcal{P}_2) \le k$. Then, by the same arguments as in Theorem 5.7, there is a pair $\{X_1, X_2\} \subset \mathcal{P}_1$ for which (4.a) would succeed up to step (4.a.iv), which happens if $\mathrm{bw}\big(M_1, \mathcal{P}_1 \cup \{X_1 \cup X_2\} \setminus \{X_1, X_2\}\big) \le k$. We call such a pair X_1, X_2 *admissible*. It remains to argue that all admissible pairs $\{X_1, X_2\} \subset \mathcal{P}_1$ belong also to $\mathcal{Q}_2$, which is trivial only during the first round of (4.). For a contradiction, assume that $\{X_1, X_2\} \notin \mathcal{Q}_2$ at the least round $i > 1$. Consider now the values of our variables $\mathcal{P}_1, \mathcal{Q}_1, \mathcal{Q}_2$ at the previous round $i - 1$: It was $\{X_1, X_2\} \cap \mathcal{Q}_1 = \emptyset$ by the assignment to $\mathcal{Q}_2$ in (4.b), and so $\{X_1, X_2\} \subset \mathcal{P}_1$ is already there. That means the pair X_1, X_2 has been admissible since round $i - 1$ started, but it has not been processed only due to $\{X_1, X_2\} \notin \mathcal{Q}_2$ at round $i-1$, which contradicts our least choice of i.

Concerning (III), each iteration of (4.) adds at least one new node to the decomposition D by (II), and hence no more than $O(n)$ iterations occur. The same argument also bounds the total number of calls to the crucial steps (4.a.iv–v). The situation with steps i, ii, iii is more versatile, and we bound the total number of calls to them from above by the total number of iterations of the cycle in (4.a): During the initial round of the main cycle (4.), there are clearly at most $|\mathcal{Q}_2| = O(n^2)$ iterations of (4.a). For each subsequent round $i > 1$, the number of iterations is at most $|\mathcal{Q}_2| \le q_i \cdot |\mathcal{P}_1|$, where $q_i = |\mathcal{Q}_1|$ at the end of the previous run $i - 1$. Hence, the total number of iterations of the cycle in (4.a) is at most $O(n^2) + O(n) \cdot \sum_{i=2}^{r} q_i$ since $|\mathcal{P}_1| = O(n)$ always. It remains to argue that $\sum_{i=2}^{r} q_i = O(n)$, which follows from the fact that each element ever assigned to $\mathcal{Q}_1$ in step (4.a.iv) appears as an internal node of the decomposition D, and $|V(D)| = O(n)$.

This also finishes the whole proof of Theorem 6.7. □

7. Finding a rank-decomposition of a graph. In this last section, we present a fixed-parameter tractable algorithm to find a rank-decomposition of width at most k or confirm that the input graph has rank-width larger than k. It is a direct translation of the algorithm of Theorem 6.7. Let us first review necessary definitions from [19] and [17]. We assume that all graphs in this section have no loops and no parallel edges.

We have seen in section 2 that every symmetric submodular function can be used to define branch-width. We define a symmetric submodular function on a graph, called the *cut-rank* function of a graph. For an $X \times Y$ matrix $\boldsymbol{R}$ and $A \subseteq X$, $B \subseteq Y$, let $\boldsymbol{R}[A, B]$ be the $A \times B$ submatrix of $\boldsymbol{R}$. For a graph $G = (V, E)$, let $\boldsymbol{A}(G)$ be the adjacency matrix of G, that is a $V \times V$ matrix over the binary field GF(2) such that an entry is 1 if and only if vertices corresponding to the column and the row are adjacent in G. The cut-rank function $\rho_G(X)$ of a graph $G = (V, E)$ is defined as the rank of the matrix $\boldsymbol{A}(G)[X, V \setminus X]$ for each subset X of V. Then ρ_G is symmetric and submodular; see [19]. *Rank-decomposition* and *rank-width* of a graph G is branch-decomposition

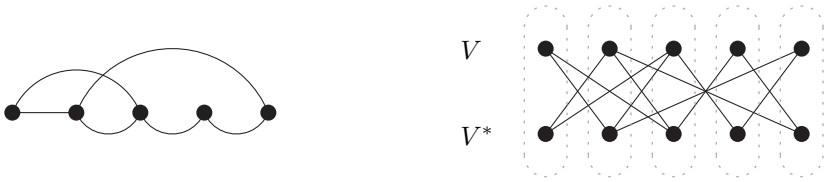


FIG. 5. *Graph G and the associated bipartite graph* $\mathrm{bip}(G)$ *with its canonical partition.*

and branch-width of the cut-rank function ρ_G of the graph G, respectively. So, if the graph has at least two vertices, then the rank-width is at most k if and only if there is a rank-decomposition of width at most k.

Now let us recall why bipartite graphs are essentially binary matroids. Oum [17] showed that the connectivity function of a binary matroid is exactly one more than the cut-rank function of its *fundamental graph*. The fundamental graph of a binary matroid M on $E = E(M)$, with respect to a basis B, is a bipartite graph on E such that two vertices in E are adjacent if and only if one vertex v is in B, another vertex w is not in B, and $(B \setminus \{v\}) \cup \{w\}$ is independent in M. Given a bipartite graph G, we can easily construct a binary matroid having G as a fundamental graph; if (C, D) is a bipartition of $V(G)$, then take the matrix

$$\begin{array}{c} \\ C \end{array} \begin{array}{c} \begin{array}{ccc} C & & D \end{array} \\ \left(\begin{array}{ccc|c} 1 & & 0 & \\ & \ddots & & \boldsymbol{A}(G)[C,D] \\ 0 & & 1 & \begin{array}{c} C \times D \text{ submatrix of} \\ \text{the adjacency matrix} \end{array} \end{array}\right) \end{array}$$

as the representation of a binary matroid. Thus, the column indices are elements of the binary matroid, and a set of columns is independent in the matroid if and only if its vectors are linearly independent. After all, finding the rank-decomposition of a bipartite graph is equivalent to finding the branch-decomposition of the associated binary matroid, that is essentially Theorem 6.7.

To find a rank-decomposition of nonbipartite graphs, we transform the graph into a canonical bipartite graph. For a finite set V, let V^* be a disjoint copy of V, that is, formally speaking, $V^* = \{v^* : v \in V\}$ such that $v^* \neq w$ for all $w \in V$ and $v^* \neq w^*$ for all $w \in V \setminus \{v\}$. For a subset X of V, let $X^* = \{v^* : v \in X\}$. For a graph $G = (V, E)$, let $\mathrm{bip}(G)$ be the bipartite graph on $V \cup V^*$ such that vw^* are adjacent in $\mathrm{bip}(G)$ if and only if v and w are adjacent in G (see Figure 5). Let $P_v = \{v, v^*\}$ for each $v \in V$. Then $\Pi(G) = \{P_v : v \in V\}$ is a canonical partition of $V(\mathrm{bip}(G))$.

LEMMA 7.1. *For every subset X of $V(G)$, $2\rho_G(X) = \rho_{\mathrm{bip}(G)}(X \cup X^*)$.*

Proof. This is clear from the construction of $\mathrm{bip}(G)$. Let $Y = V(G) \setminus X$. Let $N = \boldsymbol{A}(G)[X, Y]$. Since

$$\rho_{\mathrm{bip}(G)}(X \cup X^*) = \operatorname{rank} \begin{array}{c} \\ X \\ X^* \end{array} \begin{array}{c} \begin{array}{cc} Y & Y^* \end{array} \\ \begin{pmatrix} 0 & N \\ N^t & 0 \end{pmatrix} \end{array},$$

we conclude that $\rho_{\mathrm{bip}(G)}(X \cup X^*) = 2 \operatorname{rank} N = 2\rho_G(X)$. ∎

COROLLARY 7.2. *Let $p : V(G) \to \Pi(G)$ be the bijective function such that $p(x) = P_x$. If (T, μ) is a branch-decomposition of $\rho_{\mathrm{bip}(G)}^{\Pi(G)}$ of width k, then $(T, \mu \circ p)$*

is a branch-decomposition of ρ_G of width $k/2$. Conversely, if (T, μ') is a branch-decomposition of ρ_G of width k, then $(T, \mu' \circ p^{-1})$ is a branch-decomposition of $\rho^{\Pi(G)}_{\mathrm{bip}(G)}$ of width $2k$. Therefore, the branch-width of ρ_G is equal to half of the branch-width of $\rho^{\Pi(G)}_{\mathrm{bip}(G)}$.

Let $M = \mathrm{mat}(G)$ be the binary matroid on $V \cup V^*$ represented by the matrix

$$\begin{array}{c} \\ V \end{array} \begin{array}{c} \begin{array}{cc} V & V^* \end{array} \\ \left(\begin{array}{c|c} \text{Identity matrix} & \boldsymbol{A}(G) \end{array}\right) \end{array}.$$

Since the bipartite graph $\mathrm{bip}(G)$ is a fundamental graph of M, we have $\lambda_M(X) = \rho_{\mathrm{bip}(G)}(X) + 1$ for all $X \subseteq V \cup V^*$ (see Oum [17]) and therefore (T, μ) is a branch-decomposition of a partitioned matroid $(M, \Pi(G))$ of width $k+1$ if and only if it is a branch-decomposition of $\rho^{\Pi(G)}_{\mathrm{bip}(G)}$ of width k. Corollary 7.2 implies that a branch-decomposition of $\rho^{\Pi(G)}_{\mathrm{bip}(G)}$ of width k is equivalent to that of ρ_G of width $k/2$. So, we can deduce the following theorem from Theorem 6.7.

THEOREM 7.3. *Let k be a constant. Let $n \geq 2$. For an n-vertex graph G, we can output the rank-decomposition of width at most k or confirm that the rank-width of G is larger than k in time $O(n^3)$.*

Proof. We apply Theorem 6.7 to find a branch-decomposition of a partitioned matroid $(\mathrm{mat}(G), \Pi(G))$ of width at most $2k+1$. If such a branch-decomposition is found, then one can canonically transform it into a rank-decomposition of G of width at most k by Corollary 7.2. If there is no such branch-decomposition, then the rank-width of G is larger than k. □

Acknowledgments. The authors are very grateful to Jim Geelen for his comments on the possible approach to the problem. We would also like to thank the anonymous referees for helpful comments.

REFERENCES

[1] D. G. CORNEIL, M. HABIB, J.-M. LANLIGNEL, B. A. REED, AND U. ROTICS, *Polynomial time recognition of clique-width ≤ 3 graphs* (extended abstract), in LATIN 2000: Theoretical informatics, G. H. Gonnet, D. Panario, and A. Viola, eds., Lecture Notes in Comput. Sci. 1776, Springer, Berlin, 2000, pp. 126–134.

[2] D. G. CORNEIL, Y. PERL, AND L. K. STEWART, *A linear recognition algorithm for cographs*, SIAM J. Comput., 14 (1985), pp. 926–934.

[3] B. COURCELLE, J. A. MAKOWSKY, AND U. ROTICS, *Linear time solvable optimization problems on graphs of bounded clique-width*, Theory Comput. Syst., 33 (2000), pp. 125–150.

[4] B. COURCELLE AND S. OLARIU, *Upper bounds to the clique width of graphs*, Discrete Appl. Math., 101 (2000), pp. 77–114.

[5] B. COURCELLE AND S. OUM, *Vertex-minors, monadic second-order logic, and a conjecture by Seese*, J. Combin. Theory Ser. B, 97 (2007), pp. 91–126.

[6] W. ESPELAGE, F. GURSKI, AND E. WANKE, *How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time*, in Graph-Theoretic Concepts in Computer Science (Boltenhagen, 2001), Lecture Notes in Comput. Sci. 2204, Springer, Berlin, 2001, pp. 117–128.

[7] M. R. FELLOWS, F. A. ROSAMOND, U. ROTICS, AND S. SZEIDER, *Clique-width minimization is NP-hard*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2006, pp. 354–362.

[8] J. F. GEELEN, A. M. H. GERARDS, N. ROBERTSON, AND G. P. WHITTLE, *On the excluded minors for the matroids of branch-width k*, J. Combin. Theory Ser. B, 88 (2003), pp. 261–265.

[9] J. F. Geelen, A. M. H. Gerards, and G. Whittle, *Tangles, Tree-Decompositions, and Grids in Matroids*, Research report 04-5, School of Mathematical and Computing Sciences, Victoria University of Wellington, Wellington, New Zealand, 2004.
[10] M. U. Gerber and D. Kobler, *Algorithms for vertex-partitioning problems on graphs with fixed clique-width*, Theoret. Comput. Sci., 299 (2003), pp. 719–734.
[11] I. V. Hicks and N. B. McMurray, Jr., *The branchwidth of graphs and their cycle matroids*, J. Combin. Theory Ser. B, 97 (2007), pp. 681–692.
[12] J. W. P. Hirschfeld and L. Storme, *The packing problem in statistics, coding theory and finite projective spaces: Update* 2001, in Finite Geometries, Dev. Math. 3, Kluwer Academic Publishers, Dordrecht, 2001, pp. 201–246.
[13] P. Hliněný, *A parametrized algorithm for matroid branch-width*, SIAM J. Comput., 35 (2005), pp. 259–277.
[14] P. Hliněný, *Branch-width, parse trees, and monadic second-order logic for matroids*, J. Combin. Theory Ser. B, 96 (2006), pp. 325–351.
[15] D. Kobler and U. Rotics, *Edge dominating set and colorings on graphs with fixed clique-width*, Discrete Appl. Math., 126 (2003), pp. 197–221.
[16] F. Mazoit and S. Thomassé, *Branchwidth of graphic matroids*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 346, Cambridge University Press, Cambridge, 2007, pp. 275–286.
[17] S. Oum, *Rank-width and vertex-minors*, J. Combin. Theory Ser. B, 95 (2005), pp. 79–100.
[18] S. Oum, *Approximating rank-width and clique-width quickly*, submitted, 2006.
[19] S. Oum and P. Seymour, *Approximating clique-width and branch-width*, J. Combin. Theory Ser. B, 96 (2006), pp. 514–528.
[20] S. Oum and P. Seymour, *Testing branch-width*, J. Combin. Theory Ser. B, 97 (2007), pp. 385–393.
[21] J. G. Oxley, *Matroid Theory*, Oxford University Press, New York, 1992.
[22] N. Robertson and P. Seymour, *Graph minors.* X. *Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.
[23] P. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.
[24] E. Wanke, *k-NLC graphs and polynomial algorithms. Efficient algorithms and partial k-trees*, Discrete Appl. Math., 54 (1994), pp. 251–266.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1033–1057

QUERY-EFFICIENT ALGORITHMS FOR POLYNOMIAL INTERPOLATION OVER COMPOSITES*

PARIKSHIT GOPALAN†

Abstract. The problem of polynomial interpolation is to reconstruct a polynomial based on its valuations on a set of inputs I. We consider the problem over $\mathbb{Z}_m$ when m is composite. We ask the following question: Given $I \subseteq \mathbb{Z}_m$, how many evaluations of a polynomial at points in I are required to compute its value at every point in I? Surprisingly for composite m, this number can vary exponentially between $\log |I|$ and $|I|$, in contrast to the prime case where $|I|$ evaluations are necessary. While we show this minimization problem to be NP-hard, we give an efficient algorithm of query complexity within a factor t of the optimum, where t is the number of prime factors of m. We use our interpolation algorithm to design algorithms for zero testing and distributional learning of polynomials over $\mathbb{Z}_m$. In some cases, we get an exponential improvement over known algorithms in query complexity and running time. Our main technical contribution is the notion of an interpolating set for I which is a subset S of I such that a polynomial which is 0 over S must be 0 at every point in I. Any interpolation algorithm needs to query an interpolating set for I. Our query-efficient algorithms are obtained by constructing interpolating sets whose size is close to optimal.

Key words. polynomials, composites, interpolation

AMS subject classifications. 68Q25, 12Y05

DOI. 10.1137/060661259

1. Introduction.

The problem of polynomial interpolation is to reconstruct a polynomial from its evaluations. This is a fundamental algorithmic question in algebra with numerous applications. The problem is especially well studied when the polynomial is over a field such as $\mathbb{R}$ or $\mathbb{Z}_p$ dating back to Newton and Lagrange. Relatively less is known about interpolation over rings which contain zero divisors, in particular over $\mathbb{Z}_m$ with m composite. The zero-testing problem is a special case of the interpolation problem where we want to know if the polynomial is 0 everywhere. In this paper we study the problem of learning a univariate polynomial in $\mathbb{Z}_m[X]$ based on its evaluations at a set $I \subseteq \mathbb{Z}_m$. We ask the following question: *Given $I \subseteq \mathbb{Z}_m$, how many evaluations of a polynomial at points in I are required to compute its value at every point in I?* Throughout, we will consider a polynomial as a function rather than a formal sum, and our aim will be to correctly predict its values at every point in I.

Polynomials and specifically the problem of interpolation over $\mathbb{Z}_m$ are well studied in mathematics [18, 31]. Dueball (see [31]) shows that when $I = \mathbb{Z}_m$, there is a subset S whose size can lie between $\log m$ and m such that the evaluations at S are sufficient for interpolation. However, this result leaves open the question of whether a similar statement holds for subsets of $\mathbb{Z}_m$. Polynomials over $\mathbb{Z}_m$ have many applications in computer science. They are used in algorithms for primality testing [1, 2], in the construction of explicit Ramsey graphs and extremal set systems [21, 24], and in circuit lower bounds [7, 36]. As we will elaborate shortly, many of these applications

*Received by the editors May 30, 2006; accepted for publication (in revised form) March 14, 2008; published electronically June 25, 2008. An extended abstract of this paper appeared in SODA'06. This work was done while the author was a graduate student at Georgia Tech, supported by NSF grant CCR-3606B64.

http://www.siam.org/journals/sicomp/38-3/66125.html

†Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 (parik@cs.washington.edu).

implicitly address questions related to polynomial interpolation. We hope that a better understanding of the interpolation problem will throw new light on these problems.

The polynomial interpolation problem over $\mathbb{Z}_m$ is very different from $\mathbb{Z}_p$, since it is no longer true that a degree d polynomial has at most d zeroes. For instance $X^k \equiv 0 \bmod 2^k$ has 2^{k-1} roots. This implies that even two polynomials of small degree can agree on a large fraction of points in $\mathbb{Z}_m$. Hence, unlike over $\mathbb{Z}_p$ one cannot interpolate even low degree polynomials from their evaluations at a few arbitrarily chosen points. On the other hand, not every function $f : \mathbb{Z}_m \to \mathbb{Z}_m$ is a polynomial, since functions defined by polynomials need to satisfy certain congruences. For instance let $m = pq$ and $x, y \in \mathbb{Z}_m$ such that $x \equiv y \bmod p$. Then $P(x) \equiv P(y) \bmod p$ for any polynomial $P(X) \in \mathbb{Z}_m[X]$. Thus the values of a polynomial at a point give some information about its values at other points. This raises the possibility of learning a polynomial by looking at its evaluations at only a few carefully chosen points.

We give an query-efficient algorithm to solve the following problem.

Problem 1. Generalized Polynomial Interpolation: *Given m, a set $I \subseteq \mathbb{Z}_m$, and black-box access to the values of a polynomial $P(X) \in \mathbb{Z}_m[X]$ at points in I, compute $P(X)$ and minimize the number of black-box queries.*

The query complexity of our algorithm is within a factor t of the optimum, where t is the number of prime divisors of m. One has to settle for approximation, since we prove that the problem of minimizing the number of queries is NP-hard. Our main technical contribution is the notion of *interpolating sets for I* which are subsets of I such that a polynomial which is 0 over that subset must in fact be 0 at every point in I. We show that any interpolation algorithm needs to query an interpolating set for I. Our query-efficient algorithms are obtained by constructing interpolating sets whose size is close to optimal. We use our interpolation algorithm to design algorithms for zero testing and distributional learning of polynomials over $\mathbb{Z}_m$. In some cases, we get an exponential improvement over known algorithms in query complexity and running time. We show some new results about the structure of polynomials over $\mathbb{Z}_m$, which may be useful in other applications.

1.1. History and motivation. Given a commutative ring R, a function $f : R \to R$ which can be computed by a polynomial in $R[X]$ is called a polynomial function. Polynomial functions over various commutative rings are well studied in algebra [15, 31, 18]. The problem of characterizing polynomial functions over $\mathbb{Z}_m$ was first studied by Carlitz and Spira (see [35] and the book by Narkiewicz and the references therein [31]). Kempner gave a canonical polynomial for every polynomial function over $\mathbb{Z}_m$ [30]. Dueball studied the problem of interpolation over $\mathbb{Z}_m$ [31]. He proved that one can solve the interpolation problem over $\mathbb{Z}_m$ with as few as $O(\log m)$ queries for some composites m. More precisely, he showed the following result:

Let $k(m)$ be the smallest integer such that $k(m)! \equiv 0 \bmod m$. Every polynomial function over $\mathbb{Z}_m$ can be learned from its values at $\{0, \ldots, k(m) - 1\}$.

In the zero-testing problem, we are given an implicit representation of a polynomial P, either as a circuit or a black box which returns the value $P(x)$ on query x. We wish to determine if P is the 0 polynomial. The problem of zero testing for polynomials over $\mathbb{Z}_m$ was studied by Agrawal and Biswas [1], motivated by primality testing. They give a randomized algorithm for this problem. However, they view polynomials as formal sums rather than as functions, and this is important for their application. Karpinski, van der Poorten, and Shparlinski [28] give a black-box algorithm for zero testing over $\mathbb{Z}_m$. However, they require that all nonzero coefficients of the polynomial are relatively prime to m. Bshouty, Tamon, and Wilson give a randomized algorithm

for interpolation over $\mathbb{Z}_m$ [14]. However, if the smallest prime dividing m is p, they require the degree to be at most $\frac{p}{2}$. The results of [14, 28] hold for multivariate polynomials, but in the univariate case Dueball's result is stronger.

Interpolation and zero testing for polynomials over $\mathbb{Z}_p$ have been studied extensively in computer science, motivated by applications in coding theory, proof checking, and several other areas (see, for instance, [20]). Most of these applications crucially use the fact that a degree d polynomial over $\mathbb{Z}_p$ has at most d roots, which does not hold for polynomials over $\mathbb{Z}_m$. Nevertheless, polynomials over $\mathbb{Z}_m$ have found surprising applications in algorithms, combinatorics, and complexity, which rely on the fact that they behave differently from polynomials over $\mathbb{Z}_p$.

- *Primality and factoring:* The primality testing algorithms of Agrawal and Biswas and the AKS algorithm reduce testing primality to testing a polynomial identity over $\mathbb{Z}_m[X]$ [1, 2]. They then devise algorithms to solve the problem of zero testing over $\mathbb{Z}_m[X]$. Shamir [33] shows that the problem of factoring polynomials over $\mathbb{Z}_m[X]$ is as hard as integer factoring.
- *Boolean function complexity:* A frontier open problem in complexity theory is to show lower bounds for circuits with Mod-m gates. Strong lower bounds are known when the circuit contains only Mod-p gates for a single prime p [32, 34]. In contrast much less is known if Mod-m gates are allowed with m composite, or Mod-p and Mod-q gates for distinct primes [26, 16]. Motivated by this problem, Barrington, Beigel, and Rudich [7] studied representations of Boolean functions by polynomials over $\mathbb{Z}_m$. They proved that functions like OR can have low degree representations when m is composite, unlike in the prime case. The problem of showing tight bounds for such representations over $\mathbb{Z}_m$ is wide open [4, 13, 36].
- *Extremal set theory:* A set system on $[n]$ is said to have restricted intersections modulo m if there exists $L \subseteq \mathbb{Z}_m$ such that the pairwise intersections mod m lie in L but the set sizes lie outside it [5]. Polynomials over $\mathbb{Z}_m$ have been used to explicitly construct large set systems and also to prove upper bounds on their size. A surprising insight from this area is that when m is a prime or a prime power, the size of such set systems is polynomial in the number of elements in the universe [5, 6, 21]. In contrast, when m has two or more prime divisors, the size can be superpolynomial [24]. These results about set systems have important combinatorial applications [5].
- *Explicit Ramsey constructions:* A Ramsey graph is a graph with no large cliques and independent sets. The problem of explicitly constructing good Ramsey graphs is an important open problem in combinatorics. Recently, the author [21] showed that the algebraic Ramsey graph constructions of Alon [3], Frankl and Wilson [17], and Grolmusz [24, 25] can be derived in a unified manner from low degree polynomials over $\mathbb{Z}_m$. Further, facts about interpolating sets over $\mathbb{Z}_{p^a}$ from this work are used in [21] in order to show that certain approaches (based on symmetric polynomials) cannot yield better Ramsey graphs.

The last three applications deal with whether certain functions can be computed by low degree polynomials over $\mathbb{Z}_m$, and hence they all implicitly address questions related to polynomial interpolation.

1.2. Our results.

The generalized interpolation problem. Our main result is an efficient algorithm to solve the generalized interpolation problem. We prove that minimizing

the number of queries is NP-hard, and hence one can hope only to approximately minimize the query complexity. Our algorithm has query complexity close to optimal.

THEOREM 1. *Let t be the number of distinct prime factors of m. There is an algorithm to solve the general interpolation problem over $\mathbb{Z}_m$, with query complexity within a factor t of the optimum.*

In fact the guarantee is slightly stronger. When the algorithm terminates, it produces a factorization of m into $t' \leq t$ relatively prime factors. The approximation factor is in fact bounded by t'. Thus input sets I which force the algorithm to make several queries must also reveal the factorization of m. The algorithm first computes a set S of queries to ask based on the input set I. Thus the set of queries is chosen nonadaptively. The size of S is within a factor t' of the optimal query complexity. This step takes time proportional to $|S| \cdot |I|$. Once the set S is found, the polynomial can be computed with $|S|$ queries in time $\text{poly}(\log m, |S|)$.

While Dueball's result gives an efficient algorithm for the case when $I = \mathbb{Z}_m$, it does not imply anything for the general interpolation problem. The naive approach for this problem would be to write a linear equation for each point in I. We can replace each equation $\sum_j a_{ij}X_{ij} = b_i \bmod m$ with $\sum_j a_{ij}X_{ij} = b_i + y_i m$, where the y_i's are integer variables, and find integer solutions to the resulting system of equations. This has query complexity $|I|$, which can be exponentially larger than the complexity of our algorithm.

The generalized zero-testing problem is a special case of the interpolation problem, where we wish to know if some identity holds for every point in I. Theorem 1 implies a query-efficient algorithm for this problem. It improves on the algorithms of [14, 28], since there are no restrictions on the degree or coefficients of the polynomial. Our results are incomparable with those of Agrawal and Biswas [1], who view polynomials as formal sums.

Learning under a distribution. We give the first efficient algorithms for learning polynomials over $\mathbb{Z}_m$ under a distribution. Here we are given evaluations of the polynomial at points which are drawn from some distribution, and we are asked to learn the polynomial. See section 5 for precise problem definitions.

THEOREM 2. *Polynomials in $\mathbb{Z}_m[X]$ are exactly learnable under the uniform distribution and PAC learnable under an arbitrary distribution in polynomial time.*

The algorithm for the uniform distribution learns the polynomial exactly, but its running time is a random variable. These algorithms use the algorithm for the general interpolation problem as a subroutine. For distributional learning it is essential that our algorithm solves the general interpolation problem, where inputs come from some subset $I \subseteq \mathbb{Z}_m$ rather than all of $\mathbb{Z}_m$.

Interpolating sets. The crux of our algorithm is the notion of an *interpolating set* which we introduce and study here. A set $S \subseteq I$ is an interpolating set for I if knowing the values of any polynomial at S fixes its value at every point in I. We show that the set of queries of an interpolation algorithm must correspond to an interpolating set for I, and thus the problem of designing query-efficient algorithms reduces to finding small interpolating sets.

Let $k(I)$ denote the size of the smallest interpolating set for I. In general $k(I)$ can lie between $\log |I|$ and $|I|$. However, we prove that the problem of computing a minimum interpolating set for I is NP-hard. We define a related quantity $\bar{k}(I)$, which is the smallest integer such that there is a degree $\bar{k}(I)$ monic polynomial $M(X) \in \mathbb{Z}_m[X]$ which is 0 over I. This quantity can be computed in polynomial time by solving a

system of linear equations. We show that for $I \subseteq \mathbb{Z}_m$, where m has t prime divisors, the following relation holds:

$$\bar{k}(I) \;\leq\; k(I) \;\leq\; t \cdot \bar{k}(I).$$

Thus $\bar{k}(I)$ is a factor t approximation to $k(I)$, where t is the number of prime divisors of m. This is where the approximation factor of t in the query complexity of our algorithm comes from.

We sketch the idea behind the algorithm for computing an interpolating set. For the prime-power case, we use a greedy algorithm. There is a natural metric on the points $I \subseteq \mathbb{Z}_{p^a}$, namely p-adic distance. Our algorithm finds a set of points so that the sum of pairwise distances is maximized; this is done by picking a new point in a natural greedy manner. We show that this in fact gives an interpolating set. For the composite case, we essentially try and repeat this greedy approach. However, this approach might fail: first, we do not know the factorization of m, and, second, distinct prime divisors p and q give different metrics on the set I. However, we show that when it fails, one can get a factorization $m = m_1 \cdot m_2$, where $(m_1, m_2) = 1$. This allows us to use divide and conquer: we find interpolating sets modulo m_1 and m_2 independently and combine the result using the Chinese remainder theorem.

Interpolating sets over $\mathbb{Z}_{p^a}$ have rich algebraic and combinatorial structure which we study in detail; these properties are also useful in analyzing our algorithm. In proving these properties, we make crucial use of the fact that the underlying space is in fact an ultrametric space (metrics where the following strengthening of the triangle inequality holds: $d(x, y) \leq \max(d(x, z), d(y, z))$. Ultrametric spaces are well studied in computer science [8, 9, 10]. We show that many algebraic properties of polynomials can be reinterpreted as geometric properties of ultrametric spaces. Further, the proof of these properties for general ultrametric spaces follows directly from the proof for polynomials over $\mathbb{Z}_{p^a}$. We note that our notion of interpolating sets over $\mathbb{Z}_{p^a}$ is closely related to very well distributed and well-ordered sequences that have been studied in mathematics [15].

1.3. Organization of this paper. In the next section, we give some basic definitions and results about interpolation over $\mathbb{Z}_m$. We study interpolating sets in section 3. We present our algorithms for interpolation in section 4 and learning algorithms in section 5. We present other algebraic and combinatorial characterizations of interpolating sets in section 6. In section 7, we use these characterizations to establish some geometric properties of ultrametric spaces. An extended abstract of this paper appeared in SODA'06 [22].

2. Preliminaries. We will use X to denote a variable and x for a constant. Let (a, b) denote the greatest common divisor of a and b.

Given $x \in \mathbb{Z}_{p^a}$, $x \neq 0$, let its p-adic valuation $\mathrm{val}_p(x)$ be the highest power of p which divides x. Set $\mathrm{val}_p(0) = \infty$. We have the so-called ultrametric inequality, which states that

$$\mathrm{val}_p(x + y) \geq \min(\mathrm{val}_p(x), \mathrm{val}_p(y)).$$

The p-adic norm of x is defined as

$$|x|_p = p^{-\,\mathrm{val}_p(x)}.$$

The p-adic norm satisfies the following condition:

$$|x + y|_p \leq \max(|x|_p, |y|_p).$$

This induces a metric (the p-adic metric) on $\mathbb{Z}_{p^a}$ given by $d(x,y) = |x-y|_p$. This metric satisfies the following strong form of the triangle inequality:

$$d(x,z) \leq \max(d(x,y), d(y,z)). \tag{1}$$

Metrics which satisfy (1) are known as ultrametrics.

We will also define valuations over $\mathbb{Z}_m$, where m is not a prime power. Assume that p divides m, and let p^e be the highest power of p dividing m. For $x \in \mathbb{Z}_m$, we define

$$\mathrm{val}_p(x) \;=\; \mathrm{val}_p(x \bmod p^e).$$

All our results can be stated either in terms of p-adic valuations or norms. For our algorithmic results, it is more natural to work with valuations. For our combinatorial results, we will use p-adic norms, since it is easier to translate these results into general ultrametric spaces.

We start with some basic algebraic facts that will be useful to us.

PROPOSITION 3 (see [27]). *Let $a, b \in \mathbb{Z}_m$ and $a \not\equiv 0$. The equation*

$$aX \;\equiv\; b \bmod m$$

has a solution in $\mathbb{Z}_m$ iff $(a,m)|b$. If this condition holds, there is a unique solution in the interval $\left[0, \ldots, \frac{m}{(a,m)} - 1\right]$.

PROPOSITION 4. *Let $M(X)$ be a monic polynomial in $\mathbb{Z}_m[x]$ of degree k. Given $P(X) \in \mathbb{Z}_m[X]$, we can divide it by $M(X)$ and get a remainder of degree at most $k-1$.*

Proof. This is just Euclidean division. Let $P(X) = \sum_{i \leq d} c_i X^i$, where $d \geq k$. Since $M(X)$ is monic, $P(X) - c_d X^{d-k} M(X)$ has degree $d-1$. Now repeat the same procedure until we are left with a polynomial of degree $\leq k-1$. ∎

PROPOSITION 5. *Let $N_0(X), \ldots, N_k(X)$ be polynomials in $\mathbb{Z}_m[X]$, where $N_i(X)$ is a monic polynomial of degree i. Every polynomial $P(X)$ of degree at most k can be written as*

$$P(X) = \sum_{i=0}^{k} c_i N_i(X).$$

Further, if $P(X)$ is a monic polynomial of degree k, then $c_k = 1$.

Proof. The proof is by induction on k. When $k = 0$, $N_0(X) = 1$, so there is nothing to prove. Assume the claim holds for $k-1$. Let $P(X) = \sum_{i \leq k} a_i X^i$. Since $N_k(X)$ is monic, $P(X) - a_k N_k(X)$ has degree $k-1$, so we can apply induction to it. Note that the leading coefficient in the monomial basis and the $\{N_i(X)\}$ basis is the same. This shows the second part of the claim. ∎

We can use this to give a canonical form for polynomial functions over $\mathbb{Z}_m$ due to Kempner [30]. Let $N_0(X) = 1$, and, for $j \geq 1$, let

$$N_j(X) = \prod_{i=0}^{j-1} (X - i).$$

Let the elements $\{0, \ldots, m-1\}$ of $\mathbb{Z}_m$ be endowed with the ordering $0 < 1 < \cdots < m-1$. Let $k(m)$ be the smallest integer such that $k(m)! \equiv 0 \bmod m$.

LEMMA 6 (see [30]). *Every polynomial function over $\mathbb{Z}_m$ is computed by a unique polynomial of the form*

$$P(X) = \sum_{j=0}^{k(m)-1} c_j N_j(X), \qquad 0 \le c_j < \frac{m}{(m, j!)}. \tag{2}$$

Proof. For any $x \in \mathbb{Z}$,

$$N_j(x) \;=\; \prod_{i=0}^{j-1}(x-i) \;=\; \binom{x}{j} j!.$$

Hence $N_j(x)$ is divisible by $j!$. So the polynomial $\frac{m}{(m,j!)}N_j(X)$ is zero over $\mathbb{Z}_m$. In particular, $N_{k(m)}(X)$ is a degree $k(m)$ monic polynomial which is 0 over $\mathbb{Z}_m$.

Given an arbitrary polynomial $Q(X)$, we first divide by $N_{k(m)}(X)$ to get a polynomial $Q'(X)$ of degree $k(m)-1$. Since the polynomial $N_j(X)$ is monic and of degree j for $j \in \{0, \ldots, k(m)-1\}$, we can write $Q(X)$ as

$$Q'(X) = \sum_{j=0}^{k(m)-1} c_j N_j(X).$$

We can reduce this to the form of (2) by subtracting an appropriate multiple of $\frac{m}{(m,j!)}N_j(X)$ for $j \le k(m)$. Since we are subtracting only polynomials that are 0 over $\mathbb{Z}_m$, the resulting polynomial computes the same function as the polynomial $Q(X)$ that we started with.

To show that this representation is unique, take two polynomials

$$P(X) = \sum_{j=0}^{k(m)-1} c_j N_j(X), \qquad 0 \le c_j < \frac{m}{(m, j!)},$$
$$Q(X) = \sum_{j=0}^{k(m)-1} d_j N_j(X), \qquad 0 \le d_j < \frac{m}{(m, j!)}.$$

Pick the smallest index j so that $c_j \ne d_j$, and assume that $c_j > d_j$. We claim that $P(j) \not\equiv Q(j) \bmod m$. Since $N_i(j) = 0$ for $i > j$ and $c_i = d_i$ for $i < j$, we have

$$P(j) - Q(j) \;=\; (c_j - d_j)N_j(j) \;=\; (c_j - d_j)j! \;\not\equiv\; 0 \bmod m,$$

since

$$0 \;<\; c_j - d_j \;<\; \frac{m}{(m, j!)}. \qquad \square$$

An easy consequence is the following result of Dueball (see [31]).

COROLLARY 7 (see [31]). *Every polynomial function over $\mathbb{Z}_m$ can be interpolated from its evaluations at the points $\{0, \ldots, k(m)-1\}$.*

Proof. Let

$$P(X) = \sum_j c_j N_j(X), \qquad 0 \le c_j < \frac{m}{(m, j!)}.$$

We let $c_0 = P(0)$. Assuming we know $c_0, \ldots, c_{j-1}$, we solve for c_j from the equation

$$c_j j! \equiv P(j) - \sum_{i<j} c_i N_i(j) \bmod m.$$

The coefficients of $P(X)$ in the canonical form are a solution to this equation. Further, the solution must be unique, since the canonical form is unique. □

The following estimates for $k(m)$ show that the number of queries can be significantly smaller than m if m is smooth.

LEMMA 8. *For prime powers,*

$$p(a-1)+1 \;\leq\; k(p^a) \;\leq\; pa.$$

If $m = \prod_j p_j^{a_j}$, *then*

$$k(m) = \max_j k(p_j^{a_j}).$$

Proof. Since $(pa)! \equiv 0 \bmod p^a$, $k(p^a) \leq pa$. Let $k = \sum_i k_i p^i$ be the base-p expansion of k. Using a formula due to Legendre (see [23]),

$$\mathrm{val}_p(k!) \;=\; \sum_i \left\lfloor \frac{k}{p^i} \right\rfloor \;=\; \frac{k - \sum k_i}{p-1}. \tag{3}$$

Hence if $k! \equiv 0 \bmod p^a$, then

$$\frac{k - \sum k_i}{p-1} \geq a \quad\Rightarrow\quad k \;\geq\; (p-1)a + \sum_i k_i \;\geq\; (p-1)a + 1.$$

For $m = \prod_j p_j^{a_j}$, by Chinese remaindering, $k! \equiv 0 \bmod m$ is equivalent to $k! \equiv 0 \bmod p_j^{a_j}$ for all j. So $k(m) = \max_j k(p_j^{a_j})$. □

Next we show that the problem of computing $k(m)$ from m is as hard as factoring m.

LEMMA 9. *The problem of computing* $k(m)$ *given* m *as input is equivalent to factoring* m.

Proof. One can check in polynomial time if m is a prime power [11], so assume it is not. We will show that $(k(m), m)$ gives a nontrivial factor of m. Note that $k(m) = \max_j k(p_j^{a_j})$. Assume this maximum is attained for the prime p_i. Note that $k(p^a) \equiv 0 \bmod p$; else

$$\mathrm{val}_p(k(p^a)!) = \mathrm{val}_p((k(p^a)-1)!),$$

which contradicts the definition of $k(p^a)$. Then $k(m) = k(p_i^{a_i})$ is divisible by p_i. Further,

$$k(p_i^{a_i}) \;\leq\; p_i a_i \;\leq\; p_i^{a_i} \;<\; m,$$

since m is not a prime power. Hence

$$p_i \;\leq\; (k(m), m) \;<\; m.$$

Thus we get a nontrivial factor of m. If $(k(m), m)$ is not a prime power, we can repeat this procedure until we get a prime-power divisor of m. □

3. Interpolating sets. We say that a polynomial $P(X)$ is 0 over set S if it evaluates to 0 at every point in S.

DEFINITION 1. *Given $I \subseteq \mathbb{Z}_m$, $S \subseteq I$ is an interpolating set for I if every polynomial which is 0 over S is 0 over I. Let $k(I)$ denote the size of the smallest interpolating set for I.*

Note that I itself is trivially an interpolating set. However, in general there can be interpolating sets which are significantly smaller than I. Note that two polynomials $P(X)$ and $Q(X)$ that agree at S must in fact agree at every point in I by considering $P(X) - Q(X)$. Thus the values of a polynomial over I are uniquely determined by the values at points in an interpolating set. The next lemma shows that the minimum number of queries to interpolate a polynomial over I is $k(I)$.

LEMMA 10. *The set of black-box queries of any interpolation algorithm is an interpolating set for I.*

Proof. Assume that the set S of queries to the black box is not an interpolating set. Then there exists polynomial $Q(X) \in \mathbb{Z}_m[X]$ such that $Q(x)$ is 0 at all $x \in S$ but nonzero at some point $y \in I$. Hence the algorithm cannot distinguish between polynomials $P(X)$ and $P(X)+Q(X)$ which agree on S but are different at $y \in I$. ∎

Note that this bound holds even if the algorithm chooses its queries adaptively.

DEFINITION 2. *Let $\bar{k}(I)$ be the smallest integer such that there is a degree $\bar{k}(I)$ monic polynomial $M(X) \in \mathbb{Z}_m[X]$ which is 0 over I.*

If S is an interpolating set of size $k(I)$, then the polynomial $\prod_{\alpha \in S}(X - \alpha)$ is a monic polynomial of degree $k(I)$. It is zero over S and hence over I. Hence

$$\bar{k}(I) \leq k(I). \tag{4}$$

This lets us prove lower bounds on $k(I)$ by showing that any polynomial that is 0 over I must have a certain degree.

Example 1. For $I \subseteq \mathbb{Z}_p$, $\bar{k}(I) = k(I) = |I|$. Since $\mathbb{Z}_p$ is a field, the smallest degree monic polynomial which is 0 over I is $M(X) = \prod_{\alpha \in I}(X - \alpha)$, and hence $\bar{k}(I) = |I|$.

Example 2. For $I = \mathbb{Z}_m$, $\bar{k}(I) = k(I) = k(m)$. By Corollary 7, the set $S = \{0, \ldots, k(m) - 1\}$ is an interpolating set, so $k(I) \leq k(m)$. To show that $\bar{k}(I) \geq k(m)$, assume that $M(X)$ is a monic polynomial of degree $d < k(m)$. Writing it in the canonical form, we get $M(X) = \sum_{i \leq d} c_i N_i(X)$, where $c_d = 1$. So $M(X)$ cannot be zero over $\mathbb{Z}_m$ by Lemma 6.

One can use the Chinese remainder theorem to relate the problem of computing $k(I)$ and $\bar{k}(I)$ for $I \subseteq \mathbb{Z}_m$ for composite m to the prime-power case. First, we need to introduce some notation. Let $m = \prod_{j=1}^{t} p_i^{a_i}$. Given a set $L \subseteq \mathbb{Z}_m$ we define the *projection* L_j of L mod $p_j^{a_j}$ as

$$L_j = \{y \in \mathbb{Z}_{p_j{}^{a_j}} | \exists\, x \in L, x \equiv y \bmod p_j^{a_j}\}.$$

For a polynomial $P(X) \in \mathbb{Z}_m[X]$, we define $P_j(X) \in \mathbb{Z}_{p_j{}^{a_j}}[X]$ to be its *projection* modulo $p_j^{a_j}$ obtained by taking each coefficient of $P(X)$ modulo $p_j^{a_j}$. Conversely, given polynomials $P_j(X) \in \mathbb{Z}_{p_j{}^{a_j}}[X]$ we can combine the coefficients using the Chinese remainder theorem to get a unique polynomial $P(X) \in \mathbb{Z}_m[X]$ whose projections are the polynomials $P_j(X)$. We call $P(X)$ the *lift* of the $P_j(X)$'s.

LEMMA 11. *Let $I \subseteq \mathbb{Z}_m$. Then*

$$\bar{k}(I) \quad = \quad \max_j \ \bar{k}(I_j). \tag{5}$$

Proof. It is easy to show using the Chinese remainder theorem that a polynomial $P(X)$ is zero over I iff $P_j(X)$ is zero over I_j for all j. Let $M(X)$ be a monic polynomial which is zero over $I \subseteq \mathbb{Z}_m$. Then, by the Chinese remainder theorem, $M_j(X)$ is a monic polynomial which is 0 over $I_j \subseteq \mathbb{Z}_{p_j{}^{a_j}}$. Hence $k(I) \geq k(I_j)$ for every j.

Conversely let $\max_j \bar{k}(I_j) = d$. For each j, there is a monic polynomial $M_j(X)$ of degree $d_j \leq d$ which is zero over I_j. The polynomial

$$M'_j(X) = X^{d-d_j} M_j(X)$$

is a degree d monic polynomial which is zero over I_j. Let $M'(X) \in \mathbb{Z}_m[X]$ be the lift of the $M'_j(X)$s. It follows that $M'(X)$ is a monic polynomial of degree d and that it is zero over I. □

LEMMA 12. *Let $I \subseteq \mathbb{Z}_m$. The set S is an interpolating set for I iff S_j is an interpolating set for I_j for every j.*

Proof. Assume that S_j is an interpolating set for I_j for every j, but S is not an interpolating set for I. Then there is a polynomial $Q(X) \in \mathbb{Z}_m[X]$ such that at every point $x \in S, Q(x) \equiv 0 \bmod m$, but for some $y \in I,\ Q(y) \not\equiv 0 \bmod m$. But then $Q(y) \not\equiv 0 \bmod p_j^{a_j}$ for some j. Consider the polynomial $Q_j(X)$. Since $Q(X)$ is zero over S, $Q_j(X)$ is zero over S_j. However, there exists $y' \equiv y \bmod p_j^{a_j}$ in I_j such that $Q_j(y') \not\equiv 0 \bmod p_j^{a_j}$. This contradicts the assumption that S_j is an interpolating set for I_j.

In the other direction, assume that S is an interpolating set for I, but S_j is not an interpolating set for I_j. Then there is a polynomial $Q_j(X) \in \mathbb{Z}_{p_j{}^{a_j}}[X]$ such that for every $x \in S_j, Q(x) \equiv 0 \bmod p_j^{a_j}$, but there exists $y \in I_j$ such that $Q(y) \not\equiv 0 \bmod p_j^{a_j}$. Take $Q_i(X) = 0$ for $i \neq j$, and set $Q(X) \in \mathbb{Z}_m[X]$ to be the lift of the $Q_i(X)$'s. Then $Q(X)$ is zero over S, since it is zero over every S_j. But it is not zero at some point in I, since Q_j is not zero over I_j. This contradicts the assumption that S is an interpolating set. □

COROLLARY 13. *Let $I \subseteq \mathbb{Z}_m$. Then*

$$\max_j\ k(I_j) \quad \leq \quad k(I) \quad \leq \quad \sum_{j=1}^{t} k(I_j). \tag{6}$$

Proof. The bound $k(I) \geq k(I_j)$ follows trivially since $|S| \geq |S_j| \geq k(I_j)$. To prove the other direction, let S_j be a minimum interpolating set for I_j. For $y \in S_j$, there exists a preimage $x \in I$ such that $x \equiv y \bmod p_j^{a_j}$. Define $S'_j \subseteq I$ by choosing one preimage for each y. Set $T = \cup_j S'_j$. T is an interpolating set for I, since $S_j \subseteq T_j$ is an interpolating set for I_j. Also $|T| \leq \sum_j k(I_j)$. □

Example 3. We give an example where $\bar{k}(I) < k(I)$ and the upper bound in (6) is (near) tight. Let $m = p_1 p_2$, and let

$$I = \{ap_1 | 1 \leq a \leq p_2 - 1\} \cup \{bp_2 | 1 \leq b \leq p_1 - 1\}.$$

It is easy to see that $\bar{k}(I_1) = k(I_1) = p_1$, $\bar{k}(I_2) = k(I_2) = p_2$, and hence $\bar{k}(I) = \max(p_1, p_2)$. On the other hand, the only interpolating set for I is I itself. Each point of the form ap_1 must be included, since it is the only point in its congruence class modulo p_2. Similarly, every point bp_2 must be included. Thus $k(I) = p_1 + p_2 - 2$.

The following extension of the above lemmas can be proved similarly using the Chinese remainder theorem.

COROLLARY 14. *Let* $m = \prod_{j=1}^{t'} m_j$ *and* $(m_i, m_j) = 1$. *Let* I_j *denote the projection of* I *modulo* m_j*:*

$$\begin{aligned} \bar{k}(I) &= \max_j \bar{k}(I_j), \\ \max_j k(I_j) \leq\ k(I) &\leq \sum_j k(I_j). \end{aligned}$$

THEOREM 15. *The problem of computing* $k(I)$ *given* I *and* m *as input is* NP*-hard.*

Proof. Consider the following decision problem.

PROBLEM 2. MIN-INTERPOLATING-SET: *Given* m *and* $I \subseteq \mathbb{Z}_m$, *is* $k(I) \leq n$*?*

We prove this problem is NP-hard by reduction from 3-dimensional matching [19].

PROBLEM 3. 3-DIMENSIONAL MATCHING: *Given sets* U, V, W *of size* n *and a set of edges* $E \subseteq U \times V \times W$, *is there a subset of* E *of size* n *that covers all the vertices?*

Take p_1, p_2, p_3 to be three distinct primes greater than n. Let $m = p_1 p_2 p_3$. For each triple $(u_i, v_j, w_k) \in E$ with $i, j, k \leq n$, we add a number $x \in \mathbb{Z}_m$ to I, where $x \equiv i \bmod p_1,\ x \equiv j \bmod p_2,\ x \equiv k \bmod p_3$. We claim that there is a matching of size n iff the set I has an interpolating set of size n. We may assume that every vertex occurs in some edge, and hence $|I_1| = |I_2| = |I_3| = n$. Thus S is an interpolating set for I iff $S_j = I_j$ for $1 \leq j \leq 3$. Thus an interpolating set corresponds to a set of edges that cover every vertex. If there is an interpolating set of size n, then there is a cover of size n and vice versa.

In fact, it is possible to show that the MIN-INTERPOLATING-SET problem is NP-complete, so we skip the proof. □

In Theorem 17, we will show that for $I \subseteq \mathbb{Z}_{p^a}$, $\bar{k}(I) = k(I)$. Combining this with (5) and (6),

$$\bar{k}(I) \leq k(I) \leq t\bar{k}(I). \tag{7}$$

Thus $\bar{k}(I)$ is a factor t approximation to $k(I)$, where t is the number of prime divisors of m.

4. Algorithms for interpolation.

4.1. The prime-power case. We give an algorithm to solve the polynomial interpolation problem over $\mathbb{Z}_{p^a}$ using exactly $k(I)$ queries. We first give a (greedy) algorithm to find a minimum interpolating set.

We start by picking an arbitrary element in I. Suppose that we have chosen $\{\alpha_0, \ldots, \alpha_{i-1}\}$ so far. If the polynomial $N_i^S(X) = \prod_{j<i}(X - \alpha_j)$ is 0 over I, we stop. Else we choose the next element $\alpha_i \in I$ so that $\mathrm{val}_p(N_i(\alpha_i))$ is minimized.

ALGORITHM 1. **IntSet**(I, p^a)
Input: Set $I \subseteq \mathbb{Z}_{p^a}$.
Output: Interpolating set S for I.

Pick $\alpha_0 \in I$ arbitrarily. Set $S = \{\alpha_0\}$, $i = 1$.
Repeat
  Let $N_i^S(X) = \prod_{j<i}(X - \alpha_j)$.
  If $N_i^S(x)$ is zero for all $x \in I$,
    Output $S = \{\alpha_0, \ldots, \alpha_{i-1}\}$. Stop.
  Else
    Find $x \in I$ that minimizes $\mathrm{val}_p(N_i^S(x))$.
    Set $\alpha_i = x$, $i = i + 1$.

Assume that the algorithm outputs a set $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$ of size k, and let $e_i = \mathrm{val}_p(N_i^S(\alpha_i)$.

LEMMA 16. *Every polynomial function over I is computed by a unique polynomial of the form*

$$P(X) = \sum_{j=0}^{k-1} c_j N_j^S(X), \qquad 0 \le c_j < p^{a-e_j}. \tag{8}$$

Proof. The proof is similar to that of Lemma 6. Given any polynomial $Q(X)$, we give an algorithmic procedure to construct $P(X)$ with the above form that agrees with $Q(X)$ on I. By the termination condition, the polynomial $N_k^S(X) = \prod_{j<k}(X - \alpha_j)$ is identically zero over I. Dividing $Q(X)$ by $N_k^S(X)$ and taking the remainder, we get $Q'(X)$ of degree $k-1$ that computes the same function on I. Let us set $N_0^S(X) = 1$. Since the polynomial $N_j^S(X)$ is monic and of degree j for $j \in \{0, \ldots, k-1\}$, we can write any polynomial of degree at most $k-1$ as a linear combination of these polynomials. Hence we have

$$Q'(X) = \sum_{j<k} c_j N_j^S(X).$$

Note that, by our choice of α_j,

$$e_j \;=\; \mathrm{val}_p(N_j^S(\alpha_j)) \;\le\; \mathrm{val}_p(N_j^S(x)) \;\text{ for } x \in I.$$

So the polynomials $p^{a-e_j} N_j^S(X)$ are 0 over I. So by subtracting appropriate multiples of these polynomials from $Q'(X)$ we can get a polynomial $P(X)$, where $0 \le c_j < p^{a-e_j}$, that computes the same function as $Q(X)$.

To show uniqueness of this representation, consider two polynomials

$$P(X) = \sum_{j=0}^{k-1} c_j N_j^S(X), \qquad 0 \le c_j < p^{a-e_j},$$

$$Q(X) = \sum_{j=0}^{k-1} d_j N_j^S(X), \qquad 0 \le d_j < p^{a-e_j}$$

with different canonical forms. Pick the smallest j such that $c_j \neq d_j$. We claim that $P(\alpha_j) \neq Q(\alpha_j)$. Note that

$$P(\alpha_j) - Q(\alpha_j) = \sum_i (c_i - d_i) N_i^S(\alpha_j).$$

Since $N_i^S(\alpha_j) = 0$ for $i > j$ and $c_i = d_i$ for $i < j$, we have

$$P(\alpha_j) - Q(\alpha_j) = (c_j - d_j)N_j^S(\alpha_j).$$

Since $\mathrm{val}_p(N_j^S(\alpha_j)) = e_j$ and $\mathrm{val}(c_j - d_j) < a - e_j$, hence

$$P(\alpha_j) - Q(\alpha_j) \not\equiv 0 \bmod p^a. \qquad \square$$

THEOREM 17. *The set S is a minimum interpolating set. In fact, $\bar{k}(I) = k(I) = |S|$.*

Proof. Since S is an interpolating set of size k, $k(I) \le k$, it suffices to show that $\bar{k}(I) \ge k$. Let $M(X)$ be a monic polynomial of degree $d \le k-1$. We can put $M(X)$ in the canonical form using the procedure above to get

$$M(X) = \sum_{j \le d} c_j N_j^S(X), \qquad 0 \le c_j < p^{a-e_j}.$$

Since $M(X)$ is monic, it follows that $c_d = 1$. Thus $M(X)$ does not compute the 0 function by Lemma 16. So $\bar{k}(I) \ge k$. $\square$

Note that Algorithm 1 for picking a minimum interpolating set is essentially a greedy algorithm: at each stage it picks a new element x that minimizes $\sum_{j<i} \mathrm{val}_p(x - \alpha_j)$. One can ask what objective function is being optimized by this greedy algorithm. In Theorem 30 (proved in section 6), we prove that this algorithm minimizes the power of p that divides the Vandermonde determinant of $\prod_{i<j}(\alpha_i - \alpha_j)$. In other words, the minimum interpolating sets of I are all subsets $S = \{\beta_i\}$ of size $k(I)$ that minimize

$$\sum_{i<j \le k(I)} \mathrm{val}_p(\beta_i - \beta_j).$$

This gives a simple algorithm to check if a set $T = \{\beta_i\}$ is a minimum interpolating set for I. We first compute an interpolating set $S = \{\alpha_i\}$ using Algorithm 1 and then check that $|T| = |S|$ and that $\sum \mathrm{val}_p(\beta_i - \beta_j) = \sum \mathrm{val}_p(\alpha_i - \alpha_j)$.

We now give an algorithm for polynomial interpolation over $\mathbb{Z}_{p^a}$ whose query complexity is optimal. One can show that this algorithm computes the canonical form of $P(X)$ using an argument similar to Corollary 7.

ALGORITHM 2. **Interpolate**$(I, \mathbb{Z}_{p^a})$
Input: Set $I \subseteq \mathbb{Z}_{p^a}$, a black box for $P(X)$ evaluated at I.
Output: The polynomial $P(X)$.

Compute $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$ using IntSet(I, p^a).
For $i = 0, \ldots, k-1$,
  Query $P(\alpha_i)$.
  Compute c_i so that $0 \le c_i < p^{a-e_i}$ and

$$P(\alpha_i) \equiv \sum_{j \le i} c_j N_j^S(\alpha_i) \bmod p^a.$$

Output $P(X) = \sum_{i<k} c_i N_i^S(X)$.

4.2. The general interpolation problem. We first give an algorithm to find interpolating sets over $\mathbb{Z}_m$. The algorithm is given I as input; it does not have the factorization of m. It computes an interpolating set for I. We sketch the idea of the algorithm for $m = pq$. The algorithm tries to add elements to S greedily as in the prime-power case. Assume we have picked $\{\alpha_0, \ldots, \alpha_{i-1}\}$, and let $N_i^S(X) = \prod_{j<i}(X - \alpha_i)$. We compute $g(x) = (N_i^S(x), m)$ for every $x \in I$. This quantity plays the role of $\mathrm{val}_p(N_i^S(X))$ in Algorithm 1.

1. If there is an x such that $\mathrm{val}_p(N_i^S(X))$ and $\mathrm{val}_q(N_i^S(X))$ are both minimized at x, then $g(x)|g(y)$ for all $y \in I$. We add x to S and proceed.
2. If $\mathrm{val}_p(N_i^S(X))$ and $\mathrm{val}_q(N_i^S(X))$ are minimized at distinct points x and y, then $g(x) \not| g(y)$ and vice versa. Here the greedy approach fails. But in this case we can efficiently factor $m = pq$ using $g(x)$ and $g(y)$. We then use divide and conquer.

For general m, in case 2 we compute a factorization $m = m_1 m_2$, where $(m_1, m_2) = 1$ using the subroutine Factor, and then use divide and conquer.

ALGORITHM 3. **IntSet(I, m)**
Input: Set $I \subseteq \mathbb{Z}_m$.
Output: A factorization $m = \prod_{j=1}^{t'} m'_j$, where $(m'_i, m'_j) = 1$ and a minimum interpolating set S_j for $I_j = I \bmod m'_j$.

```
Pick α_0 ∈ I arbitrarily. Set S = {α_0}, i = 1.
Repeat
   Let N_i^S(X) = ∏_{j<i}(X − α_j).
   If N_i^S(x) is zero for all x ∈ I,
      Output S = {α_0, ..., α_{i−1}}, m. Stop.
   Else
      For each x ∈ I, set g(x) = (N_i^S(x), m).
      If some g(x) divides g(y) for all y ∈ I,
         Set α_i = x, i = i + 1.
      Else
         Find g(x), g(y) that do not divide each other.
         Factor(m, g(x), g(y)) = m_1 · m_2.
         Return IntSet(I_1, m_1), IntSet(I_2, m_2). Stop.
```

We first analyze the algorithm when Factor is not called. We then present the factoring subroutine. In particular, Lemmas 18, 19, and 20 all assume that Factor was not called. In this case, the behavior of the algorithm is similar to the prime-power case.

Let $m = \prod_{j=1}^{t} p_j^{a_j}$. Let $S = \{\alpha_i\}$ be the set output. Let I_j and S_j be the projections of I and S modulo $p_j^{a_j}$. We show that if Factor is not called, then the algorithm finds a minimum interpolating set by showing $\bar{k}(I) = k(I) = |S|$. This is done by simulating Algorithm 1 on I_j and showing that it would produce the same outcome.

Fix a prime p_j. Let $\alpha'_i \equiv \alpha_i \bmod p_j^{a_j}$. We take T to be the projection of the first $k(I_j)$ elements of I. In other words, let $T = \{\alpha'_1, \ldots, \alpha'_{k(I_j)}\}$. Note that $T \subseteq S_j$.

LEMMA 18. *The set T is a minimum interpolating set for I_j.*

Proof. We will show that $\mathrm{val}_{p_j}(N_i^T(x))$ is minimized over I_j at α'_i. Hence the set T is a possible output when we run Algorithm 1 on the set I_j.

Assume that there is a $y' \in I_j$ such that

$$\mathrm{val}_{p_j}(N_i^T(y')) < \mathrm{val}_{p_j}(N_i^T(\alpha_i')).$$

Choose $y \in I$ so that $y \equiv y' \bmod p_j^{a_j}$. Note that

$$\mathrm{val}_{p_j}(N^T(y')) = \mathrm{val}_{p_j}(g(y)), \qquad \mathrm{val}_{p_j}(N^T(\alpha_i')) = \mathrm{val}_{p_j}(g(\alpha_i));$$

$$\text{hence} \qquad \mathrm{val}_{p_j}(g(y)) < \mathrm{val}_{p_j}(g(\alpha_i)).$$

So $g(\alpha_i)$ cannot divide $g(y)$. But since Factor is not used, α_i satisfies $g(\alpha_i)|g(y)$ for all $y \in I$, which is a contradiction. □

LEMMA 19. *The set S is a minimum interpolating set for I. In fact, $\bar{k}(I) = k(I) = |S|$.*

Proof. By Lemma 18, the set S_j is an interpolating set for I_j, so S is an interpolating set for I. We will show that $\bar{k}(I) = k(I) = |S|$.

For each j, the polynomial $\prod_{i<k(I_j)}(X-\alpha_i)$ is 0 mod $p_j^{a_j}$ over I_j because the first $k(I_j)$ elements are an interpolation set for I_j. Take $k = \max_j k(I_j)$. The polynomial $\prod_{i<k}(X-\alpha_i)$ is 0 mod m over I. Since this is the termination condition for Algorithm 3, it will stop after k steps and output S of size $k = \max_j k(I_j)$. By (6), we have

$$\max_j k(I_j) \;\le\; k(I).$$

Hence the set S is a minimum interpolating set. Further, by (5),

$$\bar{k}(I) \;= \max_j \bar{k}(I_j).$$

But for prime powers, $\bar{k}(I_j) = k(I_j)$. So we conclude that

$$\bar{k}(I) \;=\; \max_j \bar{k}(I_j) \;=\; \max_j k(I_j) \;=\; k(I). \qquad \square$$

$P(X)$ can be computed by a procedure similar to Algorithm 2.

LEMMA 20. *The polynomial $P(X)$ can be computed from the values at points in S.*

Proof. The proof of correctness is similar to Corollary 7. For $i \le k$, the polynomials $\frac{m}{(m,N_i^S(\alpha_i))}N_i^S(X)$ are 0 over I. This is because

$$\frac{m}{(m, N_i^S(\alpha_i))} N_i^S(\alpha_i) \equiv 0 \bmod m$$

and, for every $x \in I$,

$$N_i^S(x) = y \cdot N_i^S(\alpha_i) \bmod m \quad \text{for some } y \in \mathbb{Z}_m.$$

Hence every polynomial function over I can be canonically represented as

$$P(X) = \sum_{i=0}^{k-1} c_i N_i^S(X), \qquad 0 \le c_i < \frac{m}{(m, N_i^S(\alpha_i))}.$$

To compute the canonical form of $P(X)$, we query the value of $P(X)$ at every point in S. For $i \leq k-1$ we solve the equation

$$c_i N_i^S(\alpha_i) \equiv P(\alpha_i) - \sum_{j<i} c_j N_j^S(\alpha_i) \bmod m, \qquad 0 \leq c_i < \frac{m}{(m, N_i^S(\alpha_i))}.$$

The unique solution to this system is the canonical from of $P(X)$. □

We now turn to the subroutine for factoring. The idea is to use $g(x)$ and $g(y)$ to get m_1, m_2, which divide m and are relatively prime. Their product $m_1 m_2$ might be less than m. At each step, we take a nontrivial divisor of $\frac{m}{m_1 m_2}$ and multiply either m_1 or m_2 by it. We do this in such a way that they stay relatively prime.

ALGORITHM 4. **Factor(m,g(x),g(y))**
Input: A number m and $g(x), g(y)$ that divide m but do not divide each other.
Output: $m_1 \cdot m_2 = m$ and $(m_1, m_2) = 1$.

Let $g = (g(x), g(y))$. Let $m_1 = \frac{g(x)}{g}$, $m_2 = \frac{g(y)}{g}$.
Repeat
  Set $c = \frac{m}{m_1 \cdot m_2}$.
  If $(c, m_1) = 1$, Set $m_2 = m_2 \cdot c$.
  Else, Set $m_1 = m_1 \cdot (c, m_1)$.
  If $m_1 \cdot m_2 = m$, Output m_1, m_2. Stop.

LEMMA 21. *Factor(m, g(x), g(y)) returns m_1, m_2 such that $m_1 \cdot m_2 = m$ and $(m_1, m_2) = 1$.*

Proof. At the start of the algorithm,

$$m_1 = \frac{g(x)}{(g(x), g(y))}, \qquad m_2 = \frac{g(y)}{(g(x), g(y))},$$

so $(m_1, m_2) = 1$. Also m_1, m_2 are nontrivial divisors of m, since $g(x)$ and $g(y)$ do not divide each other.

Let

$$c = \frac{m}{m_1 m_2}.$$

If $(m_1, c) = 1$, since $(m_1, m_2) = 1$, we have $(m_1, cm_2) = 1$. So we set $m_2 = cm_2$, and we are done. If $(c, m_1) = d > 1$, then since d divides m_1, we have $(d, m_2) = 1$. So we set $m_1 = dm_1$. In either case, the product $m_1 m_2$ increases by a factor of 2, and hence the algorithm terminates in $O(\log m)$ iterations. □

The subroutine above is somewhat inefficient. The running time can be considerably improved by running the factor refinement algorithm of Bernstein [12] on $g(x), g(y)$, and m. This algorithm gives a factorization into coprimes in near linear time.

If we find factors m_1, m_2 which are relatively prime, then we run IntSet(I_1, m_1) and IntSet(I_2, m_2). In doing so we could find further factors of m_1 and m_2, but these will be relatively prime, since m_1 and m_2 are relatively prime. So finally, the algorithm returns a factorization $m = \prod_{i \leq t'} m_i'$, where the m_i''s are relatively prime. If m has t distinct prime factors, then clearly $t' \leq t$. We now solve the interpolation problem

modulo m'_j using Lemma 20 and combine the results using the Chinese remainder theorem.

ALGORITHM 5. **Interpolate**$(I, \mathbb{Z}_m)$
Input: Set $I \subseteq \mathbb{Z}_m$, a black box for $P(X)$ evaluated at I.
Output: The polynomial $P(X)$.

Using IntSet(I, m), compute $m = \prod_{j=1}^{t'} m'_j$ and interpolating sets S_j for I_j.
For each $j \in 1, \ldots, t'$
  For each $y \in S_j$,
    Query $P(X)$ at $x \in I$ so that $x \equiv y \bmod m'_j$.
  Use these to compute $P_j(X) \bmod m'_j$.
Lift the polynomials $P_j(X)$ to a polynomial $P(X) \in \mathbb{Z}_m[X]$ using Chinese remaindering.

LEMMA 22. *Algorithm* 5 *solves the interpolation problem over* $\mathbb{Z}_m[X]$. *The number of queries is within a factor* t' *of the optimal.*

Proof. The proof that the polynomial $P(X)$ is correct follows by the Chinese remainder theorem. The number of queries is at most $\sum_{j \le t'} |S_j|$. By Lemma 19, since each m_j is not factored further, the set S_j is a minimum interpolating set for I_j. Hence $|S_j| = k(I_j)$. Hence by Corollary 14,

$$\max_j |S_j| \;\le\; k(I) \;\le\; \sum_{j \le t'} |S_j| \;\le\; t' k(I). \qquad \square$$

Also by Corollary 14,

$$\bar{k}(I) \;=\; \max_j \bar{k}(I_j) \;=\; \max_j |S_j|.$$

Hence Algorithm 3 can be used to compute $\bar{k}(I)$ exactly. (Note that this can also be done by solving a system of linear equations.)

5. Learning algorithms. One can use the algorithms in the previous section to design efficient algorithms for interpolation over $\mathbb{Z}_m$ in various learning theoretic settings. We consider the problem of learning under the uniform distribution and PAC learning under an arbitrary distribution. In the uniform distribution problem, we are given evaluations of a polynomial $P(X)$ at points x chosen at random from $\mathbb{Z}_m$. In the PAC-learning problem, the samples are drawn from an unknown distribution $\mathcal{D}$ over $\mathbb{Z}_m$. We are required to output a polynomial that computes $P(X)$ correctly with good probability on points chosen from the same distribution. In this setting, it is necessary to allow some error probability. Consider a distribution D which is concentrated on a set I which does not contain an interpolating set for $\mathbb{Z}_m$. A polynomial time algorithm cannot distinguish between the 0 function and a function which is 0 on I but nonzero elsewhere.

For learning algorithms, the notion of polynomial running time needs to be defined carefully. Let $F(m)$ denote the number of polynomial functions over $\mathbb{Z}_m$. The algorithm is required to output some polynomial function which requires at least $\log F(m)$ bits to represent. Hence we say the algorithm runs in polynomial time if the running time is $\text{poly}(\log F(m))$.

From Lemma 6, we get

$$F(m) = \prod_{0 \le j < k(m)} \frac{m}{(m, j!)}. \tag{9}$$

Note that $\log F(m)$ can vary between $\log m$ and m depending on the prime factorization of m. We compute a rough lower bound on $\log F(m)$ in terms of its factorization. Note that, if $m = p^a$, it follows from Lemma 6 that

$$F(p^a) \ \geq \ (p^a)^p \ = \ p^{ap}.$$

Hence if $m = \prod_{j \le t} p_j^{a_j}$, then

$$F(m) \geq \prod_j p_j^{a_j p_j} \ \Rightarrow \ \log F(m) \geq \sum_{j \le t} a_j p_j \log p_j.$$

5.1. Learning under the uniform distribution. We first consider the problem of learning polynomials from their evaluations at random points in $\mathbb{Z}_m$.

Problem 4. Learning under the Uniform Distribution: *Given samples $(x, f(x))$, where x is drawn uniformly from $\mathbb{Z}_m$ and f is a polynomial function, find a polynomial $P(X)$ that computes f.*

Algorithm 6. **Interpolation under the Uniform Distribution**
Input: Black box for evaluations of $P(X)$ under the uniform distribution.
Output: The polynomial $P(X)$.

Compute the factorization $m = \prod_{j=1}^{t} p_j^{a_j}$.
Draw samples until we have an interpolating set S_j for $\mathbb{Z}_{{p_j}^{a_j}}$.
Compute $P_j(X)$ for each j.
Let $P(X)$ be the lift of the $P_j(X)$'s.

We compute the factorization using brute force which takes time $O(\sum_j p_j a_j) = O(F(m))$. We now bound the number of samples needed until we have an interpolating set modulo p^a. Let p^b be the smallest power of p such that $p^b > k(p^a)$. By Lemma 8, $p^b < p^2 a$. Let $S = \{\alpha_0, \ldots, \alpha_{k(p^a)-1}\}$, where $\alpha_i \equiv i \bmod p^b$.

Lemma 23. *The set S is an interpolating set for $\mathbb{Z}_{p^a}$.*

Proof. By Corollary 7, $T = \{0, \ldots, k(p^a) - 1\}$ is an interpolating set for $\mathbb{Z}_{p^a}$. By the choice of α_i, α_j,

$$\alpha_i - \alpha_j \ \equiv \ i - j + cp^b \bmod p^a.$$

Since $0 \le i \ne j < p^b$, $\mathrm{val}_p(\alpha_i - \alpha_j) = \mathrm{val}_p(i - j)$. Hence

$$\sum_{i < j \le k(p^a)} (\alpha_i - \alpha_j) = \sum_{i < j \le k(p^a)} (i - j).$$

So, by Theorem 30, S is an interpolating set. ∎

Lemma 24. *Algorithm 6 requires $O(\log^2 F(m))$ samples with high probability.*

Proof. The uniform distribution over $\mathbb{Z}_m$ induces the uniform distribution over congruence classes modulo $p_j^{b_j}$, since $b_j < a_j$. By the coupon collector's problem,

in time $O(p_j^{b_j} \log(p_j^{b_j}))$ we will see a sample from each congruence class with high probability. By Lemma 23, this gives an interpolating set modulo $p_j^{a_j}$. Overall the number of samples needed can be bounded by $O(\log^2 F(m))$ with high probability. □

THEOREM 25. *Algorithm* 6 *learns the polynomial* $P(X)$ *exactly under the uniform distribution. It runs in time* $O(\log^2 F(m))$ *with high probability.*

We needed to factor m to check whether the set of points seen so far is an interpolating set for $\mathbb{Z}_m$. Is there an algorithm to check if S is an interpolating set for $\mathbb{Z}_m$ that does not need to factor m?

5.2. PAC learning. Next we consider the problem of PAC-learning polynomials. We refer the reader to the book of Kearns and Vazirani for details of the PAC-learning model [29].

PROBLEM 5. PAC LEARNING: *Given samples* $(x, f(x))$*, where* x *is drawn from an unknown distribution* $\mathcal{D}$ *and* f *is a polynomial function, find a polynomial* $P(X)$ *that computes* f *over* $\mathcal{D}$ *with probability* $1 - \varepsilon$.

We show that polynomial functions are PAC learnable under an arbitrary distribution in polynomial time. Once we have drawn the set of samples, the problem reduces to one of general interpolation. The number of samples to be drawn can be determined from $F(m)$ using Occam's razor [29]. We first compute $F(m)$ using (9). This can be done in time $O(\log F(m))$. We then draw $\frac{1}{\varepsilon} \log \frac{F(m)}{\delta}$ samples from $\mathcal{D}$ and solve the interpolation problem on these inputs using Algorithm 5. The proof that this suffices for PAC learning is standard [29].

THEOREM 26. *Polynomials over* $\mathbb{Z}_m$ *are PAC learnable in polynomial time using* $\frac{1}{\varepsilon} \log \frac{F(m)}{\delta}$ *queries.*

6. Algebraic structure of interpolating sets modulo prime powers. In this section we study the algebraic properties of interpolating sets modulo prime powers. We give alternate algebraic characterizations of such sets (Theorems 29 and 30). In this section and the next, we use p-adic distance as opposed to valuations.

Recall that by the definition of interpolating sets, every polynomial which is nonzero over I is in fact nonzero over some point in S. The next lemma generalizes this to show that, in fact, the norm of every polynomial is maximized over I at some point in S.

LEMMA 27. *A set* S *is an interpolating set iff, for any polynomial* $P(X)$*, there exists* $\alpha \in S$ *such that*

$$|P(\alpha)|_p \geq |P(x)|_p \qquad \forall x \in I. \tag{10}$$

Proof. ($\Rightarrow$) Assume there exists $P(X) \in \mathbb{Z}_{p^a}$ such that $|P(x)|_p > |P(\alpha)|_p$ for all $\alpha \in S$. But then for an appropriately chosen e, $p^e P(X)$ is nonzero at x but 0 everywhere in S. Hence S cannot be an interpolating set.

($\Leftarrow$) Assume that S satisfies (10). There cannot exist a polynomial $P(X)$ which is 0 on S, but $P(x) \neq 0$ for some $x \in \mathbb{Z}_{p^a}$, since this implies that $|P(x)|_p > |P(\alpha)|_p$ for all $\alpha \in S$. Hence S is an interpolating set. □

This property of interpolating sets allows us to order its elements in a natural manner. Given an ordered set $S = \{\alpha_0, \alpha_1, \ldots\}$, let $N_j^S(X) = \prod_{i<j}(X - \alpha_i)$.

ALGORITHM 7. **Ordering an Interpolating Set**
Input: An interpolating set T of I.
Output: An ordered set $S \subseteq T$ which is a (minimal) interpolating set.

Pick $\alpha_0 \in T$ arbitrarily, and put it in S.
Given $S = \{\alpha_0, \ldots, \alpha_{j-1}\}$.
  If $N_j^S(X)$ is 0 over T, stop and output $S = \{\alpha_0, \ldots, \alpha_{j-1}\}$.
  Else pick $\alpha_j \in T$, which maximizes $|N_j^S(\alpha_j)|_p$, and add it to S.

Assume that the above procedure outputs an ordered set $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$ of size k. Let $e_j = \mathrm{val}_p(N_j^S(\alpha_j))$ for $i \le j \le k-1$. Observe that $0 \le e_j < a$. Using the argument of Lemma 16, we can show that every polynomial function over I is computed by a unique polynomial of the form

$$P(X) = \sum_{j=0}^{t} c_j N_j^S(X), \qquad 0 \le c_j < p^{a-e_j}.$$

Using the canonical form above, one can show that all minimal interpolating sets over $\mathbb{Z}_{p^a}$ have the same size. The proof is similar to that of Theorem 17.

COROLLARY 28. *The set $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$ is an interpolating set iff $|N_j^S(\alpha_j)|_p \ge |N_j^S(x)|_p$ for all $x \in I$.*

Proof. Clearly an interpolating set with the canonical ordering has this property. To prove the other direction, simply take $T = I$ in Algorithm 7. Since $|N_j^S(x)|_p$ is maximized at α_j, we can add α_j to S at step j, giving the interpolating set $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$. □

Henceforth we will assume that interpolating sets are canonically ordered and that polynomials are in the canonical form. Lemma 27 states that for any polynomial function $P(X)$, $|P(x)|_p$ is maximized at some point $\alpha \in S$. We strengthen this to show that if the degree of $P(X)$ is d, such an α can be found among the first $d+1$ elements in S.

THEOREM 29. *The set $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$ is an interpolating set iff, for every polynomial $P(X)$ of degree d, there exists $\alpha \in \{\alpha_0, \ldots, \alpha_d\}$ such that $|P(\alpha)|_p \ge |P(x)|_p$ for all $x \in I$.*

Proof. Clearly a set with this property is an interpolating set by Lemma 27. We prove the other direction. The proof is by induction on d. The base case when $d = 0$ is trivial. Assume the claim holds for $d-1$. Let $P(X) = Q(X) + c_d N_k^S(X)$, where $\deg(Q) \le d-1$.

$$|P(x)|_p \le \max(|Q(x)|_p, |c_d N_d^S(x)|_p) \qquad \text{(ultrametric inequality)}. \tag{11}$$

We bound $|Q(x)|_p$ using the inductive hypothesis. Since $Q(X)$ has degree $d-1$,

$$|Q(x)|_p \le \max(|Q(\alpha_0)|_p, \ldots, |Q(\alpha_{d-1})|_p). \tag{12}$$

By our choice of α_d,

$$\begin{aligned} |c_d N_d^S(x)|_p &\le |c_d N_d^S(\alpha_d)| \\ &= |P(\alpha_d) - Q(\alpha_d)|_p \\ &\le \max(|Q(\alpha_d)|_p, |P(\alpha_d)|_p) \\ &\le \max(|Q(\alpha_0)|_p, \ldots, |Q(\alpha_{d-1})|_p, |P(\alpha_d)|_p) \qquad \text{(induction on } Q(X)). \end{aligned} \tag{13}$$

Hence from (11), (12), and (13) we get

$$|P(x)|_p \le \max(|Q(\alpha_0)|_p, \ldots, |Q(\alpha_{d-1})|_p, P(\alpha_d)|_p).$$

Since $N_d^S(\alpha_j) = 0$ for $j < d$, we have $Q(\alpha_j) = P(\alpha_j)$ for $j < d$. Hence

$$|P(x)|_p \le \max(|P(\alpha_0)|_p, \ldots, |P(\alpha_{d-1})|_p, |P(\alpha_d)|_p). \qquad (14)$$ □

We use this to show that interpolating sets are greedy solutions for the problem of maximizing the p-adic norm of the Vandermonde determinant. Since the determinant could vanish mod p^a, we define the norm of the Vandermonde determinant as follows:

$$\text{Let} \quad |V(\alpha_0, \ldots, \alpha_{k-1})|_p \;=\; \prod_{0 \le i < j \le k-1} |(\alpha_i - \alpha_j)|_p \;=\; \prod_{j=1}^{k-1} |N_j^S(\alpha_j)|_p.$$

This is equivalent to regarding the determinant as an integer and taking its norm.

THEOREM 30. *The set $S = \{\alpha_0, \ldots, \alpha_{k-1}\}$ is an interpolating set for I iff, for all subsets $\{x_0, \ldots, x_{k-1}\}$ of I,*

$$|V(\alpha_0, \ldots, \alpha_{k-1})|_p \ge |V(x_0, \ldots, x_{k-1})|_p. \qquad (15)$$

Proof. ($\Rightarrow$). We will show a stronger statement: for $1 \le j \le k-1$, for any subset $\{x_0, \ldots, x_j\}$ of I,

$$|V(\alpha_0, \ldots, \alpha_j)|_p \ge |V(x_0, \ldots, x_j)|_p. \qquad (16)$$

Consider the polynomial $Q(X) = \prod_{i<j}(X - x_i)$. By Theorem 29, there exists $\alpha_i \in \{\alpha_0, \ldots, \alpha_j\}$ such that $|Q(\alpha_i)|_p \ge |Q(x_j)|_p$. Hence one can replace x_j by α_i without decreasing the norm of the Vandermonde determinant. Now repeat the same argument for the set $\{x_0, \ldots, x_{j-1}, \alpha_i\}$ and the element x_{j-1} and so on. We get

$$|V(\alpha_0, \ldots, \alpha_j)|_p \;\ge\; \cdots \;\ge\; |V(x_0, \ldots, x_{j-1}, \alpha_i)|_p \;\ge\; |V(x_0, \ldots, x_j)|_p.$$

($\Leftarrow$). Assume we have a set S satisfying (15). Assume that the α_i's are ordered canonically. We will show that $|N_j^S(\alpha_j)|_p \ge |N_j^S(x)|_p$ for all $x \in I$. This implies S is an interpolating set by Corollary 28.

Assume there exists β such that $|N_j^S(\alpha_j)|_p < |N_j^S(\beta)|_p$. Pick $\alpha_i \in \{\alpha_j, \ldots, \alpha_{k-1}\}$ such that $|\beta - \alpha_i|_p$ is minimized. We will show that replacing α_i by β will increase the norm of the Vandermonde determinant. Observe that, for any $\ell \ne i$ and $\ell \ge j$,

$$\begin{aligned} |\alpha_\ell - \alpha_i|_p &\le \max(|\beta - \alpha_\ell|_p, |\beta - \alpha_i|_p) \\ \text{but} \quad |\alpha_\ell - \beta|_p &\ge |\beta - \alpha_i|_p \qquad \text{by choice of } \alpha_i; \\ \text{hence} \quad |\alpha_i - \alpha_\ell|_p &\le |\beta - \alpha_\ell|_p \\ \Rightarrow \prod_{\ell > j, \ell \ne i} |\alpha_i - \alpha_\ell|_p &\le \prod_{\ell \ge j, \ell \ne i} |\beta - \alpha_\ell|_p. \end{aligned} \qquad (17)$$

We also have

$$|N_j^S(\alpha_i)|_p \;\le\; |N_j^S(\alpha_j)|_p \;<\; |N_j^S(\beta)|_p.$$

The first inequality is because S is ordered canonically, and the second is by the definition of β. Hence, from the definition of $N_j^S(X)$,

$$\left|\prod_{\ell<j}(\alpha_i-\alpha_\ell)\right|_p \;<\; \left|\prod_{\ell<j}(\beta-\alpha_\ell)\right|_p . \tag{18}$$

Combining (17) and (18), we get

$$\left|\prod_{\ell\neq i}(\alpha_i-\alpha_\ell)\right|_p \;<\; \left|\prod_{\ell\neq i}(\beta-\alpha_\ell)\right|_p . \tag{19}$$

Hence replacing α_i with β increases the norm of the Vandermonde determinant, which contradicts the assumption that the norm is maximized at S. □

COROLLARY 31. *The parameters $e_1,\ldots,e_{k-1}$ are independent of the choice of interpolating set.*

Proof. Note that

$$|N_j^S(\alpha_j)| \;=\; p^{-e_j}$$

and

$$|V(\alpha_0,\ldots,\alpha_j)|_p \;=\; \prod_{i\leq j}|N_i^S(\alpha_i)|_p \;=\; p^{-\sum_{i\leq j}e_i}.$$

This quantity is maximized over subsets of I at every interpolating set. So $\sum_{i\leq j}e_i$, and hence e_i is the same for every interpolating set of I. □

7. Some combinatorial properties of ultrametric spaces. We show that many of our results for interpolating sets can be translated into properties of general ultrametric spaces. Further, the proof of these properties for general ultrametric spaces follows directly from the proof for polynomials over $\mathbb{Z}_{p^a}$.

DEFINITION 3. *Let T be a tree rooted at a vertex r such that the distances of all leaves from the root r are equal. The metric space whose points are the leaves of the tree and distance is the shortest path in the tree is called an* equidistant tree *and is denoted by (T,d).*

It is easy to show that (T,d) is an ultrametric . In fact, the converse is also true.

FACT 32. *Every finite ultrametric space embeds isometrically into an equidistant tree.*

Every equidistant tree can in turn be associated with $I\subseteq\mathbb{Z}_{p^a}$ for appropriate choices of p,a, and I.

LEMMA 33. *There is a mapping from any equidistant tree T to $I\subseteq\mathbb{Z}_{p^a}$ for some p,a such that*

$$|x-y|_p = p^{\frac{d(x,y)}{2}-a} \quad \textit{for } x\neq y.$$

Proof. There is a natural way to associate $\mathbb{Z}_{p^a}$ with an equidistant tree of degree p and depth a [6]. The root is at depth 0. The edges from each vertex to its descendants are labeled $\{0,\ldots,p-1\}$. Given a point $x=\sum_i x_ip^i\in\mathbb{Z}_{p^a}$, we associate it with a leaf of the tree as follows: Start from the root. At depth i, follow the edge labeled x_i. Thus the leaf nodes correspond to points in $\mathbb{Z}_{p^a}$, while nodes at depth d correspond

to congruence classes modulo p^d. If $d(x,y)$ is the tree distance between the points, then $|x-y|_p = p^{\frac{d(x_i,x_j)}{2}-a}$.

Given an equidistant tree T, we take p to be a prime larger than the maximum degree of T and a to be the depth of the tree. For any node, we arbitrarily label the edges to its descendants with $\{0,\dots,p-1\}$. This can be done, since there are at most p of them. This will map the leaves of T to $I \subseteq \mathbb{Z}_{p^a}$, and it is easy to verify that the distance satisfies the desired condition. □

Based on this correspondence, we can translate properties of interpolating sets into properties of ultrametric spaces. We consider the following NP-hard optimization problem.

PROBLEM 6. MAX-DIST-K: *Given a metric space (X,d), pick a subset S of k points such that the sum of pairwise distances is maximized.*

For ultrametrics, the problem can be solved greedily.

ALGORITHM 8. **Greedy Algorithm for Max-Dist-k**
Input: An n point ultrametric space (T,d).
Output: A subset S of size k maximizing the sum of pairwise distances.

Pick $\alpha_0 \in T$ arbitrarily.
For $j \le k-1$,
  Pick α_j so that $\sum_{i<j} d(\alpha_j,\alpha_i)$ is minimized.
Output $S = \{\alpha_0,\dots,\alpha_{k-1}\}$.

LEMMA 34. *The greedy algorithm solves Max-Dist-k over ultrametric spaces.*

Proof. Associate T with $I \subseteq \mathbb{Z}_{p^a}$. For any subset $(x_0,\dots,x_{k-1})$ of size k,

$$\prod_{i<j} |x_i - x_j|_p = p^{\sum_{i,j} \frac{d(x_i,x_j)}{2} - \binom{k}{2}a}.$$

Hence MAX-DIST-K on an ultrametric reduces to choosing k points in I such that $|V(x_0,\dots,x_{k-1})|_p$ is maximized; by Theorem 30 this can be done by choosing the points greedily. □

Next we consider the problem of finding a point in a metric space that is farthest from a given set of points.

PROBLEM 7. FARTHEST-POINT: *Given a metric space (X,d) and a set of points $\{y_1,\dots,y_{k-1}\}$ of size $k-1$, find the point $x \in X$ that maximizes $\sum_{i<k} d(x,y_i)$.*

This problem is easy to solve for arbitrary metric spaces; we can just try every point in X and pick the best. However, ultrametric spaces admit a more efficient solution with some preprocessing. In the preprocessing step, we find a solution S to MAX-DIST-K using the greedy algorithm. This step is oblivious of the y_i's. We then return the point $x \in S$ that maximizes $\sum_{i<k} d(x,y_i)$. The running time of this step depends only on k; it is independent of the number of points n.

LEMMA 35. *Let $\{y_1,\dots,y_{k-1}\}$ be any subset of size $k-1$ in X. Let $S = \{\alpha_0,\dots,\alpha_{k-1}\}$ be a greedy solution to Max-Dist-k. The quantity $\sum_{i<k} d(x,y_i)$ is maximized over X at a point $\alpha \in S$.*

Proof. Associate X with $I \subseteq \mathbb{Z}_{p^a}$. Given points y_i, consider the polynomial

$$P(X) = \prod_{i<k} (X - y_i).$$

Note that

$$|P(x)|_p = p^{\sum \frac{d(x,y_i)}{2} - a(k-1)}.$$

Hence, maximizing the distance is equivalent to maximizing $|P(x)|_p$. Since $P(X)$ is of degree $k-1$, by Theorem 29 its norm over I is maximized at some point in $\{\alpha_0, \ldots, \alpha_{k-1}\}$. □

The case $k = 2$ of this lemma is a direct consequence of the ultrametric inequality. We are unaware of a direct combinatorial proof of Lemma 35 for $k \geq 3$ and higher.

Acknowledgments. We thank Sophie Frisch, Igor Shparlinski, and an anonymous referee for many useful references and Saugata Basu and Ernie Croot for helpful discussions. We thank T. S Jayram, Ravi Kumar, and Satyen Kale for their comments, which greatly improved the presentation. We thank Dick Lipton for numerous illuminating discussions on this and related problems.

REFERENCES

[1] M. Agrawal and S. Biswas, *Primality and identity testing via Chinese remaindering*, J. ACM, 50 (2003), pp. 429–443.
[2] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math. (2), 160 (2004), pp. 781–793.
[3] N. Alon, *The Shannon capacity of a union*, Combinatorica, 18 (1998), pp. 301–310.
[4] N. Alon and R. Beigel, *Lower bounds for approximations by low degree polynomials over $\mathbb{Z}_m$*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity (CCC'01), 2001.
[5] L. Babai and P. Frankl, *Linear Algebra Methods in Combinatorics*, preliminary version 2, University of Chicago, Chicago, IL, 1992.
[6] L. Babai, P. Frankl, S. Kutin, and D. Štefankovič, *Set systems with restricted intersections modulo prime powers*, J. Combin. Theory Ser. A, 95 (2001), pp. 39–73.
[7] D. A. Barrington, R. Beigel, and S. Rudich, *Representing Boolean functions as polynomials modulo composite numbers*, Comput. Complexity, 4 (1994), pp. 367–382.
[8] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS'96), 1996, pp. 184–193.
[9] Y. Bartal, *On approximating arbitrary metrices by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC'98), 1998, pp. 161–168.
[10] Y. Bartal and M. Mendel, *Dimension reduction for ultrametrics*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04), 2004, pp. 664–665.
[11] D. J. Bernstein, *Detecting perfect powers in essentially linear time*, Math. Comp., 67 (1998), pp. 1253–1283.
[12] D. J. Bernstein, *Factoring into coprimes in essentially linear time*, J. Algorithms, 54 (2005), pp. 1–30.
[13] N. Bhatnagar, P. Gopalan, and R. J. Lipton, *Symmetric polynomials over $\mathbb{Z}_m$ and simultaneous communication protocols*, J. Comput. System Sci., 72 (2006), pp. 252–285.
[14] N. H. Bshouty, C. Tamon, and D. K. Wilson, *Learning matrix functions over rings*, Algorithmica, 22 (1998), pp. 91–111.
[15] P.-J. Cahen and J.-L. Chabert, *Integer-Valued Polynomials*, Math. Surveys Monogr. 48, AMS, Providence, RI, 1996.
[16] A. Chattopadhyay, N. Goyal, P. Pudlak, and D. Therien, *Lower bounds for circuits with MOD_m gates*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 709–718.
[17] P. Frankl and R. Wilson, *Intersection theorems with geometric consequences*, Combinatorica, 1 (1981), pp. 357–368.
[18] S. Frisch, *Polynomial functions on finite commutative rings*, in Advances in Commutative Ring Theory (Fez, 1997), Lecture Notes in Pure and Appl. Math. 205, Dekker, New York, 1999, pp. 323–336.
[19] M. Garey and D. Johnson, *Computers and Intractability*, Freeman, New York, 1979.

[20] O. Goldreich, R. Rubinfeld, and M. Sudan, *Learning polynomials with queries: The highly noisy case*, SIAM J. Discrete Math., 13 (2000), pp. 535–570.
[21] P. Gopalan, *Constructing Ramsey graphs from Boolean function representations*, in Proceedings of the 21st IEEE Conference on Computational Complexity (CCC'06), 2006.
[22] P. Gopalan, *Query-efficient algorithms for polynomial interpolation over composites*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 908–917.
[23] A. Granville, *Arithmetic properties of binomial coefficients*, in Organic Mathematics, CMS Conf. Proc. 20, AMS, Providence, RI, 1997, pp. 253–276.
[24] V. Grolmusz, *Superpolynomial size set-systems with restricted intersections mod* 6 *and explicit Ramsey graphs*, Combinatorica, 20 (2000), pp. 71–86.
[25] V. Grolmusz, *Constructing set systems with prescribed intersection sizes*, J. Algorithms, 44 (2002), pp. 321–337.
[26] V. Grolmusz and G. Tardos, *Lower bounds for (*MODp-MODm*) circuits*, SIAM J. Comput., 29 (2000), pp. 1209–1222.
[27] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Clarendon Press, Oxford, UK, 1985.
[28] M. Karpinski, A. van der Poorten, and I. Shparlinski, *Zero testing of p-adic and modular polynomials*, Theoret. Comput. Sci., 233 (2000), pp. 309–317.
[29] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory*, MIT Press, Cambridge, MA, 1994.
[30] A. Kempner, *Polynomials and their residue systems*, Trans. Amer. Math. Soc., 22 (1921), pp. 240–288.
[31] W. Narkiewicz, *Polynomial Mappings*, Lecture Notes in Math. 1600, Springer-Verlag, Berlin, 1995.
[32] A. Razborov, *Lower bounds for the size of circuits of bounded depth with basis* $\{\wedge, \oplus\}$, Mathematical Notes of the Academy of Science of the USSR, 41 (1987), pp. 333–338.
[33] A. Shamir, *On the generation of multivariate polynomials which are hard to factor*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC'93), 1993, pp. 796–804.
[34] R. Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC'87), 1987, pp. 77–82.
[35] R. Spira, *Polynomial interpolation over commutative rings*, Amer. Math. Monthly, 75 (1968), pp. 638–640.
[36] G. Tardos and D. Barrington, *A lower bound on the mod* 6 *degree of the OR function*, Comput. Complexity, 7 (1998), pp. 99–108.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1058–1079

EXACT ALGORITHMS FOR TREEWIDTH AND MINIMUM FILL-IN*

FEDOR V. FOMIN†, DIETER KRATSCH‡, IOAN TODINCA§, AND YNGVE VILLANGER†

Abstract. We show that the treewidth and the minimum fill-in of an n-vertex graph can be computed in time $\mathcal{O}(1.8899^n)$. Our results are based on combinatorial proofs that an n-vertex graph has $\mathcal{O}(1.7087^n)$ minimal separators and $\mathcal{O}(1.8135^n)$ potential maximal cliques. We also show that for the class of asteroidal triple–free graphs the running time of our algorithms can be reduced to $\mathcal{O}(1.4142^n)$.

Key words. exact exponential algorithm, treewidth, fill-in, minimal separators, potential maximal clique, minimal triangulation

AMS subject classifications. 05C35, 05C85, 68R10, 68W40

DOI. 10.1137/050643350

1. Introduction. Exact exponential algorithms.

The interest in exact (fast) exponential algorithms dates back to Held and Karp's paper [28] on the travelling salesman problem in the early 1960s. We mention just a few examples: an $\mathcal{O}(1.4422^n)$ time algorithm for knapsack (Horowitz and Sahni [29]); $\mathcal{O}(1.2600^n)$ and $\mathcal{O}(1.2109^n)$ time algorithms for independent set (Tarjan and Trojanowski [43] and Robson [40]); 3-coloring in time $\mathcal{O}(1.4422^n)$ (Lawler [35]); and 3-SAT in time $\mathcal{O}(1.6181^n)$ (Monien and Speckenmeyer [36]).[1]

Nowadays, it is commonly believed that NP-hard problems cannot be solved in polynomial time. For a number of NP-hard problems, we even have strong evidence that they cannot be solved in subexponential time [30]. In order to obtain exact solutions to these problems, the only hope is to design exact algorithms with good exponential running times. In recent years there has been emerging interest in attacking this question for concrete combinatorial problems: there are, for example, an $\mathcal{O}^*(2^n)$ time algorithm for coloring (Björklund and Husfeldt [5] and Koivisto [34]); an $\mathcal{O}(1.3289^n)$ time algorithm for 3-coloring (Beigel and Eppstein [3]); an $\mathcal{O}(1.7325^n)$ time algorithm for Max-Cut (Williams [45]); an algorithm for 3-SAT in time $\mathcal{O}(1.4726^n)$ (Brueggemann and Kern [15]); and an $\mathcal{O}(1.5129^n)$ time algorithm for dominating set (Fomin, Grandoni, and Kratsch [23]).

There are several explanations for the current revival of interest in fast exponential algorithms within the algorithmic community.

- The design and analysis of exact algorithms leads to a better understanding of NP-hard problems and initiates interesting new combinatorial and algorithmic challenges.

*Received by the editors October 24, 2005; accepted for publication (in revised form) January 17, 2008; published electronically July 2, 2008. This project was supported by The Aurora Programme Collaboration Research Project between Norway and France. A preliminary version of these results appeared in [25] and [44].

http://www.siam.org/journals/sicomp/38-3/64335.html

†Department of Informatics, University of Bergen, 5020 Bergen, Norway (fomin@ii.uib.no, yngvev@ii.uib.no). The work of the first author was supported by the Norwegian Research Council.

‡LITA, Université de Metz, 57045 Metz Cedex 01, France (kratsch@univ-metz.fr).

§LIFO, Université d'Orléans, 45067 Orléans Cedex 2, France (Ioan.Todinca@univ-orleans.fr).

[1] Because $c^n \cdot n^{\mathcal{O}(1)} = \mathcal{O}((c+\epsilon)^n)$ for any $\epsilon > 0$, we omit polynomial factors in the big-Oh notation every time we round the base of the exponent. We also use a modified big-Oh notation that suppresses all other (polynomially bounded) terms. Thus for functions f and g we write $f(n) = \mathcal{O}^*(g(n))$ if $f(n) = \mathcal{O}(g(n) \cdot n^{\mathcal{O}(1)})$.

- For certain applications it is important to find exact solutions. With the increased speed of modern computers, fast algorithms, even though they have exponential running times in the worst case, may actually lead to practical algorithms for certain NP-hard problems, at least for moderate instance sizes.
- Approximation algorithms, randomized algorithms, and different heuristics are not always satisfactory. Each of these approaches has weak points such as the necessity of exact solutions, difficulty of approximation, limited power of the method itself, and many others.
- A reduction of the base of the exponential running time, say from $\mathcal{O}(2^n)$ to $\mathcal{O}(1.8^n)$, increases the size of the instances solvable within a given amount of time by a constant *multiplicative* factor. However, running a given exponential algorithm on a faster computer can enlarge the mentioned size by only a constant *additive* factor.

For overviews and introductions to the field see the recent surveys by Fomin, Grandoni, and Kratsch [24], Iwama [31], Schöning [41], and Woeginger [46, 47].

Treewidth and minimum fill-in. Treewidth is one of the most basic parameters in graph algorithms [7], and it plays an important role in structural graph theory. It serves as one of the main tools in Robertson and Seymour's graph minors project [39]. Treewidth also plays a crucial role in parameterized complexity theory [19]. The minimum fill-in problem (also known as minimum chordal graph completion) has important applications in sparse matrix computations and computational biology.

The problems of computing the treewidth and minimum fill-in of a graph are known to be NP-hard even when the input is restricted to complements of bipartite graphs (so called cobipartite graphs) [2, 48]. Despite the importance of treewidth almost nothing is known about how to cope with its intractability. For a long time the best known approximation algorithm for treewidth had a factor $\log OPT$ [1, 11] (see also [10]). Recently, Feige, Hajiaghayi, and Lee [21] obtained a factor $\sqrt{\log OPT}$ approximation algorithm for treewidth. Furthermore, it is an old open question whether the treewidth can be approximated within a constant factor.

Treewidth is known to be fixed parameter tractable. Moreover, for any fixed k, there is a linear time algorithm to compute the treewidth of graphs of treewidth at most k (unfortunately there is a huge hidden constant in the running time) [6]. There is a number of algorithms that, for a given graph G and integer k, either report that the treewidth of G is at least k or produce a tree decomposition of width at most $c_1 \cdot k$ in time ${c_2}^k \cdot n^{O(1)}$, where c_1, c_2 are some constants (see, e.g., [1]). Fixed parameter algorithms are known for the minimum fill-in problem as well [16, 32].

There exists an exact $\mathcal{O}(2.9512^n)$ time algorithm that computes the treewidth of a graph in polynomial space [9]. We are not aware of any previous work on exact algorithms for the treewidth or minimum fill-in problem that solves the problem in $\mathcal{O}(c^n)$ time where $c < 2$. There are three relatively simple approaches resulting in time $\mathcal{O}^*(2^n)$ algorithms:

- One can reformulate the problems as problems of finding special vertex elimination orderings and then find an optimal permutation by using the dynamic programming based technique as in the article of Held and Karp [28] for the travelling salesman problem. The algorithm of Bodlaender et al. [9] also uses this approach.
- With some modifications, the algorithm of Arnborg, Corneil, and Proskurowski [2] for a given k deciding in time $O(n^{k+1})$ if the treewidth of a graph is at most k can be used to compute the treewidth (and similarly fill-in) in time $\mathcal{O}^*(2^n)$.

- Both problems can be solved by making use of the game theoretic approach, by finding a specific path in the graph of possible states of a cop and robber game [22].

However, it is not clear whether any of the aforementioned approaches can bring us to an $\mathcal{O}(c^n)$ time algorithm for some $c < 2$. Prior to our work, no exact algorithm computing the treewidth or minimum fill-in of a graph in time $\mathcal{O}(c^n)$ for some $c < 2$ was known.

Our results. In this paper we obtain the first exact algorithm computing the treewidth in time $\mathcal{O}(c^n)$ for $c < 2$. Additionally it can be adapted to solve a number of other minimal triangulation problems such as minimum fill-in.

Our main result is an $\mathcal{O}(1.8899^n)$ algorithm computing the treewidth and minimum fill-in of a graph on n vertices. The algorithm can be regarded as dynamic programming across partial solutions and is based on results of Bouchitté and Todinca [13, 14]. The analysis of the running time is difficult and is based on combinatorial properties of special structures in a graph, namely, potential maximal cliques, i.e., vertex subsets in a graph that can be maximal cliques in some minimal triangulation of this graph. (See the next section for the definition.)

More precisely, first we modify the algorithm of Bouchitté and Todinca [13] which computes the treewidth and minimum fill-in of a graph G with the given set Π_G of all potential maximal cliques of G and then improve the analysis of its running time to obtain an $\mathcal{O}^*(|\Pi_G|)$ time complexity. The most technical and difficult part of the paper is the proof that all potential maximal cliques can be listed in time $\mathcal{O}(1.8899^n)$. Very roughly, our listing algorithm is based on the following combinatorial result: every "large" potential maximal clique either is "almost" a minimal separator or can be represented by a "small" vertex subset. The technique developed to prove this combinatorial result can be interesting on its own.

For several special graph classes, for which both problems remain NP-complete, we are able to prove that our approach guarantees significantly better bounds. To exemplify this we show that, for the class of asteroidal triple (AT)-free graphs, the number of minimal separators and the number of potential maximal cliques, and thus the running time of our algorithm, is $\mathcal{O}^*(2^{n/2})$.

This paper is organized as follows. In section 2 we give basic definitions. In section 3 we show how Bouchitté and Todinca's approach can be used to compute the treewidth and fill-in in time linear in the number of potential maximal cliques. In section 4 we prove that every graph on n vertices has $\mathcal{O}(n{\cdot}1.7087^n)$ minimal separators. In section 5 we show that an n-vertex graph contains $\mathcal{O}(1.8135^n)$ potential maximal cliques. This bound is of only combinatorial interest because it is not constructive, in a sense, that we do not know how to use this bound to list all potential maximal cliques in time $\mathcal{O}(1.8135^n)$. In order to obtain a fast algorithm computing the treewidth and the fill-in of a graph, we need an algorithm listing all potential maximal cliques. In the remaining part of section 5 we derive the most difficult and important algorithmic result of this paper, namely, that all potential maximal cliques of a graph can be listed in time $\mathcal{O}(1.8899^n)$. This result is based on a novel characterization of potential maximal cliques. Combined with the results from section 3, this yields the main result of the paper, that the treewidth and minimum fill-in can be computed in time $\mathcal{O}(1.8899^n)$. In section 6 we design a faster $\mathcal{O}^*(2^{n/2})$ time algorithm for AT-free graphs. We conclude with open problems and final remarks in section 7.

2. Basic definitions. We denote by $G = (V, E)$ a finite, undirected, and simple graph with $|V| = n$ vertices and $|E| = m$ edges. For any nonempty subset $W \subseteq V$,

the subgraph of G induced by W is denoted by $G[W]$. For $S \subseteq V$ we often use $G \setminus S$ to denote $G[V \setminus S]$. The *neighborhood* of a vertex v is $N(v) = \{u \in V : \{u, v\} \in E\}$, $N[v] = N(v) \cup \{v\}$, and for a vertex set $S \subseteq V$ we set $N(S) = \bigcup_{v \in S} N(v) \setminus S$, $N[S] = N(S) \cup S$. A *clique* C of a graph G is a subset of V such that all the vertices of C are pairwise adjacent. By $\omega(G)$ we denote the maximum clique-size of a graph G.

Treewidth and minimum fill-in of graphs. The notion of treewidth is due to Robertson and Seymour [38]. A *tree decomposition* of a graph $G = (V, E)$, denoted by $TD(G)$, is a pair (X, T) in which $T = (V_T, E_T)$ is a tree and $X = \{X_i \mid i \in V_T\}$ is a family of subsets of V such that

(i) $\bigcup_{i \in V_T} X_i = V$;
(ii) for each edge $e = \{u, v\} \in E$ there exists an $i \in V_T$ such that both u and v belong to X_i; and
(iii) for all $v \in V$, the set of nodes $\{i \in V_T \mid v \in X_i\}$ induces a connected subtree of T.

The maximum of $|X_i| - 1$, $i \in V_T$, is called the *width* of the tree decomposition. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width taken over all tree decompositions of G.

A graph H is *chordal* (or *triangulated*) if every cycle of length at least four has a chord, i.e., an edge between two nonconsecutive vertices of the cycle. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, E')$ such that $E \subseteq E'$. H is a *minimal triangulation* if, for any intermediate set E'' with $E \subseteq E'' \subset E'$, the graph $F = (V, E'')$ is not chordal.

The following result is very useful for our algorithms.

THEOREM 2.1 (folklore). *For any graph G, $\mathrm{tw}(G) \leq k$ if and only if there is a triangulation H of G such that $\omega(H) \leq k + 1$.*

Thus the treewidth of a graph G can be defined as the minimum of $\omega(H) - 1$ taken over all triangulations H of G, of $\omega(H) - 1$.

The *minimum fill-in* of a graph $G = (V, E)$, denoted by $\mathrm{mfi}(G)$, is the smallest value of $|E_H - E|$, where the minimum is taken over all triangulations $H = (V, E_H)$ of G.

In other words, computing the treewidth of G means finding a (minimal) triangulation with the smallest maximum clique-size, while computing the minimum fill-in means finding a (minimal) triangulation with the smallest number of edges. Clearly, in both cases it is sufficient to consider only minimal triangulations of G, which makes minimal separators and potential maximal cliques important tools of our algorithmic approach.

Minimal separators. Minimal separators and potential maximal cliques are the most important tools used in our proofs. Let a and b be two nonadjacent vertices of a graph $G = (V, E)$. A set of vertices $S \subseteq V$ is an *a, b-separator* if a and b are in different connected components of the graph $G \setminus S$. A connected component C of $G \setminus S$ is a *full* component (associated to S) if $N(C) = S$. S is a *minimal a, b-separator* of G if no proper subset of S is an a, b-separator. We say that S is a *minimal separator* of G if there are two vertices a and b such that S is a minimal a, b-separator. Notice that a minimal separator can be strictly included in another one. We denote by Δ_G the set of all minimal separators of G. A set of vertices $\Omega \subseteq V$ of a graph G is called a *potential maximal clique* if there is a minimal triangulation H of G such that Ω is a maximal clique of H. We denote by Π_G the set of all potential maximal cliques of G. Clearly, $|\Delta_G| \leq 2^n$ and $|\Pi_G| \leq 2^n$ for every graph G on n vertices, and no better upper bounds had been known prior to our work.

The following result will be used to list all minimal separators of a graph.

THEOREM 2.2 (see [4]). *There is an algorithm listing all minimal separators of an input graph G in $\mathcal{O}(n^3|\Delta_G|)$ time.*

There is a very useful relationship between the minimal separators of a graph and its minimal triangulations. Two minimal separators S and T of a graph G are said to be *crossing* if S is a minimal u,v-separator for a pair of vertices $u,v \in T$, in which case T is a minimal x,y-separator for a pair $x,y \in S$. (See [33] and [37] for a full proof.)

THEOREM 2.3 (see [37]). *The graph H is a minimal triangulation of the graph G if and only if there is a maximal set of pairwise noncrossing minimal separators $\{S_1, S_2, \ldots, S_p\}$ of G such that H can be obtained from G by completing each S_i, $i \in \{1, 2, \ldots, p\}$, into a clique.*

Although we do not use this characterization explicitly it is fundamental for our paper.

Potential maximal cliques. The following structural characterization of potential maximal cliques is extremely useful for our purposes.

For a set $K \subseteq V$, a connected component C of $G \backslash K$ is a *full component associated to* K if $N(C) = K$.

THEOREM 2.4 (see [13]). *Let $K \subseteq V$ be a set of vertices of the graph $G = (V, E)$. Let $\mathcal{C}(K) = \{C_1(K), \ldots, C_p(K)\}$ be the set of the connected components of $G \setminus K$ and let $\mathcal{S}(K) = \{S_1(K), S_2(K), \ldots, S_p(K)\}$, where $S_i(K)$, $i \in \{1, 2, \ldots, p\}$, is the set of those vertices of K which are adjacent to at least one vertex of the component $C_i(K)$. Then K is a potential maximal clique of G if and only if*

1. *$G \setminus K$ has no full component associated to K, and*
2. *the graph on the vertex set K obtained from $G[K]$ by completing each $S_i \in \mathcal{S}(K)$ into a clique is a complete graph.*

Moreover, if K is a potential maximal clique, then $\mathcal{S}(K)$ is the set of the minimal separators of G contained in K.

Remark 2.5. By Theorem 2.4, for every potential maximal clique Ω of G, the sets $S_i(\Omega)$ are exactly the minimal separators of G contained in Ω. For each minimal separator $S_i = S_i(\Omega)$, all vertices of $\Omega \setminus S_i$ are contained in the same component of $G \setminus S_i$.

The following result is an easy consequence of Theorem 2.4.

THEOREM 2.6 (see [13]). *There is an algorithm that, given a graph $G = (V, E)$ and a set of vertices $K \subseteq V$, verifies if K is a potential maximal clique of G. The time complexity of the algorithm is $\mathcal{O}(nm)$.*

According to [14], the number of potential maximal cliques of a graph G is at least $|\Delta_G|/n$ and at most $n|\Delta_G|^2 + n|\Delta_G| + 1$.

3. Computing treewidth and minimum fill-in. We describe a modification of the algorithm of [13] that, given a graph, all its minimal separators, and all its potential maximal cliques, computes the treewidth and the minimum fill-in of the graph. The running time stated in [13] could be reformulated as $\mathcal{O}(n^2\,|\Delta_G| \cdot |\Pi_G|)$. We show how the algorithm can be implemented to run in time $\mathcal{O}(n^3 \cdot |\Pi_G|)$.

For a minimal separator S and a component $C \in \mathcal{C}(S)$ of $G \backslash S$, we say that (S, C) is a *block* associated to S. We sometimes use the notation (S, C) to denote the set of vertices $S \cup C$ of the block. It is easy to notice that if $X \subseteq V$ corresponds to the set of vertices of a block, then this block (S, C) is unique: indeed, $S = N(V \setminus X)$ and $C = X \setminus S$.

A block (S, C) is called *full* if C is a full component associated to S. The graph $R(S, C) = G_S[S \cup C]$ obtained from $G[S \cup C]$ by completing S into a clique is called the *realization* of the block B.

THEOREM 3.1 (see [33]). *Let G be a noncomplete graph. Then*

$$\mathrm{tw}(G) = \min_{S \in \Delta_G} \max_{C \in \mathcal{C}(S)} \mathrm{tw}(R(S, C)),$$

$$\mathrm{mfi}(G) = \min_{S \in \Delta_G} \left(\mathrm{fill}(S) + \sum_{C \in \mathcal{C}(S)} \mathrm{mfi}(R(S, C)) \right),$$

where $\mathrm{fill}(S)$ *is the number of nonedges of* $G[S]$.

Remark 3.2. In the equations of Theorem 3.1 we may take the minimum only over the inclusion-minimal separators of G. Then all the blocks in the equations are full.

Unfortunately, Theorem 3.1 is not sufficient for computing the treewidth and the minimum fill-in. Therefore we now express the treewidth and the minimum fill-in of realizations of full blocks from realizations of smaller full blocks. Let Ω be a potential maximal clique of G. We say that a block (S', C') is *associated to* Ω if C' is a component of $G \setminus \Omega$ and $S' = N(C')$.

THEOREM 3.3 (see [13]). *Let (S, C) be a full block of G. Then*

$$\mathrm{tw}(R(S, C)) = \min_{S \subset \Omega \subseteq (S,C)} \max(|\Omega| - 1, \mathrm{tw}(R(S_i, C_i))),$$

$$\mathrm{mfi}(R(S, C)) = \min_{S \subset \Omega \subseteq (S,C)} \left(\mathrm{fill}(\Omega) - \mathrm{fill}(S) + \sum \mathrm{mfi}(R(S_i, C_i)) \right),$$

where the minimum is taken over all potential maximal cliques Ω such that $S \subset \Omega \subseteq (S, C)$ and (S_i, C_i) are the blocks associated to Ω in G such that $S_i \cup C_i \subset S \cup C$.

THEOREM 3.4. *There is an algorithm that, given a graph G together with the list of its minimal separators Δ_G and the list of its potential maximal cliques Π_G, computes the treewidth and the minimum fill-in of G in $\mathcal{O}(n^3 |\Pi_G|)$ time. Moreover, the algorithm constructs optimal triangulations for the treewidth and the minimum fill-in.*

Proof. W.l.o.g. we may assume that the input graph G is connected (otherwise we can run the algorithm for each connected component of G).

The algorithm for computing the treewidth and the minimum fill-in of a graph, using its minimal separators and its potential maximal cliques, is given below. It is a slightly different version of the algorithm given in [13].

`Input:` G, all its potential maximal cliques and all its minimal separators
`Output:` $\mathrm{tw}(G)$ and $\mathrm{mfi}(G)$
`begin`
  compute all the full blocks (S, C) and sort them by the number of vertices
  `for` each full block (S, C) taken in increasing order
    $\mathrm{tw}(R(S, C)) := |S \cup C| - 1$ if (S, C) is inclusion-minimal
      and $\mathrm{tw}(R(S, C)) := \infty$ otherwise
    $\mathrm{mfi}(R(S, C)) := \mathrm{fill}(S \cup C)$ if (S, C) is inclusion-minimal
      and $\mathrm{mfi}(R(S, C)) := \infty$ otherwise
    `for` each p.m.c. Ω with $S \subset \Omega \subseteq (S, C)$

compute the blocks (S_i, C_i) associated to Ω s.t. $S_i \cup C_i \subset S \cup C$
$\mathrm{tw}(R(S,C)) := \min(\mathrm{tw}(R(S,C)),$
$\max_i(|\Omega| - 1, \mathrm{tw}(R(S_i, C_i))))$
$\mathrm{mfi}(R(S,C)) := \min(\mathrm{mfi}(R(S,C)),$
$\mathrm{fill}(\Omega) - \mathrm{fill}(S) + \sum_i(\mathrm{mfi}(R(S_i, C_i))))$
`end_for`
`end_for`
let Δ_G^* be the set of inclusion-minimal separators of G
$\mathrm{tw}(G) := \min_{S \in \Delta_G^*} \max_{C \in \mathcal{C}(S)} \mathrm{tw}(R(S,C))$
$\mathrm{mfi}(G) := \min_{S \in \Delta_G^*}(\mathrm{fill}(S) + \sum_{C \in \mathcal{C}(S)} \mathrm{mfi}(R(S,C)))$
`end`

For the sake of completeness we shortly discuss the correctness proof. According to Theorem 3.3, at the end of the outer `for` loop the values of $\mathrm{tw}(R(S,C))$ and $\mathrm{mfi}(R(S,C))$ are correctly computed for each full block (S,C) of G. Then the treewidth and the minimum fill-in of the graph are computed using Theorem 3.1 and the fact that in Theorem 3.1 one can work only with inclusion-minimal separators.

Let us show how the algorithm can be implemented such that its running time is $\mathcal{O}(n^3 \cdot |\Pi_G|)$.

To store and manipulate the minimal separators, potential maximal cliques, and blocks we use data structures that allow us to search, to insert, or to check whether an element is inclusion-minimal in $\mathcal{O}(n)$ time.

During a preprocessing step, we realize the following operations.

- Compute the list of all full blocks and, for each minimal separator S, store a pointer towards each full block of type (S,C). These operations are performed as follows. For each minimal separator S, we compute the connected components of $G \setminus S$. For each such component C, if $N(C) = S$, then the block (S,C) is full, so we add it to the list of full blocks and store the pointer from S to (S,C). Note that this procedure will generate all the full blocks, and each of them is encountered exactly once. For a given minimal separator S, there are at most n full blocks associated to it, and thus at most n pointers to be stored. The insertion of these blocks into the list of full blocks requires $\mathcal{O}(n)$ time for each block. Hence the whole step costs $\mathcal{O}(n^2|\Delta_G|)$ time.
- For each potential maximal clique Ω, store a pointer to each full block associated to it as follows: compute the components C_i of $G \setminus \Omega$, and then $(N(C_i), C_i)$ are precisely the blocks associated to Ω. In particular there are at most n such blocks. This computation can be done in $\mathcal{O}(n^2)$ time for each potential maximal clique, hence globally in $\mathcal{O}(n^2|\Pi_G|)$ time.
- Compute all the *good triples* (S,C,Ω), where (S,C) is a full block and Ω is a potential maximal clique such that $S \subset \Omega \subseteq S \cup C$. Moreover, for each good triple we store a pointer from (S,C) to Ω. By Theorem 2.4, there are at most n minimal separators $S \subset \Omega$, each of them being the neighborhood of a component of $G \setminus \Omega$, and for each such S there is exactly one component $G \setminus S$ intersecting Ω (in particular there are at most $n|\Pi_G|$ good triples). For each component C' of $G \setminus \Omega$ we take $S = N(C')$, find the component C of $G \setminus S$ intersecting Ω, and store the pointer from (S,C) to Ω. Thus this computation takes $\mathcal{O}(nm)$ time for each potential maximal clique, hence $\mathcal{O}(nm|\Pi_G|)$ globally.

Hence this preprocessing step costs $\mathcal{O}(n^2|\Delta_G| + nm|\Pi_G|)$. Sorting the blocks by their size can be done in $\mathcal{O}(n|\Delta_G|)$ time using a bucket sort.

Observe that the cost of one iteration of the inner `for` loop is $\mathcal{O}(n^2)$, by the fact that there are at most n blocks associated to a potential maximal clique. With the data structures obtained during the preprocessing step, each full block (S, C) keeps a pointer towards each potential maximal clique Ω such that (S, C, Ω) form a good triple. Thus the number of iterations of the two nested loops is exactly the number of good triples, that is, at most $n|\Pi_G|$. It follows that the two loops cost $\mathcal{O}(n^3|\Pi_G|)$ time.

After the execution of the loops, computing the set Δ_G^* of inclusion-minimal separators costs $\mathcal{O}(n|\Delta_G|)$ time. Each inclusion-minimal separator S keeps the list of its associated blocks, obtained during the preprocessing step. Computing the maximum required by the two last instructions costs $\mathcal{O}(n)$ time for a given S. This last step costs $\mathcal{O}(n|\Delta_G|)$ time.

Altogether, the algorithm runs in time $\mathcal{O}(n^2 \cdot |\Delta_G| + n^3 \cdot |\Pi_G|)$. It is known [14] that each minimal separator is contained in at least one potential maximal clique. According to Theorem 2.4, each potential maximal clique contains at most n minimal separators. Therefore $|\Pi_G| \geq |\Delta_G|/n$. We conclude that the algorithm runs in $\mathcal{O}(n^3 \cdot |\Pi_G|)$ time.

The algorithm can be easily transformed in order to output not only the treewidth and the minimum fill-in of the graph, but also optimal triangulations with respect to these parameters. It is sufficient to keep, for each full block, the set of potential maximal cliques realizing the minimum treewidth and fill-in of its realization. At the end of the algorithm, the potential maximal cliques of the chosen blocks will be the maximal cliques of the computed optimal triangulation: optimal tree decomposition or minimum triangulation. □

Using Theorem 3.4, the only missing ingredient of our treewidth and minimum fill-in algorithms is an algorithm listing all (minimal separators and) potential maximal cliques of a graph in time $\mathcal{O}^*(c^n)$ for some $c < 2$. This requires exponential upper bounds of the type $\mathcal{O}^*(c^n)$ for some $c < 2$ for the number of minimal separators and for the number of potential maximal cliques in a graph on n vertices. In the next two sections we discuss this issue.

4. The number of minimal separators. In this section we show that any graph with n vertices has $\mathcal{O}(1.7087^n)$ minimal separators. For the main algorithm of this paper the upper bound $\mathcal{O}(1.8899^n)$ would be sufficient. However, bounding the number of minimal separators in a graph is a nice combinatorial problem, and we prefer to give here the best upper bound we were able to find.

Let S be a separator in a graph $G = (V, E)$. For $x \in V \setminus S$, we denote by $C_x(S)$ the component of $G \setminus S$ containing x. The following lemma is an exercise in [27].

LEMMA 4.1 (folklore). *A set S of vertices of G is a minimal a, b-separator if and only if a and b are in different full components associated to S. In particular, S is a minimal separator if and only if there are at least two distinct full components associated to S.*

Here is one of the main combinatorial results of our paper.

THEOREM 4.2. *For any graph G, $|\Delta_G| = \mathcal{O}(1.7087^n)$.*

Proof. For a constant α, $0 < \alpha < 1$, we distinguish two types of minimal separators: *small* separators, of size at most αn, and *big* separators, of size more than αn. We denote the cardinalities of these sets by #`small sep` and #`big sep`. Notice that $|\Delta_G| = \#\texttt{small sep} + \#\texttt{big sep}$.

The number of big separators. Let S be a minimal separator. By Lemma 4.1, there are at least two full components associated to S. Hence at least one of these full

components has at most $n(1-\alpha)/2$ vertices. For every $S \in \Delta_G$ we choose one of these full components and call it the *small* component of S, denoted by $\mathrm{s}(S)$.

By the definition of a full component, $S = N(\mathrm{s}(S))$. In particular, for distinct minimal separators S and T, we have that $\mathrm{s}(S) \neq \mathrm{s}(T)$. Therefore the number #`big sep` of big minimal separators is at most the number of small components, and we conclude that #`big sep` does not exceed the number of subsets of V of cardinality at most $n(1-\alpha)/2$; i.e.,

$$\#\texttt{big sep} \leq \sum_{i=1}^{\lceil n(1-\alpha)/2 \rceil} \binom{n}{i}.$$

By making use of Stirling's formula we deduce that

$$\#\texttt{big sep} \leq \frac{n(1-\alpha)}{2} \left(\pi n(1-\alpha)\frac{(1+\alpha)}{2}\right)^{-\frac{1}{2}} \left[\left(\frac{1-\alpha}{2}\right)^{-\frac{1-\alpha}{2}} \left(\frac{1+\alpha}{2}\right)^{-\frac{1+\alpha}{2}}\right]^n.$$

The number of small separators. To count small separators we use a different technique. Let S be a minimal separator, let x be a vertex of a full component $C_x(S)$ associated to S with minimum number of vertices, and let $X \subset V$ be a vertex subset. We say that (x, X) is a *bad pair* associated to S if $C_x(S) \subseteq X \subseteq V \setminus S$.

Claim 1. *Let $S \neq T$ be two minimal separators and let (x, X) and (y, Y) be two bad pairs associated to S and T, respectively. Then $(x, X) \neq (y, Y)$.*

Proof. Since $C_x(S) \subseteq X$ and $X \cap S = \emptyset$, we have that the connected component of $G[X]$ containing x is $C_x(S)$. Similarly, the connected component of $G[Y]$ containing y is $C_y(T)$.

Thus if $x = y$ and $X = Y$, then $C_x(S) = C_y(T)$. Since $C_x(S)$ is a full component associated to S in G, we have that $S = N(C_x(S))$ and $T = N(C_y(T))$. Therefore $S = T$, which is a contradiction. □

By Lemma 4.1, there are at least two full components associated to every small separator S. For a full component $C_x(S)$ associated to S with the minimum number of vertices, $|V \setminus (S \cup C_x(S))| \geq n \cdot (1-\alpha)/2$. For any $Z \subseteq V \setminus (S \cup C_x(S))$, the pair $(x, Z \cup C_x(S))$ is a bad pair associated to S. Therefore there are at least $2^{n\cdot(1-\alpha)/2}$ distinct bad pairs associated to S. Hence by Claim 1, the total number of bad pairs is at least $\#\texttt{small sep} \cdot 2^{n\cdot(1-\alpha)/2}$. On the other hand, the number of bad pairs is at most $n \cdot 2^n$. We conclude that

$$\#\texttt{small sep} \leq n2^{n\cdot(1+\alpha)/2}.$$

Finally, choosing $\alpha = 0.5456$, we obtain

$$|\Delta_G| = \#\texttt{small sep} + \#\texttt{big sep} = \mathcal{O}(n \cdot 1.7087^n). \qquad \square$$

Let us note that, by Theorem 2.2, Theorem 4.2 yields that all minimal separators of a graph can be listed in time $\mathcal{O}(1.7087^n)$.

5. The number of potential maximal cliques. In this section we prove that the number of potential maximal cliques in a graph with n vertices is $\mathcal{O}(1.8135^n)$ and then show that there exists an algorithm to list all potential maximal cliques of any graph in time $\mathcal{O}(1.8899^n)$.

We bound the number of potential maximal cliques by counting specific potential maximal cliques called nice potential maximal cliques. Later these nice potential maximal cliques are used to generate and to bound the number of all potential maximal cliques.

DEFINITION 5.1. *Let Ω be a potential maximal clique of a graph G, and let $S \subset \Omega$ be a minimal separator of G. We say that S is an* active separator for Ω *if Ω is not a clique in the graph $G_{\mathcal{S}(\Omega)\setminus\{S\}}$, obtained from G by completing all the minimal separators contained in Ω except S. If S is active, a pair of vertices $x, y \in S$ nonadjacent in $G_{\mathcal{S}(\Omega)\setminus\{S\}}$ is called an* active pair. *Otherwise, S is called* inactive for Ω.

DEFINITION 5.2. *We say that a potential maximal clique Ω is* nice *if at least one of the minimal separators contained in Ω is active for Ω.*

We define Π_n as the maximum number of nice potential maximal cliques in a graph on n vertices. By the theorem and lemma below, the number of potential maximal cliques is polynomially bounded by the number of nice potential maximal cliques and minimal separators.

THEOREM 5.3 (see [14]). *Let Ω be a potential maximal clique of G, and let a be a vertex of G and $G' = G \setminus \{a\}$. Then one of the following cases holds:*

1. *either Ω or $\Omega \setminus \{a\}$ is a potential maximal clique of G';*
2. *$\Omega = S \cup \{a\}$, where S is a minimal separator of G;*
3. *Ω is nice.*

LEMMA 5.4. *A graph G on n vertices has at most $n^2|\Delta_G|+n\Pi_n$ potential maximal cliques.*

Proof. Let $x_1, x_2, \ldots, x_n$ be the vertices of G and $G_i = G[\{x_1, \ldots, x_i\}]$ for all $i \in \{1, 2, \ldots, n\}$. By Theorem 5.3, for each $i \in \{2, 3, \ldots, n\}$, $|\Pi_{G_i}| \leq |\Pi_{G_{i-1}}|+n|\Delta_{G_i}|+\Pi_i$. By [14, Corollary 4], $|\Delta_{G_i}| \leq |\Delta_G|$ for any $i \in \{1, \ldots, n\}$. This yields that

$$|\Pi_{G_n}| \leq \sum_{i=1}^{n} n|\Delta_{G_i}| + \Pi_i \leq n^2|\Delta_{G_n}| + n\Pi_n. \qquad \square$$

5.1. Nonconstructive upper bound on the number of potential maximal cliques. We show that the number of potential maximal cliques in a graph is $\mathcal{O}(1.8135^n)$. This bound is obtained by finding an upper bound on the number of nice potential maximal cliques. We do this by computing two numbers (as for the separator bound): the number of nice potential maximal cliques of size less than αn and the number of nice potential maximal cliques of size at least αn for $0 < \alpha < 1$.

DEFINITION 5.5. *We say that the pair (Z, z) is a* vertex representation *of a potential maximal clique Ω in G where $Z \subset V$ and $z \in Z$ if $\Omega = N(Z) \cup \{z\}$ and $G[Z]$ is connected.*

LEMMA 5.6. *Let Ω be a potential maximal clique of G and let $z \in \Omega$. Then (Z, z) is a vertex representation of Ω if and only if Z is the vertex set of the connected component of $G \setminus (\Omega \setminus \{z\})$ containing z.*

Proof. Suppose that (Z, z) is a vertex representation of Ω. Vertex z is contained in Z, and thus every neighbor of z not in Ω has to be contained in Z. By applying this argument recursively, every connected component C of $G \setminus \Omega$, where $z \in N(C)$, is contained in Z. On the other hand, these are the only vertices in Z, because the rest are separated by $\Omega \setminus \{z\}$.

Conversely, let Z be the vertex set of the component of $G \setminus (\Omega \setminus \{z\})$ containing z. Clearly $N(Z) \cup \{z\}$ is contained in Ω, so it remains to prove that any $x \in \Omega \setminus \{z\}$ is contained in $N(Z)$. By Theorem 2.4, x is adjacent to z or there is a component C of $G \setminus \Omega$ such that both x and z are in $N(C)$. Since $C \subset Z$ the conclusion follows. $\square$

LEMMA 5.7. *Let Ω be a nice potential maximal clique of size αn. Then there exists a* vertex representation *(U, u) of Ω such that $|U| \leq 2n(1-\alpha)/3 + 1$.*

Proof. Let S be a minimal separator active for Ω, let $x, y \in S$ be an active pair, and let z be a vertex contained in $\Omega \setminus S$. By Lemma 5.6, every vertex in a potential maximal clique defines a unique vertex representation. Let $(X, x), (Y, y), (Z, z)$ be the unique vertex representations defined by Ω and, respectively, x, y, and z (Lemma 5.6).

Let us now prove that one of the sets X, Y, Z contains at most $2n(1-\alpha)/3 + 1$ vertices. By Lemma 5.6, we can observe that if (U, u) is a vertex representation of Ω, then $G[U \setminus \{u\}]$ is formed exactly by the set of connected components of $G \setminus \Omega$ having u in their neighborhood. We partition the connected components of $G \setminus \Omega$ into three sets:

- $A_1 = (X \setminus \{x\}) \cap (Y \setminus \{y\})$,
- $A_2 = (X \setminus \{x\}) \setminus (Y \setminus \{y\})$, and
- $A_3 = (V \setminus \Omega) \setminus (A_1 \cup A_2)$.

Let us emphasize that

- $|A_1 \cup A_2 \cup A_3| = n(1-\alpha)$ (because $A_1 \cup A_2 \cup A_3 = V \setminus \Omega$ and $|\Omega| = \alpha n$),
- the sets A_1, A_2, A_3 are pairwise disjoint,
- $X \setminus \{x\} = A_1 \cup A_2$,
- $Y \setminus \{y\} \subseteq A_1 \cup A_3$, and
- $Z \setminus \{z\} \subseteq A_2 \cup A_3$ (by construction, $G[A_1]$ is the union of components of $G \setminus \Omega$ that see both x and y; since S is an active separator for Ω, x, y is an active pair, and $z \notin S$, it follows by Definition 5.1 that none of these components can see z; thus $A_1 \cap Z = \emptyset$).

One of the vertex sets A_1, A_2, A_3, say A_1, is of size at least $n(1-\alpha)/3$; then $|A_2| + |A_3| \leq 2n(1-\alpha)/3$. Since $Z \setminus \{z\} \subseteq A_2 \cup A_3$, we have that $|Z \setminus \{z\}| \leq 2n(1-\alpha)/3$, and thus there exists a vertex representation $(U, z) = (Z, z)$ of Ω such that $|U| \leq (2n(1-\alpha)/3) + 1$. □

LEMMA 5.8. *For a constant $0 < \alpha < 1$ and a graph G, the number $\Pi_{\geq \alpha n}$ of nice potential maximal cliques of size at least αn is at most $n \sum_{i=1}^{2n(1-\alpha)/3} \binom{n}{i}$.*

Proof. By Lemma 5.7, every potential maximal clique Ω of size at least αn has a vertex representation (X, x) such that $|X \setminus \{x\}| \leq 2n(1-\alpha)/3$. Thus $\Pi_{\geq \alpha n}$ is at most the number of pairs (X, x), where $|X \setminus \{x\}| \leq 2n(1-\alpha)/3$ and $x \in V \setminus X$, which is at most $n \sum_{i=1}^{2n(1-\alpha)/3} \binom{n}{i}$. □

LEMMA 5.9. *For a constant $0 < \alpha < 1$ and a graph G, the number $\Pi_{<\alpha n}$ of nice potential maximal cliques of size less than αn is at most $n \cdot 2^{n(2+\alpha)/3}$.*

Proof. We say that (x, X) is a *bad pair* associated to Ω if $\Omega = N(C_x) \cup \{x\}$, where C_x is the connected component of $G[X \cup \{x\}]$ containing x.

To prove that a bad pair is unique for a potential maximal clique, we let (x, X) be a bad pair associated to Ω_x, and let (y, Y) be a bad pair associated to Ω_y, where $\Omega_x \neq \Omega_y$. We claim that $(x, X) \neq (y, Y)$. Targeting a contradiction, we assume that $x = y$ and that $X = Y$. From the definition of a bad pair, we know that $N(X) \cup \{x\} = N(Y) \cup \{y\}$. But this is a contradiction because $N(X) \cup \{x\} = \Omega_x$, $N(Y) \cup \{y\} = \Omega_y$, and $\Omega_x \neq \Omega_y$.

By Lemma 5.7, every potential maximal clique Ω of size less than αn has a vertex representation (U, u) such that $|V \setminus (\Omega \cup U)| \geq n(1-\alpha)/3$. Thus we can create $2^{n(1-\alpha)/3}$ unique bad pairs (u, X) for Ω by selecting $X = U \cup Z$, where Z is any of the $2^{n(1-\alpha)/3}$ subsets of $V \setminus N[U]$. The number of bad pairs is at most $n \cdot 2^n$, and we get that $n \cdot 2^n \geq \Pi_{<\alpha n} \cdot 2^{n(1-\alpha)/3}$. □

LEMMA 5.10. *The number of nice potential maximal cliques in a graph G with n vertices is $\mathcal{O}(1.8135^n)$.*

Proof. The number of nice potential maximal cliques Π_n is at most $\Pi_{\geq \alpha n} + \Pi_{<\alpha n}$ for $0 \leq \alpha \leq 1$. By using Lemmas 5.8 and 5.9, we have that $\Pi_n \leq n \cdot \sum_{i=1}^{2n(1-\alpha)/3} \binom{n}{i} +$

$n \cdot 2^{n(2+\alpha)/3}$. By making use of Stirling's formula for $\alpha = 0.5763$, we obtain the bound $\mathcal{O}(1.81349^n)$. □

THEOREM 5.11. *For every graph G on n vertices, $|\Pi_G| = \mathcal{O}(1.8135^n)$.*

Proof. By Lemma 5.4, G has at most $n^2|\Delta_G| + n\Pi_n$ potential maximal cliques. By Theorem 4.2 and Lemma 5.10, this yields that $\Pi_G = \mathcal{O}(n^2 1.7087^n + n1.81349^n) = \mathcal{O}(1.8135^n)$. □

5.2. Listing potential maximal cliques. Notice that the proof of Lemma 5.9 is nonconstructive; i.e., the proof cannot be turned into an algorithm listing potential maximal cliques, which is required by the algorithms computing treewidth and fill-in.

Roughly speaking, the idea of this subsection is to show that each potential maximal clique of a graph can be identified by a set of vertices of size at most $n/3$. The algorithm for generating all the potential maximal cliques of a graph lists all the sets of vertices of size at most $n/3$ and then, by applying a polynomial time procedure for each set, generates all the potential maximal cliques of the input graph. A potential maximal clique can be recognized by the following three representations.

DEFINITION 5.12. *Let Ω be a potential maximal clique of G. The triple (S, a, b) is called a* separator representation *of Ω if S is a minimal separator of G, $a \in S$, $b \in V \setminus S$, and $\Omega = S \cup (N(a) \cap C_b)$, where C_b is the component of $G \setminus S$ containing b.*

The number of all possible separator representations of a graph is at most $n^2|\Delta_G|$. Unfortunately, not every nice potential maximal clique has a separator representation. The two definitions below allow us to represent a potential maximal clique by using a small vertex set.

DEFINITION 5.13. *For a potential maximal clique Ω of G, we say that a pair (X, c), where $X \subset V$ and $c \in X$, is a* partial representation *of Ω if $\Omega = N(C_c) \cup (X \setminus C_c)$, where C_c is the connected component of $G[X]$ containing c.*

DEFINITION 5.14. *For a potential maximal clique Ω of G, we say that a triple (X, x, c), where $X \subset V$ and $x, c \notin X$, is an* indirect representation *of Ω if $\Omega = N(C_c \cup D_x \cup \{x\}) \cup \{x\}$, where*

- *C_c is the connected component of $G \setminus N[X]$ containing c;*
- *D_x is the vertex set of the union of all connected components C' of $G[X]$ such that $x \in N(C')$.*

Let us note that for a given vertex set X and two vertices x, c one can check in polynomial time whether the pair (X, c) is a partial representation or if the triple (X, x, c) is a separator representation or indirect representation of a (unique) potential maximal clique Ω.

The next step is to partition the vertex sets of the graph into smaller sets that can be used to create one of the three representations for the nice potential maximal clique. First the following theorem is required.

THEOREM 5.15 (see [14]). *Let Ω be a potential maximal clique of G and $S \subset \Omega$ a minimal separator, active for Ω. Let (S, C) be the block associated to S containing Ω, and let $x, y \in \Omega$ be an active pair. Then $\Omega \setminus S$ is a minimal x, y-separator in $G[C \cup \{x, y\}]$.*

We are now ready to divide the set of connected components of $G \setminus \Omega$ into subsets.

LEMMA 5.16. *Let Ω be a nice potential maximal clique, S be a minimal separator active for Ω, $x, y \in S$ be an active pair, and C be the component of $G \setminus S$ containing $\Omega \setminus S$. There is a partition (D_x, D_y, D_r) of $C \setminus \Omega$ such that $N(D_x \cup \{x\}) \cap C = N(D_y \cup \{y\}) \cap C = \Omega \setminus S$.*

Proof. By Theorem 5.15, $\Omega \setminus S$ is a minimal x, y-separator in $G[C \cup \{x, y\}]$. Let C_x be the full component associated to $\Omega \setminus S$ in $G[C \cup \{x, y\}]$ containing x, $D_x = C_x \setminus \{x\}$,

and let C_y be the full component associated to $\Omega \setminus S$ in $G[C \cup \{x, y\}]$ containing y, $D_y = C_y \setminus \{y\}$, and $D_r = C \setminus (\Omega \cup D_x \cup D_y)$. Since $D_x \cup \{x\}$ and $D_y \cup \{y\}$ are full components of $\Omega \setminus S$, we have that $N(D_x \cup \{x\}) \cap C = N(D_y \cup \{y\}) \cap C = \Omega \setminus S$. □

By the lemma below we get that every potential maximal clique Ω that is not defined by the closed neighborhood of a vertex or defined by a separator representation has a neighbor outside of Ω in a full connected component of S.

LEMMA 5.17. *Let Ω be a potential maximal clique of G, S be a minimal separator contained in Ω, and C be the component of $G \setminus S$ intersecting Ω. Then one of the following holds:*

1. *there is a vertex a such that $\Omega = N[a]$;*
2. *Ω has a separator representation;*
3. *$\Omega = N(C \setminus \Omega)$.*

Proof. Suppose that there is a vertex $a \in \Omega$ having no neighbor in $C \setminus \Omega$. We consider first the case $a \in \Omega \setminus S$. We claim that in this case $\Omega = N[a]$. Because $a \in \Omega \setminus S \subseteq C$, we conclude that $N[a] \subseteq \Omega$. Thus to prove the claim we need to show that $\Omega \subseteq N[a]$. For sake of contradiction, suppose that there is $b \in \Omega$ which is not adjacent to a. By Theorem 2.4, every two nonadjacent vertices of a potential maximal clique are contained in some minimal separator $S_i(\Omega)$. Thus both a and b should have neighbors in a component $C_i(\Omega)$ of $G \setminus \Omega$. Since $a \in \Omega \setminus S \subseteq C$, we have that $C_i(\Omega) \subseteq C \setminus \Omega$. But this contradicts the assumption that a has no neighbors in $C \setminus \Omega$.

The case $a \in S$ is similar. Suppose that $\Omega \setminus S \neq N(a) \cap C$; i.e., there is a vertex $b \in \Omega \setminus S$ nonadjacent to a. Then again, a and b are contained in some minimal separator and thus should have neighbors in a component $C_i(\Omega) \subseteq C$ of $G \setminus \Omega$ which is a contradiction.

Since C is a component of $G \setminus S$ and S is contained in Ω, we have that $N(C \setminus \Omega) \subseteq \Omega$. If every vertex of Ω is adjacent to a vertex of $C \setminus \Omega$, then $\Omega = N(C \setminus \Omega)$. □

We state now the main tool for upper bounding the time required to list the set of nice potential maximal cliques.

LEMMA 5.18. *Let Ω be a nice potential maximal clique of G. Then one of the following holds:*

1. *there is a vertex a such that $\Omega = N[a]$;*
2. *Ω has a separator representation;*
3. *Ω has a partial representation (X, c) such that $|X| \leq n/3$;*
4. *Ω has a indirect representation (X, x, c) such that $|X| \leq n/3$.*

Proof. Let S be a minimal separator active for Ω, $x, y \in S$ be an active pair, and C be the component of $G \setminus S$ containing $\Omega \setminus S$. By Lemma 5.16, there is a partition (D_x, D_y, D_r) of $C \setminus \Omega$ such that $N(D_x \cup \{x\}) \cap C = N(D_y \cup \{y\}) \cap C = \Omega \setminus S$. If one of the sets D_x, D_y, say D_x, is empty, then $N(D_x \cup \{x\}) \cap C = N(x) \cap C = \Omega \setminus S$, and thus the triple (S, x, z) is a separator representation of Ω.

Suppose that none of the first two conditions of the lemma holds. Then D_x and D_y are nonempty. In order to argue that Ω has a partial representation (X, c) or an indirect representation (X, x, c) such that $|X| \leq n/3$, we partition the graph further. Let $R = \Omega \setminus S$, and let D_S be the union of all full components associated to S in $G \setminus \Omega$. The vertex set D_x is the union of vertex sets of all connected components C' of $G \setminus (\Omega \cup D_S)$ such that x is contained in the neighborhood of C'. Thus a connected component C' of $G \setminus (\Omega \cup D_S)$ is contained in D_x if and only if $x \in N(C')$. Similarly, a connected component C' of $G \setminus (\Omega \cup D_S)$ is contained in D_y if and only if $y \in N(C')$. We also define $D_r = V \setminus (\Omega \cup D_S \cup D_x \cup D_y)$, which is the set of vertices of the components of $G \setminus (\Omega \cup D_S)$ which are not in D_x and D_y.

We partition S in the following sets:

- $S_{\overline{x}} = (S \setminus N(D_x)) \cap N(D_y)$;
- $S_{\overline{y}} = (S \setminus N(D_y)) \cap N(D_x)$;
- $S_{\overline{xy}} = S \setminus (N(D_y) \cup N(D_x))$;
- $S_{xy} = S \cap N(D_y) \cap N(D_x)$.

Thus $S_{\overline{x}}$ is the set of vertices in S with no neighbor in D_x and with at least one neighbor in D_y, $S_{\overline{y}}$ is the set of vertices in S with no neighbor in D_y and with at least one neighbor in D_x, $S_{\overline{xy}}$ is the set of vertices in S with neighbors in neither D_x nor D_y, and, finally, S_{xy} is the set of vertices in S with neighbors in both D_x and D_y. Notice that the vertex sets D_S, D_x, D_y, D_r, R, $S_{\overline{x}}$, $S_{\overline{y}}$, $S_{\overline{xy}}$, and S_{xy} are pairwise disjoint. The set S_{xy} is mentioned only to complete the partition of S and will not be used in the rest of the proof.

Both for size requirements and because of the definition of indirect representation we cannot use the sets $S_{\overline{x}}$, $S_{\overline{y}}$, and $S_{\overline{xy}}$ directly; they have to be represented by the sets $Z_{\overline{x}}$, $Z_{\overline{y}}$, and $Z_{\overline{r}}$, which are subsets of the vertex sets D_y, D_x, and D_r. By the definition of $S_{\overline{x}}$ and $S_{\overline{y}}$ it follows that there exist two vertex sets $Z_{\overline{x}} \subseteq D_y$ and $Z_{\overline{y}} \subseteq D_x$ such that $S_{\overline{x}} \subseteq N(Z_{\overline{x}})$ and $S_{\overline{y}} \subseteq N(Z_{\overline{y}})$. Let Z be such a set of minimum cardinality. By Lemma 5.17, $\Omega = N(D_x \cup D_y \cup D_r)$ since cases 1 and 2 of Lemma 5.17 correspond to cases 1 and 2 of the lemma we are proving. Thus, there exists a vertex set $Z_{\overline{r}} \subseteq D_r$ such that $S_{\overline{xy}} \subseteq N(Z_{\overline{r}})$. Let Z be such a set of minimum cardinality. A sketch of how these vertex sets relate to each other is given in Figure 5.1.

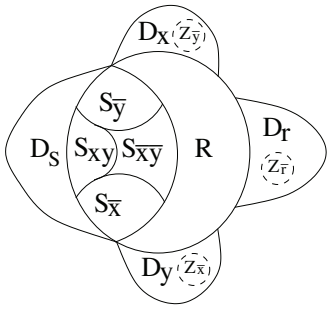


FIG. 5.1. *The figure shows a sketch of how the vertex sets $D_S, D_x, D_y, D_r, R, S_{\overline{x}}, S_{\overline{y}}, S_{\overline{xy}}$, and S_{xy} partition the graph G, and how the sets $Z_{\overline{x}}, Z_{\overline{y}}$, and $Z_{\overline{r}}$ relate to this partition.*

Let C^* be a connected component of $G[D_S]$ (let us remind that $N(C^*) = S$). We define the following sets:

- $X_1 = C^* \cup R$;
- $X_2 = D_x \cup Z_{\overline{x}} \cup Z_{\overline{r}}$;
- $X_3 = D_y \cup Z_{\overline{y}} \cup Z_{\overline{r}}$.

First we claim that

- the pair (X_1, c), where $c \in C^*$, is a partial representation of Ω;
- the triple (X_2, x, c), where $c \in C^*$, is an indirect representation of Ω;
- the triple (X_3, y, c), where $c \in C^*$, is an indirect representation of Ω.

In fact, the pair $(X_1, c) = (C^* \cup R, c)$ is a partial representation of Ω because $N(C^*) \cap R = \emptyset$, C^* induces a connected graph, and $\Omega = N(C^*) \cup R$.

To prove that $(X_2, x, c) = (D_x \cup Z_{\overline{x}} \cup Z_{\overline{r}}, x, c)$ is an indirect representation of Ω, we have to show that $\Omega = N(C_c \cup D'_x \cup \{x\}) \cup \{x\}$, where C_c is the connected component of $G \setminus N[X_2]$ containing c, and D'_x is the vertex set of the union of all connected components C' of $G[X_2]$ such that $x \in N(C')$. Notice that $(S \cup C^*) \cap X_2 = \emptyset$ and

that $S \subseteq N(X_2)$ since $S \subseteq N(D_x \cup Z_{\overline{x}} \cup Z_{\overline{r}})$ and $X_2 = D_x \cup Z_{\overline{x}} \cup Z_{\overline{r}}$. Hence the connected component C_c of $G \setminus N[X_2]$ containing c is C^*.

Every connected component C' of $G[X_2]$ is contained in $D_x, Z_{\overline{x}}$, or $Z_{\overline{r}}$ since $\Omega \cap (D_x \cup Z_{\overline{x}} \cup Z_{\overline{r}}) = \emptyset$ and Ω separates $D_x, Z_{\overline{x}}$, and $Z_{\overline{r}}$. From the definition of D_x it follows that $x \in N(C')$ for every component C' of $G[D_x]$, and from the definition of D_y and D_r it follows that $x \notin N(C')$ for every component C' of $G[Z_{\overline{x}} \cup Z_{\overline{r}}]$. We can now conclude that D_x is the vertex set of the union of all connected components C' of $G[X_2]$ such that $x \in N(C')$. It remains to prove that $\Omega = N(C^* \cup D_x \cup \{x\}) \cup \{x\}$. By Lemma 5.16, we have that $\Omega \setminus S = R$ is a subset of $N(D_x \cup \{x\})$ and $N(D_y \cup \{y\})$, and we remember that $N(C^*) = S$. From this observation it follows that $\Omega = N(C^* \cup D_x \cup \{x\}) \cup \{x\}$ since $N(C^* \cup D_x \cup \{x\}) = (S \cup R) \setminus \{x\}$.

By similar arguments, (X_3, y, c) is an indirect representation of Ω.

To conclude the proof of the lemma, we argue that at least one of the vertex sets X_1, X_2, or X_3 used to represent Ω contains at most $n/3$ vertices.

We partition the graph into the following three sets:

- $V_1 = D_S \cup R$;
- $V_2 = D_x \cup S_{\overline{x}} \cup S_{\overline{xy}}$;
- $V_3 = D_y \cup S_{\overline{y}} \cup D_r$.

These sets are pairwise disjoint, and at least one of them is of size at most $n/3$; to prove the lemma we show that $|X_1| \leq |V_1|$, $|X_2| \leq |V_2|$, and $|X_3| \leq |V_3|$.

$|X_1| \leq |V_1|$. Since $C^* \subseteq D_S$, we have that $X_1 = C^* \cup R \subseteq V_1 = D_S \cup R$.

$|X_2| \leq |V_2|$. To prove the inequality we need the additional result

$$|Z_{\overline{x}}| \leq |S_{\overline{x}}|, \ |Z_{\overline{y}}| \leq |S_{\overline{y}}|, \text{ and } |Z_{\overline{r}}| \leq |S_{\overline{xy}}|. \tag{5.1}$$

In fact, since $Z_{\overline{x}}$ is the smallest subset of D_y such that $S_{\overline{x}} \subseteq N(Z_{\overline{x}})$, we have that for any vertex $u \in Z_{\overline{x}}$, $S_{\overline{x}} \not\subseteq N(Z_{\overline{x}} \setminus \{u\})$. Thus u has a private neighbor in $S_{\overline{x}}$, or in other words there exists $v \in S_{\overline{x}}$ such that $\{u\} = N(v) \cap Z_{\overline{x}}$. Therefore $S_{\overline{x}}$ contains at least one vertex for every vertex in $Z_{\overline{x}}$, which yields $|Z_{\overline{x}}| \leq |S_{\overline{x}}|$. The proofs of inequalities $|Z_{\overline{y}}| \leq |S_{\overline{y}}|$ and $|Z_{\overline{r}}| \leq |S_{\overline{xy}}|$ are similar.

Now the proof of $|X_2| \leq |V_2|$, which is equivalent to $|D_x \cup Z_{\overline{x}} \cup Z_{\overline{r}}| \leq |D_x \cup S_{\overline{x}} \cup S_{\overline{xy}}|$, follows from (5.1) and the fact that all subsets on each side of the inequality are pairwise disjoint.

$|X_3| \leq |V_3|$. This inequality is equivalent to $|D_y \cup Z_{\overline{y}} \cup Z_{\overline{r}}| \leq |D_y \cup S_{\overline{y}} \cup D_r|$. Again, the sets on each side of the inequality are pairwise disjoint. $|Z_{\overline{r}}| \leq |D_r|$ because $Z_{\overline{r}} \subseteq D_r$, and $|Z_{\overline{y}}| \leq |S_{\overline{y}}|$ by (5.1).

Thus $\min\{|X_1|, |X_2|, |X_3|\} \leq n/3$, which concludes the proof of the lemma. ∎

LEMMA 5.19. *The set of nice potential maximal cliques in a graph G on n vertices can be listed in listed in $\mathcal{O}^*(\binom{n}{n/3})$ time.*

Proof. By Lemma 5.18, the number of possible partial representations (X, c) and indirect representations (X, x, c) with $|X| \leq n/3$ is at most $2n^2 \sum_{i=1}^{n/3} \binom{n}{i}$. By Theorem 4.2, the number of all possible separator representations is at most $n^2|\Delta_G| \leq n^2\binom{n}{n/3}$, and we deduce that the number of nice potential maximal cliques is at most $2n^2 \sum_{i=1}^{n/3} \binom{n}{i}$. Moreover, these potential maximal cliques can be computed in $\mathcal{O}^*(\binom{n}{n/3})$ time as follows. We enumerate all the triples (S, a, b) where S is a minimal separator and a, b are vertices, and check if the triple is the separator representation of a potential maximal clique Ω; if so, we store this potential maximal clique. We also enumerate all the potential maximal cliques of type $N[a], a \in V(G)$ in polynomial time. Finally, by listing all the sets X of at most $n/3$ vertices and all the couples

of vertices (x, c), we compute all the nice potential maximal cliques with a partial representation (X, c) or an indirect representation (X, x, c). □

THEOREM 5.20. *There is an algorithm to list all potential maximal cliques of a graph G on n vertices in time $\mathcal{O}(1.8899^n)$.*

Proof. Let $x_1, x_2, \ldots, x_n$ be the vertices of G and $G_i = G[\{x_1, \ldots, x_i\}]$ for all $i \in \{1, 2, \ldots, n\}$. Theorem 5.3 and Lemma 5.19 imply that $|\Pi_{G_i}| \leq |\Pi_{G_{i-1}}| + n|\Delta_{G_i}| + 2n^2 \sum_{i=1}^{n/3} \binom{n}{i}$ for all $i \in \{2, 3, \ldots, n\}$. By Theorem 4.2, $|\Pi_G| \leq 2n^3 \sum_{i=1}^{n/3} \binom{n}{i}$.

Clearly, if we have the potential maximal cliques of G_{i-1}, the potential maximal cliques of G_i can be computed in $\mathcal{O}^*(|\Pi_{G_{i-1}}| + \binom{n}{n/3})$ time by making use of Theorems 5.3 and 4.2 and Lemma 5.19. The graph G_1 has a unique potential maximal clique, namely, $\{x_1\}$. Therefore Π_G can be listed in time $\mathcal{O}^*(\binom{n}{n/3})$ time which is $\mathcal{O}(1.8899^n)$. □

Theorems 3.4 and 5.20 imply the main result of this paper.

THEOREM 5.21. *For a graph G on n vertices, the treewidth and the minimum fill-in of G can be computed in $\mathcal{O}(1.8899^n)$ time.*

6. AT-free graphs. In this section we establish exact algorithms to compute the treewidth and the minimum fill-in of AT-free graphs which are faster than the ones obtained for general graphs in the previous section. Both algorithms are based on new upper bounds on the number of minimal separators and the number of potential maximal cliques in AT-free graphs.

Three pairwise nonadjacent vertices of a graph G form an *asteroidal triple* (AT) if any two of them are connected by a path avoiding the neighborhood of the third vertex. Graphs without asteroidal triples are called *AT-free*.

Corneil, Olariu, and Stewart studied structural properties of AT-free graphs in their fundamental paper [17]. Among other results, they showed that every connected AT-free graph has a dominating pair, where two vertices x and y of G form a *dominating pair* (DP for short) if the vertex set of each x, y-path is a dominating set of G.

AT-free graphs contain cocomparability graphs, permutation graphs, interval graphs, and cobipartite graphs. Thus the treewidth problem and the minimum fill-in problem remain NP-hard when restricted to AT-free graphs [2, 48].

Remark 6.1. There is a well-known cobipartite (and thus AT-free) graph consisting of two cliques of size $n/2$ and a perfect matching between them which has precisely $2^{n/2} - 2$ minimal separators. It is not hard to show that this is indeed the largest number of minimal separators of a cobipartite graph on n vertices.

First we show that $|\Pi_G| = \mathcal{O}^*(|\Delta_G|)$ for AT-free graphs, improving a result in [13, Corollary 5.2]. This also establishes an algorithm to list the potential maximal cliques of an AT-free graph in $\mathcal{O}^*(|\Delta_G|)$ time. Then we prove that an AT-free graph on n vertices has at most $2^{n/2+3}$ minimal separators.

First let us summarize some structural properties of potential maximal cliques in AT-free graphs.

LEMMA 6.2 (Proposition 5.1 of [13]). *Let Ω be a potential maximal clique of an AT-free graph G. Then the set $\mathcal{S}(\Omega)$ of minimal separators contained in Ω has at most two inclusion-maximal elements.*

LEMMA 6.3 (Theorem 3.10 of [13]). *Let G be a graph and Ω be a potential maximal clique of G such that $\mathcal{S}(\Omega)$ has a unique inclusion-maximal element S. Then $\Omega \setminus S$ is a connected component of $G \setminus S$.*

Let S and T be two noncrossing minimal separators of G, incomparable with respect to inclusion. Thus S meets a unique component of $G \setminus T$, say $C_S(T)$, and T

meets a unique component of $G \setminus S$, say $C_T(S)$. We define the *piece between* S and T as $P(S,T) = S \cup T \cup (C_T(S) \cap C_S(T))$.

LEMMA 6.4 (Theorem 3.11 of [13]). *Let G be a graph and Ω be a potential maximal clique of G such that $\mathcal{S}(\Omega)$ has exactly two inclusion-maximal elements S and T. Then $\Omega = P(S,T)$.*

LEMMA 6.5. *Let G be an AT-free graph and Ω be a potential maximal clique of G such that $\mathcal{S}(\Omega)$ has two inclusion-maximal elements S and T. Choose $s \in S \setminus T$. Then $\Omega = S \cup (N(s) \cap C_T(S))$.*

Proof. By Lemma 6.4, $\Omega = P(S,T)$. Clearly s is in the unique component $C_S(T)$ of $G \setminus T$ meeting S, so $N(s) \cap C_T(S) \subseteq P(S,T)$. Consequently, $S \cup (N(s) \cap C_T(S)) \subseteq \Omega$.

Conversely, suppose there is a vertex $t \in \Omega$, not contained in $S \cup (N(s) \cap C_T(S))$. Let $S' = (S \setminus \{s\}) \cup (N(s) \cap C_T(S))$. Clearly S' separates s and any vertex of $C_T(S) \setminus S'$ in G; in particular S' separates s and t. It follows that there is a minimal separator $S'' \subseteq S'$ of G, contained in Ω and separating two vertices of Ω. According to Theorem 2.4, for each minimal separator U contained in Ω, Ω intersects exactly one component of $G \setminus U$, which is a contradiction. □

THEOREM 6.6. *An AT-free graph G has at most $n^2|\Delta_G| + n|\Delta_G| + 1$ potential maximal cliques. Furthermore, there is an algorithm to list the potential maximal cliques of an AT-free graph in $\mathcal{O}^*(|\Delta_G|)$ time.*

Proof. If G has no minimal separator, then G is a complete graph, and its vertex set is the unique potential maximal clique of G.

Suppose now that G is not complete. Fix a minimal separator S of G. By Lemma 6.3, the number of potential maximal cliques Ω such that S is the unique inclusion-maximal element of $\mathcal{S}(\Omega)$ is bounded by the number of connected components of $G \setminus S$. Hence, there are at most n such potential maximal cliques.

Now let us consider the potential maximal cliques Ω for which S is one of the two inclusion-maximal separators contained in $\mathcal{S}(\Omega)$. For any component C of $G \setminus S$, there are, by Lemma 6.5, at most $|S|$ such potential maximal cliques contained in $S \cup C$. It follows that there are at most n^2 potential maximal cliques of this type.

Therefore, G contains at most $(n^2+n)|\Delta_G|+1$ potential maximal cliques. These combinatorial arguments can easily be transformed into an algorithm listing the potential maximal cliques of an AT-free graph in time $\mathcal{O}^*(|\Delta_G|)$. □

Hence Theorem 3.4 implies that to construct an $O(1.4142^n)$ algorithm computing the treewidth and the minimum fill-in of an AT-free graph it is enough to prove that the number of minimal separators in an AT-free graph is $O(1.4142^n)$.

Our proof that the number of minimal separators in an AT-free graph is at most $2^{n/2+3}$ relies on properties of 2LexBFS, i.e., a combination of two runs of lexicographic breadth-first-search (also called 2-sweep LexBFS), on AT-free graphs established by Corneil, Olariu, and Stewart in [18].

DEFINITION 6.7. *A vertex ordering $x_n, x_{n-1}, \ldots, x_1$ is said to be a* 2LexBFS ordering *of G if some* 2LexBFS(G) *returns the vertices in this order (starting with x_n) during the second sweep of LexBFS on G where x_n is supposed to be the last vertex of the first sweep of LexBFS on G.*

We shall write $u \prec v$ if $u = x_i$, $v = x_j$, and $i < j$. A 2LexBFS ordering and the levels $L_0 = \{x_n\}, L_1 = N(x_n), \ldots, L_i = \{x_j : d(x_j, x_n) = i\}, \ldots, L_r$ are called a 2*LexBFS scheme* of G. Consider any 2LexBFS scheme. Clearly all neighbors of a vertex $v \in L_i$ are contained in $L_{i-1} \cup L_i \cup L_{i+1}$. For a vertex $v \in L_i$ we denote $N(v) \cap L_{i-1}$ by $N^{\uparrow}(v)$, and we denote $N(v) \cap L_{i+1}$ by $N_{\downarrow}(v)$.

THEOREM 6.8 (see [18]). *Every 2LexBFS ordering $x_n, x_{n-1}, \ldots, x_1$ of a connected AT-free graph has the* dominating pair-property *(DP-property); i.e., for all*

$i \in \{1, 2, \ldots, n\}$, (x_n, x_i) is a dominating pair of the graph $G[\{x_i, x_{i+1}, \ldots, x_n\}]$.

The following easy consequence of Theorem 6.8 is useful.

LEMMA 6.9. *Let $x_n, x_{n-1}, \ldots, x_1$ be a 2LexBFS ordering of an AT-free graph G, and let $L_0, L_1, \ldots, L_r$ be the corresponding 2LexBFS scheme. Let $s > r$, $x_s, x_r \in L_i$ and $\{x_r, x_s\} \notin E$. Then $N^{\uparrow}(x_r) \subseteq N^{\uparrow}(x_s)$.*

Proof. Let $w \in N^{\uparrow}(x_r) \setminus N^{\uparrow}(x_s)$. Then the path $x_r, w, u_{i-2}, \ldots, u_1, x_n$ with $u_j \in L_j$ and $u_{j-1} \in N^{\uparrow}(u_j)$ for all $j = i-2, \ldots, 1$ contains no neighbor of x_s, contradicting the DP-property of a 2LexBFS scheme of an AT-free graph. ∎

THEOREM 6.10. *An AT-free graph on n vertices has at most $2^{n/2+3}$ minimal separators.*

Proof. Let G be an AT-free graph. Let $x_n, x_{n-1}, \ldots, x_1$ be a 2LexBFS ordering of G, and let $L_0, L_1, \ldots, L_r$ be the levels of the corresponding 2LexBFS scheme.

Let S be any minimal separator of G. Let C and C' be two (not necessarily full) components of $G \setminus S$. We claim that at most one level of the 2LexBFS scheme may contain vertices of C and C'. Suppose not. Let L_i and L_{i+1} be levels containing vertices of C and C'. Then there are edges $\{u, v\}$ in C and $\{w, x\}$ in C' such that $u, w \in L_i$ and $v, x \in L_{i+1}$. W.l.o.g. assume $u \prec w$. Then Lemma 6.9 implies that w and v are adjacent, a contradiction.

Let C and C' be two (not necessarily full) components of $G \setminus S$ such that both contain vertices of some level of the 2LexBFS scheme, say L_i. Furthermore, assume $C \cap L_{i-1} \neq \emptyset$ and $C' \cap L_{i-1} = \emptyset$. Hence there is an edge $\{u, v\}$ in C such that $u \in L_i$ and $v \in L_{i-1}$. Then for each $w \in C'$ holds $w \prec u$. Otherwise $u \prec w$, $w \in L_i$, and Lemma 6.9 would imply that w and v are adjacent, a contradiction.

Finally we claim that in this case $c' \prec c$ for each vertex $c \in C$ and each vertex $c' \in C'$. This is obviously true if one of c and c' is not in L_i. It remains to consider the case $c \in L_i$, $c' \in L_i$. To the contrary assume $c \prec c'$. Since C contains vertices of L_i and L_{i-1}, there is a path in C starting in c passing through vertices of $C \cap L_i$ only until it passes through an edge $\{u, v\}$ in C with $u \in L_i$ and $v \in L_{i-1}$. This path can be extended to a path from c to x_n that does not contain a neighbor of c' although $c \prec c'$, a contradiction to the DP-property.

Now we are able to upper bound the number of those minimal separators in an AT-free graph in which no full component contains only vertices of one level. Simply divide the vertex set into two halves: $A = \{x_n, x_{n-1}, \ldots, x_{\lceil n/2 \rceil + 1}\}$ and $B = \{x_{\lfloor n/2 \rfloor}, \ldots, x_1\}$. Now consider two full components C and C' of a minimal separator S of G, i.e., $S = N(C) = N(C')$. Then either C or C' is a subset of either A or B, and surely each of C and C' uniquely determines S. Hence we simply consider all subsets of A and all subsets of B as possible full components of a minimal separator of G. Consequently, there are at most $2^{n/2+1}$ minimal separators of this type.

It remains to upper bound the number of all those minimal separators S of an AT-free graph G for which each full component is neither a subset of A nor a subset of B. Hence at least one full component of S contains only vertices from one level of the 2LexBFS scheme.

Let S be such a minimal separator of G. Let C and C' be two full components of $G \setminus S$. W.l.o.g. assume $C \subseteq L_i$. Hence $x_{\lfloor n/2 \rfloor} \in L_i$, and thus the level L_i is uniquely determined.

$C' \cap \bigcup_{j=0}^{i-1} L_j = \emptyset$ since otherwise $c \prec c'$ for all $c \in C$ and all $c' \in C'$, and either C or C' is a subset of A or B. Similarly C' must contain vertices of L_i. Consequently, $C' \subseteq \bigcup_{j=i}^{r} L_j$. It is easy to see that $C \subseteq L_i$ and $S = N(C)$ imply $N(C') = S \subseteq \bigcup_{j=i-1}^{i+1} L_j$. Furthermore, $N(C) = N(C') = S$ implies $S \cap L_{i-1} = N^{\uparrow}(C \cap L_i) = N^{\uparrow}(C' \cap L_i)$.

Now let us consider the graph $G' = G \setminus \bigcup_{j=0}^{i-1} L_j$. Then $S' = S \setminus \bigcup_{j=0}^{i-1} L_j$ is

a separator of G'; C and C' are components of $G' \setminus S'$. Furthermore, every vertex of $S' \subseteq S$ has a neighbor in C and C', and thus S' is a minimal separator of G'. Consequently, every minimal separator S of G for which no full component is a subset of A or B corresponds uniquely to a minimal separator of G'. Notice that G' has at most $n-1$ vertices since we remove at least one vertex of L_{i-1} from G to obtain G'.

Let $f(n)$ be a function such that $f(n)$ is an upper bound for the number of minimal separators in an n-vertex AT-free graph. Then we establish the recurrence $f(n) \geq 2^{n/2+1} + f(n-1)$ and conclude with $f(n) = 4 \cdot 2^{n/2+1} = 8 \cdot 2^{n/2}$. □

Combining Theorems 3.4, 6.6, and 6.10, we obtain algorithms for AT-free graphs that are faster than the corresponding ones for general graphs.

THEOREM 6.11. *There are algorithms to compute the treewidth and the minimum fill-in of an AT-free graph in $\mathcal{O}(1.4142^n)$ time.*

7. Open problems and final remarks. Planar graphs. The computational complexity of treewidth restricted to planar graphs is a longstanding open problem in graph algorithms. The treewidth of planar graphs can be approximated within a constant factor of 1.5. More precisely, Seymour and Thomas [42] gave a polynomial algorithm for computing the *branchwidth* of planar graphs, and the latter parameter differs by at most a factor of 1.5 from the treewidth.

In the case of planar graphs with n vertices, the treewidth is at most $\mathcal{O}(\sqrt{n})$.

THEOREM 7.1 (see [26]). *For any planar graph G on n vertices,* $\mathrm{tw}(G) \leq 3.182\sqrt{n} + \mathcal{O}(1)$.

Also, given a graph G and a number k, one can decide if $\mathrm{tw}(G) \leq k$ in $\mathcal{O}^*(n^k)$ time, either using the algorithm of Arnborg Corneil, and Proskurowski [2] or our technique, restricted to potential maximal cliques of size at most $k+1$.

Consequently, the treewidth of planar graphs can be computed in time $\mathcal{O}^*(n^{3.182\sqrt{n}}) = 2^{\mathcal{O}(\sqrt{n}\log n)}$.

Unfortunately, although the structure of potential maximal cliques in planar graphs is very particular [12], our approach cannot be used for obtaining algorithms of running time $2^{\mathcal{O}(\sqrt{n})}$ for planar treewidth. This is because the number of "large" potential maximal cliques in planar graphs can be "large."

CLAIM 2. *For any integer N, there is a planar graph on $n > N$ vertices with at least $2^{0.49\sqrt{n}\log n}$ potential maximal cliques of size at least $2\sqrt{n}+2$.*

Proof. Consider the planar graph G_p depicted in Figure 7.1. It has $n = p^2+p+3$ vertices. The set of vertices $S = \{a_1, b_{1i_1}, a_2, b_{2i_2}, \ldots, a_p, b_{pi_p}, a_{p+1}\}$ forms a c,d-minimal separator for any values $i_1, i_2, \ldots, i_p$ between 1 and p. By making use of Theorem 2.4, it is not hard to see that $S \cup \{c\}$ is a potential maximal clique of size $p+1$ in G_p. Consequently, G has at least p^p potential maximal cliques. If $p \geq 2$, we have $p > \sqrt{n} - 1$; thus the number of potential maximal cliques is at least $(\sqrt{n}-1)^{\sqrt{n}-1}$. □

Since we do not know if the treewidth of a planar graph can be computed in polynomial time, an interesting task is to design an algorithm of running time $2^{\mathcal{O}(\sqrt{n})}$. As we mentioned, this will need new techniques.

Combinatorial bounds. The running time estimation of our algorithms is based on combinatorial upper bounds on the number of minimal separators and an upper bound for the time to list all potential maximal cliques of a graph. Finding better bounds on the number of minimal separators and potential maximal cliques in a graph is an interesting combinatorial challenge.

How many potential maximal cliques can be in a graph? We have shown that the number of potential maximal cliques in a graph on n vertices is at most $\mathcal{O}(1.8135^n)$. Unfortunately, it is not clear if the same bound can be obtained by an algorithm

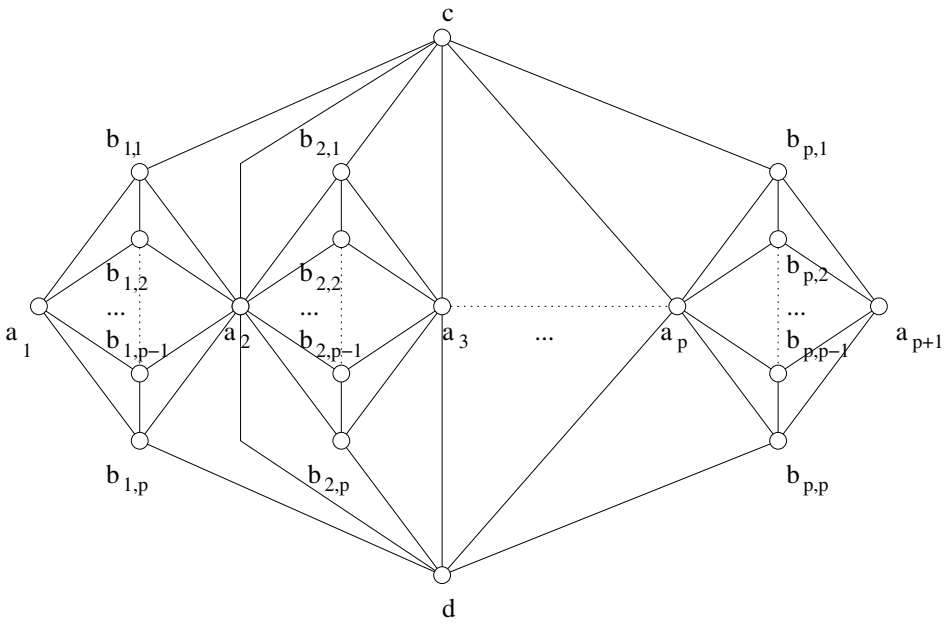


FIG. 7.1. *Planar graphs with many large potential maximal cliques.*

listing potential maximal cliques. Of course, such an algorithm can be used to speed up our algorithm for treewidth and fill-in. A related interesting question is whether it is possible to list potential maximal cliques with polynomial time delay.

How many minimal separators can be in a graph? We are aware of the following construction providing the lower bound $3^{n/3} \approx 1.4422^n$ on the number of minimal separators: Let G be a graph on $n = 3k + 2$ vertices. G has two vertices a, b that are connected by k vertex disjoint paths of length 4. Every minimal a, b-separator in G contains exactly one inner vertex of each a, b-path. Thus the number of minimal separators in G is at least $3^{n/3} \approx 1.4422^n$. However, the gap between the lower bound and the upper bound $\mathcal{O}(1.7087^n)$ from Theorem 4.2 is still big. A related question is whether it is possible to list the minimal separators with polynomial delay.

For some special graph classes, the use of minimal separators can imply faster algorithms for triangulation problems. For example, we have shown that every AT-free graph on n vertices has at most $2^{n/2+3}$ minimal separators and that this upper bound is tight up to a multiplicative constant factor. The interesting question here is whether similar techniques can be used for other graph classes, such as bipartite graphs and graphs of small degree.

Related problems. Our algorithms for treewidth and minimum fill-in can also be used for solving other problems that can be expressed in terms of minimal triangulations such as finding a tree decomposition of minimum cost [8] or computing treewidth of weighted graphs. However, there are two "width" parameters related to treewidth, namely bandwidth and pathwidth, and one parameter called profile, related to minimum fill-in, that do not fit into this framework. Bandwidth can be computed in time $\mathcal{O}^*(10^n)$ [20], and reducing Feige's bounds is a challenging problem. Pathwidth (and profile) can be expressed as vertex ordering problems and thus solved in $\mathcal{O}^*(2^n)$ time by applying a dynamic programming approach similar to Held and Karp's approach [28] for the travelling salesman problem. Let us note that reach-

ing time complexity $\mathcal{O}^*(c^n)$ for any constant $c < 2$, even for the Hamiltonian cycle problem, is a longstanding problem. So it is unlikely that some modification of Held and Karp's approach would provide us with a better exact algorithm for pathwidth or profile. It is tempting to ask if one can reach time complexity $\mathcal{O}^*(c^n)$, for any constant $c < 2$, for these problems.

Acknowledgment. We are grateful to Hans Bodlaender for nice discussions and for pointing out that the algorithm of Arnborg, Corneil, and Proskurowski [2] can be used to compute treewidth and minimum fill-in in time $\mathcal{O}^*(2^n)$.

REFERENCES

[1] E. Amir, *Efficient approximation for triangulation of minimum treewidth*, in Proceedings of the Seventeenth Conference in Uncertainty in Artificial Intelligence (UAI-2001), Morgan Kaufmann, San Francisco, CA, 2001, pp. 7–15.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Alg. Disc. Meth., 8 (1987), pp. 277–284.

[3] R. Beigel and D. Eppstein, 3*-coloring in time* $O(1.3289^n)$, J. Algorithms, 54 (2005), pp. 168–204.

[4] A. Berry, J. P. Bordat, and O. Cogis, *Generating all the minimal separators of a graph*, Internat. J. Found. Comput. Sci., 11 (2000), pp. 397–403.

[5] A. Björklund and T. Husfeldt, *Inclusion-exclusion algorithms for counting set partitions*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Washington, DC, 2006, pp. 575–582.

[6] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[7] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998), pp. 1–45.

[8] H. L. Bodlaender and F. V. Fomin, *Tree decompositions with small cost*, Discrete Appl. Math., 145 (2005), pp. 143–154.

[9] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos, *On exact algorithms for treewidth*, in Algorithms—ESA 2006, Lecture Notes in Comput. Sci. 4168, Springer, Berlin, 2006, pp. 672–683.

[10] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks, *Approximating treewidth, pathwidth, frontsize, and shortest elimination tree*, J. Algorithms, 18 (1995), pp. 238–255.

[11] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca, *On treewidth approximations*, Discrete Appl. Math., 136 (2004), pp. 183–196.

[12] V. Bouchitté, F. Mazoit, and I. Todinca, *Chordal embeddings of planar graphs*, Discrete Math., 273 (2003), pp. 85–102.

[13] V. Bouchitté and I. Todinca, *Treewidth and minimum fill-in: Grouping the minimal separators*, SIAM J. Comput., 31 (2001), pp. 212–232.

[14] V. Bouchitté and I. Todinca, *Listing all potential maximal cliques of a graph*, Theoret. Comput. Sci., 276 (2002), pp. 17–32.

[15] T. Brueggemann and W. Kern, *An improved deterministic local search algorithm for* 3*-SAT*, Theoret. Comput. Sci., 329 (2004), pp. 303–313.

[16] L. Cai, *Fixed-parameter tractability of graph modification problems for hereditary properties*, Inform. Process. Lett., 58 (1996), pp. 171–176.

[17] D. G. Corneil, S. Olariu, and L. Stewart, *Asteroidal triple-free graphs*, SIAM J. Discrete Math., 10 (1997), pp. 399–430.

[18] D. G. Corneil, S. Olariu, and L. Stewart, *Linear time algorithms for dominating pairs in asteroidal triple-free graphs*, SIAM J. Comput., 28 (1999), pp. 1284–1297.

[19] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer, New York, 1999.

[20] U. Feige, *Coping with the NP-hardness of the graph bandwidth problem*, in Algorithm Theory—SWAT 2000, Lecture Notes in Comput. Sci. 1851, Springer, Berlin, 2000, pp. 10–19.

[21] U. Feige, M. Hajiaghayi, and J. R. Lee, *Improved approximation algorithms for minimum-weight vertex separators*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC '05), ACM Press, New York, 2005, pp. 563–572.

[22] F. V. Fomin, P. Fraigniaud, and N. Nisse, *Nondeterministic graph searching: From pathwidth to treewidth*, in Mathematical Foundations of Computer Science 2005, Lecture Notes in Comput. Sci. 3618, Springer, Berlin, 2005, pp. 364–375.

[23] F. V. Fomin, F. Grandoni, and D. Kratsch, *Measure and conquer: Domination—a case study*, in Automata, Languages and Programming (ICALP 2005), Lecture Notes in Comput. Sci. 3580, Springer, Berlin, 2005, pp. 191–203.

[24] F. V. Fomin, F. Grandoni, and D. Kratsch, *Some new techniques in design and analysis of exact (exponential) algorithms*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 87 (2005), pp. 47–77.

[25] F. V. Fomin, D. Kratsch, and I. Todinca, *Exact (exponential) algorithms for treewidth and minimum fill-in*, in Automata, Languages and Programming (ICALP 2004), Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 568–580.

[26] F. V. Fomin and D. M. Thilikos, *New upper bounds on the decomposability of planar graphs*, J. Graph Theory, 51 (2006), pp. 53–81.

[27] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[28] M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, J. Soc. Indust. Appl. Math., 10 (1962), pp. 196–210.

[29] E. Horowitz and S. Sahni, *Computing partitions with applications to the knapsack problem*, J. Assoc. Comput. Mach., 21 (1974), pp. 277–292.

[30] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity*, J. Comput. System Sci., 63 (2001), pp. 512–530.

[31] K. Iwama, *Worst-case upper bounds for k-SAT*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 82 (2004), pp. 61–71.

[32] H. Kaplan, R. Shamir, and R. E. Tarjan, *Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs*, SIAM J. Comput., 28 (1999), pp. 1906–1922.

[33] T. Kloks, D. Kratsch, and J. Spinrad, *On treewidth and minimum fill-in of asteroidal triple-free graphs*, Theoret. Comput. Sci., 175 (1997), pp. 309–335.

[34] M. Koivisto, *An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Washington, DC, 2006, pp. 583–590.

[35] E. L. Lawler, *A note on the complexity of the chromatic number problem*, Inform. Process. Lett., 5 (1976), pp. 66–67.

[36] B. Monien and E. Speckenmeyer, *Solving satisfiability in less than 2^n steps*, Discrete Appl. Math., 10 (1985), pp. 287–295.

[37] A. Parra and P. Scheffler, *Characterizations and algorithmic applications of chordal graph embeddings*, Discrete Appl. Math., 79 (1997), pp. 171–188.

[38] N. Robertson and P. D. Seymour, *Graph minors.* II. *Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[39] N. Robertson and P. D. Seymour, *Graph minors.* X. *Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

[40] J. M. Robson, *Algorithms for maximum independent sets*, J. Algorithms, 7 (1986), pp. 425–440.

[41] U. Schöning, *Algorithmics in exponential time*, in STACS 2005, Lecture Notes in Comput. Sci. 3404, Springer, Berlin, 2005, pp. 36–43.

[42] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

[43] R. E. Tarjan and A. E. Trojanowski, *Finding a maximum independent set*, SIAM J. Comput., 6 (1977), pp. 537–546.

[44] Y. Villanger, *Improved exponential-time algorithms for treewidth and minimum fill-in*, in LATIN 2006: Theoretical Informatics, Lecture Notes in Comput. Sci. 3887, Springer, Berlin, 2006, pp. 800–811.

[45] R. Williams, *A new algorithm for optimal constraint satisfaction and its implications*, in Automata, Languages and Programming (ICALP 2004), Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 1227–1237.

[46] G. Woeginger, *Exact algorithms for NP-hard problems: A survey*, in Combinatorial Optimization—Eureka, You Shrink!, Lecture Notes in Comput. Sci. 2570, Springer, Berlin, 2003, pp. 185–207.

[47] G. Woeginger, *Space and time complexity of exact algorithms: Some open problems*, in IWPEC 2004, Lecture Notes in Comput. Sci. 3162, Springer, Berlin, 2004, pp. 281–290.

[48] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 77–79.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1080–1112

DIMENSIONS OF POINTS IN SELF-SIMILAR FRACTALS*

JACK H. LUTZ† AND ELVIRA MAYORDOMO‡

Abstract. Self-similar fractals arise as the unique attractors of iterated function systems (IFSs) consisting of finitely many contracting similarities satisfying an open set condition. Each point x in such a fractal F arising from an IFS S is naturally regarded as the "outcome" of an infinite *coding sequence* T (which need not be unique) over the alphabet $\Sigma_k = \{0, \ldots, k-1\}$, where k is the number of contracting similarities in S. A classical theorem of Moran (1946) and Falconer (1989) states that the Hausdorff and packing dimensions of a self-similar fractal coincide with its similarity dimension, which depends only on the contraction ratios of the similarities. The theory of computing has recently been used to provide a meaningful notion of the *dimensions of individual points* in Euclidean space. In this paper, we use (and extend) this theory to analyze the dimensions of individual points in fractals that are *computably self-similar*, meaning that they are unique attractors of IFSs that are computable and satisfy the open set condition. Our main theorem states that, if $F \subseteq \mathbb{R}^n$ is any computably self-similar fractal and S is any IFS testifying to this fact, then the dimension identities $\dim(x) = \mathrm{sdim}(F)\dim^{\pi_S}(T)$ and $\mathrm{Dim}(x) = \mathrm{sdim}(F)\mathrm{Dim}^{\pi_S}(T)$ hold for all $x \in F$ and all coding sequences T for x. In these equations, $\mathrm{sdim}(F)$ denotes the similarity dimension of the fractal F; $\dim(x)$ and $\mathrm{Dim}(x)$ denote the dimension and strong dimension, respectively, of the point x in Euclidean space; and $\dim^{\pi_S}(T)$ and $\mathrm{Dim}^{\pi_S}(T)$ denote the dimension and strong dimension, respectively, of the coding sequence T relative to a probability measure π_S that the IFS S induces on the alphabet Σ_k. The above-mentioned theorem of Moran and Falconer follows easily from our main theorem by relativization. Along the way to our main theorem, we develop the elements of the theory of constructive dimensions relative to general probability measures. The proof of our main theorem uses Kolmogorov complexity characterizations of these dimensions.

Key words. Billingsley dimension, computability, constructive dimension, fractal geometry, geometric measure theory, iterated function system, Hausdorff dimension, packing dimension, self-similar fractal

AMS subject classifications. 68Q15, 68Q30, 03D99, 28A78, 11K55

DOI. 10.1137/070684689

1. Introduction.

The theory of computing has recently been used to formulate effective versions of Hausdorff dimension and packing dimension, the two most important dimensions in geometric measure theory [36, 37, 11, 1]. These effective fractal dimensions have already produced quantitative insights into many aspects of algorithmic randomness, Kolmogorov complexity, computational complexity, data compression, and prediction [24]. They are also beginning to yield results in geometric measure theory itself [20].

The most fundamental effective dimensions are the constructive dimension [37] and the constructive strong dimension [1]. These two constructive dimensions (which are defined explicitly in section 3 below) are the constructive versions of the Hausdorff and packing dimensions, respectively. For each set X of (infinite) sequences over a

*Received by the editors March 8, 2007; accepted for publication (in revised form) February 11, 2008; published electronically July 2, 2008.
http://www.siam.org/journals/sicomp/38-3/68468.html

†Department of Computer Science, Iowa State University, Ames, IA 50011 (lutz@cs.iastate.edu). This author's research was supported in part by National Science Foundation grants 0344187, 0652569, and 0728806 and by Spanish Government MEC Project TIN 2005-08832-C03-02. Part of this author's research was performed during a sabbatical at the University of Wisconsin and two visits at the University of Zaragoza.

‡Departamento de Informática e Ingeniería de Sistemas, María de Luna 1, Universidad de Zaragoza, 50018 Zaragoza, Spain (elvira@unizar.es). This author's research supported in part by Spanish Government MEC Project TIN 2005-08832-C03-02.

finite alphabet Σ, the constructive dimension $\mathrm{cdim}(X)$ and the constructive strong dimension $\mathrm{cDim}(X)$ are real numbers satisfying the inequalities

$$\begin{array}{ccccccc} & & \mathrm{cdim}(X) & \leq & \mathrm{cDim}(X) & \leq & 1 \\ & & \vee\mid & & \vee\mid & & \\ 0 & \leq & \dim_{\mathrm{H}}(X) & \leq & \mathrm{Dim}_{\mathrm{P}}(X), & & \end{array}$$

where $\dim_{\mathrm{H}}(X)$ and $\mathrm{Dim}_{\mathrm{P}}(X)$ are the Hausdorff and packing dimensions, respectively, of X. These constructive dimensions are exact analogues of the constructive Lebesgue measure that Martin-Löf used to formulate algorithmic randomness [38]. As such, they are endowed with universality properties that make them especially well behaved. For example, unlike the other effective dimensions, and unlike their classical counterparts, the constructive dimensions are *absolutely stable*, meaning that the dimension of *any* union (countable or otherwise) of sets is the supremum of the dimensions of the individual sets. In particular, this implies that, if we define the dimension and strong dimension of an individual sequence $S \in \Sigma^\infty$ to be

$$\dim(S) = \mathrm{cdim}(\{S\}) \tag{1.1}$$

and

$$\mathrm{Dim}(S) = \mathrm{cDim}(\{S\}), \tag{1.2}$$

respectively, then the constructive dimensions of any set $X \subseteq \Sigma^\infty$ are determined by the equations

$$\mathrm{cdim}(X) = \sup_{S \in X} \dim(S) \tag{1.3}$$

and

$$\mathrm{cDim}(X) = \sup_{S \in X} \mathrm{Dim}(S). \tag{1.4}$$

Constructive dimensions are thus investigated entirely in terms of the dimensions of individual sequences.

The two constructive dimensions also admit precise characterizations in terms of Kolmogorov complexity [39, 1]. Specifically, for any sequence $S \in \Sigma^\infty$,

$$\dim(S) = \liminf_{j \to \infty} \frac{\mathrm{K}(S[0..j-1])}{j \log |\Sigma|} \tag{1.5}$$

and

$$\mathrm{Dim}(S) = \limsup_{j \to \infty} \frac{\mathrm{K}(S[0..j-1])}{j \log |\Sigma|}, \tag{1.6}$$

where $\mathrm{K}(S[0..j-1])$ denotes the Kolmogorov complexity of the j-symbol prefix of S [33] and the logarithm is base-2. Since $\mathrm{K}(w)$ measures the algorithmic information content (in bits) of a string w, (1.5) and (1.6) say that $\dim(S)$ and $\mathrm{Dim}(S)$ are asymptotic measures of the *algorithmic information density* of S.

Although the constructive dimensions have primarily been investigated in sequence spaces Σ^∞, they work equally well in Euclidean spaces $\mathbb{R}^n$. One of several

equivalent ways to achieve this is to fix a base $k \geq 2$ in which to expand the coordinates of each point $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$. If the expansions of the fractional parts of these coordinates are $S_1, \ldots, S_n \in \Sigma_k^\infty$, respectively, where $\Sigma_k = \{0, \ldots, k-1\}$, and if S is the interleaving of these sequences, i.e.,

$$S = S_1[0]S_2[0] \ldots S_n[0]S_1[1]S_2[1] \ldots S_n[1]S_1[2]S_2[2] \ldots,$$

then the *dimension* of the point x is

$$\dim(x) = n \dim(S), \tag{1.7}$$

and the *strong dimension* of x is

$$\mathrm{Dim}(x) = n \mathrm{Dim}(S). \tag{1.8}$$

If one or more of the coordinates of x have two base-k expansions (because they are rationals whose denominators are powers of k), it is easily seen that the numbers $\dim(x)$ and $\mathrm{Dim}(x)$ are unaffected by how we choose between these base-k expansions. Also, a theorem of Staiger [47], in combination with (1.5) and (1.6), implies that $\dim(x)$ and $\mathrm{Dim}(x)$ do not depend on the choice of the base k. The dimension and strong dimension of a point $x \in \mathbb{R}^n$ are properties of the point x itself and not properties of a particular encoding.

Clearly, $0 \leq \dim(x) \leq \mathrm{Dim}(x) \leq n$ for all $x \in \mathbb{R}^n$. In fact, this is the only restriction that holds in general; i.e., for any two real numbers $0 \leq \alpha \leq \beta \leq n$, there is a point x in $\mathbb{R}^n$ with $\dim(x) = \alpha$ and $\mathrm{Dim}(x) = \beta$ [1].

The theory of computing thus assigns a dimension $\dim(x)$ and a strong dimension $\mathrm{Dim}(x)$ to each point x in Euclidean space. This assignment is robust (i.e., several natural approaches all lead to the same assignment), but is it geometrically meaningful? Prior work already indicates an affirmative answer. By Hitchcock's correspondence principle for constructive dimension ([26], extending a result of [46]), together with the absolute stability of constructive dimension [37], if $X \subseteq \mathbb{R}^n$ is any countable (not necessarily effective) union of computably closed, i.e., Π_1^0, sets, then $\mathrm{cdim}(X) = \dim_{\mathrm{H}}(X)$. Putting this together with (1.3) and (1.7), we have

$$\dim_{\mathrm{H}}(X) = \sup_{x \in X} \dim(x) \tag{1.9}$$

for any set X that is a union of computably closed sets. That is, the *classical* Hausdorff dimension [15] of any such set is completely determined by the dimensions of its individual points. Many, perhaps most, of the sets which arise in "standard" mathematical practice are unions of computably closed sets, so (1.9) constitutes strong prima facie evidence that the dimensions of individual points are indeed geometrically meaningful.

This paper analyzes the dimensions of points in the most widely known type of fractals, the self-similar fractals. The class of self-similar fractals includes such famous members as the Cantor set, the von Koch curve, the Sierpinski triangle, and the Menger sponge, along with many more exotic sets in Euclidean space [2, 12, 13, 15]. A self-similar fractal (defined precisely in section 5 below) is constructed from an *iterated function system (IFS)* $S = (S_0, \ldots, S_{k-1})$, which is a list of two or more contracting similarities mapping an initial nonempty, closed set $D \subseteq \mathbb{R}^n$ into itself. Each set $S_i(D)$ is a strictly smaller "copy" of D inside of D, and each set $S_i(S_j(D))$

is a strictly smaller "copy" of $S_j(D)$ inside of $S_i(D)$. Continuing in this way, each sequence $T \in \Sigma_k^\infty$ encodes a nested sequence

$$D \supseteq S_{T[0]}(D) \supseteq S_{T[0]}(S_{T[1]}(D)) \supseteq \dots \tag{1.10}$$

of nonempty, closed sets in $\mathbb{R}^n$. Each of these sets is strictly smaller than the one preceding it, because each similarity S_i has a contraction ratio $c_i \in (0,1)$. Hence there is a unique point $S(T) \in \mathbb{R}^n$ that is an element of all the sets in (1.10). Figure 1 illustrates how a *coding sequence* T represents a point $S(T)$ in the Sierpinski triangle.

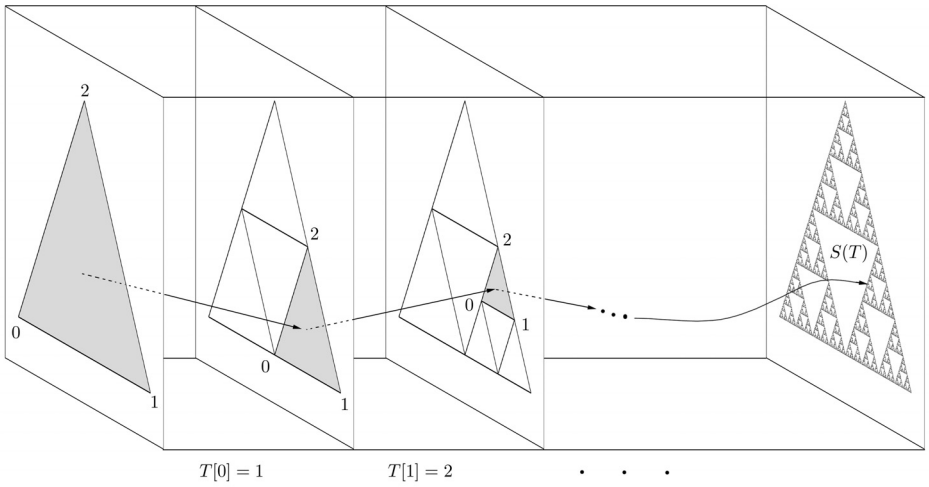


FIG. 1. *A sequence $T \in \{0,1,2\}^\infty$ codes a point $S(T)$ in the Sierpinski triangle.*

The *attractor* of the IFS S is the set

$$F(S) = \{ S(T) \mid T \in \Sigma_k^\infty \}. \tag{1.11}$$

In general, the sets $S_0(D), \dots, S_{k-1}(D)$ need not be disjoint, so a point $x \in F(S)$ may have many *coding sequences*, i.e., many sequences T for which $S(T) = x$. A *self-similar fractal* is a set $F \subseteq \mathbb{R}^n$ that is the attractor of an IFS S that satisfies a technical *open set condition* (defined in section 5), which ensures that the sets $S_0(D), \dots, S_{k-1}(D)$ are "nearly" disjoint.

The *similarity dimension* of an IFS S is the (unique) solution $\mathrm{sdim}(S)$ of the equation

$$\sum_{i=0}^{k-1} c_i^{\mathrm{sdim}(S)} = 1, \tag{1.12}$$

where $c_0, \dots, c_{k-1}$ are the contraction ratios of the similarities $S_0, \dots, S_{k-1}$, respectively. The *similarity dimension* of a self-similar fractal $F = F(S)$ is $\mathrm{sdim}(F) = \mathrm{sdim}(S)$. A classical theorem of Moran [40] and Falconer [14] says that, for any self-similar fractal F,

$$\dim_{\mathrm{H}}(F) = \mathrm{Dim}_{\mathrm{P}}(F) = \mathrm{sdim}(F), \tag{1.13}$$

i.e., the Hausdorff and packing dimensions of F coincide with its similarity dimension. In addition to its theoretical interest, the Moran–Falconer theorem has the pragmatic

consequence that the Hausdorff and packing dimensions of a self-similar fractal are easily computed from the contraction ratios by solving (1.12).

Our main theorem concerns the dimensions of points in fractals that are *computably self-similar*, meaning that they are attractors of *computable* IFSs satisfying the open set condition. (We note that most self-similar fractals occurring in practice—including the four famous examples mentioned above—are, in fact, computably self-similar.) Our main theorem says that, if F is any fractal that is computably self-similar with the IFS S as witness, then, for every point $x \in F$ and every coding sequence T for x, the dimension and strong dimension of the point x are given by the dimension formulas

$$\dim(x) = \mathrm{sdim}(F)\dim^{\pi_S}(T) \tag{1.14}$$

and

$$\mathrm{Dim}(x) = \mathrm{sdim}(F)\mathrm{Dim}^{\pi_S}(T), \tag{1.15}$$

where $\dim^{\pi_S}(T)$ and $\mathrm{Dim}^{\pi_S}(T)$ are the dimension and strong dimension of T *with respect to* the probability measure π_S on the alphabet Σ_k defined by

$$\pi_S(i) = c_i^{\mathrm{sdim}(F)} \tag{1.16}$$

for all $i \in \Sigma_k$. (We note that $\dim^{\pi_S}(T)$ is the constructive analogue of Billingsley dimension [3, 9].) This theorem gives a complete analysis of the dimensions of points in computably self-similar fractals and the manner in which the dimensions of these points arise from the dimensions of their coding sequences.

Although our main theorem applies directly only to computably self-similar fractals, we use relativization to show that the Moran–Falconer theorem (1.13) for arbitrary self-similar fractals is an easy consequence of our main theorem. Hence, as is often the case, a theorem of computable analysis (i.e., the theoretical foundations of scientific computing [6]) has an immediate corollary in classical analysis.

The proof of our main theorem has some geometric and combinatorial similarities with the classical proofs of Moran [40] and Falconer [14], but the argument here is information-theoretic. Specifically, our proof uses Kolmogorov complexity characterizations of dimensions with respect to probability measures. These characterizations (proven in section 4 below) say that, if ν is a suitable probability measure on a sequence space Σ^∞, then, for every sequence $S \in \Sigma^\infty$,

$$\dim^\nu(S) = \liminf_{j\to\infty} \frac{\mathrm{K}(S[0..j-1])}{\mathcal{I}_\nu(S[0..j-1])} \tag{1.17}$$

and

$$\mathrm{Dim}^\nu(S) = \limsup_{j\to\infty} \frac{\mathrm{K}(S[0..j-1])}{\mathcal{I}_\nu(S[0..j-1])}, \tag{1.18}$$

where

$$\mathcal{I}_\nu(w) = \log \frac{1}{\nu(w)}$$

is the Shannon self-information of the string w with respect to the probability measure ν [10]. The older characterizations (1.5) and (1.6) are the special cases of (1.17) and

(1.18) in which $\nu(w) = |\Sigma|^{-|w|}$ for all w in Σ^*. The characterizations (1.17) and (1.18) say that $\dim^\nu(S)$ and $\mathrm{Dim}^\nu(S)$ are asymptotic measures of the algorithmic information density of S, but the "density" here is now an information-to-cost ratio. In this ratio, the "information" is algorithmic information, i.e., Kolmogorov complexity, and the "cost" is the Shannon self-information. To see why this makes sense, consider the case of interest in our main theorem. In this case, (1.16) says that

$$\nu(w) = \prod_{j=0}^{|w|-1} c_{w[j]}^{\mathrm{sdim}(F)},$$

whence the cost of a string $w \in \Sigma_k^*$ is

$$\mathcal{I}_\nu(w) = \mathrm{sdim}(F) \sum_{j=0}^{|w|-1} \log \frac{1}{c_{w[j]}},$$

i.e., the sum of the costs of the symbols in w, where the cost of a symbol $i \in \Sigma_k$ is $\mathrm{sdim}(F)\log(1/c_i)$. These symbol costs are computational and realistic. A symbol i with high cost invokes a similarity S_i with a small contraction ratio c_i, thereby necessitating a high-precision computation.

We briefly mention some other recent research on fractals in theoretical computer science. Braverman and Cook [5, 6] have used computability and complexity of various fractals to explore the relationships between the two main models of real computation. Rettinger and Weihrauch [42], Hertling [23], and Braverman and Yampolsky [7] have investigated computability and complexity properties of Mandelbrot and Julia sets. Gupta, Krauthgamer, and Lee [21] have used fractal geometry to prove lower bounds on the distortions of certain embeddings of metric spaces. Most of the fractals involved in these papers are more exotic than the self-similar fractals that we investigate here. Cai and Hartmanis [8] and Fernau and Staiger [17] have investigated Hausdorff dimension in self-similar fractals and their coding spaces. This work is more closely related to the present paper, but the motivations and results are different. Our focus here is on a *pointwise* analysis of dimensions.

Some of the most difficult open problems in geometric measure theory involve establishing lower bounds on the fractal dimensions of various sets. Kolmogorov complexity has proven to be a powerful tool for lower-bound arguments, leading to the solution of many long-standing open problems in discrete mathematics [33]. There is thus reason to hope that our pointwise approach to fractal dimension, coupled with the introduction of Kolmogorov complexity techniques, will lead to progress in this classical area. In any case, our results extend computable analysis [41, 28, 54] in a new, geometric direction.

The rest of this paper is organized as follows. Section 2 summarizes basic terminology and notation. Section 3 develops the basic theory of constructive dimensions with respect to probability measures. Section 4 establishes the Kolmogorov complexity characterizations of these dimensions. Section 5 is a brief exposition of self-similar fractals for readers who are not familiar with IFSs. Section 6 proves our main theorem and derives the Moran–Falconer theorem from it.

2. Preliminaries. Given a finite alphabet Σ, we write Σ^* for the set of all (finite) *strings* over Σ and Σ^∞ for the set of all (infinite) *sequences* over Σ. If $\psi \in \Sigma^* \cup \Sigma^\infty$ and $0 \le i \le j < |\psi|$, where $|\psi|$ is the length of ψ, then $\psi[i]$ is the ith symbol in ψ (where $\psi[0]$ is the leftmost symbol in ψ), and $\psi[i..j]$ is the string consisting of the ith

through the jth symbols in ψ. If $w \in \Sigma^*$ and $\psi \in \Sigma^* \cup \Sigma^\infty$, then w is a prefix of ψ, and we write $w \sqsubseteq \psi$ if there exists $i \in \mathbb{N}$ such that $w = \psi[0..i-1]$. If $A \subseteq \Sigma^*$, then $A^{=n} = \{x \mid x \in A \wedge |x| = n\}$.

For functions on Euclidean space, we use the computability notion formulated by Grzegorczyk [19] and Lacombe [29, 30, 31] in the 1950's and exposited in the monographs by Pour-El and Richards [41], Ko [28], and Weihrauch [54] and in the recent survey paper by Braverman and Cook [6]. A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is *computable* if there is an oracle Turing machine M with the following property. For all $x \in \mathbb{R}^n$ and $r \in \mathbb{N}$, if M is given a function oracle $\varphi_x : \mathbb{N} \to \mathbb{Q}^n$ such that, for all $k \in \mathbb{N}$, $|\varphi_x(k) - x| \leq 2^{-k}$, then M, with oracle φ_x and input r, outputs a rational point $M^{\varphi_x}(r) \in \mathbb{Q}^n$ such that $|M^{\varphi_x}(r) - f(x)| \leq 2^{-r}$.

A point $x \in \mathbb{R}^n$ is *computable* if there is a computable function $\psi_x : \mathbb{N} \to \mathbb{Q}^n$ such that, for all $r \in \mathbb{N}$, $|\psi_x(r) - x| \leq 2^{-r}$.

For subsets of Euclidean space, we use the computability notion introduced by Brattka and Weihrauch [4] (see also [54, 6]). A set $X \subseteq \mathbb{R}^n$ is computable if there is a computable function $f_X : \mathbb{Q}^n \times \mathbb{N} \to \{0, 1\}$ that satisfies the following two conditions for all $q \in \mathbb{Q}^n$ and $r \in \mathbb{N}$.

(i) If there exists $x \in X$ such that $|x - q| \leq 2^{-r}$, then $f_X(q, r) = 1$.

(ii) If there is no $x \in X$ such that $|x - q| \leq 2^{1-r}$, then $f_X(q, r) = 0$.

The following two observations are well known and easy to verify.

OBSERVATION 2.1. *A nonempty set $X \subseteq \mathbb{R}^n$ is computable if and only if the associated distance function*

$$\rho_X : \mathbb{R}^n \to [0, \infty),$$
$$\rho_X(y) = \inf_{x \in X} |x - y|$$

is computable.

OBSERVATION 2.2. *If $X \subseteq \mathbb{R}^n$ is both computable and closed, then X is a computably closed, i.e., Π^0_1, set.*

All logarithms in this paper are base-2.

3. Dimensions relative to probability measures. Here we develop the basic theory of constructive fractal dimension on a sequence space Σ^∞ with respect to a suitable probability measure on Σ^∞. We first review the classical Hausdorff and packing dimensions.

Let ρ be a metric on a set $\mathcal{X}$. We use the following standard terminology. The *diameter* of a set $X \subseteq \mathcal{X}$ is

$$\operatorname{diam}(X) = \sup \{\rho(x, y) \mid x, y \in X\}$$

(which may be ∞). For each $x \in \mathcal{X}$ and $r \in \mathbb{R}$, the *closed ball* of radius r about x is the set

$$B(x, r) = \{y \in \mathcal{X} \mid \rho(y, x) \leq r\},$$

and the *open ball* of radius r about x is the set

$$B^o(x, r) = \{y \in \mathcal{X} \mid \rho(y, x) < r\}.$$

A *ball* is any set of the form $B(x, r)$ or $B^o(x, r)$. A ball B is *centered* in a set $X \subseteq \mathcal{X}$ if $B = B(x, r)$ or $B = B^o(x, r)$ for some $x \in X$ and $r \geq 0$.

For each $\delta > 0$, we let $\mathcal{C}_\delta$ be the set of all countable collections $\mathcal{B}$ of balls such that $\operatorname{diam}(B) \le \delta$ for all $B \in \mathcal{B}$, and we let $\mathcal{D}_\delta$ be the set of all $\mathcal{B} \in \mathcal{C}_\delta$ such that the balls in $\mathcal{B}$ are pairwise disjoint. For each $X \subseteq \mathcal{X}$ and $\delta > 0$, we define the sets

$$\mathcal{H}_\delta(X) = \left\{ \mathcal{B} \in \mathcal{C}_\delta \;\middle|\; X \subseteq \bigcup_{B \in \mathcal{B}} B \right\},$$
$$\mathcal{P}_\delta(X) = \{ \mathcal{B} \in \mathcal{D}_\delta \mid (\forall B \in \mathcal{B}) B \text{ is centered in } X \}.$$

If $\mathcal{B} \in \mathcal{H}_\delta(X)$, then we call $\mathcal{B}$ a *δ-cover* of X. If $\mathcal{B} \in \mathcal{P}_\delta(X)$, then we call $\mathcal{B}$ a *δ-packing* of X. For $X \subseteq \mathcal{X}$, $\delta > 0$, and $s \ge 0$, we define the quantities

$$H^s_\delta(X) = \inf_{\mathcal{B} \in \mathcal{H}_\delta(X)} \sum_{B \in \mathcal{B}} \operatorname{diam}(B)^s,$$
$$P^s_\delta(X) = \sup_{\mathcal{B} \in \mathcal{P}_\delta(X)} \sum_{B \in \mathcal{B}} \operatorname{diam}(B)^s.$$

Since $H^s_\delta(X)$ and $P^s_\delta(X)$ are monotone as $\delta \to 0$, the limits

$$H^s(X) = \lim_{\delta \to 0} H^s_\delta(X),$$
$$P^s_0(X) = \lim_{\delta \to 0} P^s_\delta(X)$$

exist, though they may be infinite. Let

$$P^s(X) = \inf \left\{ \sum_{i=0}^{\infty} P^s_0(X_i) \;\middle|\; X \subseteq \bigcup_{i=0}^{\infty} X_i \right\}. \tag{3.1}$$

It is routine to verify that the set functions H^s and P^s are outer measures [15]. The quantities $H^s(X)$ and $P^s(X)$—which may be infinite—are called the *s-dimensional Hausdorff (outer) ball measure* and the *s-dimensional packing (outer) ball measure* of X, respectively. The optimization (3.1) over all countable partitions of X is needed because the set function P^s_0 is *not* an outer measure.

DEFINITION 3.1. *Let ρ be a metric on a set $\mathcal{X}$, and let $X \subseteq \mathcal{X}$.*

1. *(Hausdorff* [22]*). The* Hausdorff dimension *of X with respect to ρ is*

$$\dim^{(\rho)}_{\mathrm{H}}(X) = \inf \{ s \in [0, \infty) \mid H^s(X) = 0 \}.$$

2. *(Tricot* [50]*, Sullivan* [49]*). The* packing dimension *of X with respect to ρ is*

$$\operatorname{Dim}^{(\rho)}_{\mathrm{P}}(X) = \inf \{ s \in [0, \infty) \mid P^s(X) = 0 \}.$$

When $\mathcal{X}$ is a Euclidean space $\mathbb{R}^n$ and ρ is the usual Euclidean metric on $\mathbb{R}^n$, $\dim^{(\rho)}_{\mathrm{H}}$ and $\operatorname{Dim}^{(\rho)}_{\mathrm{P}}$ are the ordinary Hausdorff and packing dimensions, also denoted by $\dim_{\mathrm{H}}$ and $\operatorname{Dim}_{\mathrm{P}}$, respectively.

We now focus our attention on sequence spaces. Let Σ be a finite alphabet with $|\Sigma| \ge 2$. A *(Borel) probability measure* on Σ^∞ is a function $\nu : \Sigma^* \to [0, 1]$ such that $\nu(\lambda) = 1$ and $\nu(w) = \sum_{a \in \Sigma} \nu(wa)$ for all $w \in \Sigma^*$. Intuitively, $\nu(w)$ is the probability that $w \sqsubseteq S$ when a sequence $S \in \Sigma^\infty$ is chosen according to the probability measure ν. A probability measure ν on Σ^∞ is *strongly positive* if there exists $\delta > 0$ such that, for all $w \in \Sigma^*$ and $a \in \Sigma$, $\nu(wa) > \delta \nu(w)$.

The following type of probability measure is used in our main theorem.

Example 3.2. Let π be a probability measure on the alphabet Σ, i.e., a function $\pi : \Sigma \to [0,1]$ such that $\sum_{a\in\Sigma} \pi(a) = 1$. Then π induces the *product probability measure* π on Σ^∞ defined by

$$\pi(w) = \prod_{i=0}^{|w|-1} \pi(w[i])$$

for all $w \in \Sigma^*$. If π is *positive* on Σ, i.e., $\pi(a) > 0$ for all $a \in \Sigma$, then the probability measure π on Σ^∞ is strongly positive.

Example 3.3. We reserve the symbol μ for the *uniform probability measure* on Σ^∞, which is the function $\mu : \Sigma^* \to [0,\infty)$ defined by

$$\mu(w) = |\Sigma|^{-|w|}$$

for all $w \in \Sigma^*$. Note that this is the special case of Example 3.2 in which $\pi(a) = 1/|\Sigma|$ for each $a \in \Sigma$.

Definition 3.4. *The* metric induced by *a strongly positive probability measure ν on Σ^∞ is the function $\rho_\nu : \Sigma^\infty \times \Sigma^\infty \to [0,1]$ defined by*

$$\rho_\nu(S,T) = \inf\{\nu(w) \mid w \sqsubseteq S \text{ and } w \sqsubseteq T\}$$

for all $S, T \in \Sigma^\infty$.

The following fact is easily verified.

Observation 3.5. *For every strongly positive probability measure ν on Σ^∞, the function ρ_ν is a metric on Σ^∞.*

Hausdorff and packing dimensions with respect to probability measures on sequence spaces are defined as follows.

Definition 3.6. *Let Σ be a finite alphabet with $|\Sigma| \geq 2$, let ν be a strongly positive probability measure on Σ^∞, and let $X \subseteq \Sigma^\infty$.*

1. *The* Hausdorff dimension *of X with respect to ν (also called the* Billingsley dimension *of X with respect to ν [3, 9]) is*

$$\dim_{\mathrm{H}}^{\nu}(X) = \dim_{\mathrm{H}}^{(\rho_\nu)}(X).$$

2. *The* packing dimension *of X with respect to ν is*

$$\mathrm{Dim}_{\mathrm{P}}^{\nu}(X) = \mathrm{Dim}_{\mathrm{P}}^{(\rho_\nu)}(X).$$

Note. We have assumed strong positivity here for clarity of presentation, but this assumption can be weakened in various ways for various results.

When ν is the probability measure μ, it is generally omitted from the terminology. Thus, the *Hausdorff dimension* of X is $\dim_{\mathrm{H}}(X) = \dim_{\mathrm{H}}^{\mu}(X)$, and the *packing dimension* of X is $\mathrm{Dim}_{\mathrm{P}}(X) = \mathrm{Dim}_{\mathrm{P}}^{\mu}(X)$.

It was apparently Wegmann [53] who first noticed that the metric ρ_ν could be used to make Billingsley dimension a special case of Hausdorff dimension. Fernau and Staiger [17] have also investigated this notion.

We now develop gale characterizations of $\dim_{\mathrm{H}}^{\nu}$ and $\mathrm{Dim}_{\mathrm{P}}^{\nu}$.

Definition 3.7. *Let Σ be a finite alphabet with $|\Sigma| \geq 2$, let ν be a probability measure on Σ^∞, and let $s \in [0,\infty)$.*

1. *A* ν-s-supergale *is a function* $d : \Sigma^* \to [0, \infty)$ *that satisfies the condition*

$$d(w)\nu(w)^s \geq \sum_{a \in \Sigma} d(wa)\nu(wa)^s \tag{3.2}$$

for all $w \in \Sigma^*$.
2. *A* ν-s-gale *is a* ν-s-*supergale that satisfies* (3.2) *with equality for all* $w \in \Sigma^*$.
3. *A* ν-supermartingale *is a* ν-1-*supergale.*
4. *A* ν-martingale *is a* ν-1-*gale.*
5. *An* s-supergale *is a* μ-s-*supergale.*
6. *An* s-gale *is a* μ-s-*gale.*
7. *A* supermartingale *is a* 1-*supergale.*
8. *A* martingale *is a* 1-*gale.*

The following observation shows how gales and supergales are affected by variation of the parameter s.

Observation 3.8 (see[37]). *Let* ν *be a probability measure on* Σ^∞, *let* $s, s' \in [0, \infty)$, *and let* $d, d' : \Sigma^* \to [0, \infty)$. *Assume that*

$$d(w)\nu(w)^s = d'(w)\nu(w)^{s'}$$

holds for all $w \in \Sigma^*$.

1. d *is a* ν-s-*supergale if and only if* d' *is a* ν-s'-*supergale.*
2. d *is a* ν-s-*gale if and only if* d' *is a* ν-s'-*gale.*

For example, if the probability measure ν is positive, then a function $d : \Sigma^* \to [0, \infty)$ is a ν-s-gale if and only if the function $d' : \Sigma^* \to [0, \infty)$ defined by $d'(w) = \nu(w)^{s-1}d(w)$ is a ν-martingale.

Martingales were introduced by Lévy [32] and Ville [52]. They have been used extensively by Schnorr [43, 44, 45] and others in investigations of randomness and by Lutz [34, 35] and others in the development of resource-bounded measure. Gales are a convenient generalization of martingales introduced by Lutz [36, 37] in the development of effective fractal dimensions.

The following generalization of Kraft's inequality [10] is often useful.

Lemma 3.9 (see [37]). *Let* d *be a* ν-s-*supergale, where* ν *is a probability measure on* Σ^∞ *and* $s \in [0, \infty)$. *Then, for all* $w \in \Sigma^*$ *and all prefix sets* $B \subseteq \Sigma^*$,

$$\sum_{u \in B} d(wu)\nu(wu)^s \leq d(w)\nu(w)^s.$$

Intuitively, a ν-s-gale d is a strategy for betting on the successive symbols in a sequence $S \in \Sigma^\infty$. We regard the value $d(w)$ as the amount of money that a gambler using the strategy d will have after betting on the symbols in w if w is a prefix of S. If $s = 1$, then the ν-s-gale identity

$$d(w)\nu(w)^s = \sum_{a \in \Sigma} d(wa)\nu(wa)^s \tag{3.3}$$

ensures that the payoffs are fair in the sense that the conditional ν-expected value of the gambler's capital after the symbol following w, given that w has occurred, is precisely $d(w)$, the gambler's capital after w. If $s < 1$, then (3.3) says that the payoffs are less than fair. If $s > 1$, then (3.3) says that the payoffs are more than fair. Clearly, the smaller s is, the more hostile the betting environment is.

There are two important notions of success for a supergale.

DEFINITION 3.10. *Let d be a ν-s-supergale, where ν is a probability measure on Σ^∞ and $s \in [0,\infty)$, and let $S \in \Sigma^\infty$.*

1. *We say that d* succeeds *on S, and we write $S \in S^\infty[d]$ if* $\limsup_{t\to\infty} d(S[0..t-1]) = \infty$.
2. *We say that d* succeeds strongly *on S and we write $S \in S^\infty_{\mathrm{str}}[d]$ if*

$$\liminf_{t\to\infty} d(S[0..t-1]) = \infty.$$

Notation. Let ν be a probability measure on Σ^∞, and let $X \subseteq \Sigma^\infty$.

1. $\mathcal{G}^\nu(X)$ is the set of all $s \in [0,\infty)$ such that there is a ν-s-gale d for which $X \subseteq S^\infty[d]$.
2. $\mathcal{G}^{\nu,\mathrm{str}}(X)$ is the set of all $s \in [0,\infty)$ such that there is a ν-s-gale d for which $X \subseteq S^\infty_{\mathrm{str}}[d]$.
3. $\widehat{\mathcal{G}}^\nu(X)$ is the set of all $s \in [0,\infty)$ such that there is a ν-s-supergale d for which $X \subseteq S^\infty[d]$.
4. $\widehat{\mathcal{G}}^{\nu,\mathrm{str}}(X)$ is the set of all $s \in [0,\infty)$ such that there is a ν-s-supergale d for which $X \subseteq S^\infty_{\mathrm{str}}[d]$.

The following theorem gives useful characterizations of the classical Hausdorff and packing dimensions with respect to probability measures on sequence spaces.

THEOREM 3.11 (gale characterizations of $\dim^\nu_{\mathrm{H}}(X)$ and $\mathrm{Dim}^\nu_{\mathrm{P}}(X)$). *If ν is a strongly positive probability measure on Σ^∞, then, for all $X \subseteq \Sigma^\infty$,*

$$\dim^\nu_{\mathrm{H}}(X) = \inf \mathcal{G}^\nu(X) = \inf \widehat{\mathcal{G}}^\nu(X) \tag{3.4}$$

and

$$\mathrm{Dim}^\nu_{\mathrm{P}}(X) = \inf \mathcal{G}^{\nu,\mathrm{str}}(X) = \inf \widehat{\mathcal{G}}^{\nu,\mathrm{str}}(X). \tag{3.5}$$

Proof. In this proof we will use the following notation for each $w \in \Sigma^*$, $\mathbf{C}_w = \{S \in \Sigma^\infty \mid w \sqsubseteq S\}$.

Notice that for each $S \in \Sigma^\infty$, $r > 0$, the balls $B(S,r) = \mathbf{C}_v$, $B^o(S,r) = \mathbf{C}_w$ for some $v, w \in \Sigma^*$. Therefore, either two balls $\mathbf{C}_w$, $\mathbf{C}_{w'}$ are disjoint or one is contained in the other.

In order to prove (3.4) it suffices to show that for all $s \in [0,\infty)$,

$$H^s(X) = 0 \implies s \in \mathcal{G}^\nu(X) \implies s \in \widehat{\mathcal{G}}^\nu(X) \implies H^s(X) = 0.$$

First, assume that $H^s(X) = 0$. Then $H^s_1(X) = 0$, which implies that for each $r \in \mathbb{N}$ there is a disjoint cover $\mathcal{B} \in \mathcal{C}_1$ such that $\sum_{B\in\mathcal{B}} \mathrm{diam}(B)^s < 2^{-r}$. Let $A_r = \{w \in \Sigma^* \mid \mathbf{C}_w \in \mathcal{B}\}$.

We define a function $d : \Sigma^* \to [0,\infty)$ as follows. Let $w \in \Sigma^*$. If there exists $v \sqsubseteq w$ such that $v \in A_r$, then

$$d_r(w) = \left(\frac{\nu(w)}{\nu(v)}\right)^{1-s}.$$

Otherwise,

$$d_r(w) = \sum_{\substack{u,\\ wu\in A_r}} \left(\frac{\nu(wu)}{\nu(w)}\right)^s.$$

It is routine to verify that the following conditions hold for all $r \in \mathbb{N}$.

(i) d_r is a ν-s-gale.
(ii) $d_r(\lambda) < 2^{-r}$.
(iii) For all $w \in A_r$, $d_r(w) = 1$.

Let $d = \sum_{r=0}^{\infty} 2^r d_{2r}$. Notice that d is a ν-s-gale. To see that $X \subseteq S^\infty[d]$, let $T \in X$, and let $r \in \mathbb{N}$ be arbitrary. Since $\mathcal{B}$ covers X, there exists $w \in A_{2r}$ such that $w \sqsubseteq T$. Then by (iii) above, $d(w) \geq 2^r d_{2r}(w) = 2^r$. Since $r \in \mathbb{N}$ is arbitrary, this shows that $T \in S^\infty[d]$, confirming that $X \subseteq S^\infty[d]$.

We have now shown that d is a ν-s-gale such that $X \subseteq S^\infty[d]$, whence $s \in \mathcal{G}^\nu(X)$.

Conversely, assume that $s \in \widehat{\mathcal{G}}^\nu(X)$. To see that $H^s(X) = 0$, let $\delta > 0$, $r \in \mathbb{N}$. It suffices to show that $H^s(X) \leq 2^{-r}$. If $X = \emptyset$, this is trivial, so assume that $X \neq \emptyset$.

Since $s \in \widehat{\mathcal{G}}^\nu(X)$, there is a ν-s-supergale d such that $X \subseteq S^\infty[d]$. Note that $d(\lambda) > 0$ because $X \neq \emptyset$. Let

$$A = \{w \in \Sigma^* \mid \nu(w) < \delta, d(w) \geq 2^r d(\lambda) \text{ and } (\forall v)[v \sqsubset w \implies v \notin A]\}.$$

It is clear that A is a prefix set. It is also clear that $\mathcal{B} = \{\mathbf{C}_w \mid w \in A\}$ is a δ-cover of $S^\infty[d]$, and since $X \subseteq S^\infty[d]$, $\mathcal{B}$ is also a δ-cover of X. By Lemma 3.9 and the definition of A, we have

$$d(\lambda) \geq \sum_{w \in A} \nu(w)^s d(w) \geq 2^r d(\lambda) \sum_{w \in A} \nu(w)^s.$$

Since $B \in \mathcal{C}_\delta(X)$ and $d(\lambda) > 0$, it follows that

$$H^s_\delta(X) \leq \sum_{w \in A} \nu(w)^s \leq 2^{-r}.$$

This completes the proof of (3.4).

The proof of (3.5) is based on the following three claims.

Claim 1. *For each family $X_i \subseteq \Sigma^\infty$, $i \in \mathbb{N}$, $\inf \mathcal{G}^{\nu,\mathrm{str}}(\cup_i X_i) = \sup_i \inf \mathcal{G}^{\nu,\mathrm{str}}(X_i)$.*

Claim 2. *For each $X \subseteq \Sigma^\infty$, if $P^s_0(X) < \infty$, then $\inf \mathcal{G}^{\nu,\mathrm{str}}(X) \leq s$.*

Claim 3. *For each $X \subseteq \Sigma^\infty$, if $s > \inf \widehat{\mathcal{G}}^{\nu,\mathrm{str}}(X)$, then $P^s(X) = 0$.*

Proof of Claim 1. The $\geq$ inequality follows from the definition of $\mathcal{G}^{\nu,\mathrm{str}}()$. To prove that $\inf \mathcal{G}^{\nu,\mathrm{str}}(\cup_i X_i) \leq \sup_i \inf \mathcal{G}^{\nu,\mathrm{str}}(X_i)$, let $s > \sup_i \inf \mathcal{G}^{\nu,\mathrm{str}}(X_i)$. Assume that $X_i \neq \emptyset$ for every i, since otherwise the proof is similar taking only nonempty X_i's. Then for each $i \in \mathbb{N}$ there is a ν-s-gale d_i such that $X_i \subseteq S^\infty_{\mathrm{str}}[d_i]$. We define a ν-s-gale d by

$$d(w) = \sum_i \frac{2^{-i}}{d_i(\lambda)} d_i(w)$$

for all $w \in \Sigma^*$. Then for each i, for any $S \in X_i$, we have

$$d(S[0..n-1]) \geq \frac{2^{-i}}{d_i(\lambda)} d_i(S[0..n-1])$$

for all n, so $S \in S^\infty_{\mathrm{str}}[d]$. Therefore, $\cup_i X_i \subseteq S^\infty_{\mathrm{str}}[d]$ and the claim follows. □

Proof of Claim 2. Assume that $P^s_0(X) < \infty$. Let $\epsilon > 0$. Let

$$A = \{w \mid w \in \Sigma^* \text{ and } C_w \cap X \neq \emptyset\}.$$

Notice that there is a constant c such that for every n, $\sum_{w \in A^{=n}} \nu(w)^s < c$ and that for each $T \in X$, for every n, $T[0..n-1] \in A$. For each $n \in \mathbb{N}$ we define $d_n : \Sigma^* \to [0, \infty)$

similarly to the first part of this proof; that is, we let $w \in \Sigma^*$. If there exists $v \sqsubseteq w$ such that $v \in A^{=n}$, then

$$d_n(w) = \left(\frac{\nu(w)}{\nu(v)}\right)^{1-s}.$$

Otherwise,

$$d_n(w) = \sum_{\substack{u,\\ wu \in A^{=n}}} \left(\frac{\nu(wu)}{\nu(w)}\right)^{s}.$$

d_n is a ν-s-gale, $d_n(\lambda) = \sum_{u \in A^{=n}} \nu(u)^s$, and for all $w \in A^{=n}$, $d_n(w) = 1$.

Let $d(w) = \sum_{n=0}^{\infty} \nu(w)^{-\epsilon} d_n(w)$. Notice that d is a ν-$(s+\epsilon)$-gale. To see that $X \subseteq S^{\infty}_{\mathrm{str}}[d]$, let $T \in X$ and let n be arbitrary. Since $T[0..n-1] \in A$,

$$d(T[0..n-1]) \geq \nu(T[0..n-1])^{-\epsilon} d_n(T[0..n-1]) \geq \nu(T[0..n-1])^{-\epsilon}.$$

Since $\nu(T[0..n-1]) \stackrel{n\to\infty}{\longrightarrow} 0$, this shows that $T \in S^{\infty}_{\mathrm{str}}[d]$. Therefore, $X \subseteq S^{\infty}_{\mathrm{str}}[d]$ and $\inf \mathcal{G}^{\nu,\mathrm{str}}(X) \leq s + \epsilon$ for arbitrary ϵ, so the claim follows. □

Proof of Claim 3. Let $s > t > \inf \widehat{\mathcal{G}}^{\nu,\mathrm{str}}(X)$. To see that $P^s(X) = 0$, let d be a ν-t-supergale such that $X \subseteq S^{\infty}_{\mathrm{str}}[d]$. Let $i \in \mathbb{N}$ and

$$X_i = \{T \mid \forall n \geq i, d(T[0..n-1]) > d(\lambda)\}.$$

Then $X \subseteq \cup_i X_i$. For each $i \in \mathbb{N}$ we prove that $P^s_0(X_i) = 0$.

Let $\delta_i = \min_{|w| \leq i} \nu(w)$. Let $\delta < \delta_i$ and B be a δ-packing of X_i; then $B \subseteq \{w \mid d(w) > d(\lambda)\}$ and $\sum_{w \in B} \nu(w)^t \leq 1$. Therefore, $P^t_0(X_i) \leq 1$ and $P^s_0(X_i) = 0$ (since $\sum_{w \in B} \nu(w)^s \leq \delta^{s-t} \stackrel{\delta\to 0}{\longrightarrow} 0$). Therefore $P^s(X) = 0$ and the claim follows. □

We next prove (3.5). $\inf \mathcal{G}^{\nu,\mathrm{str}}(X) \leq \mathrm{Dim}^{\nu}_{\mathrm{P}}(X)$ follows from Claims 1 and 2, and $\mathrm{Dim}^{\nu}_{\mathrm{P}}(X) \leq \widehat{\mathcal{G}}^{\nu,\mathrm{str}}(X)$ from Claim 3. □

We note that the case $\nu = \mu$ of (3.4) was proven by Lutz [36], and the case $\nu = \mu$ of (3.5) was proven by Athreya et al. [1].

Guided by Theorem 3.11, we now develop the constructive fractal ν-dimensions.

DEFINITION 3.12. *A ν-s-supergale d is constructive if it is lower semicomputable, i.e., if there is an exactly computable function $\widehat{d} : \Sigma^* \times \mathbb{N} \to \mathbb{Q}$ with the following two properties.*

(i) *For all $w \in \Sigma^*$ and $t \in \mathbb{N}$, $\widehat{d}(w,t) \leq \widehat{d}(w,t+1) < d(w)$.*

(ii) *For all $w \in \Sigma^*$, $\lim_{t\to\infty} \widehat{d}(w,t) = d(w)$.*

Notation. For each probability measure ν on Σ^∞ and each set $X \subseteq \Sigma^\infty$, we define the sets $\mathcal{G}^{\nu}_{\mathrm{constr}}(X)$, $\mathcal{G}^{\nu,\mathrm{str}}_{\mathrm{constr}}(X)$, $\widehat{\mathcal{G}}^{\nu}_{\mathrm{constr}}(X)$, and $\widehat{\mathcal{G}}^{\nu,\mathrm{str}}_{\mathrm{constr}}(X)$ exactly like the sets $\mathcal{G}^{\nu}(X)$, $\mathcal{G}^{\nu,\mathrm{str}}(X)$, $\widehat{\mathcal{G}}^{\nu}(X)$, and $\widehat{\mathcal{G}}^{\nu,\mathrm{str}}(X)$, respectively, except that the gales and supergales d are now required to be constructive.

DEFINITION 3.13. *Let ν be a probability measure on Σ^∞, and let $X \subseteq \Sigma^\infty$.*

1. *The* constructive ν-dimension *of X is* $\mathrm{cdim}^{\nu}(X) = \inf \widehat{\mathcal{G}}^{\nu}_{\mathrm{constr}}(X)$.
2. *The* constructive strong ν-dimension *of X is* $\mathrm{cDim}^{\nu}(X) = \inf \widehat{\mathcal{G}}^{\nu,\mathrm{str}}_{\mathrm{constr}}(X)$.
3. *The* constructive dimension *of X is* $\mathrm{cdim}(X) = \mathrm{cdim}^{\mu}(X)$.
4. *The* constructive strong dimension *of X is* $\mathrm{cDim}(X) = \mathrm{cDim}^{\mu}(X)$.

The fact that the "unhatted" $\mathcal{G}$-classes can be used in place of the "hatted" $\widehat{\mathcal{G}}$-classes is not as obvious in the constructive case as in the classical case. Nevertheless,

Fenner [16] proved that this is the case for constructive ν-dimension. (Hitchcock [25] proved this independently for the case $\nu = \mu$.) The case of strong ν-dimension also holds with a more careful argument [1].

THEOREM 3.14 (Fenner [16]). *If ν is a strongly positive, computable probability measure on Σ^∞, then, for all $X \subseteq \Sigma^\infty$,*

$$\mathrm{cdim}^\nu(X) = \inf \mathcal{G}^\nu_{\mathrm{constr}}(X)$$

and

$$\mathrm{cDim}^\nu(X) = \inf \mathcal{G}^{\nu,\mathrm{str}}_{\mathrm{constr}}(X).$$

A *correspondence principle* for an effective dimension is a theorem stating that the effective dimension coincides with its classical counterpart on sufficiently "simple" sets. The following such principle, proven by Hitchcock [26], extended a correspondence principle for computable dimension that was implicit in results of Staiger [46].

THEOREM 3.15 (correspondence principle for constructive dimension [26]). *If $X \subseteq \Sigma^\infty$ is any union (not necessarily effective) of computably closed, i.e., Π^0_1, sets, then* $\mathrm{cdim}(X) = \dim_{\mathrm{H}}(X)$.

We now define the constructive dimensions of individual sequences.

DEFINITION 3.16. *Let ν be a probability measure on Σ^∞, and let $S \in \Sigma^\infty$. Then the ν-dimension of S is*

$$\dim^\nu(S) = \mathrm{cdim}^\nu(\{S\}),$$

and the strong ν-dimension of S is

$$\mathrm{Dim}^\nu(S) = \mathrm{cDim}^\nu(\{S\}).$$

4. Kolmogorov complexity characterizations. In this section we prove characterizations of constructive ν-dimension and constructive strong ν-dimension in terms of Kolmogorov complexity. These characterizations are used in the proof of our main theorem in section 6.

Let Σ be a finite alphabet, with $|\Sigma| \geq 2$. The *Kolmogorov complexity* of a string $w \in \Sigma^*$ is the natural number

$$\mathrm{K}(w) = \min\{|\pi| \mid \pi \in \{0,1\}^* \text{ and } U(\pi) = w\},$$

where U is a fixed optimal universal prefix Turing machine. This is a standard notion of (prefix) Kolmogorov complexity. The reader is referred to the standard text by Li and Vitányi [33] for background on prefix Turing machines and Kolmogorov complexity.

If ν is a probability measure on Σ^∞, then the *Shannon self information* of a string $w \in \Sigma^*$ with respect to ν is

$$\mathcal{I}_\nu(w) = \log \frac{1}{\nu(w)}.$$

Note that $0 \leq \mathcal{I}_\nu(w) \leq \infty$. Equality holds on the left here if and only if $\nu(w) = 1$, and equality holds on the right if and only if $\nu(w) = 0$. Since our results here concern strongly positive probability measures, we will have $0 < \mathcal{I}_\nu(w) < \infty$ for all $w \in \Sigma^+$.

The following result is the main theorem of this section. It gives characterizations of the ν-dimensions and the strong ν-dimensions of sequences in terms of Kolmogorov complexity.

THEOREM 4.1. *If ν is a strongly positive, computable probability measure on Σ^∞, then, for all $S \in \Sigma^\infty$,*

$$\dim^\nu(S) = \liminf_{m\to\infty} \frac{\mathrm{K}(S[0..m-1])}{\mathcal{I}_\nu(S[0..m-1])} \tag{4.1}$$

and

$$\mathrm{Dim}^\nu(S) = \limsup_{m\to\infty} \frac{\mathrm{K}(S[0..m-1])}{\mathcal{I}_\nu(S[0..m-1])}. \tag{4.2}$$

Proof. Let $S \in \Sigma^\infty$. Let $s > s' > \liminf_m \frac{\mathrm{K}(S[0..m-1])}{\mathcal{I}_\nu(S[0..m-1])}$. For infinitely many m, $\mathrm{K}(S[0..m-1]) < s'\,\mathcal{I}_\nu(S[0..m-1])$, and so $\nu(S[0..m-1])^{s'} < 2^{-\mathrm{K}(S[0..m-1])}$.

Let $m \in \mathbb{N}$. We define the computably enumerable (c.e.) set

$$A = \{w \mid \mathrm{K}(w) < s'\,\mathcal{I}_\nu(w)\}$$

and the ν-s-constructive supergale d_m as follows. If there exists $v \sqsubseteq w$ such that $v \in A^{=m}$, then

$$d_m(w) = \left(\frac{\nu(w)}{\nu(v)}\right)^{1-s}.$$

Otherwise,

$$d_m(w) = \sum_{\substack{u,\\ wu\in A^{=m}}} \left(\frac{\nu(wu)}{\nu(w)}\right)^{s}.$$

First, notice that d_m is well defined since $d_m(\lambda) < \infty$ and d_m is a supergale,

$$d_m(\lambda) = \sum_{u\in A^{=m}} \nu(u)^s \le \sum_{u\in A^{=m}} 2^{-\mathrm{K}(u)}(1-\delta)^{m(s-s')} \le (1-\delta)^{m(s-s')},$$

for $\delta \in (0,1)$ a constant testifying that ν is strongly positive.

We define the ν-s-constructive supergale

$$d(w) = \sum_m (1-\delta)^{-m(s-s')} d_{2m}(w) + \sum_m (1-\delta)^{-m(s-s')} d_{2m+1}(w).$$

Notice that the fact that A is c.e. is necessary for the constructivity of d. Since for $w \in A$, $d_{|w|}(w) = 1$, we have that $d(w) \ge (1-\delta)^{-|w|(s-s')/2}$ for $w \in A$. Since for infinitely many m, $S[0..m-1] \in A$, we have that $S \in S^\infty[d]$ and $\dim^\nu(S) \le s$. This finishes the proof of the first inequality of (4.1).

For the other direction, let $s > \dim^\nu(S)$. Let d be a ν-s-constructive gale succeeding on S. Let $c \ge d(\lambda)$ be a rational number.

Let $B = \{w \mid d(w) > c\}$, and notice that B is c.e. For every m,

$$\sum_{w\in B^{=m}} \nu(w)^s \le 1.$$

Let $\theta_m : B^{=m} \to \{0,1\}^*$ be the Shannon–Fano–Elias code given by the probability submeasure p defined as $p(w) = \nu(w)^s$ for $w \in B^{=m}$ (this code assigns shorter code

words $\theta_m(w)$ to words with a larger probability $p(w)$; see, for example, [10]). Specifically, for each $w \in B^{=m}$, $\theta_m(w)$ is defined as the most significant $1 + \lceil \log \frac{1}{p(w)} \rceil$ bits of the real number

$$\sum_{\substack{|v|=m,\\ v<_B w}} p(v) + \frac{1}{2}\, p(w),$$

where $<_B$ corresponds to the words in B ordered according to their appearance in the computable enumeration of B.

Then

$$|\theta_m(w)| = 1 + \left\lceil \log \frac{1}{p(w)} \right\rceil = 1 + \lceil s\, \mathcal{I}_\nu(w) \rceil$$

for $w \in B^{=m}$.

Since B is c.e., codification and decodification can be computed given the length; that is, every $w \in B$ can be computed from $|w|$ and $\theta_{|w|}(w)$. Therefore, if $w \in B$, $\mathrm{K}(w) \le 2 + s\, \mathcal{I}_\nu(w) + 2 \log(|w|)$.

Notice that since ν is strongly positive, $\mathcal{I}_\nu(w) = \Omega(|w|)$, and since there exist infinitely many m for which $S[0..m-1] \in B$,

$$\liminf_{m\to\infty} \frac{\mathrm{K}(S[0..m-1])}{\mathcal{I}_\nu(S[0..m-1])} \le s.$$

The proof of (4.2) is analogous. □

If ν is a strongly positive probability measure on Σ^∞, then there is a real constant $\alpha > 0$ such that, for all $w \in \Sigma^*$, $\mathcal{I}_\nu(w) \ge \alpha |w|$. Since other notions of Kolmogorov complexity, such as the plain complexity $\mathrm{C}(w)$ and the monotone complexity $\mathrm{K_m}(w)$ [33], differ from $\mathrm{K}(w)$ by at most $O(\log |w|)$, it follows that Theorem 4.1 also holds with $\mathrm{K}(S[0..m-1])$ replaced by $\mathrm{C}(S[0..m-1])$, $\mathrm{K_m}(S[0..m-1])$, etc.

The following known characterizations of dimension and strong dimension are simply the special case of Theorem 4.1 in which $\Sigma = \{0,1\}$ and $\nu = \mu$.

Corollary 4.2 (see [39, 1]). *For all $S \in \{0,1\}^\infty$,*

$$\dim(S) = \liminf_{m\to\infty} \frac{\mathrm{K}(S[0..m-1])}{m}$$

and

$$\mathrm{Dim}(S) = \limsup_{m\to\infty} \frac{\mathrm{K}(S[0..m-1])}{m}.$$

Later, alternative proofs of Corollary 4.2 appear in [37, 48].

We define the dimension and strong dimension of a point x in Euclidean space as in (1.7) and (1.8). It is convenient to characterize these dimensions in terms of Kolmogorov complexity of rational approximations in Euclidean space. Specifically, for each $x \in \mathbb{R}^n$ and $r \in \mathbb{N}$, we define the *Kolmogorov complexity* of x at *precision* r to be the natural number

$$\mathrm{K}_r(x) = \min\left\{ K(q) \;\middle|\; q \in \mathbb{Q}^n \text{ and } |q - x| \le 2^{-r} \right\}.$$

That is, $\mathrm{K}_r(x)$ is the minimum length of any program $\pi \in \{0,1\}^*$ for which $U(\pi) \in \mathbb{Q}^n \cap B(x, 2^{-r})$. (Related notions of approximate Kolmogorov complexity have recently

been considered by Vereshchagin and Vitányi [51] and Fortnow, Lee, and Vereshchagin [18].) We also mention the quantity

$$\mathrm{K}_r(r,x) = \min\left\{K(r,q) \;\middle|\; q \in \mathbb{Q}^n \text{ and } |q-x| \le 2^{-r}\right\},$$

in which the program π must specify the precision parameter r as well as a rational approximation q of x to within 2^{-r}. The following relationship between these two quantities is easily verified by standard techniques.

OBSERVATION 4.3. *There exist constants $a, b \in \mathbb{N}$ such that, for all $x \in \mathbb{R}^n$ and $r \in \mathbb{N}$,*

$$\mathrm{K}_r(x) - a \le \mathrm{K}_r(r,x) \le \mathrm{K}_r(x) + \mathrm{K}(r) + b.$$

We now show that the quantity $\mathrm{K}_r(r,x)$ is within a constant of the Kolmogorov complexity of the first nr bits of an interleaved binary expansion of the fractional part of the coordinates of x, which was defined in section 1, together with the integer part of x.

LEMMA 4.4. *There is a constant $c \in \mathbb{N}$ such that, for all $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, all interleaved binary expansions S of the fractional parts of $x_1, \ldots, x_n$, and all $r \in \mathbb{N}$,*

$$|\mathrm{K}_r(r,x) - \mathrm{K}(\lfloor x \rfloor, S[0..nr-1])| \le c, \tag{4.3}$$

where $\lfloor x \rfloor$ is the interleaved binary expansion of $(\lfloor x_1 \rfloor, \ldots, \lfloor x_n \rfloor)$.

Proof. We first consider the case $x \in [0,1]^n$. For convenience, let $l = \lceil \frac{\log n}{2} \rceil$ (notice that both n and l are constants). Let M be a prefix Turing machine such that, if $\pi \in \{0,1\}^*$ is a program such that $U(\pi) = w \in \{0,1\}^*$ and $|w|$ is divisible by n, and if $v \in \{0,1\}^{nl}$, then $M(\pi v) = (|w|/n, q)$, where $q \in \mathbb{Q}^n$ is the dyadic rational point whose interleaved binary expansion is wv. Let $c_1 = nl + c_M$, where c_M is an optimality constant for M. Let $x \in \mathbb{R}^n$, let S be an interleaved binary expansion of x, and let $r \in \mathbb{N}$. Let $\pi \in \{0,1\}^*$ be a witness to the value of $\mathrm{K}_r(S[0..nr-1])$, and let $v = S[nr..n(r+l)-1]$. Then $M(\pi v) = (r,q)$, where q is the dyadic rational point whose interleaved binary expansion is $S[0..n(l+r)-1]$. Since

$$|q-x| = \sqrt{n(2^{-(r+l)})^2} = 2^{-(r+l)}\sqrt{n} \le 2^{-r},$$

it follows that

$$\mathrm{K}_r(r,x) \le |\pi v| + c_M = \mathrm{K}(S[0..nr-1]) + c_1, \tag{4.4}$$

which is one of the inequalities we need to get (4.3).

We now turn to the reverse inequality. For each $r \in \mathbb{N}$ and $q \in \mathbb{Q}^n$, let $A_{r,q}$ be the set of all r-dyadic points within $2^{l-r} + 2^r$ of q. That is, $A_{r,q}$ is the set of all points $q' = (q'_1, \ldots, q'_n) \in \mathbb{Q}^n$ such that $|q-q'| \le 2^{l-r} + 2^r$ and each q'_i is of the form $2^{-r}a'_i$ for some integer a'_i.

Let $q', q'' \in A_{r,q}$. For each $1 \le i \le n$, let $q'_i = 2^{-r}a'_i$ and $q''_i = 2^{-r}a''_i$ be the ith coordinates of q' and q'', respectively. Then, for each $1 \le i \le n$, we have

$$\begin{aligned} |a'_i - a''_i| &= 2^r|q'_i - q''_i| \\ &\le 2^r(|q'-q| + |q''-q|) \\ &\le 2^{r+1}(2^{l-r} + 2^{-r}) \\ &= 2^{l+1} + 2. \end{aligned}$$

This shows that there are at most $2^{l+1}+3$ possible values of a'_i. It follows that

$$|A_{r,q}| \le (2^{l+1}+3)^n. \tag{4.5}$$

Let M' be a prefix Turing machine such that, if $\pi \in \{0,1\}^*$ is a program such that $U(\pi) = (r,q) \in \mathbb{N} \times \mathbb{Q}^n$, $0 \le m < |A_{r,q}|$, and s_m is the mth string in the standard enumeration $s_0, s_1, s_2, \ldots$ of $\{0,1\}^*$, then $M'(\pi 0^{|s_m|}1 s_m)$ is the nr-bit interleaved binary expansion of the fractional points of the coordinates of the mth element of a canonical enumeration of $A_{r,q}$. Let $c_2 = n(2l'+1) + c_{M'}$, where $l' = \lceil \log(2^{l+1}+3) \rceil$ and $c_{M'}$ is an optimality constant for M'.

Let $x \in \mathbb{R}^n$, let S be an interleaved binary expansion of x, and let $r \in \mathbb{N}$. Let q' be the r-dyadic point whose interleaved binary expansion is $S[0..nr-1]$, and let $\pi \in \{0,1\}^*$ be a witness to the value of $\mathrm{K}_r(r,x)$. Then $U(\pi) = (r,q)$ for some $q \in \mathbb{Q}^n \cap B(x, 2^{-r})$. Since

$$\begin{aligned} |q'-q| &\le |q'-x| + |q-x| \\ &\le 2^{-r}\sqrt{n} + 2^{-r} \\ &\le 2^{l-r} + 2^{-r}, \end{aligned}$$

we have $q' \in A_{q,r}$. It follows that there exists $0 \le m < |A_{r,q}|$ such that $M'(\pi 0^{|s_m|}1 s_m) = S[0..nr-1]$. This implies that

$$\begin{aligned} \mathrm{K}(S[0..nr-1]) &\le |\pi 0^{|s_m|}1 s_m| + c_{M'} \\ &= \mathrm{K}_r(r,x) + 2|s_m| + c_{M'} + 1 \\ &\le \mathrm{K}_r(r,x) + 2|s_{|A_{r,q}|-1}| + c_{M'} + 1 \\ &= \mathrm{K}_r(r,x) + 2\lfloor \log |A_{r,q}| \rfloor + c_{M'} + 1 \\ &\le \mathrm{K}_r(r,x) + 2\lfloor n \log(2^{l+1}+3) \rfloor + c_{M'} + 1 \\ &\le \mathrm{K}_r(r,x) + c_2. \end{aligned} \tag{4.6}$$

If we let $c = \max\{c_1, c_2\}$, then (4.4) and (4.6) imply (4.3).

For the general case, notice that $\mathrm{K}_r(r, \lfloor x \rfloor) = \mathrm{K}(\lfloor x \rfloor) + O(1)$. □

We now have the following characterizations of the dimensions and strong dimensions of points in Euclidean space.

THEOREM 4.5. *For all $x \in \mathbb{R}^n$,*

$$\dim(x) = \liminf_{r\to\infty} \frac{\mathrm{K}_r(x)}{r}, \tag{4.7}$$

and

$$\mathrm{Dim}(x) = \limsup_{r\to\infty} \frac{\mathrm{K}_r(x)}{r}. \tag{4.8}$$

Proof. Let $x \in \mathbb{R}^n$, and let S be an interleaved binary expansion of the fractional parts of the coordinates of x. By (1.7) and Corollary 4.2, we have

$$\begin{aligned} \dim(x) &= n \dim(S) \\ &= n \liminf_{m\to\infty} \frac{\mathrm{K}(S[0..m-1])}{m}. \end{aligned}$$

Since all values of $\mathrm{K}(S[0..m-1])$ with $nr \leq m < n(r+1)$ are within a constant (that depends on the constant n) of one another, it follows that

$$\begin{aligned}\dim(x) &= n \liminf_{r\to\infty} \frac{\mathrm{K}(S[0..nr-1])}{nr} \\ &= \liminf_{r\to\infty} \frac{\mathrm{K}(S[0..nr-1])}{r}.\end{aligned}$$

Since $\mathrm{K}(r) = O(\log r)$ [33], it follows by Observation 4.3 and Lemma 4.4 that (4.7) holds. The proof that (4.8) holds is analogous. □

5. Self-similar fractals. This expository section reviews a fragment of the theory of self-similar fractals that is adequate for understanding our main theorem and its proof. Our treatment is self-contained but of course far from complete. The interested reader is referred to any of the standard texts [2, 12, 13, 15] for more extensive discussion.

DEFINITION 5.1. *A* contracting similarity *on a set $D \subseteq \mathbb{R}^n$ is a function $S : D \to D$ for which there exists a real number $c \in (0,1)$, called a* contraction ratio *of S, satisfying $|S(x) - S(y)| = c|x-y|$ for all $x, y \in D$.*

DEFINITION 5.2. *An* iterated function system (IFS) *is a finite sequence $S = (S_0, \ldots, S_{k-1})$ of two or more contracting similarities on a nonempty, closed set $D \subseteq \mathbb{R}^n$. We call D the* domain *of S, writing $D = \mathrm{dom}(S)$.*

We note that IFSs are often defined more generally. For example, IFSs consisting of *contractions*, which merely satisfy inequalities of the form $|S(x)-S(y)| \leq c|x-y|$ for some $c \in (0,1)$, are often considered. Since our results here are confined to self-similar fractals, we have used the more restrictive definition.

We use the standard notation $\mathcal{K}(D)$ for the set of all nonempty compact (i.e., closed and unbounded) subsets of a nonempty closed set $D \subseteq \mathbb{R}^n$. For each IFS S, we write $\mathcal{K}(S) = \mathcal{K}(\mathrm{dom}(S))$.

With each IFS $S = (S_0, \ldots, S_{k-1})$, we define the transformation $S : \mathcal{K}(S) \to \mathcal{K}(S)$ by

$$S(A) = \bigcup_{i=0}^{k-1} S_i(A)$$

for all $A \in \mathcal{K}(S)$, where $S_i(A)$ is the image of A under the contracting similarity S_i.

OBSERVATION 5.3. *For each IFS S, there exists $A \in \mathcal{K}(S)$ such that $S(A) \subseteq A$.*

Proof. Assume the hypothesis, with $S = (S_0, \ldots, S_{k-1})$ and $\mathrm{dom}(S) = D$, and let $c_0, \ldots, c_{k-1}$ be contraction ratios of $S_0, \ldots, S_{k-1}$, respectively. Let $c = \min\{c_0, \ldots, c_{k-1}\}$, noting that $c \in (0,1)$, and fix $z \in D$. Let

$$r = \frac{1}{1-c} \max_{0\leq i<k} |S_i(z) - z|,$$

and let $A = D \cap \mathcal{B}_r(z)$. Then A is a closed subset of the compact set $\mathcal{B}_r(z)$, and $z \in A$, so $A \in \mathcal{K}(S)$. For all $x \in A$ and $0 \leq i < k$, we have

$$\begin{aligned}|S_i(x) - z| &\leq |S_i(x) - S_i(z)| + |S_i(z) - z| \\ &= c|x - z| + |S_i(z) - z| \\ &\leq cr + (1-c)r \\ &= r,\end{aligned}$$

so each $S_i(A) \subseteq A$, and so $S(A) \subseteq A$. □

For each IFS $S = (S_0, \ldots, S_{k-1})$ and each set $A \in \mathcal{K}(S)$ satisfying $S(A) \subseteq A$, we define the function $S_A : \Sigma_k^* \to \mathcal{K}(S)$ by the recursion

$$S_A(\lambda) = A,$$

$$S_A(iw) = S_i(S_A(w))$$

for all $w \in \Sigma_k^*$ and $i \in \Sigma_k$.

If $c = \max\{c_0, \ldots, c_{k-1}\}$, where $c_0, \ldots, c_{k-1}$ are contraction ratios of $S_0, \ldots, S_{k-1}$, respectively, then routine inductions establish that, for all $w \in \Sigma_k^*$ and $i \in \Sigma_k$,

$$S_A(iw) \subseteq S_A(w) \tag{5.1}$$

and

$$\operatorname{diam}(S_A(w)) \leq c^{|w|}\operatorname{diam}(A). \tag{5.2}$$

Since $c \in (0,1)$, it follows that, for each sequence $T \in \Sigma_k^\infty$, there is a unique point $S_A(T) \in \mathbb{R}^n$ such that

$$\bigcap_{w \sqsubseteq T} S_A(w) = \{S_A(T)\}. \tag{5.3}$$

In this manner, we have defined a function $S_A : \Sigma_k^\infty \to \mathbb{R}^n$. The following observation shows that this function does not really depend on the choice of A.

Observation 5.4. *Let S be an IFS. If $A, B \in \mathcal{K}(S)$ satisfy $S(A) \subseteq A$ and $S(B) \subseteq B$, then $S_A = S_B$.*

Our proof of Observation 5.4 uses the *Hausdorff metric* on $\mathcal{K}(\mathbb{R}^n)$, which is the function $\rho_H : \mathcal{K}(\mathbb{R}^n) \times \mathcal{K}(\mathbb{R}^n) \to [0, \infty)$ defined by

$$\rho_H(A, B) = \max\{\sup_{x \in A} \inf_{y \in B} |x - y|, \sup_{y \in B} \inf_{x \in A} |x - y|\}$$

for all $A, B \in \mathcal{K}(\mathbb{R}^n)$. It is easy to see that ρ_H is a *metric* on $\mathcal{K}(\mathbb{R}^n)$. It follows that ρ_H is a metric on $\mathcal{K}(S)$ for every IFS S.

Proof of Observation 5.4. Assume the hypothesis, with $S = (S_0, \ldots, S_{k-1})$, and let $c_0, \ldots, c_{k-1}$ be contraction ratios of $S_0, \ldots, S_{k-1}$, respectively. The definition of ρ_H implies immediately that, for all $E, F \in \mathcal{K}(S)$ and $0 \leq i < k$, $\rho_H(S_i(E), S_i(F)) = c_i \rho_H(E, F)$. It follows by an easy induction that, if we let $c = \max\{c_0, \ldots, c_{k-1}\}$, then, for all $w \in \Sigma_k^*$,

$$\rho_H(S_A(w), S_B(w)) \leq c^{|w|}\rho_H(A, B). \tag{5.4}$$

To see that $S_A = S_B$, let $T \in \Sigma_k^\infty$, and let $\epsilon > 0$. For each $w \sqsubseteq T$, (5.1), (5.2), and (5.3) tell us that

$$\rho_H(\{S_A(T)\}, S_A(w)) \leq \operatorname{diam}(S_A(w)) \leq c^{|w|}\operatorname{diam}(A) \tag{5.5}$$

and

$$\rho_H(\{S_A(T)\}, S_B(w)) \leq \operatorname{diam}(S_B(w)) \leq c^{|w|}\operatorname{diam}(B). \tag{5.6}$$

Since $c \in (0,1)$, (5.4), (5.5), and (5.6) tell us that there is a prefix $w \sqsubseteq T$ such that

$$\begin{aligned}
&\rho_H(\{S_A(T)\},\{S_B(T)\}) \\
&\leq \rho_H(\{S_A(T)\},S_A(w)) + \rho_H(S_A(w),S_B(w)) + \rho_H(\{S_B(T)\},S_B(w)) \\
&< \epsilon/3 + \epsilon/3 + \epsilon/3 \\
&= \epsilon.
\end{aligned}$$

Since this holds for all $\epsilon > 0$, it follows that $\rho_H(\{S_A(T)\},\{S_B(T)\}) = 0$, i.e., that $S_A(T) = S_B(T)$. □

For each IFS S, we define the induced function $S : \Sigma_k^\infty \to \mathbb{R}^n$ by setting $S = S_A$, where A is *any* element of $\mathcal{K}(S)$ satisfying $S(A) \subseteq A$. By Observations 5.3 and 5.4, this induced function S is well defined.

We now have the machinery to define a rich collection of fractals in $\mathbb{R}^n$.

DEFINITION 5.5. *The* attractor *(or* invariant set*) of an IFS $S = (S_0,\ldots,S_{k-1})$ is the set*

$$F(S) = S(\Sigma_k^\infty),$$

i.e., the range of the induced function $S : \Sigma_k^\infty \to \mathbb{R}^n$.

It is well known that the attractor $F(S)$ is the unique fixed point of the induced transformation $S : \mathcal{K}(S) \to \mathcal{K}(S)$, but we do not use this fact here.

For each $T \in \Sigma_k^\infty$, we call T a *coding sequence*, or an *S-code*, of the point $S(T) \in F(S)$.

Example 5.6 (generalized Sierpinski triangles S in $\mathbb{R}^2$). Let D be the set consisting of the triangle in $\mathbb{R}^2$ with vertices $v_0 = (0,0)$, $v_1 = (1,0)$, and $v_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, together with this triangle's interior. Given $c_0, c_1, c_2 \in (0,1)$, define $S_0, S_1, S_2 : D \to D$ by

$$S_i(p) = v_i + c_i(p - v_i)$$

for $i \in \{0,1,2\}$ and $p \in D$. Then S_0, S_1, and S_2 are contracting similarities with contraction ratios c_0, c_1, and c_2, respectively, and so $S = (S_0, S_1, S_2)$ is an IFS with domain D. Intuitively, a coding sequence $T \in \{0,1,2\}^\infty$ can be regarded as an abbreviation of the procedure

$\Delta_0 := D$;
for $j := 0$ to ∞ do
    $\Delta_{j+1} :=$ the $c_{T[j]}$ − reduced copy of Δ_j lying in corner $T[j]$ of Δ_j.

The point $S(T)$ of $F(S)$ is then the unique point of $\mathbb{R}^2$ lying in all the triangles $\Delta_0, \Delta_1, \Delta_2, \ldots$. The attractor $F(S)$ is thus a generalized Sierpinski triangle. Figure 2(a,b) illustrates this construction in the case where $c_0 = \frac{1}{2}$, $c_1 = \frac{1}{4}$, and $c_2 = \frac{1}{3}$. If $c_0 = c_1 = c_2 = 1/2$, then $F(S)$ is the familiar Sierpinski triangle of Figure 2(c).

In general, the attractor of an IFS $S = (S_0,\ldots,S_{k-1})$ is easiest to analyze when the sets $S_0(\mathrm{dom}(S)),\ldots,S_{k-1}(\mathrm{dom}(S))$ are "nearly disjoint." (Intuitively, this prevents each point $x \in F(S)$ from having "too many" coding sequences $T \in \Sigma_k^\infty$.) The following definition makes this notion precise.

DEFINITION 5.7. *An IFS $S = (S_0,\ldots,S_{k-1})$ with domain D satisfies the* open set condition *if there exists a nonempty, bounded, open set $G \subseteq D$ such that $S_0(G),\ldots,S_{k-1}(G)$ are disjoint subsets of G.*

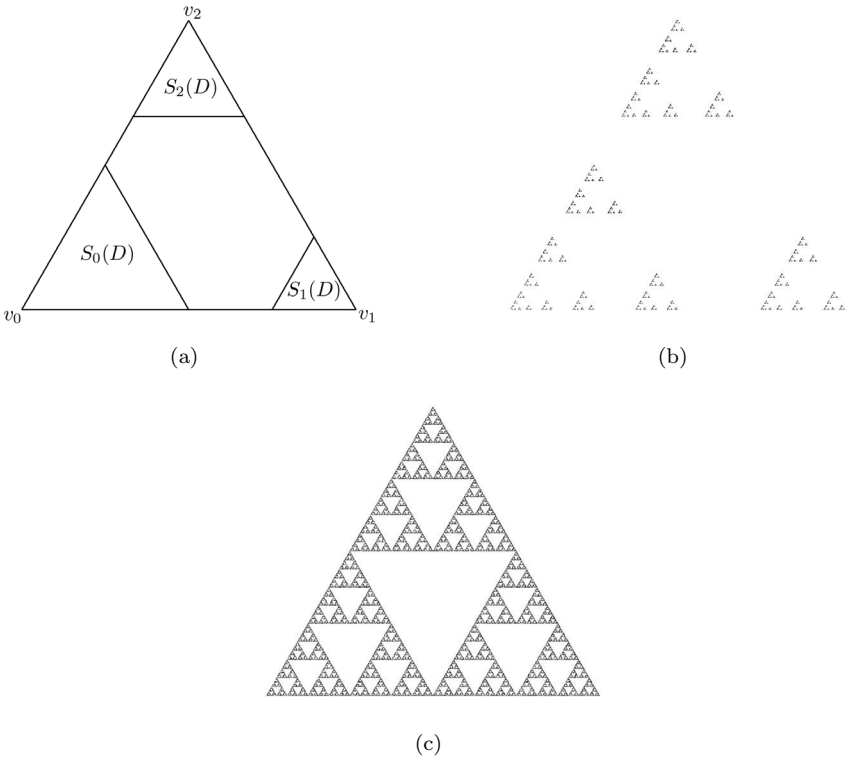


FIG. 2. (a) *The IFS S of Example* 5.6, *with* $c_0 = \frac{1}{2}$, $c_1 = \frac{1}{4}$, *and* $c_2 = \frac{1}{3}$. (b) *The attractor $F(S)$ of this IFS.* (c) *The attractor $F(S)$ when* $c_0 = c_1 = c_2 = \frac{1}{2}$.

We now define the most widely known type of fractal.

DEFINITION 5.8. *A* self-similar fractal *is a set $F \subseteq \mathbb{R}^n$ that is the attractor of an IFS that satisfies the open set condition.*

Example 5.9 (Example 5.6 continued). If $c_0+c_1 \leq 1$, $c_0+c_2 \leq 1$, and $c_1+c_2 \leq 1$, then the set $G = D^o$ (the topological interior of D) testifies that S satisfies the open set condition, whence $F(S)$ is a self-similar fractal. If $c_0 + c_1 > 1$ or $c_0 + c_2 > 1$ or $c_1 + c_2 > 1$, then S does not satisfy the open set condition.

The following quantity plays a central role in the theory of self-similar fractals.

DEFINITION 5.10. *The* similarity dimension *of an IFS $S = (S_0, \ldots, S_{k-1})$ with contraction ratios $c_0, \ldots, c_{k-1}$ is the (unique) solution* $\operatorname{sdim}(S) = s \in [0, \infty)$ *of the equation*

$$\sum_{i=0}^{k-1} c_i^s = 1.$$

If $F = F(S)$ is a self-similar fractal, where S is an IFS satisfying the open set condition, then the classical Hausdorff and packing dimensions of F are known to coincide with the similarity dimension of S. (The fact that this holds for Hausdorff dimension was proven by Moran [40]. The fact that it holds for packing dimension was proven by Falconer [14]. As we shall see, both facts follow from our main theorem.) In particular, this implies that the following definition is unambiguous.

DEFINITION 5.11. *The* similarity dimension *of a self-similar fractal F is the number*

$$\mathrm{sdim}(F) = \mathrm{sdim}(S),$$

where S is an IFS satisfying $F(S) = F$ and the open set condition.

It should be noted that some authors define a fractal to be self-similar if it is the attractor of *any* IFS S. With this terminology, i.e., in the absence of the open set condition, the Hausdorff and packing dimensions may be less than the similarity dimension.

6. Pointwise analysis of dimensions. In this section we prove our main theorem, which gives a precise analysis of the dimensions of individual points in computably self-similar fractals. We first recall the known fact that such fractals are computable.

DEFINITION 6.1. *An IFS $S = (S_0, \ldots, S_{k-1})$ is computable if* $\mathrm{dom}(S)$ *is a computable set and the functions $S_0, \ldots, S_{k-1}$ are computable.*

THEOREM 6.2 (Kamo and Kawamura [27]). *For every computable IFS S, the attractor $F(S)$ is a computable set.*

One consequence of Theorem 6.2 is the following.

COROLLARY 6.3. *For every computable IFS S,* $\mathrm{cdim}(F(S)) = \dim_{\mathrm{H}}(F(S))$.

Proof. Let S be a computable IFS. Then $F(S)$ is compact, hence closed, and is computable by Theorem 6.2, so $F(S)$ is computably closed by Observation 2.2. It follows by the correspondence principle for constructive dimension (Theorem 3.15) that $\mathrm{cdim}(F(S)) = \dim_{\mathrm{H}}(F(S))$. □

We next present three lemmas that we use in the proof of our main theorem. The first is a well-known geometric fact (e.g., it is Lemma 9.2 in [15]) whose proof is short enough to repeat here.

LEMMA 6.4. *Let $\mathcal{G}$ be a collection of disjoint open sets in $\mathbb{R}^n$, and let $r, a, b \in (0, \infty)$. If every element of $\mathcal{G}$ contains a ball of radius ar and is contained in a ball of radius br, then no ball of radius r meets more than $\left(\frac{1+2b}{a}\right)^n$ of the closures of the elements of $\mathcal{G}$.*

Proof. Assume the hypothesis, and let B be a ball of radius r. Let

$$\mathcal{M} = \left\{ G \in \mathcal{G} \;\middle|\; B \cap \overline{G} \neq \emptyset \right\},$$

and let $m = |\mathcal{M}|$. Let B' be a closed ball that is concentric with B and has radius $(1 + 2b)r$. Then B' contains $\overline{G}$ for every $G \in \mathcal{M}$. Since each $G \in \mathcal{M}$ contains a ball B_G of radius ar, and since these balls are disjoint, it follows that

$$\mathrm{volume}(B') \geq \sum_{G \in \mathcal{M}} \mathrm{volume}(B_G).$$

This implies that

$$[(1 + 2b)r]^n \geq m(ar)^n,$$

whence $m \leq \left(\frac{1+2b}{a}\right)^n$. □

Our second lemma gives a computable means of assigning rational "hubs" to the various open sets arising from a computable IFS satisfying the open set condition.

DEFINITION 6.5. *A* hub function *for an IFS $S = (S_0, \ldots, S_{k-1})$ satisfying the open set condition with G as witness is a function $h : \Sigma_k^* \to \mathbb{R}^n$ such that $h(w) \in$*

$S_G(w)$ for all $w \in \Sigma_k^*$. In this case, we call $h(w)$ the hub *that h assigns to the set $S_G(w)$.*

LEMMA 6.6. *If $S = (S_0, \ldots, S_{k-1})$ is a computable IFS satisfying the open set condition with G as witness, then there is an exactly computable, rational-valued hub function $h : \Sigma_k^* \to \mathbb{Q}^n$ for S and G.*

Proof. Assume the hypothesis. From oracle Turing machines computing $S_0, \ldots, S_{k-1}$, it is routine to construct an oracle Turing machine M computing the function

$$\tilde{S} : \mathrm{dom}(S) \times \Sigma_k^* \to \mathrm{dom}(S)$$

defined by the recursion

$$\begin{aligned} \tilde{S}(x, \lambda) &= x, \\ \tilde{S}(x, iw) &= S_i(\tilde{S}(x, w)) \end{aligned}$$

for all $x \in \mathrm{dom}(S)$, $w \in \Sigma_k^*$, and $i \in \Sigma_k$. Fix a rational point $q \in G \cap \mathbb{Q}^n$, and let C_q be the oracle that returns the value q on all queries, noting that

$$|M^{C_q}(w, r) - \tilde{S}(q, w)| \leq 2^{-r} \tag{6.1}$$

holds for all $w \in \Sigma_k^*$ and $r \in \mathbb{N}$. Fix $l \in \mathbb{Z}^+$ large enough to satisfy the following conditions.

(i) G contains the closed ball of radius 2^{-l} about q.

(ii) For each $i \in \Sigma_k$, $2^{-l} \leq c_i$, where c_i is the contraction ratio of S_i.

Then a routine induction shows that, for each $w \in \Sigma_k^*$, $S_G(w)$ contains the closed ball of radius $2^{-l(1+|w|)}$ about $\tilde{S}(q, w)$. It follows by (6.1) that the function $h : \Sigma_k^* \to \mathbb{Q}^n$ defined by

$$h(w) = M^{C_q}(w, l(1 + |w|))$$

is a hub function for S and G. It is clear that h is rational-valued and exactly computable. □

Iterated function systems induce probability measures on alphabets in the following manner.

DEFINITION 6.7. *The* similarity probability measure *of an IFS $S = (S_0, \ldots, S_{k-1})$ with contraction ratios $c_0, \ldots, c_{k-1}$ is the probability measure π_S on the alphabet Σ_k defined by*

$$\pi_S(i) = c_i^{\mathrm{sdim}(S)}$$

for all $i \in \Sigma_k$. For $w \in \Sigma_k^*$, we use the abbreviation $\mathcal{I}_S(w) = \mathcal{I}_{\pi_S}(w)$.

Our third lemma provides a decidable set of well-behaved "canonical prefixes" of sequences in Σ_k^∞.

LEMMA 6.8. *Let $S = (S_0, \ldots, S_{k-1})$ be a computable IFS, and let $c_{\min}$ be the minimum of the contraction ratios of $S = (S_0, \ldots, S_{k-1})$. For any real number*

$$\alpha > \mathrm{sdim}(S) \log \frac{1}{c_{\min}}, \tag{6.2}$$

there exists a decidable set $A \subseteq \mathbb{N} \times \Sigma_k^*$ such that, for each $r \in \mathbb{N}$, the set

$$A_r = \{ w \in \Sigma_k^* \mid (r, w) \in A \}$$

has the following three properties.

(i) *No element of A_r is a proper prefix of any element of $A_{r'}$ for any $r' \leq r$.*
(ii) *Each sequence in Σ_k^∞ has a (unique) prefix in A_r.*
(iii) *For all $w \in A_r$,*

$$r \cdot \mathrm{sdim}(S) < \mathcal{I}_S(w) < r \cdot \mathrm{sdim}(S) + \alpha. \tag{6.3}$$

Proof. Let S, $c_{\min}$, and α be as given, and let $c_0, \ldots, c_{k-1}$ be the contraction ratios of $S_0, \ldots, S_{k-1}$, respectively. Let $c_{\max} = \max\{c_0, \ldots, c_{k-1}\}$, and let $\delta = \frac{1}{2}\min\{c_{\min}, 1 - c_{\max}\}$, noting that $\delta \in (0, \frac{1}{2k}]$. Since S is computable, there is, for each $i \in \Sigma_k$, an exactly computable function

$$\widehat{c}_i : \mathbb{N} \to \mathbb{Q} \cap [\delta, 1 - \delta]$$

such that, for all $t \in \mathbb{N}$,

$$|\widehat{c}_i(t) - c_i| \leq 2^{-t}. \tag{6.4}$$

For all $T \in \Sigma_k^\infty$ and $l, t \in \mathbb{N}$, we have

$$\begin{aligned}
&\prod_{i=0}^{l-1} \widehat{c}_{T[i]}(t+i) - \prod_{i=0}^{l-1} c_{T[i]} \\
&= \sum_{i=0}^{l-1} \left[\left(\prod_{j=0}^{i-1} c_{T[j]} \right) \left(\prod_{j=i}^{l-1} \widehat{c}_{T[j]}(t+j) \right) \right. \\
&\quad \left. - \left(\prod_{j=0}^{i} c_{T[j]} \right) \left(\prod_{j=i+1}^{l-1} \widehat{c}_{T[j]}(t+j) \right) \right] \\
&= \sum_{i=0}^{l-1} (\widehat{c}_{T[i]}(t+i) - c_{T[i]}) p_i,
\end{aligned}$$

where

$$p_i = \left(\prod_{j=0}^{i-1} c_{T[j]} \right) \left(\prod_{j=i+1}^{l-1} \widehat{c}_{T[j]}(t+j) \right).$$

Since each $|p_i| \leq 1$, it follows by (6.4) that

$$\left| \prod_{i=0}^{l-1} \widehat{c}_{T[i]}(t+i) - \prod_{i=0}^{l-1} c_{T[i]} \right| < 2^{1-t} \tag{6.5}$$

holds for all $T \in \Sigma_k^\infty$ and $l, t \in \mathbb{N}$.

By (6.2), we have $2^{-\alpha/\mathrm{sdim}(S)}/c_{\min} < 1$, so we can fix $m \in \mathbb{Z}^+$ such that

$$2^{1-m} < 1 - 2^{-\alpha/\mathrm{sdim}(S)}/c_{\min}. \tag{6.6}$$

For each $T \in \Sigma_k^\infty$ and $r \in \mathbb{N}$, let

$$l_r(T) = \min \left\{ l \in \mathbb{N} \;\middle|\; \prod_{i=0}^{l-1} \widehat{c}_{T[i]}(r+m+i+1) \leq 2^{-r} - 2^{-(r+m)} \right\},$$

and let

$$A = \{(r, T[0..l_r(T)]) \mid T \in \Sigma_k^\infty \text{ and } r \in \mathbb{N}\}.$$

Since the functions $\widehat{c}_0, \ldots, \widehat{c}_{k-1}$ are rational-valued and exactly computable, the set A is decidable. It is clear that each A_r has properties (i) and (ii).

Let $r \in \mathbb{N}$. To see that A_r has property (iii), let $w \in A_r$. Let $l = |w|$, and fix $T \in \Sigma_k^\infty$ such that $l = l_r(T)$ and $w = T[0..l-1]$. By the definition of $l_r(T)$ and (6.5), we have

$$\prod_{i=0}^{l-1} c_{w[i]} < 2^{-r},$$

which implies that

$$\mathcal{I}_S(w) > r \cdot \operatorname{sdim}(S). \tag{6.7}$$

If $l > 0$, then the minimality of $l_r(T)$ tells us that

$$\prod_{i=0}^{l-2} \widehat{c}_{w[i]}(r+m+i+1) > 2^{-r} - 2^{-(r+m)}.$$

It follows by (6.5) and (6.6) that

$$\begin{aligned}\prod_{i=0}^{l-2} c_{w[i]} &> 2^{-r} - 2^{1-(r+m)} \\ &= 2^{-r}(1 - 2^{1-m}) \\ &> 2^{-(r+\alpha/\operatorname{sdim}(S))}/c_{\min},\end{aligned}$$

whence

$$\begin{aligned}\prod_{i=0}^{l-1} c_{w[i]} &> \frac{c_{w[l-1]}}{c_{\min}} 2^{-(r+\alpha/\operatorname{sdim}(S))} \\ &\geq 2^{-(r+\alpha/\operatorname{sdim}(S))}.\end{aligned}$$

This implies that

$$\pi_S(w) > 2^{-(r\cdot\operatorname{sdim}(S)+\alpha)}. \tag{6.8}$$

If $l = 0$, then $\pi_S(w) = 1$, so (6.8) again holds. Hence, in any case, we have

$$\mathcal{I}_S(w) < r \cdot \operatorname{sdim}(S) + \alpha. \tag{6.9}$$

By (6.7) and (6.9), A_r has property (iii). □

Our main theorem concerns the following type of fractal.

DEFINITION 6.9. *A* computably self-similar fractal *is a set $F \subseteq \mathbb{R}^n$ that is the attractor of an IFS that is computable and satisfies the open set condition.*

Most self-similar fractals occurring in the literature are, in fact, computably self-similar. For instance, let F be a generalized Sierpinski triangle with contraction ratios $c_0, c_1, c_2 \in (0,1)$, defined as in Example 5.6. As we have noted, F is self-similar if

$c_0 + c_1 \leq 1$, $c_0 + c_2 \leq 1$, and $c_1 + c_2 \leq 1$. It is easy to see that F is computably self-similar if c_0, c_1, and c_2 are also computable real numbers.

We now have the machinery to give a complete analysis of the dimensions of points in computably self-similar fractals.

THEOREM 6.10 (main theorem). *If $F \subseteq \mathbb{R}^n$ is a computably self-similar fractal and S is an IFS testifying this fact, then, for all points $x \in F$ and all S-codes T of x,*

$$\dim(x) = \mathrm{sdim}(F)\dim^{\pi_S}(T) \tag{6.10}$$

and

$$\mathrm{Dim}(x) = \mathrm{sdim}(F)\mathrm{Dim}^{\pi_S}(T). \tag{6.11}$$

Proof. Assume the hypothesis, with $S = (S_0, \ldots, S_{k-1})$. Let $c_0, \ldots, c_{k-1}$ be the contraction ratios of $S_0, \ldots, S_{k-1}$, respectively, let G be a witness to the fact that S satisfies the open set condition, and let $l = \max\{0, \lceil \log \mathrm{diam}(G) \rceil\}$. Let $h : \Sigma_k^* \to \mathbb{Q}^n$ be an exactly computable, rational-valued hub function for S and G as given by Lemma 6.6. Let $\alpha = 1 + \mathrm{sdim}(F) \log \frac{1}{c_{\min}}$ for $c_{\min} = \min\{c_0, \ldots, c_{k-1}\}$, and choose a decidable set A for S and α as in Lemma 6.8.

For all $w \in \Sigma_k^*$, we have

$$\begin{aligned} \mathrm{diam}(S_G(w)) &= \mathrm{diam}(G) \prod_{i=0}^{|w|-1} c_{w[i]} \\ &= \mathrm{diam}(G)\pi_S(w)^{\frac{1}{\mathrm{sdim}(F)}}. \end{aligned}$$

It follows by (6.3) that, for all $r \in \mathbb{N}$ and $w \in A_{r+l}$,

$$2^{-r} a_1 \leq \mathrm{diam}(S_G(w)) \leq 2^{-r}, \tag{6.12}$$

where $a_1 = 2^{-\left(l + \frac{\alpha}{\mathrm{sdim}(F)}\right)} \mathrm{diam}(G)$.

Let $x \in F$, and let $T \in \Sigma_k^\infty$ be an S-code of x, i.e., $S(T) = x$. For each $r \in \mathbb{N}$, let w_r be the unique element of A_{r+l} that is a prefix of T. Much of this proof is devoted to deriving a close relationship between the Kolmogorov complexities $\mathrm{K}_r(x)$ and $\mathrm{K}(w_r)$. Once we have this relationship, we will use it to prove (6.10) and (6.11).

Since the hub function h is computable, there is a constant a_2 such that, for all $w \in \Sigma_k^*$,

$$\mathrm{K}(h(w)) \leq \mathrm{K}(w) + a_2. \tag{6.13}$$

Since $h(w_r) \in S_G(w_r)$ and $x = S(T) \in S_{\overline{G}}(w_r) = \overline{S_G(w_r)}$, (6.12) tells us that

$$|h(w_r) - x| \leq \mathrm{diam}(S_G(w_r)) \leq 2^{-r},$$

whence

$$\mathrm{K}_r(x) \leq \mathrm{K}(h(w_r))$$

for all $r \in \mathbb{N}$. It follows by (6.13) that

$$\mathrm{K}_r(x) \leq \mathrm{K}(w_r) + a_2 \tag{6.14}$$

for all $r \in \mathbb{N}$. Combining (6.14) and the right-hand inequality in (6.3) gives

$$\frac{\mathrm{K}_r(x)}{r \cdot \mathrm{sdim}(F)} \leq \frac{\mathrm{K}(w_r) + a_2}{\mathcal{I}_S(w_r) - \alpha} \tag{6.15}$$

for all $r \in \mathbb{N}$.

Let E be the set of all triples (q, r, w) such that $q \in \mathbb{Q}^n$, $r \in \mathbb{N}$, $w \in A_{r+l}$, and

$$|q - h(w)| \leq 2^{1-r}. \tag{6.16}$$

Since the set A and the condition (6.16) are decidable, the set E is decidable.

For each $q \in \mathbb{Q}^n$ and $r \in \mathbb{N}$, let

$$E_{q,r} = \{ w \in \Sigma_k^* \mid (q, r, w) \in E \}.$$

We prove two key properties of the sets $E_{q,r}$. First, for all $q \in \mathbb{Q}^n$ and $r \in \mathbb{N}$,

$$|q - x| \leq 2^{-r} \Rightarrow w_r \in E_{q,r}. \tag{6.17}$$

To see that this holds, assume that $|q-x| \leq 2^{-r}$. Since $x = S(T) \in S_{\overline{G}}(w_r) = \overline{S_G(w_r)}$, the right-hand inequality in (6.12) tells us that

$$|q - h(w_r)| \leq |q - x| + |x - h(w_r)| \leq 2^{-r} + \mathrm{diam}(S_G(w_r)) \leq 2^{1-r},$$

confirming (6.17).

The second key property of the sets $E_{q,r}$ is that they are small, namely, that

$$|E_{q,r}| \leq \gamma \tag{6.18}$$

holds for all $q \in \mathbb{Q}^n$ and $r \in \mathbb{N}$, where γ is a constant that does not depend on q or r. To see this, let $w \in E_{q,r}$. Then $w \in A_{r+l}$ and $|q - h(w)| \leq 2^{1-r}$, so $h(w) \in S_G(w) \cap B(q, 2^{1-r})$. This argument establishes that

$$w \in E_{q,r} \Rightarrow B(q, 2^{1-r}) \text{ meets } S_G(w). \tag{6.19}$$

Now let

$$\mathcal{G}_r = \{ S_G(w) \mid w \in A_{r+l} \}.$$

By our choice of G, $\mathcal{G}_r$ is a collection of disjoint open sets in $\mathbb{R}^n$. By the right-hand inequality in (6.12), each element of $\mathcal{G}_r$ is contained in a closed ball of radius 2^{-r}. Since G is open, it contains a closed ball of some radius $a_3 > 0$. It follows by the left-hand inequality in (6.12) that $S_G(w)$, being a contraction of G, contains a closed ball of radius $2^{1-r}a_4$, where $a_4 = \frac{a_1 a_3}{2\mathrm{diam}(G)}$. By Lemma 6.4, this implies that $B(q, 2^{1-r})$ meets no more than γ of the (closures of the) elements of $\mathcal{G}_r$, where $\gamma = (\frac{2}{a_4})^n$. By (6.19), this confirms (6.18).

Now let M be a prefix Turing machine with the following property. If $U(\pi) = q \in \mathbb{Q}^n$ (where U is the universal prefix Turing machine), s_r is the rth string in a standard enumeration $s_0, s_1, \ldots$ of $\{0,1\}^*$, and $0 \leq m < |E_{q,r}|$, then $M(\pi 0^{|s_r|} 1 s_r 0^m 1)$ is the mth element of $E_{q,r}$. There is a constant a_5 such that, for all $w \in \Sigma_k^*$,

$$\mathrm{K}(w) \leq \mathrm{K}_M(w) + a_5. \tag{6.20}$$

Taking π to be a program testifying to the value of $\mathrm{K}_r(x)$ and applying (6.17) and (6.18) shows that

$$\begin{aligned}\mathrm{K}_M(w_r) &\le |\pi 0^{|s_r|}1 s_r 0^m 1| \\ &= \mathrm{K}_r(x) + 2|s_r| + m + 2 \\ &\le \mathrm{K}_r(x) + 2\log(r+1) + |E_{q,r}| + 1 \\ &\le \mathrm{K}_r(x) + 2\log(r+1) + \gamma + 1,\end{aligned}$$

whence (6.20) tells us that

$$\mathrm{K}(w_r) \le \mathrm{K}_r(x) + \epsilon(r) \tag{6.21}$$

for all $r \in \mathbb{N}$, where $\epsilon(r) = 2\log(r+1) + a_5 + \gamma + 1$. Combining (6.21) and the left-hand inequality in (6.3) gives

$$\frac{\mathrm{K}_r(x)}{r \cdot \mathrm{sdim}(F)} \ge \frac{\mathrm{K}(w_r) - \epsilon(r)}{\mathcal{I}_S(w_r)} \tag{6.22}$$

for all $r \in \mathbb{N}$. Note that $\epsilon(r) = o(\mathcal{I}_S(w_r))$ as $r \to \infty$.

By (6.15) and (6.22), we now have

$$\frac{\mathrm{K}(w_r) - \epsilon(r)}{\mathcal{I}_S(w_r)} \le \frac{\mathrm{K}_r(x)}{r \cdot \mathrm{sdim}(F)} \le \frac{\mathrm{K}(w_r) + a_2}{\mathcal{I}_S(w_r) - \alpha} \tag{6.23}$$

for all $r \in \mathbb{N}$. In order to use this relationship between $\mathrm{K}_r(x)$ and $\mathrm{K}(w_r)$, we need to know that the asymptotic behavior of $\frac{\mathrm{K}(w_r)}{\mathcal{I}_S(w_r)}$ for $r \in \mathbb{N}$ is the same as the asymptotic behavior of $\frac{\mathrm{K}(w)}{\mathcal{I}_S(w)}$ for arbitrary prefixes w of T. Our verification of this fact makes repeated use of the *additivity* of $\mathcal{I}_S$, by which we mean that

$$\mathcal{I}_S(uv) = \mathcal{I}_S(u) + \mathcal{I}_S(v) \tag{6.24}$$

holds for all $u, v \in \Sigma_k^*$.

Let $r \in \mathbb{N}$, and let $w_r \sqsubseteq w \sqsubseteq w_{r+1}$, writing $w = w_r u$ and $w_{r+1} = wv$. Then (6.24) tells us that

$$\mathcal{I}_S(w_r) \le \mathcal{I}_S(w) \le \mathcal{I}_S(w_{r+1}),$$

and (6.3) tells us that

$$\mathcal{I}_S(w_{r+1}) - \mathcal{I}_S(w_r) \le \mathrm{sdim}(F) + \alpha,$$

so we have

$$\mathcal{I}_S(w_r) \le \mathcal{I}_S(w) \le \mathcal{I}_S(w_r) + a_6, \tag{6.25}$$

where $a_6 = \mathrm{sdim}(F) + \alpha$. We also have

$$\begin{aligned}a_6 &\ge \mathcal{I}_S(w_{r+1}) - \mathcal{I}_S(w_r) \\ &= \mathcal{I}_S(uv) \\ &= \log \frac{1}{\pi_S(uv)} \\ &\ge \log c_{\min}^{-\mathrm{sdim}(F)|uv|} \\ &= |uv|\mathrm{sdim}(F) \log \frac{1}{c_{\min}},\end{aligned}$$

i.e.,

$$|w_{r+1}| - |w_r| \leq a_7, \tag{6.26}$$

where $a_7 = \frac{a_6}{\mathrm{sdim}(F) \log \frac{1}{c_{\min}}}$.

Since (6.26) holds for all $r \in \mathbb{N}$ and a_7 does not depend on r, there is a constant a_8 such that, for all $r \in \mathbb{N}$ and $w_r \sqsubseteq w \sqsubseteq w_{r+1}$,

$$|\mathrm{K}(w) - \mathrm{K}(w_r)| \leq a_8. \tag{6.27}$$

It follows by (6.25) that

$$\frac{\mathrm{K}(w_r) - a_8}{\mathcal{I}_S(w_r) + a_6} \leq \frac{\mathrm{K}(w)}{\mathcal{I}_S(w)} \leq \frac{\mathrm{K}(w_r) + a_8}{\mathcal{I}_S(w_r)} \tag{6.28}$$

holds for all $r \in \mathbb{N}$ and $w_r \sqsubseteq w \sqsubseteq w_{r+1}$.

By (6.23), (6.28), Theorem 4.5, and Theorem 4.1, we now have

$$\begin{aligned}
\dim(x) &= \liminf_{r\to\infty} \frac{\mathrm{K}_r(x)}{r} \\
&= \mathrm{sdim}(F) \liminf_{r\to\infty} \frac{\mathrm{K}(w_r)}{\mathcal{I}_S(w_r)} \\
&= \mathrm{sdim}(F) \liminf_{j\to\infty} \frac{\mathrm{K}(T[0..j-1])}{\mathcal{I}_S(T[0..j-1])} \\
&= \mathrm{sdim}(F) \dim^{\pi_S}(T)
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{Dim}(x) &= \limsup_{r\to\infty} \frac{\mathrm{K}_r(x)}{r} \\
&= \mathrm{sdim}(F) \limsup_{r\to\infty} \frac{\mathrm{K}(w_r)}{\mathcal{I}_S(w_r)} \\
&= \mathrm{sdim}(F) \limsup_{j\to\infty} \frac{\mathrm{K}(T[0..j-1])}{\mathcal{I}_S(T[0..j-1])} \\
&= \mathrm{sdim}(F) \mathrm{Dim}^{\pi_S}(T);
\end{aligned}$$

i.e., (6.10) and (6.11) hold. □

Finally, we use relativization to derive the following well-known classical theorem from our main theorem.

Corollary 6.11 (Moran [40], Falconer [14]). *For every self-similar fractal $F \subseteq \mathbb{R}^n$,*

$$\dim_{\mathrm{H}}(F) = \mathrm{Dim}_{\mathrm{P}}(F) = \mathrm{sdim}(F).$$

Proof. Let $F \subseteq \mathbb{R}^n$ be self-similar. Then there is an IFS S satisfying $F(S) = F$ and the open set condition. For any such S, there is an oracle $A \subseteq \{0,1\}^*$ relative to

which S is computable. We then have

$$\begin{aligned}
\dim_{\mathrm{H}}(F) &\le \mathrm{Dim}_{\mathrm{P}}(F) \\
&= \mathrm{Dim}_{\mathrm{P}}^{A}(F) \\
&\le \mathrm{cDim}^{A}(F) \\
&= \sup_{x\in F} \mathrm{Dim}^{A}(x) \\
&=^{(a)} \mathrm{sdim}(F) \sup_{T\in\Sigma_k^\infty} \mathrm{Dim}^{\pi_S,A}(T) \\
&= \mathrm{sdim}(F) \\
&= \mathrm{sdim}(F) \sup_{T\in\Sigma_k^\infty} \dim^{\pi_S,A}(T) \\
&=^{(b)} \sup_{x\in F} \dim^{A}(x) \\
&= \mathrm{cdim}^{A}(F) \\
&=^{(c)} \dim_{\mathrm{H}}^{A}(F) \\
&= \dim_{\mathrm{H}}(F),
\end{aligned}$$

which implies the corollary. Equalities (a) and (b) hold by Theorem 6.10, relativized to A. Equality (c) holds by Corollary 6.3, relativized to A. □

7. Conclusion. Our main theorem gives a complete analysis of the dimensions of points in computably self-similar fractals. It will be interesting to see whether larger classes of fractals are also amenable to such pointwise analyses of dimensions.

Acknowledgments. The first author thanks Dan Mauldin for useful discussions. We thank Xiaoyang Gu for pointing out that $\dim_{\mathrm{H}}^{\nu}$ is Billingsley dimension, and we thank the referees for careful reading and several corrections and improvements.

REFERENCES

[1] K. B. Athreya, J. M. Hitchcock, J. H. Lutz, and E. Mayordomo, *Effective strong dimension in algorithmic information and computational complexity*, SIAM J. Comput., 37 (2007), pp. 671–705.
[2] M.F. Barnsley, *Fractals Everywhere*, 2nd ed., Academic Press, Boston, 1993.
[3] P. Billingsley, *Hausdorff dimension in probability theory*, Illinois J. Math, 4 (1960), pp. 187–209.
[4] V. Brattka and K. Weihrauch, *Computability on subsets of Euclidean space* I: *Closed and compact subsets*, Theoret. Comput. Sci., 219 (1999), pp. 65–93.
[5] M. Braverman, *On the complexity of real functions*, in Proceedings of the Forty-Sixth Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 155–164.
[6] M. Braverman and S. Cook, *Computing over the reals: Foundations for scientific computing*, Notices Amer. Math. Soc., 53 (2006), pp. 318–329.
[7] M. Braverman and M. Yampolsky, *Constructing non-computable Julia sets*, in Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, ACM, New York, 2007, pp. 709–716.
[8] J. Cai and J. Hartmanis, *On Hausdorff and topological dimensions of the Kolmogorov complexity of the real line*, J. Comput. System Sci., 49 (1994), pp. 605–619.
[9] H. Cajar, *Billingsley Dimension in Probability Spaces*, Springer Lecture Notes in Math. 892, Springer-Verlag, New York, 1981.
[10] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
[11] J. J. Dai, J. I. Lathrop, J. H. Lutz, and E. Mayordomo, *Finite-state dimension*, Theoret. Comput. Sci., 310 (2004), pp. 1–33.

[12] G. A. Edgar, *Integral, Probability, and Fractal Measures*, Springer-Verlag, New York, 1998.
[13] K. Falconer, *The Geometry of Fractal Sets*, Cambridge University Press, Cambridge, UK, 1985.
[14] K. Falconer, *Dimensions and measures of quasi self-similar sets*, Proc. Amer. Math. Soc., 106 (1989), pp. 543–554.
[15] K. Falconer, *Fractal Geometry: Mathematical Foundations and Applications*, John Wiley & Sons, New York, 2003.
[16] S. A. Fenner, *Gales and Supergales are Equivalent for Defining Constructive Hausdorff Dimension*, Technical report cs.CC/0208044, http://arxiv.org/abs/cs.CC/0208044, 2002.
[17] H. Fernau and L. Staiger, *Iterated function systems and control languages*, Inform. and Comput., 168 (2001), pp. 125–143.
[18] L. Fortnow, T. Lee, and N. Vereshchagin, *Kolmogorov complexity with error*, in Proceedings of the 23rd Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 3884, Springer-Verlag, New York, 2006, pp. 137–148.
[19] A. Grzegorczyk, *Computable functionals*, Fund. Math., 42 (1955), pp. 168–202.
[20] X. Gu, J. H. Lutz, and E. Mayordomo, *Points on computable curves*, in Proceedings of the Forty-Seventh Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2006, pp. 469–474.
[21] A. Gupta, R. Krauthgamer, and J. R. Lee, *Bounded geometries, fractals, and low-distortion embeddings*, in Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2003, pp. 534–543.
[22] F. Hausdorff, *Dimension und äußeres Maß*, Math. Ann., 79 (1919), pp. 157–179.
[23] P. Hertling, *Is the Mandelbrot set computable?*, Mathematical Logic Quarterly, 51 (2005), pp. 5–18.
[24] J. M. Hitchcock, *Effective fractal dimension bibliography*, available online at http://www.cs.uwyo.edu/~jhitchco/bib/dim.shtml.
[25] J. M. Hitchcock, *Gales suffice for constructive dimension*, Inform. Process. Lett., 86 (2003), pp. 9–12.
[26] J. M. Hitchcock, *Correspondence principles for effective dimensions*, Theory Comput. Syst., 38 (2005), pp. 559–571.
[27] H. Kamo and K. Kawamura, *Computability of self-similar sets*, Math. Logic Quarterly, 45 (1999), pp. 23–30.
[28] K. Ko, *Complexity Theory of Real Functions*, Birkhäuser Boston, Boston, 1991.
[29] D. Lacombe, *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles,* I, C. R. Acad. Sci. Paris, 240 (1955), pp. 2478–2480.
[30] D. Lacombe, *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles,* II, III, C. R. Acad. Sci. Paris, 241 (1955), pp. 13–14, 151–153.
[31] D. Lacombe, *Remarques sur les opérateurs récursifs et sur les fonctions récursives d'une variable réelle,* C. R. Acad. Sci. Paris, 241 (1955), pp. 1250–1252.
[32] P. Lévy, *Théorie de l'Addition des Variables Aleatoires*, 2nd ed., Gauthier–Villars, Paris, 1937, 1954.
[33] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, 2nd ed., Springer-Verlag, Berlin, 1997.
[34] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.
[35] J. H. Lutz, *Resource-bounded measure*, in Proceedings of the 13th IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 236–248.
[36] J. H. Lutz, *Dimension in complexity classes*, SIAM J. Comput., 32 (2003), pp. 1236–1259.
[37] J. H. Lutz, *The dimensions of individual strings and sequences*, Inform. and Comput. 187 (2003), pp. 49–79.
[38] P. Martin-Löf, *The definition of random sequences*, Inform. and Control, 9 (1966), pp. 602–619.
[39] E. Mayordomo, *A Kolmogorov complexity characterization of constructive Hausdorff dimension*, Inform. Process. Lett., 84 (2002), pp. 1–3.
[40] P. A. Moran, *Additive functions of intervals and Hausdorff dimension*, Proceedings of the Cambridge Philosophical Society, 42 (1946), pp. 5–23.
[41] M. B. Pour-El and J. I. Richards, *Computability in Analysis and Physics*, Springer-Verlag, New York, 1989.
[42] R. Rettinger and K. Weihrauch, *The computational complexity of some Julia sets*, in Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 177–185.

[43] C. P. Schnorr, *A unified approach to the definition of random sequences*, Mathematical Systems Theory, 5 (1971), pp. 246–258.
[44] C. P. Schnorr, *Zufälligkeit und Wahrscheinlichkeit*, Lecture Notes in Math. 218, Springer-Verlag, Berlin, 1971.
[45] C. P. Schnorr, *A survey of the theory of random sequences*, in Basic Problems in Methodology and Linguistics, R. E. Butts and J. Hintikka, eds., D. Reidel, Dordrecht, The Netherlands, 1977, pp. 193–210.
[46] L. Staiger, *A tight upper bound on Kolmogorov complexity and uniformly optimal prediction*, Theory Comput. Syst., 31 (1998), pp. 215–229.
[47] L. Staiger, *The Kolmogorov complexity of real numbers*, Theoret. Comput. Sci., 284 (2002), pp. 455–466.
[48] L. Staiger, *Constructive dimension equals Kolmogorov complexity*, Inform. Process. Lett., 93 (2005), pp. 149–153.
[49] D. Sullivan, *Entropy, Hausdorff measures old and new, and limit sets of geometrically finite Kleinian groups*, Acta Math., 153 (1984), pp. 259–277.
[50] C. Tricot, *Two definitions of fractional dimension*, Mathematical Proceedings of the Cambridge Philosophical Society, 91 (1982), pp. 57–74.
[51] K. Vereshchagin and P. M. B. Vitányi, *Algorithmic rate-distortion function*, in Proceedings of the IEEE International Symposium on Information Theory, IEEE Information Theory Society, Washington, DC, 2006, pp. 798–802.
[52] J. Ville, *Étude Critique de la Notion de Collectif*, Gauthier–Villars, Paris, 1939.
[53] H. Wegmann, *Uber den dimensionsbegriff in wahrsheinlichkeitsraumen von P. Billingsley* I *and* II, Z. Wahrscheinlichkeitstheorie verw. Geb., 9 (1968), pp. 216–221, 222–231.
[54] K Weihrauch, *Computable Analysis. An Introduction*, Springer-Verlag, New York, 2000.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1113–1140

THE COMPLEXITY OF MONADIC SECOND-ORDER UNIFICATION*

JORDI LEVY†, MANFRED SCHMIDT-SCHAUß‡, AND MATEU VILLARET§

Abstract. Monadic second-order unification is second-order unification where all function constants occurring in the equations are unary. Here we prove that the problem of deciding whether a set of monadic equations has a unifier is NP-complete, where we use the technique of compressing solutions using singleton context-free grammars. We prove that monadic second-order matching is also NP-complete.

Key words. second-order unification, theorem proving, lambda calculus, context-free grammars, rewriting systems

AMS subject classifications. 03B15, 68Q42, 68T15

DOI. 10.1137/050645403

1. Introduction.

The solvability of monadic second-order unification problems considered in this paper is a decision problem with many natural relationships with other well-known decision problems (higher-order unification, word unification, context unification and specializations thereof). For various members of this family, the exact complexity is still not known. In this paper we show that solvability of monadic second-order unification problems is NP-complete, thus adding a new piece to the larger puzzle. The techniques we use in our proof turned out to be relevant for other problems as well. Before we describe the organization of the paper we illuminate in more detail the relationship of monadic second-order unification with the above problems, give a brief survey on the techniques used in our proof, and indicate further contributions obtained from our method.

1.1. Monadic second-order unification from a higher-order perspective.

Higher-order unification (HOU) is unification in the simply typed λ-calculus, i.e., the problem of, given two λ-terms with the same type, deciding if there is a substitution (of free variables by equally typed terms) that, applied to both terms, makes them equivalent w.r.t. $\alpha\beta\eta$-equality (see chapter about HOU in [1]).

Second-order unification (SOU) is a restriction of HOU where all variables are at most second-order typed and constants are at most third-order typed. Some authors restrict constants in SOU to also be second-order typed, and it is also common to consider just one base type. Here, we will also make these assumptions. It is well known that the problem of deciding if an SOU problem has a solution is undecidable [5], even

*Received by the editors November 16, 2005; accepted for publication (in revised form) February 28, 2008; published electronically July 2, 2008. A preliminary version of this work appeared in *Proceedings of the* 15*th Annual International Conference on Rewriting Techniques and Applications (RTA* '04*)*, where it was awarded the best paper prize. This research was partially supported by the CYCIT research projects iDEAS (TIN2004-04343) and Mulog2 (TIN2007-68005-C04-01).

http://www.siam.org/journals/sicomp/38-3/64540.html

†Institut d'Investigació en Intel·ligència Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Campus de la UAB, Bellaterra, 08193 Barcelona, Spain (levy@iiia.csic.es, http://www.iiia.csic.es/~levy).

‡Institut für Informatik, Johann Wolfgang Goethe-Universität, Postfach 11 19 32, D-60054 Frankfurt, Germany (schauss@ki.informatik.uni-frankfurt.de, http://www.ki.informatik.uni-frankfurt.de/persons/schauss/schauss.html).

§Informàtica i Matemàtica Aplicada, Universitat de Girona, Campus de Montilivi, 17071 Girona, Spain (villaret@ima.udg.es, http://ima.udg.es/~villaret).

if we impose additional restrictions on the number of second-order variables (just one), their number of occurrences (just four), and their arity (just one) [3, 9, 13]. All these undecidability proofs require a language with at least a binary or higher arity constant. In fact, one single binary constant is enough [3, 14].

Monadic second-order unification (MSOU) is a restriction of SOU where all function constants occurring in the problem are at most unary. Contrary to general SOU, MSOU is decidable [6, 24, 2]. In [23], it is proved that the problem is NP-hard. In this paper, our main contribution is to prove that MSOU is in NP.

Assuming that second-order variables, like constants, are also unary does not affect the decidability of MSOU or its complexity (see Proposition 3.1). Also, the use of a unique first-order constant does not affect the complexity of MSOU. Here we will use a single one called $\flat$. Thus, in instantiations $\lambda y\,.\,t$ for variables X of the problem, the variable y can occur at most once in t. This leads to the specialization where every equation is of the form $f_1(f_2(\dots f_n(\flat)\dots)) \stackrel{?}{=} g_1(g_2(\dots g_n(\flat)\dots))$, where f_i, g_i are unary function symbols or unary variables, and which is rather similar to a word equation.

Example 1. The equation $X(a(X(\flat))) \stackrel{?}{=} a(Y(Y(\flat)))$ has, among others, the solutions $[X \mapsto \lambda x\,.\,a(a(a(x))), Y \mapsto \lambda x\,.\,a(a(a(x)))]$ and $[X \mapsto \lambda x\,.\,a(a(a(\flat))), Y \mapsto \lambda x\,.\,a(x)]$. In the second solution the instantiation of X does not use its argument.

Since all terms are monadic, we will avoid the use of parenthesis in all cases where this is possible and write the terms as words.

1.2. Monadic second-order unification from a word unification perspective. Word unification (WU) is the problem of solving equations on strings. Given a finite alphabet of constants Σ and variables $\mathcal{X}$, a word equation $s \stackrel{?}{=} t$ is defined by a pair of words $s, t \in (\Sigma \cup \mathcal{X})^+$. A solution of $s \stackrel{?}{=} t$ is a substitution of variables by words in Σ^* such that, after replacing, the words obtained from s and t are equal.

In MSOU, apart from Σ and $\mathcal{X}$, we also have a special symbol denoted by $\flat$. A basic MSOU equation $s \stackrel{?}{=} t$ is defined by a pair of words $s, t \in (\Sigma \cup \mathcal{X})^+\flat$. A solution of $s \stackrel{?}{=} t$ is a substitution of variables by either $\lambda x\,.\,w\,x$ or $\lambda x\,.\,w\flat$, where $w \in \Sigma^*$ and where we use β-reduction after the substitution. In the first case we say that the instantiation uses its argument. The substitution of a variable X by $\lambda x\,.\,w\,x$ in $w_1\,X\,w_2$, where w_1, w_2 do not contain X, results in $w_1\,w\,w_2$, as in word unification, whereas the substitution of X by $\lambda x\,.\,w\flat$ in $w_1\,X\,w_2$ results in $w_1\,w\flat$. Therefore, compared to WU in MSOU some part of the original equation can be removed after instantiation. Moreover, the set of solutions of an MSOU equation is wider than the set of solutions of the corresponding WU equation.

Example 2. All solutions of the word equation $X\,a\,X \stackrel{?}{=} a\,Y\,Y$ have the form $[X \mapsto a^n, Y \mapsto a^n]$. The monadic equation $X\,a\,X\flat \stackrel{?}{=} a\,Y\,Y\flat$ has, apart from solutions of the form $[X \mapsto \lambda x\,.\,a^n\,x, Y \mapsto \lambda x\,.\,a^n\,x]$, other solutions of the form $[X \mapsto \lambda x\,.\,a\,w\,w\flat, Y \mapsto \lambda x\,.\,w\,x]$, $[X \mapsto \lambda x\,.\,a\,w\,x, Y \mapsto \lambda x\,.\,w\,a\,a\,w\flat]$, $[X \mapsto \lambda x\,.\,x, Y \mapsto \lambda x\,.\,\flat]$, and $[X \mapsto \lambda x\,.\,a\,w\flat, Y \mapsto \lambda x\,.\,w\flat]$, for any $w \in \Sigma^*$.

MSOU problems can be decided by guessing for every variable whether it uses its argument or not, modifying the equation by dropping symbols to the right of variables that do not use their arguments, and then calling WU (see also Proposition 2.5).

Example 3. The solutions of the MSOU equation $X\,a\,X\flat \stackrel{?}{=} a\,Y\,Y\flat$ can be found by solving the disjunction of the word equations $X\,a\,X \stackrel{?}{=} a\,Y\,Y$, $X \stackrel{?}{=} a\,Y\,Y$, $X\,a\,X \stackrel{?}{=} a\,Y$, and $X \stackrel{?}{=} a\,Y$.

This simple reduction shows that MSOU is decidable, since WU is decidable [16],

and also that MSOU is in PSPACE, since the reduction is in NP and by using the result that WU is in PSPACE [20]. Since this is the best currently known upper bound for WU, our result that MSOU is NP-complete gives a sharp bound that (currently) cannot be obtained from results on WU.

1.3. Techniques. To prove that MSOU is in NP, we first show for any solvable set of equations how we can represent (at least) one of the solutions (unifiers) in polynomial space. Then, we prove that we can check in polynomial time if a substitution (written in such a representation) is a solution.

We combine two key results to obtain this sharp bound: One is the result on the exponential upper bound on the exponent of periodicity of size-minimal unifiers [16, 8, 23] (see Lemma 2.4). This upper bound allows us to represent exponents in linear space. The other key is a result of Plandowski [17, 18] (see Theorem 4.2). He proves that, given two context-free recursion-free grammars with just one rule for every nonterminal symbol (here called *singleton* grammars), we can check if they define the same (singleton) language in polynomial time (in the size of the grammars). This result is used to check that applying a substitution (represented using this kind of grammar) to both sides of an equation will result in the same term. The successful combination requires us to prove a polynomial upper bound for the size increase of the singleton grammars obtained after a series of extensions (see section 4).

1.4. Further contributions. The method presented in this paper appears to be new and powerful for obtaining sharp complexity bounds and efficient algorithms for unification problems.

WU can be seen as a restriction of MSOU with the extra condition that every variable must be instantiated with a λ-term $\lambda x\,.\,t$, where t has exactly one occurrence of the bound variable x. Our results suggests that MSOU is an easier problem—from the complexity point of view—than WU. Moreover, all naive attempts to encode WU as MSOU have failed. The direct application of our method to WU fails, since Lemma 6.8 does not hold for WU; i.e., if all equations are of the form $X \ldots \stackrel{?}{=} Y \ldots$, then there is a trivial solution as MSOU equations, but this syntactic form does not imply any easy solution method for the equations interpreted as word equations.

Context unification (CU) is a variant of SOU where instantiations of second-order variables use their arguments exactly once. Hence, WU is monadic CU. Decidability of CU is currently unknown. During revision of this paper we were able to apply variants of this technique to determine the complexity of a fragment of CU: stratified context unification [12].

Bounded second-order unification (BSOU) is another variant of SOU where, given a positive integer k, instantiations of second-order variables can use their arguments at most k times. Hence, MSOU is also a subproblem of BSOU, because in MSOU instantiations of variables can use their arguments at most once. It is also known that BSOU is decidable [22], which provides another proof of decidability of MSOU, but no tight upper complexity bound. On the other hand, our proof and results suggest an application to BSOU, which has recently (during revision of this paper) resulted in proving a precise upper complexity bound for BSOU [11].

1.5. Paper organization. This paper proceeds as follows. In section 2 we define a *basic* version of the MSOU problem and give some complexity bounds. We will prove in the rest of the paper that this problem is in NP. Then, in section 3 we define the MSOU problem in its most general form as a specialization of SOU, and we prove that it can be NP-reduced to the basic version. We prove some properties of singleton

context-free grammars in section 4, and in section 5 we use them to compact the representation of equations and solutions. We use a graph in order to describe the instantiation of some variable w.r.t. a given solution (section 6). Sometimes, we need to rewrite such graphs (section 7). Based on this graph, we prove that for any size-minimal solution we can represent the values of all variable instantiations using a polynomial-sized singleton grammar (Theorem 7.7). In section 8, we conclude the NP-completeness of MSOU and also of the corresponding matching problem.

2. Basic monadic second-order unification. In this section we present a simplification of the MSOU problem, called the *basic* MSOU problem. As we will see in section 3 (Proposition 3.1), general MSOU can be NP-reduced to this basic case; therefore this will not cause a loss of generality. The simplification consists in (1) considering only unary free variables (notice that, in the general case, the "monadic" restriction affects constants only, not variables) and (2) considering only the function symbols occurring in the equations and a unique (zero-ary) constant, called $\flat$. In this presentation we will limit the use of concepts and notation of the λ-calculus as much as possible. Moreover, we will use words to represent monadic second-order terms. This will make the use of context-free grammars in section 4 more comprehensible.

Let Σ be a finite set of unary function symbols, denoted by lowercase letters $f, g, \ldots$, and let $\mathcal{X}$ be an infinite and denumerable set of unary variables, denoted by uppercase letters $X, Y, \ldots$. Apart from these sets, we also consider a unique constant $\flat$.

Words of $(\Sigma \cup \mathcal{X})^*$ are denoted by lowercase letters $w, u, v, \ldots$, and ε is the empty word. The length of a word w is denoted by $|w|$. Concatenation is juxtaposition. The notation $v \preceq w$ means that the word v is a prefix of the word w.

Monadic terms, denoted by letters $s, t, \ldots$, are defined by the grammar $t ::= \flat \mid f(t) \mid X(t)$, where $f \in \Sigma$ and $X \in \mathcal{X}$. We remove parentheses and represent monadic terms, e.g., $f(X(g(Y(\flat))))$, as words, e.g., $f\,X\,g\,Y\,\flat$. Therefore, terms are redefined as words of $(\Sigma \cup \mathcal{X})^*\,\flat$. The size of a term $t = w\,\flat$, noted $|t|$, is defined by $|t| = |w|$.

Monadic functions, denoted by Greek letters $\varphi, \ldots$, may be of the form $\lambda x\,.\,w\,\flat$ or $\lambda x\,.\,w\,x$, where $w \in (\Sigma \cup \mathcal{X})^*$. In the first case we say that the function *does not use* the argument. In both cases, the size of the function is $|w|$, and x is said to be a *bound variable*, i.e., not *free*.

A *monadic substitution*, denoted by Greek letters $\sigma, \rho, \tau, \ldots$, is a mapping from a finite subset of variables to monadic functions. We represent these mappings as $\sigma = [X_1 \mapsto \varphi_1, \ldots, X_n \mapsto \varphi_n]$, where $Dom(\sigma) = \{X_1, \ldots, X_n\}$ is the *domain* of the substitution. We extend substitutions to functions from monadic terms to monadic terms recursively as follows:

$$\begin{aligned}
&\sigma(\flat) = \flat,\\
&\sigma(f\,w_1) = f\,\sigma(w_1),\\
&\sigma(X\,w_1) = w_2\,\flat \qquad \text{if } \sigma(X) = \lambda x\,.\,w_2\,\flat,\\
&\sigma(X\,w_1) = w_2\,\sigma(w_1) \quad \text{if } \sigma(X) = \lambda x\,.\,w_2\,x.
\end{aligned}$$

The size of a substitution is defined as $|\sigma| = \sum_{X \in Dom(\sigma)} |\sigma(X)|$. Given two substitutions σ and ρ, their composition is defined by $(\sigma \circ \rho)(t) = \sigma(\rho(t))$, for any term t, and is also a substitution; hence $Dom(\sigma \circ \rho)$ is finite. Given a set of variables V and a substitution σ, the restriction of σ to the domain V is denoted by $\sigma|_V$. Given a set of variables V, we say that a substitution σ is *more general* w.r.t. V than another substitution ρ, denoted $\sigma \preceq_V \rho$, if there exists a substitution τ such that $\rho(X) = \tau(\sigma(X))$

for all variables $X \in V$, i.e., $\rho = (\tau \circ \sigma)|_V$. (Usually, V will be the set of variables occurring in a set of equations; in this case, we do not mention V if it is clear from the context). This defines a preorder relation on substitutions. An equivalence relation can also be defined as $\sigma \approx_V \rho$ if $\sigma \preceq_V \rho$ and $\rho \preceq_V \sigma$. A substitution σ is said to be *ground* if $\sigma(X)$ does not contain (free occurrences of) variables for all $X \in Dom(\sigma)$. Notice that if σ and τ are ground, then $\sigma \approx_V \tau$ is equivalent to $\sigma|_V = \tau|_V$ (otherwise they are only equivalent modulo variable renaming). We say that a substitution σ *introduces* a constant a (or a variable X) if, for some $Y \in Dom(\sigma)$, $\sigma(Y)$ has an occurrence of a (or X).

DEFINITION 2.1. *A* basic monadic second-order unification problem *(basic MSOU problem) E is a finite set of pairs of monadic terms a.k.a.* monadic equations*, represented as $E = \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$.*

The set of variables occurring in E is denoted as $FV(E)$. The *size* of E is the sum of the sizes of its terms s_i and t_i and is denoted as $|E|$. We denote the number of equations of E as $\#Eq(E)$.

DEFINITION 2.2. *A* unifier *of E is a monadic substitution σ, mapping variables of $FV(E)$ to monadic functions and solving all equations: $\sigma(s_i) = \sigma(t_i)$ for $i = 1, \ldots, n$. It is said to be* ground *if $\sigma(s_i)$ and $\sigma(t_i)$ do not contain free occurrences of variables for $i = 1, \ldots, n$.* Most general unifiers *are unifiers that are minimal w.r.t. $\preceq_{FV(E)}$.*

A solution *of E is a ground unifier. A solution σ of E is said to be* size-minimal *if $Dom(\sigma) = FV(E)$ and has minimal size among all solutions of E; i.e., it minimizes $\sum_{X \in FV(E)} |\sigma(X)|$.*

The problem E is said to be unifiable *if it has a unifier, and* solvable *if it has a ground unifier or solution.*

Example 4. Let f and g be unary function symbols and X and Y unary variables. Consider the following basic MSOU problem:

$$\{f\, g\, Y\, X\, \flat \stackrel{?}{=} X\, f\, g\, Y\, \flat\}.$$

It has infinitely many solutions, for instance, $\sigma_1 = [X \mapsto \lambda x\,.\,(f\, g)^n\, x, \quad Y \mapsto \lambda x\,.\,(f\, g)^m x]$ for any $n, m \geq 0$ or $\sigma_2 = [X \mapsto \lambda x\,.\,(f\, g)^{n+1}\, \flat, \quad Y \mapsto \lambda x\,.\,(f\, g)^n\, \flat]$ for any $n \geq 0$. Obviously σ_1 with $n = m = 0$ is a size-minimal solution. Observe also that, interpreting the problem as a WU equation, we can use σ_1 to get the corresponding solution for the word equation because the monadic functions of the solutions use their argument, whereas this is not the case for σ_2.

2.1. The exponent of periodicity bound. The following lemma will provide us with an upper bound on the number of iterations of subwords within solutions.

DEFINITION 2.3. *For a ground substitution σ, its* exponent of periodicity*, denoted as* $\mathrm{eop}(\sigma)$*, is the maximal number $n \in \mathbb{N}$, such that for words u, v, and w over Σ^*, where v is not empty, $\sigma(X) = \lambda y\,.\,u\, v^n\, w\, y$ or $\sigma(X) = \lambda y\,.\,u\, v^n\, w\, \flat$ for some $X \in Dom(\sigma)$.*

We know that any size-minimal ground unifier (i.e., solution) of a set of MSOU equations satisfies the following exponent of periodicity lemma [16, 8, 23, 22].

LEMMA 2.4 (see [22, Lemma 4.1]). *There exists a constant $\alpha \in \mathbb{R}^+$ such that for every basic MSOU problem $\langle \Sigma, E \rangle$ and every size-minimal solution σ we have $\mathrm{eop}(\sigma) \leq 2^{\alpha |E|}$.*

2.2. Some upper and lower complexity bounds. We show the relation of MSOU problems to WU. The NP-reduction of MSOU to WU (and its NP-hardness) allows us to translate any upper bound from WU to MSOU. It does not appear to be

possible to encode WU as an MSOU problem, which provides evidence that MSOU may be an easier problem than WU.

PROPOSITION 2.5. *MSOU is in PSPACE.*

Proof. MSOU is NP-reducible to solvability of basic MSOU problems (Proposition 3.1), and this in turn to WU: Given a basic MSOU problem E, we solve it using WU as follows. It is only necessary to search solutions where $\sigma(X)$, for $X \in FV(E)$, is of the form $\lambda x \,.\, a_1 \dots a_n \, x$ or of the form $\lambda x \,.\, a_1 \dots a_n \, \flat$. Thus, the first step is guessing, for every variable occurring in E, whether it uses its argument or not, i.e., whether it is of the first or second form. Then, we translate E into a set of word equations by first replacing every occurrence of $X\, s$ by $X \, \flat$, when $\sigma(X)$ does not use its argument, and then removing $\flat$ at the end of the terms and interpreting them as words. Now, we can apply an algorithm solving WU.

This nondeterministic reduction is correct, since if E is solvable as a basic MSOU problem, then the resulting word equations are solvable (for the convenient guessing). It is easy to see that the converse is also true.

A theorem of Plandowski [20] showing that WU is in PSPACE now implies that MSOU is in PSPACE. □

It is well known that MSOU is NP-hard [23]. We show that this also holds for monadic second-order matching. The proof gives a good feeling of what one can express in MSOU.

THEOREM 2.6. *Basic monadic second-order matching is NP-hard.*

Proof. We use the ONE-IN-THREE-SAT problem, which is known to be NP-complete [4]. An instance of the ONE-IN-THREE-SAT problem consists of the following: a set of propositional variables $p_1, \dots, p_n$ and m clauses $C_i = \{q_{i,1}, q_{i,2}, q_{i,3}\}$, where $q_{i,j} \in \{p_1, \dots, p_n\}$ for every $i = 1, \dots, m$ and $j = 1, 2, 3$. A solution is an assignment of the truth values *true* and *false* to the propositional variables, such that in every clause exactly one variable is assigned the value *true*.

We construct a basic MSOU problem where equations have ground right-hand sides and where $\Sigma = \{a, b, c\}$. For every $i = 1, \dots, n$ let X_i, Y_i be unary second-order variables. For $i = 1, \dots, n$, we use the equations

$$\begin{aligned} X_i \, Y_i \, b \flat &\stackrel{?}{=} a \, b \flat, \\ X_i \, Y_i \, c \flat &\stackrel{?}{=} a \, c \flat. \end{aligned}$$

These equations enforce that for all i either X_i is instantiated by $\lambda x \,.\, x$ or by $\lambda x \,.\, a \, x$, and similarly for Y_i, and that there are at most two possibilities for the instantiation of the pair X_i, Y_i for every i: The assignment $[X_i \mapsto \lambda x \,.\, a \, x, Y_i \mapsto \lambda x \,.\, x]$ is interpreted as *true*, and the assignment $[X_i \mapsto \lambda x \,.\, x, Y_i \mapsto \lambda x \,.\, a \, x]$ as *false*. Every clause $C = \{p_i, p_j, p_k\}$ is encoded as an equation

$$X_i \, X_j \, X_k \, b \flat \stackrel{?}{=} a \, b \flat.$$

Now it is obvious that the set of constructed equations has a unifier, if and only if the instance of ONE-IN-THREE-SAT is solvable. The equations form a monadic second-order matching problem and can be generated in linear time. Hence, the claim follows. □

3. General monadic second-order unification. In the rest of this paper we will prove the complexity estimation for *basic* MSOU problems. In this section we will argue that the restriction to "basic" does not compromise generality. The main claim is that there is a nondeterministic reduction from (general) MSOU problems to

basic MSOU problems that can be done in nondeterministic polynomial time. As a subcase of HOU, the definition of the problem in all its generality requires the use of the λ-calculus. However, we will limit its use to this section, which can be skipped by those readers not familiar with the λ-calculus.

We will use the standard notation and definitions of the simply typed λ-calculus, and we inherit the definitions of the previous section, unless we explicitly overwrite them here.

We consider only one (first-order) *base type* o and all the *second-order types* constructed from it, i.e., the ones described by the syntax $\tau ::= o \to o \mid o \to \tau$, with the usual convention that $\to$ is associative to the right. Hence, every type is o which has order one, or it is of the form $o \to o \to \cdots \to o$ and has order two.[1]

We consider a *signature* $\Sigma = \Sigma_0 \cup \Sigma_1$ of constants, denoted by $a, b, c, \ldots$, where constants of Σ_0 have type o and constants of Σ_1 have type $o \to o$. There is also a *set of variables* $\mathcal{X} = \bigcup_{i \geq 0} \mathcal{X}_i$, denoted by $x, y, z, \ldots$, where every set $\mathcal{X}_i$ contains infinitely and denumerable many variables with type $\underbrace{o \to \cdots \to o}_{i+1}$. Constants of Σ_0 and variables of $\mathcal{X}_0$ are first-order typed and are said to have arity zero, whereas those of Σ_i and $\mathcal{X}_i$, for $i > 0$, are second-order typed and have arity i.

Well-typed *terms* over the signature Σ and the set of variables $\mathcal{X}$ are built as usual in the simply typed λ-calculus:

(i) any constant $a \in \Sigma_i$ and any variable $x \in \mathcal{X}_i$ is a well-typed term of type $o \overset{i+1}{\to \cdots \to} o$,

(ii) if t is a well-typed term of type τ, and $x \in \mathcal{X}_0$, then $(\lambda x \,.\, t)$ is also a well-typed term of type $o \to \tau$, and

(iii) if s of type $o \to \tau$ and t of type o are well-typed terms, then $(s\, t)$ is also a well-typed term of type τ.

General second-order terms are defined using a signature $\Sigma = \bigcup_{i \geq 0} \Sigma_i$ and a set of variables $\mathcal{X} = \bigcup_{i \geq 0} \mathcal{X}_i$, where constants of Σ_i and variables of $\mathcal{X}_i$ have arity i. Therefore, monadic terms are second-order terms built without using constants of arity greater than one. Notice that there is no restriction on the arity of variables, whereas in basic MSOU we consider only unary variables.

Any term of type $\underbrace{o \to \cdots \to o}_{n+1}$ is said to have *arity* n. It is called of *first-order type* when $n = 0$, and of *second-order type* when $n > 0$. Hence, the arity of a term or of a symbol determines its type, and we will usually specify the arity instead of the type. When we say *normal form* we mean η-long β-reduced normal form, defined as usual. Since we do not consider third- or higher-order constants, first-order typed terms in normal form do not contain λ-abstractions, and second-order typed terms contain λ-abstractions only in outermost positions. The set of *free variables* of a term t is denoted by $FV(t)$. A term without occurrences of free variables is said to be *closed*. The *size* of a term t is denoted $|t|$ and defined as its number of symbols (variables and constants), when written in normal form.

Second-order substitutions are functions from terms to terms, defined as usual. The application of a substitution σ to a term t is written as $\sigma(t)$, where we implicitly assume that $\sigma(t)$ (after some β-reductions) is written in normal form. For any substitution σ, the set of variables x, such that $\sigma(x) \neq_{\beta\eta} x$, is finite and is called the *domain* of the substitution and denoted $Dom(\sigma)$. A substitution σ can be represented

[1] This also means that we do not allow symbols or expressions of third- or a higher-order type.

as $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$, where $x_i \in Dom(\sigma)$ and t_i has the same type as x_i and satisfies $t_i = \sigma(x_i)$.

An instance of the *(general) MSOU problem* is a pair $\langle \Sigma, E \rangle$, where $\Sigma = \Sigma_0 \cup \Sigma_1$ is a monadic signature and E is a *set of equations* $E = \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$, where s_i and t_i are normalized first-order terms over Σ, i.e., terms not containing λ-abstractions.

Note that, in monadic signatures, closed second-order terms are of two forms: $\lambda x_1 . \cdots . \lambda x_n . (a_1 (\ldots (a_m x_i) \ldots))$ or $\lambda x_1 . \cdots . \lambda x_n . (a_1 (\ldots (a_m b) \ldots))$ for unary constants a_i and a zero-ary constant b. Since solutions σ map variables to closed terms, in the first case we say that $\sigma(x)$ *uses one* of its arguments, and if x is unary, we say that $\sigma(x)$ *uses* its argument, and in the second case we say that $\sigma(x)$ is constant or that it *ignores* its argument.

Unifiability of MSOU problems does not depend on the signature, as long as the symbols of the equations are in the signature. However, its solvability has some dependence on the signature; more precisely it depends on the existence of at least one first-order constant.

In MSOU, as in general SOU, we can prove that *for every unifiable set of equations E, and every most general unifier σ, all constants occurring in σ also occur in E.* The proof is by contradiction. If there is a most general unifier using constants not occurring in the equations, we can replace these constants by fresh variables, obtaining a more general unifier. The statement does not hold for variable occurrences in unifiers. Even if the set of equations is built from unary variables and unary constants, most general unifiers may introduce fresh n-ary variables with $n \geq 2$. For instance, the set of equations $\{(x\, a) \stackrel{?}{=} (y\, b)\}$ has only one most general second-order unifier $[x \mapsto \lambda x . ((z\, x)\, b), y \mapsto \lambda x . ((z\, a)\, x)]$ that introduces a binary variable z.

PROPOSITION 3.1. *Unifiability of MSOU problems is NP-reducible to solvability of basic MSOU problems.*

Proof. The reduction is done in three steps.

(i) First, we reduce unifiability of MSOU problems to solvability, provided that Σ_0 contains at least one constant. In other words, for any signature Σ, any set of equations E over Σ, and first-order constant b, $\langle \Sigma, E \rangle$ is unifiable, if and only if $\langle \Sigma \cup \{b\}, E \rangle$ is solvable.

For the *if* direction, assume given a solution σ. If $b \notin \Sigma$, then $b \notin E$ and we can replace b by a fresh first-order variable x_b everywhere in σ and obtain a (maybe nonground) unifier of $\langle \Sigma, E \rangle$. For the *only if* part, assume given a unifier σ. Then we can define a substitution ρ such that, for every $n \geq 0$, every n-ary variable $x \in FV(\sigma(E))$ is instantiated by $\lambda x_1 . \cdots . \lambda x_n . b$. Then $\rho \circ \sigma$ is a solution of $\langle \Sigma \cup \{b\}, E \rangle$.

(ii) Second, we prove that solvability of MSOU problems is reducible in polynomial time to solvability of MSOU problems with just one first-order constant.

Assume given an MSOU problem $\langle \Sigma, E \rangle$. If $\Sigma_0 = \emptyset$, then the problem is unsolvable. Otherwise, we reduce the problem as follows. We transform the signature Σ into a new signature $\Sigma' = \Sigma'_1 \cup \Sigma'_0$, where the set of unary constants is $\Sigma'_1 = \Sigma_1 \cup \Sigma_0$; i.e., the former zero-ary constants are unary ones in the new signature, and the set of zero-ary constants is $\Sigma'_0 = \{\flat\}$. We replace every first-order constant occurrence a in the equations E by $(a\, \flat)$, obtaining a set of equations E' over Σ'. We will see that any solution σ of $\langle \Sigma, E \rangle$ can be translated into a solution σ' of $\langle \Sigma', E' \rangle$, and vice versa.

Any solution σ of $\langle \Sigma, E \rangle$ can be translated into a solution of $\langle \Sigma', E' \rangle$ using the same transformation as for the equations. To show the other direction, let σ' be

a solution of $\langle \Sigma', E' \rangle$. Before retranslating σ' we transform it into σ'' as follows: For every x in $Dom(\sigma')$, we remove every occurrence of symbols $a \in \Sigma_0$ in $\sigma'(x)$ which is not of the form $(a\, \flat)$; i.e., we replace $(a\, s)$ by s when $s \neq \flat$, until this replacement is no longer applicable. The translation from E to E' ensures that in E' every occurrence of all x is in subterms of the form $((\ldots(x\, s_1)\ldots)\, s_n)$, where $s_i \neq \flat$. Looking at the different cases, the removal of symbols takes place only within instantiations of variables and does not conflict with the constants occurring in E. Hence, σ'' is a solution of $\langle \Sigma', E' \rangle$ and can be immediately retranslated to a solution of $\langle \Sigma, E \rangle$.

The translations of signature, set of equations, and solutions in either direction are polynomial.

(iii) Third, we prove that we can go a step further, assuming that all variables are unary: We show that solvability of MSOU, where $\Sigma_0 = \{\flat\}$, is nondeterministically reducible in polynomial time to solvability of MSOU with the same signature, and where all variables occurring in the equations are unary: $\mathcal{X}_n = \emptyset$ for all $n \neq 1$.

Given an MSOU problem $\langle \Sigma, E \rangle$, where $\Sigma_0 = \{\flat\}$, we consider substitutions ρ that instantiate every first-order variable $x \in FV(E)$ by $(x'\, \flat)$, where x' is a fresh unary variable, and every n-ary variable $y \in FV(E)$ (with $n \geq 2$) by either $\lambda x_1 . \cdots . \lambda x_n . (y'\, x_i)$, where $1 \leq i \leq n$, or $\lambda x_1 . \cdots . \lambda x_n . (y'\, \flat)$, where y' is a fresh unary variable, and the selection is nondeterministic. Obviously, if for some ρ as given above, $\langle \Sigma, \rho(E) \rangle$ is solvable, so is $\langle \Sigma, E \rangle$.

Conversely, if $\langle \Sigma, E \rangle$ is solvable, we prove that for some ρ satisfying the specified conditions, $\langle \Sigma, \rho(E) \rangle$ is also solvable. Mainly, we prove that there is some ρ as specified above and a substitution τ with $\sigma(x) = \tau \circ \rho(x)$ for all $x \in FV(E)$; thus τ solves $\langle \Sigma, \rho(E) \rangle$.

Since all constants have arity at most one and solutions are ground, instantiations $\sigma(x)$ of n-ary variables, for $n \geq 2$, use at most one of their arguments: $\sigma(x) = \lambda x_1 . \cdots . \lambda x_n . t$, where t has a unique occurrence of some x_i, or none. Therefore, we can take $\rho(x) = \lambda x_1 . \cdots . \lambda x_n . (x'\, x_i)$ or $\rho(x) = \lambda x_1 . \cdots . \lambda x_n . (x'\, \flat)$. Instantiations of first-order variables use at least the first-order constant $\flat$; therefore they can also be replaced by a fresh unary variable applied to this constant. It is obvious how to construct the solution τ of $\langle \Sigma, \rho(E) \rangle$ from σ.

Finally, note that we can compute the substitution ρ in nondeterministic polynomial time on the size of E because for any nonunary variable x, we can guess whether x uses one of its arguments and, in the positive case, which argument $\rho(x)$ uses. □

The following lemma states that if there is just one zero-ary constant in the signature, the set of size-minimal solutions is independent from the rest of the signature. Therefore, since we are dealing with size-minimal solutions of basic MSOU problems, in the next sections, we will not specify the signature.

LEMMA 3.2. *For any MSOU problem $\langle \Sigma, E \rangle$, where $\Sigma_0 = \{\flat\}$, every size-minimal solution σ contains only constants that also occur in E or are equal to $\flat$.*

Proof. Suppose that a size-minimal solution of $\langle \Sigma, E \rangle$ introduces a constant a that does not occur in E. Then, we could generate a solution with a strictly smaller size by replacing all subterms of the form $(a\, s)$ by $\flat$. This would contradict minimality. □

4. Singleton context-free grammars. In this section we prove some properties of context-free grammars. They will be used to *compactly* represent solutions of MSOU problems. In particular, we will use singleton context-free grammars that define languages with just one word.

A *context-free grammar (CFG)* is a 4-tuple (Σ, N, P, s), where Σ is an alphabet of *terminal* symbols, N is an alphabet of *nonterminal* symbols (contrary to the standard conventions and in order to avoid confusion between free variables (unknowns) and nonterminal symbols, all terminal and nonterminal symbols are denoted by lowercase letters), P is a finite set of rules, and $s \in N$ is the *start symbol*. In fact, we will not distinguish any particular start symbol, and we will represent a CFG as a 3-tuple (Σ, N, P). Moreover, we will use Chomsky grammars with at most two symbols on the right-hand sides of the rules.

DEFINITION 4.1. *We say that a CFG $G = (\Sigma, N, P)$* generates *a word $v \in \Sigma^*$ if there exists a nonterminal symbol $a \in N$ such that v belongs to the language defined by (Σ, N, P, a). In such a case, we also say that a* generates *v.*

We say that a CFG is singleton *if it is in Chomsky normal form, i.e., the right-hand sides of the productions consist of words of length at most* 2*, it is not recursive, and there exists just one production for each nonterminal symbol. Then, every nonterminal symbol $a \in N$ generates just one word, denoted w_a, and we say that a* generates *w_a. In general, for any sequence $\alpha \in (\Sigma \cup N)^*$, $w_\alpha \in \Sigma^*$ denotes the word generated by α.*

Plandowski [17, 18] defines singleton grammars, but he calls them *grammars defining set of words*. Note that so-called straight-line programs are an equivalent device [7]. Plandowski proves the following result.

THEOREM 4.2 (see [18, Theorem 33]). *The word equivalence problem for singleton CFGs is defined as follows: Given a singleton grammar and two nonterminal symbols a and b, decide whether $w_a = w_b$. This problem can be solved in polynomial worst-case time in the size of the grammar.*

Recent work [15] claims that this can be done in cubic time.

For nonrecursive grammars we define their depth as follows. The usage of both size and depth of the grammar is necessary for a good estimation, since they reflect balancing conditions for a singleton grammar seen as a tree. Using only a single measure leads to unsatisfactory upper bounds (see Remark 1 in section 8).

DEFINITION 4.3. *Let $G = (\Sigma, N, P)$ be a nonrecursive CFG. For any terminal symbol $a \in \Sigma$ we define $depth(a) = 0$, and for any nonterminal symbol $a \in N$ we define*

$$depth(a) = \max\{depth(b) + 1 \mid a \to \alpha \in P,\ b \text{ occurs in } \alpha\}.$$

We define the depth *of G as $depth(G) = \max\{depth(a) \mid a \in N\}$.*

Given a Chomsky CFG G, we define the size *of G, noted $|G|$, as the number of its rules.*

We say that $G' = (\Sigma', N', P')$ is an extension of $G = (\Sigma, N, P)$, denoted as $G' \supseteq G$, if and only if $\Sigma' \supseteq \Sigma$, $N' \supseteq N$, and $P' \supseteq P$, where we require only $\Sigma' = \Sigma$. We can extend a singleton grammar in order to generate concatenation, exponentiation, and prefixes and suffixes of words already generated by the grammar. We use these extension operations in the next sections to build the grammar defining some solution of the unification problem. The following three lemmas state how the size and the depth of the grammar are increased with these transformations. Since in the final step of this paper a grammar of polynomial size is guessed and checked in polynomial time, we only need the existence of polynomial-sized grammars. Thus we do not care about the algorithmic complexity of constructing these grammars.

LEMMA 4.4 (concatenation). *Let G be a singleton grammar generating the words $v_1, \ldots, v_n$ for $n \geq 1$. Then there exists a singleton grammar $G' \supseteq G$ that generates*

the word $v_1 \dots v_n$ and satisfies

$$\begin{array}{lcl} |G'| & \leq & |G| + n - 1, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G) + \lceil \log n \rceil. \end{array}$$

Proof. Let a_i be the nonterminal symbol generating v_i, for any $i = 1, \dots, n$. We define G' by adding a set of rules to G of the form

$$b_{i,j} \to b_{i,\, \lfloor \frac{i+j}{2} \rfloor}\, b_{\lfloor \frac{i+j}{2} \rfloor + 1,\, j},$$

where $1 \leq i \leq j \leq n$ and $b_{i,i}$ is a_i. Then, $b_{1,n}$ generates $v_1 \dots v_n$, and to generate it we need only to add $n - 1$ of such rules. The depth is increased by at most $\lceil \log n \rceil$. □

LEMMA 4.5 (exponentiation). *Let G be a singleton grammar generating the word v. For any $n \geq 1$, there exists a singleton grammar $G' \supseteq G$ that generates the word v^n and satisfies*

$$\begin{array}{lcl} |G'| & \leq & |G| + 2 \lfloor \log n \rfloor, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G) + \lfloor \log n \rfloor + 1. \end{array}$$

Proof. Let a be the nonterminal symbol generating v, $m = \lfloor \log n \rfloor$, and let $n = k_0 2^0 + k_1 2^1 + \dots + k_m 2^m$ be a binary representation satisfying $k_i \in \{0, 1\}$. We add the following set of rules to G:

$$\begin{array}{lcl} a_1 & \to & a\, a, \\ a_2 & \to & a_1\, a_1, \\ & \dots & \\ a_m & \to & a_{m-1}\, a_{m-1}, \end{array} \qquad \begin{array}{lcl} b_0 & \to & \begin{cases} a & \text{if } k_0 = 1, \\ \varepsilon & \text{if } k_0 = 0, \end{cases} \\ b_1 & \to & \begin{cases} a_1\, b_0 & \text{if } k_1 = 1, \\ b_0 & \text{if } k_1 = 0, \end{cases} \\ & \dots & \\ b_m & \to & \begin{cases} a_m\, b_{m-1} & \text{if } k_m = 1, \\ b_{m-1} & \text{if } k_m = 0. \end{cases} \end{array}$$

Then, the nonterminal symbol b_m generates v^n, and it is easy to see that this grammar satisfies the bounds stated by the lemma. □

LEMMA 4.6 (prefixes and suffixes). *Let G be a singleton grammar generating the word v. For any prefix or suffix v' of v, there exists a singleton grammar $G' \supseteq G$ that generates v' and satisfies*

$$\begin{array}{lcl} |G'| & \leq & |G| + \mathit{depth}(G), \\ \mathit{depth}(G') & = & \mathit{depth}(G). \end{array}$$

Proof. Let a be the nonterminal symbol generating v. By induction on $\mathit{depth}(a)$, we will prove a stronger result: For any prefix v' of w_a, there exists a grammar G' generating v' and satisfying $|G'| \leq |G| + \mathit{depth}(a)$ and $\mathit{depth}(G') = \mathit{depth}(G)$.

The base case is trivial since $\mathit{depth}(a) = 0$ implies that a is a terminal symbol, and $v' = v$ or v' is empty. For the induction case, assume that $v' \neq v$; otherwise we are done. Let $a \to \alpha$ be the rule for a. Note that $|\alpha| \leq 2$. There exists a prefix $\beta\, b$ of α, where b is a nonterminal, such that w_β is a prefix of v' and v' is a prefix of $w_{\beta\, b}$; i.e., $v' = w_\beta\, v''$, where v'' is a prefix of w_b. By induction hypothesis, there exists a grammar $G'' \supseteq G$ deriving v'' from some b' with the same depth as G and size $|G''| \leq |G| + \mathit{depth}(b) \leq |G| + \mathit{depth}(a) - 1$. We add $a' \to \beta\, b'$ to get the grammar G'

from G'' such that $w_{a'} = v'$. Notice that $|G'| = |G''| + 1$, and $depth(G') = depth(G'')$ because $depth(b') \leq depth(b)$ implies $depth(a') \leq depth(a)$.

For suffixes the proof is very similar. □

We illustrate Lemmas 4.4, 4.5, and 4.6 by means of Example 5.

Example 5. Let G be the grammar defined by the productions $\{c_1 \to c_2\, c_3$, $c_2 \to c_3\, c_4$, $c_3 \to ff$, $c_4 \to gg\}$. The words generated by the nonterminals c_1, c_2, c_3, and c_4 of G are $v_1 = ffggff$, $v_2 = ffgg$, $v_3 = ff$, and $v_4 = gg$, respectively.

(i) *Concatenation.* We first show how to build the word

$$v_1 v_2 v_3 v_4 = ffggffffggffgg$$

using the techniques of the proof of Lemma 4.4. Following the definitions of $b_{i,j} \to b_{i,\lfloor \frac{i+j}{2} \rfloor}\, b_{\lfloor \frac{i+j}{2} \rfloor + 1,\, j}$, and $b_{i,i} = c_i$, we extend G with the following rules:

$$\begin{array}{lcl} b_{1,4} & \to & b_{1,2}\, b_{3,4}, \\ b_{1,2} & \to & c_1\, c_2, \\ b_{3,4} & \to & c_3\, c_4. \end{array}$$

Then, $b_{1,4}$ generates $v_1 v_2 v_3 v_4$.

(ii) *Exponentiation.* Now we show how to build the word $(v_1)^5$ according to the techniques of the proof of Lemma 4.5. We have $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$; hence we extend G with the following rules:

$$\begin{array}{lcllcl} a_1 & \to & c_1\, c_1, & b_0 & \to & c_1, \\ a_2 & \to & a_1\, a_1, & b_1 & \to & b_0, \\ & & & b_2 & \to & a_2\, b_1. \end{array}$$

Then b_2 generates $(v_1)^5$.

(iii) *Prefix.* Finally we show how to build the word prefix $v' = ffg$ of v_1 using the techniques of the proof of Lemma 4.6. We extend G with the following rules:

$$\begin{array}{lcl} c'_1 & \to & c'_2, \\ c'_2 & \to & c_3\, c'_4, \\ c'_4 & \to & g. \end{array}$$

Then c'_1 generates v'.

5. Compact representations. In this section we use singleton grammars to compact the representation of solutions of basic MSOU problems. We go a step further and also compact the representation of equations, allowing the use of nonterminal symbols of a singleton grammar to represent large words also in the equations.

DEFINITION 5.1. *Let Σ be a signature of unary symbols, and let $\mathcal{X}$ be a set of unary variables.*

A compact representation *of a basic MSOU problem E is a pair $\langle E', G\rangle$, where $G = \langle \Sigma, N, P\rangle$ is a singleton CFG and E' is a set of equations of the form $\{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\}$, where $s_i, t_i \in (\Sigma \cup \mathcal{X} \cup N)^* \flat$, for $i = 1, \ldots, n$, such that when replacing in E' every nonterminal symbol a by the word w_a that it generates, it results in the set of equations E.*

A compact representation *of a monadic substitution σ is a pair $\langle \sigma', G\rangle$, where $G = \langle \Sigma, N, P\rangle$ is a singleton CFG and σ' is a mapping from variables X to terms of the form $\lambda x\,.\,\alpha\flat$ or $\lambda x\,.\,\alpha\, x$, where $\alpha \in (\Sigma \cup N)^*$, such that, after replacing every nonterminal symbol by the word it represents, we obtain σ.*

We say that $\langle \tau, G' \rangle$ *is a* compacted solution *of* $\langle E, G \rangle$ *if the substitution represented by* $\langle \tau, G' \rangle$ *is a solution of the set of equations represented by* $\langle E, G \rangle$*, where* G' *is an extension of* G*.*

Notice that nonterminal symbols derive into sequences of unary function symbols, that we do not consider first-order variables, and that $\flat$ is the only constant. Words of $(\Sigma \cup \mathcal{X} \cup N)^*$ are denoted by Greek letters $\alpha, \beta, \ldots$.

Example 6. Let $\Sigma = \{f\}$ and $N := \{a, c, d\}$. Consider the compacted equations $\langle E, G \rangle$ defined by

$$\begin{array}{rcl} E & = & \{a\, X\, X\, f\, \flat \stackrel{?}{=} Y\, Y\, Y\, \flat\}, \\ G & = & \{a \to c\, c,\ c \to f\, f\}. \end{array}$$

Then, the pairs $\langle \sigma_1, G_1 \rangle$ and $\langle \sigma_2, G_2 \rangle$, defined by

$$\begin{array}{rcl} \sigma_1 & = & [X \mapsto \lambda x\,.\, c\, x,\ Y \mapsto \lambda x\,.\, d\, x], \\ G_1 & = & \{a \to c\, c,\ c \to f\, f,\ d \to c\, f\} \end{array} \quad \text{and} \quad \begin{array}{rcl} \sigma_2 & = & [X \mapsto \lambda x\,.\, e\, \flat,\ Y \mapsto \lambda x\,.\, a\, \flat], \\ G_2 & = & \{a \to c\, c,\ c \to f\, f,\ e \to \varepsilon\}, \end{array}$$

are compacted representations of solutions of $\langle E, G \rangle$. The first solution is not size-minimal. The second solution is size-minimal, but it is not a most general unifier. In fact, the second is an instantiation of the most general unifiers $[X \mapsto \lambda x\,.\, Z\, x,\ Y \mapsto \lambda x\,.\, a\, Z\, Z\, f\, \flat]$ and $[X \mapsto \lambda x\,.\, Z\, a\, Z\, a\, Z\, \flat, Y \mapsto \lambda x\,.\, a\, Z\, x]$.

We generalize the basic MSOU problem in the sense that, given some compacted equations $\langle E, G \rangle$, we will try to find a compacted solution $\langle \sigma, G' \rangle$. Moreover, the grammar G' used to represent the solution will be an extension of the grammar G given to represent the equations.

Notice that solvability of a set of monadic equations and solvability of compact equations are, w.r.t. decidability, equivalent problems. With respect to their complexity, we will prove that solvability of compact equations can be decided in NP-time. This implies that solvability of MSOU is also in NP (since $\langle E, \emptyset \rangle$ is a trivial compact representation of E).

Notice that the straightforward translation of a compacted set of equations into the set of monadic equations that they represent may exponentially increase the size of the equations. Using another translation, we can show that solvability of MSOU problems and solvability of compacted MSOU problems are polynomially equivalent.

PROPOSITION 5.2. *Given a compacted set of equations* $\langle E, G \rangle$*, there is a P-time translation into a basic MSOU problem* E'*, such that* $\langle E, G \rangle$ *is solvable if and only if* E' *is solvable.*

Proof. For every nonterminal a in G, define a fresh unary variable X_a. For every production $a \to b\, c$ of the grammar, where $a, b, c \in N$, define a set of two equations $E_a = \{X_a\, \flat \stackrel{?}{=} X_b\, X_c\, \flat,\ X_a\, f\, \flat \stackrel{?}{=} X_b\, X_c\, f\, \flat\}$, where $f \in \Sigma$ and $X_a, X_b, \ldots$ are fresh variables. This is similar for the other kinds of rules, where the right-hand sides are shorter. The terminal symbols are not translated. For instance, for $a \to b\, c$, where $a \in N$ and $b, c \in \Sigma$, $E_a = \{X_a\, \flat \stackrel{?}{=} b\, c\, \flat, X_a\, f\, \flat \stackrel{?}{=} b\, c\, f\, \flat\}$. Then, $E' = \hat{E} \cup \bigcup_{a \in N} E_a$, where $\hat{E}$ is the translation of the equations E by replacing nonterminals a with the corresponding unary variable X_a. The size of E' is smaller than $|E| + 12\,|G|$, and it can be constructed in polynomial time.

The equations in $\bigcup_{a \in N} E_a$ enforce that, for every nonterminal a, $\sigma(X_a)$ uses its argument, and therefore $\sigma(X_a) = \lambda x\,.\, w_a\, x$. Now it is easy to see that $\langle E, G \rangle$ is solvable if and only if E' is solvable. □

6. The graph of surface dependencies. In this section we define graphs of surface dependencies. The purpose of these graphs is to support constructing the compact representation of a minimal solution σ of a compacted set of equations and estimating the size of this representation. Later on this will be used to show that only a polynomial-sized representation has to be guessed in order to check solvability. Observe that we only impose a bound on the size of the representation and do not care about the complexity of finding such a representation. In our proof, we start from the compact representation $\langle E, G\rangle$ of some basic MSOU problem and a given solution σ. Then, we find a variable X whose instantiation can be compactly represented. This is done by extending the grammar to $G' \supseteq G$. Then, we repeat the process starting from the same equation with the variable already instantiated. Observe that G' (apart from the instantiation of the X) is able to generate all the words represented by G. This iteration describes a proof by induction, not the unification algorithm. It could be interpreted as a nondeterministic unification procedure, however, with the restriction of finding a size-minimal solution, and moreover, without a guarantee of being in NP.

There are cases in which for some variable X of the problem, its instantiation $\sigma(X)$ is immediately given or immediately constructible from the "surface" of the equations. We identify two such cases: when σ has a *small component* (Lemma 6.6) and when the graph contains a *cycle* (Lemma 6.12). We also identify three situations which ensure that any size-minimal solution has small components: when the graph has a constant equation (Lemma 6.7), when the graph has no edges (Lemma 6.8), and when there are strong divergences (Lemma 6.10). In the rest of the cases, it becomes necessary to rewrite the graph to obtain a new graph that describes the instantiation of some variable. This graph rewriting process will be described in section 7.

The graph of surface dependencies is defined only for *simplified* equations, where a simplified equation is defined as follows.

DEFINITION 6.1. *Given a compacted set of equations $\langle E, G\rangle$, we say that they are* simplified *if E does not contain equations of the following forms (symmetric cases omitted):*

(i) $a\,s \stackrel{?}{=} b\,t$, *where* $a, b \in \Sigma \cup N$,
(ii) $s \stackrel{?}{=} a\,b\,t$, *where* $a, b \in \Sigma \cup N$, *or*
(iii) $s \stackrel{?}{=} a\,t$, *where* $a \in N$ *and* $w_a = \varepsilon$.

Note that simplified equations are of the forms (symmetric cases omitted) $X\,s \stackrel{?}{=} a\,\flat$, $X\,s \stackrel{?}{=} a\,Y\,t$, $X\,s \stackrel{?}{=} \flat$ (flexible-rigid), or $X\,s \stackrel{?}{=} Y\,t$ (flexible-flexible), where a is a nonterminal. Note also that solvable equations without variables have the form $\alpha\,\flat \stackrel{?}{=} \beta\,\flat$, where α, β are in $(\Sigma \cup N)^*$, satisfy $w_\alpha = w_\beta$, and are not simplified.

We describe a simplification algorithm in the proof of the following lemma. This algorithm will be used as a subroutine in the proof of Theorem 7.7. Notice that it can increase the size of the associated grammar as stated in the lemma.

LEMMA 6.2 (simplification). *Given the solvable and compacted set of equations $\langle E, G\rangle$, there exists a simplified and compacted set of equations $\langle E', G'\rangle$ with the same solutions, such that*

$$\begin{aligned} |G'| &= |G| + \mathcal{O}(|E|(\mathit{depth}(G) + \log|E|)), \\ \mathit{depth}(G') &= \mathit{depth}(G) + \mathcal{O}(\log|E|), \end{aligned}$$

and the number of equations, number of variables, and number of occurrences of variables in E' are not greater than the corresponding numbers for E.

Proof. For all equations $s_i \stackrel{?}{=} t_i$ of E, let α_i, β_i be the longest prefixes of s_i and t_i, respectively, that do not contain variables or $\flat$; i.e., $s_i = \alpha_i\,s'_i$, and $t_i = \beta_i\,t'_i$, where

s'_i and t'_i have a variable or $\flat$ in the head. Let v_i be the word satisfying $w_{\alpha_i} = w_{\beta_i} v_i$ or $w_{\alpha_i} v_i = w_{\beta_i}$.

Using Lemma 4.4, construct an extension of the grammar with a nonterminal for the word $\alpha_1 \beta_1 \dots \alpha_n \beta_n$. For every $i = 1, \dots, n$, since v_i is a suffix of some prefix of this word, use Lemma 4.6 to prove that there exists another extension of the grammar that generates v_i. This last process will be repeated at most $2\,\#Eq(E)$ times to obtain a new grammar G'. This ensures that $depth(G') \leq depth(G) + \lceil \log |E| \rceil$ and $|G'| \leq |G| + |E| + 2\,\#Eq(E)\,(depth(G) + \lceil \log |E| \rceil)$. This implies the estimations given in the lemma.

Now, we construct E' from E as follows. For every $i = 1, \dots, n$,

(i) if $s'_i = t'_i = \flat$, then remove the equation from E;

(ii) if $w_{\alpha_i} = w_{\beta_i}$, then replace $s_i \stackrel{?}{=} t_i$ in E by $s'_i \stackrel{?}{=} t'_i$;

(iii) if w_{α_i} is a prefix of w_{β_i}, then replace $s \stackrel{?}{=} t$ in E by $s'_i \stackrel{?}{=} b\,t'_i$, where b is the nonterminal of G' generating v_i; and

(iv) proceed similarly if w_{β_i} is a prefix of w_{α_i}.

Notice that the case where neither w_{α_i} is a prefix of w_{β_i} nor w_{β_i} is a prefix of w_{α_i} is not possible for solvable equations. Notice also that, if we simplify equations one by one, generating a suffix of a prefix of α_i or of β_i each time, we would get a worse estimation. □

DEFINITION 6.3. *A* constant equation *is a simplified and compacted equation of the form* $X\,t \stackrel{?}{=} a\flat$ *or* $X\,t \stackrel{?}{=} \flat$, *where* $t \in (\Sigma \cup \mathcal{X} \cup N)^* \flat$ *is a compacted term and* $a \in N$ *is a nonterminal symbol.*

We define the graph of surface dependencies only for solvable, simplified, and compacted sets of equations.

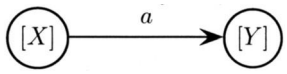
DEFINITION 6.4. *Let* $\langle E, G \rangle$ *be a solvable, simplified, and compacted set of equations. Let* $\approx$ *be the minimal equivalence relation satisfying* $X \approx Y$ *whenever* E *contains an equation of the form* $X\,s \stackrel{?}{=} Y\,t$. *This defines a partition on* $FV(E)$.

The graph of surface dependencies *of* $\langle E, G \rangle$ *is a labeled directed multigraph*[2] *defined as follows:*

Nodes: *The nodes are the* $\approx$*-equivalence classes of variables and the empty set* $\emptyset$.

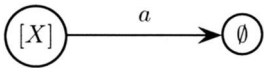
Edges: *There are two cases:*

(i) *For every equation of the form* $X\,s \stackrel{?}{=} a\,Y\,t$, *where* $X, Y \in \mathcal{X}$ *are variables and* $a \in N$ *is a nonterminal symbol, there is an edge*



Simplification yields that $w_a \neq \varepsilon$ *for the edge label* a.

(ii) *For every constant equation* $X\,s \stackrel{?}{=} a\flat$, *where* $X \in \mathcal{X}$ *and* $a \in N$, *there is an edge*



In the case of a constant equation $X\,s \stackrel{?}{=} \flat$, *we use* ε *as the label of the edge.*

The size *of a graph of surface dependencies* D, *denoted as* $|D|$, *is defined as its number of edges.*

Note that the node $\emptyset$ has no outgoing edges.

[2]There may be several edges, even labeled differently between two nodes.

6.1. Small components of solutions. We say that a solution of $\langle E, G\rangle$ has a small component if there exists a variable whose value is "small" enough to be described just as the prefix of some word defined by G. This will be helpful in the construction of a compact representation of a size-minimal solution for several reasons: It is a case where the instantiation of a variable is completely known, it can be constructed with only a small increase of the grammar, and it eases the definition and argumentation for the remaining cases.

We identify some classes of compacted equations that have a solution with a small component.

DEFINITION 6.5. *Given the simplified and compacted set of equations $\langle E, G\rangle$, a variable X occurring in E, and a solution σ, we say that X is a* small component *of σ if $\sigma(X) = \lambda x\,.\,v\,x$, or $\sigma(X) = \lambda x\,.\,v\,\flat$ and either $v = \varepsilon$ or there is a nonterminal a in G such that v is a prefix of w_a.*

LEMMA 6.6 (small component). *Let $\langle E, G\rangle$ be a simplified and compacted set of equations, and let σ be a solution of $\langle E, G\rangle$ with a small component X such that $\sigma(X) = \lambda x\,.\,v\,x$ or $\sigma(X) = \lambda x\,.\,v\,\flat$. Then, there exists a singleton CFG $G' \supseteq G$ generating v and satisfying*

$$\begin{aligned} |G'| &\leq |G| + \mathit{depth}(G), \\ \mathit{depth}(G') &= \mathit{depth}(G). \end{aligned}$$

Proof. The inequalities follow from Definition 6.5 and Lemma 4.6. □

This lemma is helpful in two ways: It allows us to eliminate variables from the problem and it restricts the cases where this elimination does not work (and it is necessary to rewrite the equations) to cases in which solutions do not have small components; i.e., they have only "large" instantiations. This will simplify the reasoning.

LEMMA 6.7. *All solutions of simplified and compacted sets of equations containing constant equations have at least one small component.*

Proof. Let $\langle E, G\rangle$ be a simplified and compacted set of equations, and let σ be a solution. Since σ solves a constant equation of the form $X\,s \stackrel{?}{=} a\,\flat$, it must instantiate X either with $\lambda x\,.\,v\,x$ or with $\lambda x\,.\,v\,\flat$ for some prefix v of w_a. Similar arguments hold for equations of the form $X\,s \stackrel{?}{=} \flat$. □

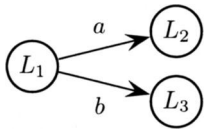
The following lemma describes the situation that reflects the difference between basic MSOU problems and WU when all equations are *flexible-flexible*. While in MSOU we can instantiate variables by terms not using their arguments, such as $\lambda x\,.\,\flat$, this is not possible when considering WU; hence the lemma does not hold for WU. This is the point that shows that our result is not straightforwardly transferable to WU.

LEMMA 6.8. *Let $\langle E, G\rangle$ be a simplified and compacted set of equations such that the graph of surface dependencies does not contain edges and such that $FV(E) \neq \emptyset$, and let σ be a size-minimal solution of $\langle E, G\rangle$. Then, for every variable $X \in FV(E)$, either $\sigma(X) = \lambda x\,.\,\flat$ or $\sigma(X) = \lambda x\,.\,x$. This also means that there is at least one small component in σ.*

Proof. If there are not edges, then all equations are of the form $X\,s \stackrel{?}{=} Y\,t$. The substitution σ, with $\sigma(X) = \lambda x\,.\,\flat$ for every variable $X \in FV(E)$, is a size-minimal solution; hence for every other size-minimal solution σ' and for every variable $X \in FV(E)$, only $\sigma'(X) = \lambda x\,.\,\flat$ or $\sigma'(X) = \lambda x\,.\,x$ is possible.[3] □

[3]We could make the solution with $\sigma'(X) = \lambda x\,.\,\flat$, for all variables, be the only size-minimal solution by changing the term size measure to make $\lambda x\,.\,\flat$ smaller than $\lambda x\,.\,x$. However, then the exponent of periodicity bound (see Lemma 2.4) must be adapted.

DEFINITION 6.9. *A dependence graph D is said to contain a* divergence $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$ *if it contains a subgraph of the following form (where L_1, L_2, and L_3 are not necessarily distinct nodes):*



If neither w_a is a prefix of w_b nor w_b is a prefix of w_a, then it is called a strong divergence *and otherwise a* weak divergence.

Strong divergences are easy to eliminate, whereas weak divergences require a more complex treatment, which will be done by rewriting the graph.

LEMMA 6.10. *Given a simplified and compacted set of equations, if its graph of surface dependencies contains a strong divergence, then every solution has small components.*

Proof. Given $\langle E, G\rangle$, let D be its graph of surface dependencies, and let σ be any of its solutions. Assume that D contains a strong divergence $L_2 \overset{a}{\leftarrow} L_1 \overset{b}{\rightarrow} L_3$. For every variable $X \in L_1$, let $v_X \in \Sigma^*$ be a word such that either $\sigma(X) = \lambda y\,.\,v_X\,y$ or $\sigma(X) = \lambda y\,.\,v_X\,\natural$. By definition of D, we have a pair of equations in E of the form $X_1 \cdots \overset{?}{=} a\,t_1$ and $X_2 \cdots \overset{?}{=} b\,t_2$, where $X_1, X_2 \in L_1$. Therefore, v_{X_1} is a prefix of $\sigma(w_a\,t_1)$, and v_{X_2} is a prefix of $\sigma(w_b\,t_2)$. Now, let $Y \in L_1$ be the variable such that v_Y is the shortest word of $\{v_X \mid X \in L_1\}$. By the equivalence relation and the dependence graph definitions, we have that v_Y is a prefix of both v_{X_1} and v_{X_2}, and thus a prefix of $\sigma(w_a\,t_1)$ and of $\sigma(w_b\,t_2)$. Since w_a is not a prefix of w_b, and vice versa, we have that v_Y is a proper prefix of both w_a and w_b. (Notice that v_Y is not necessarily the longest common prefix of w_a and w_b). Therefore, Y is a small component of σ. □

6.2. Cycles in the graph of dependencies. The cycles in the graph of surface dependencies describe the base of *some* exponentiation occurring in the instantiation of some variables. For instance, the solutions of the equation $X\,f\,\natural \overset{?}{=} f\,X\,\natural$ have the form $[X \mapsto \lambda y\,.\,f^n\,y]$ for some $n \geq 0$. The base of this power is described by a cycle in its graph of surface dependencies:

LEMMA 6.11. *Let E be a (noncompacted) set of equations with a cycle of the form*

$$\begin{array}{lcl} X_1 \cdots & \overset{?}{=} & w_1\,X_2 \cdots \\ X_2 \cdots & \overset{?}{=} & w_2\,X_3 \cdots \\ & \cdots & \\ X_m \cdots & \overset{?}{=} & w_m\,X_1 \cdots, \end{array}$$

where $w_i \in \Sigma^*$, for every $i = 1, \ldots, m$, and $w_1 \ldots w_m \neq \varepsilon$.

Then, for every solution σ of E, there is some variable X_k, with $1 \leq k \leq m$, such that $\sigma(X_k) = \lambda x \,.\, u\, x$, where u is a prefix of $(w_k \dots w_m\, w_1 \dots w_{k-1})^n$ for sufficiently large n.

Proof. The proof is by induction on the size of σ.

For any $i = 1, \dots, m$, if σ solves $X_i \cdots \stackrel{?}{=} w_i\, X_{i+1} \cdots$, then either $\sigma(X_i) = \lambda x \,.\, u_i\, x$ for some proper prefix u_i of w_i, or we have $\sigma(X_i) = \lambda x \,.\, w_i\, v_i\, x$ or $\sigma(X_i) = \lambda x \,.\, w_i\, v_i\, \flat$, for some word v_i. Therefore, there are two cases:

If, for some $k = 1, \dots, m$, we have the first situation that there is some X_k such that $\sigma(X_k) = \lambda x \,.\, u_k\, x$, where u_k is a prefix of w_k, then the claim of the lemma holds.

Otherwise, for every $i = 1, \dots, m$, we have $\sigma(X_i) = \lambda x \,.\, w_i\, v_i\, x$ or $\sigma(X_i) = \lambda x . w_i\, v_i\, \flat$. In this case we generate a new system by instantiating X_i by $\lambda x \,.\, w_i\, X'_i\, x$, where X'_i are fresh and different unary second-order variables. Then, the equations in the cycle become

$$\begin{array}{lcl} w_1\, X'_1 \cdots & \stackrel{?}{=} & w_1\, w_2\, X'_2 \cdots, \\ w_2\, X'_2 \cdots & \stackrel{?}{=} & w_2\, w_3\, X'_3 \cdots, \\ & \cdots & \\ w_m\, X'_m \cdots & \stackrel{?}{=} & w_m\, w_1\, X'_1 \cdots. \end{array}$$

Simplifying the equations, we obtain a new system of equations,

$$\begin{array}{lcl} X'_1 \cdots & \stackrel{?}{=} & w_2\, X'_2 \cdots, \\ X'_2 \cdots & \stackrel{?}{=} & w_3\, X'_3 \cdots, \\ & \cdots & \\ X'_m \cdots & \stackrel{?}{=} & w_1\, X'_1 \cdots. \end{array}$$

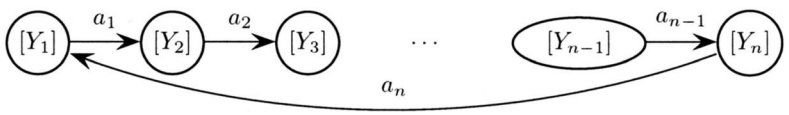
From the original solution σ we get a solution σ' of the new equations satisfying $\sigma'(X'_i) = \lambda x \,.\, v_i\, x$ or $\sigma(X'_i) = \lambda x \,.\, v_i\, \flat$, for all $i = 1, \dots, m$. Now the induction hypothesis applies since σ' is smaller than σ; hence there is a variable X'_k such that $\sigma'(X'_k) = \lambda x \,.\, v_k\, x$, where v_k is a prefix of $(w_{k+1} \dots w_m\, w_1 \dots w_k)^n$, for large enough n. Hence $\sigma(X_k) = \lambda x . w_k\, v_k\, x$, and the claim holds. □

LEMMA 6.12 (cycles). *Let $\langle E, G \rangle$ be a solvable, simplified, and compacted set of equations, with a graph of surface dependencies D with some cycle. Then, for every solution σ without small components, there exists a variable X such that $\sigma(X) = \lambda y \,.\, w\, y$ and w is generated by some grammar $G' \supseteq G$ satisfying*

$$\begin{array}{lcl} |G'| & = & |G| + \mathcal{O}(\mathit{depth}(G) + \#\mathit{Eq}(E) + \log \mathrm{eop}(\sigma)), \\ \mathit{depth}(G') & = & \mathit{depth}(G) + \mathcal{O}(\log \#\mathit{Eq}(E) + \log \mathrm{eop}(\sigma)). \end{array}$$

More precisely, the corresponding node $[X]$ is inside the cycle, and, for some $0 \leq n \leq \mathrm{eop}(\sigma)$ and some prefix v of w_α, we have $\sigma(X) = \lambda y \,.\, (w_\alpha)^n\, v\, y$, where $\alpha \in N^*$ is the sequence of labels of the edges completing the cycle from $[X]$.

Proof. Select a cycle in the graph D:



Therefore, there is a subset of equations in E of the form

$$\begin{array}{llll} Y_{1,m_1} \ldots \stackrel{?}{=} a_1\, Y_{2,1} \ldots; & Y_{2,1} \ldots \stackrel{?}{=} Y_{2,2} \ldots; & \cdots & Y_{2,m_2-1} \ldots \stackrel{?}{=} Y_{2,m_2} \ldots; \\ Y_{2,m_2} \ldots \stackrel{?}{=} a_2\, Y_{3,1} \ldots; & Y_{3,1} \ldots \stackrel{?}{=} Y_{3,2} \ldots; & \cdots & Y_{3,m_3-1} \ldots \stackrel{?}{=} Y_{3,m_3} \ldots; \\ \cdots & & & \\ Y_{n,m_n} \ldots \stackrel{?}{=} a_n\, Y_{1,1} \ldots; & Y_{1,1} \ldots \stackrel{?}{=} Y_{1,2} \ldots; & \cdots & Y_{1,m_1-1} \ldots \stackrel{?}{=} Y_{1,m_1} \ldots, \end{array}$$

where $\{Y_{i,1}, \ldots, Y_{i,m_i}\} \subseteq [Y_i]$ for $i = 1, \ldots, n$. Note that $w_{a_i} \neq \varepsilon$ for all i, since E is simplified.

Now, fix a solution σ and proceed as follows. Let E' be the set of equations represented by the compacted equations above. Notice that $w_{a_1} \cdots w_{a_n} \neq \varepsilon$, and E' fulfills the conditions of Lemma 6.11. The substitution σ also solves E', and applying Lemma 6.11, we get a variable $Y_{k,l}$ such that $\sigma(Y_{k,l}) = \lambda x.w_\alpha^n\, v\, x$, where $\alpha = a_k \cdots a_m\, a_1 \cdots a_{k-1}$ and v is a prefix of w_α. Moreover, we have $n \leq \mathrm{eop}(\sigma)$.

To prove the existence of G', we proceed by adding new rules to G. Note that all symbols labeling the edges of D are nonterminals in G. We construct a sequence of grammars $G \subseteq G_1 \subseteq G_2 \subseteq G_3 \subseteq G'$ such that G_1, apart from the words generated by G, also generates w_α, G_2 also generates v, G_3 also generates $(w_\alpha)^n$, and G' also generates $(w_\alpha)^n\, v$.

Since the length of α is at most $|D|$, by Lemma 4.4, we have

$$\begin{array}{lcl} |G_1| & \leq & |G| + |D| - 1, \\ \mathit{depth}(G_1) & \leq & \mathit{depth}(G) + \lceil \log |D| \rceil. \end{array}$$

By Lemma 4.6, we can define v with

$$\begin{array}{lcl} |G_2| & \leq & |G_1| + \mathit{depth}(G_1), \\ \mathit{depth}(G_2) & = & \mathit{depth}(G_1). \end{array}$$

By Lemma 4.5, we can define $(w_\alpha)^n$ with

$$\begin{array}{lcl} |G_3| & \leq & |G_2| + 2\, \lfloor \log \mathrm{eop}(\sigma) \rfloor, \\ \mathit{depth}(G_3) & \leq & \mathit{depth}(G_2) + \lceil \log \mathrm{eop}(\sigma) \rceil \end{array}$$

and, since we still need another rule to define $(w_\alpha)^n\, v$,

$$\begin{array}{lcl} |G'| & \leq & |G_3| + 1, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G_3) + 1. \end{array}$$

The composition of all these inequalities results in the inequalities

$$\begin{array}{lcl} |G'| & \leq & |G| + \mathit{depth}(G) + |D| + \lceil \log |D| \rceil + 2\, \lfloor \log \mathrm{eop}(\sigma) \rfloor, \\ \mathit{depth}(G') & \leq & \mathit{depth}(G) + \lceil \log |D| \rceil + \lceil \log \mathrm{eop}(\sigma) \rceil + 1. \end{array}$$

In terms of O-notation, this reduces to

$$\begin{array}{lcl} |G'| & = & |G| + \mathcal{O}(\mathit{depth}(G) + |D| + \log \mathrm{eop}(\sigma)), \\ \mathit{depth}(G') & = & \mathit{depth}(G) + \mathcal{O}(\log |D| + \log \mathrm{eop}(\sigma)). \end{array}$$

And, since $|D| \leq \#Eq(E)$,

$$\begin{array}{lcl} |G'| & = & |G| + \mathcal{O}(\mathit{depth}(G) + \#Eq(E) + \log \mathrm{eop}(\sigma)), \\ \mathit{depth}(G') & = & \mathit{depth}(G) + \mathcal{O}(\log \#Eq(E) + \log \mathrm{eop}(\sigma)), \end{array}$$

as stated in the lemma. □

7. Rewriting the graph of dependencies. In the previous section we saw that Lemmas 6.6 and 6.12 both describe the instantiation $\sigma(X)$ of some variable and allow us to eliminate it during the construction of a compact representation of the solution σ. In other words, they describe parts of the solution, i.e., a substitution $\rho = [X \mapsto \sigma(X)]$ satisfying $\rho \preceq_{FV(E)} \sigma$.

In this section we will see that, when these two lemmas are not applicable, we can *rewrite* the set of equations (and its corresponding graph of dependencies) until one of the two lemmas becomes applicable. This rewriting process is done by *partially* instantiating some variables, i.e., applying a substitution ρ also satisfying $\rho \preceq_{FV(E)} \sigma$. The substitution has the form $\rho = [X \mapsto \lambda y \,.\, w\, X'\, y]$, where X' is a fresh variable and $w \in \Sigma^*$. Substitutions of such form, as well as the total instantiations of the form $\rho = [X \mapsto \lambda y \,.\, w\, y]$ and $\rho = [X \mapsto \lambda y \,.\, w\, \flat]$ described in Lemmas 6.6 and 6.12, are all called *partial instantiations*. Formally, we define partial instantiations as follows.

DEFINITION 7.1. *We say that a substitution ρ is a* partial instantiation *if it can be decomposed as $\rho = (\rho_1 \circ \cdots \circ \rho_n)|_{Dom(\rho)}$, where each ρ_i either has the form $\rho_i = [X_i \mapsto \lambda y \,.\, w_i\, X'_i\, y]$, $\rho_i = [X_i \mapsto \lambda y \,.\, y]$, or $\rho_i = [X_i \mapsto \lambda y \,.\, \flat]$, for some $X_i, X'_i \in \mathcal{X}_1$ and some $w_i \in \Sigma^*$.*

The following lemma states the preservation of some properties of partial instantiations of equations. Notice that some of these properties do not hold for arbitrary substitutions satisfying $\rho \preceq_{FV(E)} \sigma$.

LEMMA 7.2 (preservation). *For any (noncompacted) set of equations E, any solution σ, and any partial instantiation ρ, satisfying $\rho \preceq_{FV(E)} \sigma$, there exists a substitution σ' satisfying*

(i) $\sigma = (\sigma' \circ \rho)|_{FV(E)}$,
(ii) *σ' is a solution of $\rho(E)$,*
(iii) *if σ is a size-minimal solution of E, then σ' is also a size-minimal solution of $\rho(E)$,*
(iv) $\mathrm{eop}(\sigma) \geq \mathrm{eop}(\sigma')$,
(v) $|\,FV(E)| \geq |\,FV(\rho(E))|$, *and*
(vi) *the number of occurrences of variables in E is greater than or equal to the number of occurrences of variables in $\rho(E)$.*

Proof. The requirement $\rho \preceq_{FV(E)} \sigma$ ensures that there exists a substitution σ' such that $\sigma(X) = \sigma' \circ \rho(X)$ for any $X \in FV(E)$. The restriction of σ' to the domain $FV(\rho(E))$ also satisfies this property. Therefore, the required σ' always exists.

Since σ is a solution of E, for any variable $X \in FV(E)$, $\sigma(X)$ is a closed term. Moreover, since $\sigma = (\sigma' \circ \rho)|_{FV(E)}$, for any variable $X \in FV(E)$, $\sigma(X) = \sigma' \circ \rho(X)$. The same applies to any term containing only variables of E, hence to any side of any equation of E. Therefore, for any equation $\rho(s) \stackrel{?}{=} \rho(t)$ of $\rho(E)$, we have $\sigma'(\rho(s)) = \sigma(s) = \sigma(t) = \sigma'(\rho(t))$. Hence, σ' solves $\rho(E)$.

Minimality is proved by contradiction. Assume that σ' is not size-minimal. Let τ' be a size-minimal solution of $\rho(E)$. Obviously, $\tau' \circ \rho$ is a solution of E. Since τ' is size-smaller than σ', we have $\sum_{X \in FV(\rho(E))} |\tau'(X)| < \sum_{X \in FV(\rho(E))} |\sigma'(X)|$. Now, since ρ is a partial instantiation, we have

$$\sum_{X \in FV(E)} |\tau'(\rho(X))| = \sum_{X \in FV(E)} |\rho(X)| + \sum_{Y \in FV(\rho(E))} (|\tau'(Y)| - 1) \quad \text{and}$$

$$\sum_{X \in FV(E)} |\sigma'(\rho(X))| = \sum_{X \in FV(E)} |\rho(X)| + \sum_{Y \in FV(\rho(E))} (|\sigma'(Y)| - 1).$$

Therefore, $\sum_{X\in FV(E)} |\tau'(\rho(X))| < \sum_{X\in FV(E)} |\sigma'(\rho(X))| = \sum_{X\in FV(E)} \sigma(X)$, which contradicts the assumption that σ is size-minimal.

Let $X' \in Dom(\sigma')$ be a variable, and let v be a nonempty word, such that $\sigma'(X') = \lambda y\,.\,u\,v^{\mathrm{eop}(\sigma')}\,w\,\flat$ (or similarly replacing $\flat$ by y). Since $Dom(\sigma') = FV(\rho(E))$, either $X' \in FV(E)$ or, for some variable $Y \in FV(E)$, $\rho(Y)$ contains X. Hence, in both cases, there exists a variable $X \in FV(E) \subseteq Dom(\sigma)$ such that $\sigma(X)$ contains $v^{\mathrm{eop}(\sigma')}$. Therefore $\mathrm{eop}(\sigma)$ is at least $\mathrm{eop}(\sigma')$.

The requirement that ρ is a partial instantiation ensures that after applying this substitution the number of variables and the number of occurrences of variables in E do not increase. □

Now we deal with the following case: There are compacted sets of equations whose graph of surface dependencies does not have any cycle, and the focused solution does not have any small component; i.e., we deal with the case not covered by Lemmas 6.6 and 6.12. Since there are no cycles, the edges define a partial order $\succ$ on the nodes, with $N \succ N'$ if and only if $N \to N'$ is an edge. Hence we can speak of $\succ$-maximal nodes. Since there are size-minimal solutions without small components, Lemma 6.8 shows that these graphs contain at least one edge, Lemma 6.10 states that they do not contain strong divergences, and Lemma 6.7 shows that they do not contain any edge to the node labeled with $\emptyset$. There is a $\succ$-maximal node with at least one outgoing edge to other nodes and without any strong divergence. We will transform the equations, whose graph of dependencies contains such maximal nodes, in order to obtain a description of some variable instantiation. In fact, this transformation on the equations carries over to a graph transformation. An example of this graph transformation (or rewriting) is shown in Example 7.

DEFINITION 7.3. *Let $\langle E, G\rangle$ with $FV(E) \neq \emptyset$ be a simplified and compacted set of equations without cycles such that there is a solution without small components. Then the transformation rule $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$ is defined as follows.*

Let D be the graph of surface dependencies of $\langle E, G\rangle$. Let $[X]$ be a $\succ$-maximal node in D with at least one outgoing edge. Let $\{a_1, \ldots, a_m\}$ be the set of all labels of the outgoing edges of $[X]$, where w_{a_1} is a prefix of w_{a_i}, for all $i = 1, \ldots, m$. For every $Y \in [X]$, let Y' be a fresh unary variable, and let ρ be the substitution that maps each $Y \in [X]$ to $\lambda y\,.\,w_{a_1}\,Y'\,y$. Then let $\langle E', G'\rangle$ be the simplification of $\langle \rho(E), G\rangle$.

If D' is the graph of surface dependencies of $\langle E', G'\rangle$, we write $D \Rightarrow D'$.

Notice that in the previous definition, since there are not any cycles and there are solutions without small components, by Lemma 6.8 there are edges, and hence there is a $\succ$-maximal node with some outgoing edge; by Lemmas 6.7 and 6.10 there are neither constant equations nor strong divergences. This allows us to assume that w_{a_1} is a prefix of w_{a_i} for $i = 1, \ldots, m$. Notice also that for all equations of the form $Y\,s \stackrel{?}{=} a\,Z\,t$, where $Y \in [X]$, the symbol a is the label of some outgoing edge of $[X]$. Finally, notice that the substitution ρ is a partial instantiation; hence Lemma 7.2 applies.

The transformation of $\langle E, G\rangle$ results in the compacted equations $\langle E', G'\rangle$ satisfying the following:

(i) The grammar G' is an extension of G such that for $i = 1, \ldots, m$, the nonterminal b_i generates the word v_i, which is defined by $w_{a_i} = w_{a_1}\,v_i$.

(ii) The set of equations E' is obtained from E by replacing all equations of the form $Y\,s \stackrel{?}{=} Z\,t$, where $Y, Z \in [X]$, by the equation $Y'\,\rho(s) \stackrel{?}{=} Z'\,\rho(t)$, and replacing every equation of the form $Y\,s \stackrel{?}{=} a_i\,Z\,t$, where $Y \in [X]$, by $Y'\,\rho(s) \stackrel{?}{=} b_i\,Z\,\rho(t)$.

Every solution σ without small components of E can be transformed into a solution of E' by defining it on the fresh variables Y' (see Lemma 7.5).

In the special case that $m = 2$, $w_{a_1} \neq w_{a_2}$, and $w_{a_1} \prec w_{a_2}$, the transformation on the graph of surface dependencies can be represented as a graph rewriting rule

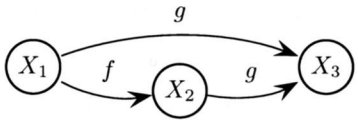
where b_2 is a new nonterminal of the grammar G', that generates a word satisfying $w_{a_2} = w_{a_1}\, w_{b_2}$.

Notice that in this rewriting process at least one arrow and also the node $[X]$ are removed. Note also that in the case $w_{a_1} = w_{a_2}$, the three nodes are merged into one node $[X'] \cup [Z_1] \cup [Z_2]$, thus removing more than one edge.

Example 7. Consider the following simplified and compacted set of equations and their set of solution components for $n \geq 0$:

$$\begin{array}{l} \Sigma = \{a, b, c, d, e\}, \\ N = \{f, g\}, \\ X_1\, c\flat \stackrel{?}{=} f\, X_2\, c\flat, \\ X_1\, d\flat \stackrel{?}{=} g\, X_3\, a\, b\, d\flat, \\ X_2\, e\flat \stackrel{?}{=} g\, X_3\, e\flat, \\ G\text{-rules: } \{f \to a\, b,\ g \to a\}. \end{array} \qquad \begin{array}{lcl} X_1 & \mapsto & \lambda x\,.\,(a\, b)^{n+2}\, x, \\ X_2 & \mapsto & \lambda x\,.\,(a\, b)^{n+1}\, x, \\ X_3 & \mapsto & \lambda x\,.\,b\,(a\, b)^{n}\, x, \end{array}$$

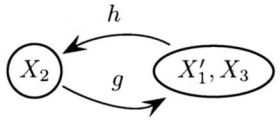
The graph of surface dependencies is



Applying the transformation rule to the only $\succ$-maximal node $[X_1]$, we get the following simplified and compacted set of equations E':

$$\begin{array}{l} \Sigma = \{a, b, c, d, e\}, \\ N' = \{f, g, h\}, \\ X_1'\, c\flat \stackrel{?}{=} h\, X_2\, c\flat, \\ X_1'\, d\flat \stackrel{?}{=} X_3\, a\, b\, d\flat, \\ X_2\, e\flat \stackrel{?}{=} g\, X_3\, e\flat, \\ G'\text{-rules: } \{f \to a\, b,\ g \to a,\ h \to b\}. \end{array}$$

The modified graph is as follows:



LEMMA 7.4 (rewriting). *Let $\langle E, G\rangle$ be a simplified and compacted set of equations, and let $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$; then*

$$\begin{array}{lcl} |G'| & \leq & |G| + \#Eq(E)\ depth(G), \\ depth(G') & = & depth(G). \end{array}$$

Moreover, $|E'| \leq |E| + m$*, where* m *is the number of variable occurrences of* E*.*

Proof. Let m be the number of outgoing edges of the removed node. The new grammar G' extends G by defining $m-1$ suffixes of words defined by G. Therefore, according to Lemma 4.6, we can obtain such a grammar G' satisfying the upper bounds:

$$\begin{array}{lcl} |G'| & \leq & |G| + (m-1)\, depth(G), \\ depth(G') & = & depth(G). \end{array}$$

It is easy to check that $m \leq \#Eq(E)$.

Finally E' is obtained by replacing in E the variable occurrences of some variables Y (belonging to $[X]$) by $b\,Y'$ and then simplifying. This simplification only has to remove the same nonterminal symbol from the head of both sides of some equations; hence it does not increase the size of the grammar. Therefore, the rewriting increases the size of E at most by 1 for each variable occurrence in E. □

LEMMA 7.5. *For any simplified and compacted set of equations* $\langle E, G\rangle$ *without cycles in its graph of surface dependencies, any solution* σ *without small components, and any transformation* $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$ *defined by the substitution* ρ*, there exists a substitution* σ' *such that*

(i) σ' *is a solution of* $\langle E', G'\rangle$*,*

(ii) σ' *satisfies* $\sigma = (\sigma' \circ \rho)|_{FV(E)}$*.*

Proof. If there are not any small components, then the substitution ρ satisfies $\rho \preceq_{FV(E)} \sigma$. Then the lemma is a direct consequence of Lemma 7.2. □

The previous lemma may be iterated: If σ' does not contain small components, and the graph of surface dependencies of $\langle E', G'\rangle$ does not contain cycles, we use it again to obtain a new solution σ'' of a new $\langle E'', G''\rangle$ and so on. By Lemma 7.6, this process cannot be repeated more than $\#Eq(E)$ times.

LEMMA 7.6. *Any graph rewriting sequence* $D \Rightarrow^* D'$ *has length at most* $|D|$*.*

Proof. This is clear, since in every transformation step at least one edge is removed. □

In Figure 7.1 we define an algorithm that, given the compacted set of equations $\langle E, G\rangle$ and a size-minimal solution σ, computes a polynomial-sized compacted solution $\langle \rho, G'\rangle$ representing σ. Note that the complexity of this algorithm is irrelevant; only the polynomial size of the obtained representation will be needed.

THEOREM 7.7 (compacted solution). *Given the initial compacted set of equations* $\langle E_0, G_0\rangle$ *and the size-minimal solution* σ_0*, the algorithm of Figure* 7.1 *computes a compacted solution* $\langle \rho, G'\rangle$ *representing* σ_0*.*

Moreover, the following inequalities hold:

$$\begin{array}{lcl} |G'| & \leq & |G_0| + \mathcal{O}(|E_0|^4\ depth(G_0) + |E_0|^6), \\ depth(G') & \leq & depth(G_0) + \mathcal{O}(|E_0|^2), \\ |\rho| & = & \mathcal{O}(|E_0|^3). \end{array}$$

Proof. The algorithm performs a sequence of transformations on the compacted equations $\langle E, G\rangle$, the compacted substitution $\langle \rho, G\rangle$, and the solution σ. These transformations are of four types: simplification (step 6), small components (step 9), cycles (step 16), and rewriting (step 23). First we show how many transformations of each type are performed and then how they modify some of the measures of the representations (number of equations, their size, etc.).

(i) *Termination*: Cycle and small component transformations remove a variable from E; therefore they cannot be executed more than $|E_0|$ times. According

Input: σ_0, $\langle E_0, G_0\rangle$
Output: ρ, G'

1. $\rho := Id$
2. $\langle E, G\rangle := \langle E_0, G_0\rangle$
3. $\sigma := \sigma_0$
4. while $FV(E) \neq \emptyset$ do
5. if $\langle E, G\rangle$ is not simplified then
6. let $\langle E', G'\rangle$ be the simplification of $\langle E, G\rangle$
7. $\langle E, G\rangle := \langle E', G'\rangle$
8. elseif σ has a small component X then
9. let G' be the grammar described in Lemma 6.6, and
10. let a be the nonterminal of G' generating v
11. if $\sigma(X) = \lambda x\,.\,v\,x$ then $\rho := [X \mapsto \lambda x\,.\,a\,x] \circ \rho$
12. if $\sigma(X) = \lambda x\,.\,v\,\flat$ then $\rho := [X \mapsto \lambda x\,.\,a\,\flat] \circ \rho$
13. $\langle E, G\rangle := \langle \rho(E), G'\rangle$
14. $\sigma = \sigma|_{FV(E)}$
15. elseif D contains a cycle then
16. let G' be the grammar,
17. let X be the variable in the cycle described in Lemma 6.12, and
18. let a be the nonterminal generating $(w_\alpha)^n\,v$, where $\sigma(X) = \lambda y\,.\,(w_\alpha)^n\,v\,y$
19. $\rho := [X \mapsto \lambda x\,.\,a\,x] \circ \rho$
20. $\langle E, G\rangle := \langle \rho(E), G'\rangle$
21. $\sigma = \sigma|_{FV(E)}$
22. else
23. compute $\langle E, G\rangle \Rightarrow \langle E', G'\rangle$
24. let $[X]$ be the $\succ$-maximal class transformed by this rewriting, and
25. let a_1 be the label of the outgoing edge of $[X]$ generating the shortest word
26. $\tau := Id$
27. for all $Y \in [X]$
28. $\tau := [Y \mapsto \lambda y\,.\,a_1\,Y'\,y] \circ \tau$
29. let σ' be the solution of $\langle E', G'\rangle$ satisfying $\sigma = (\sigma' \circ \tau)|_{FV(E)}$ given by Lemma 7.5
30. $\rho = \tau \circ \rho$
31. $\langle E, G\rangle := \langle E', G'\rangle$
32. $\sigma := \sigma'$
33. endwhile
34. $\rho := \rho|_{FV(E_0)}$

FIG. 7.1. *Pseudocode of the algorithm to compute a representation ρ of a solution.*

to Lemma 7.6, rewriting sequences cannot be longer than $|D|$. The size $|D|$ of the graph of surface dependencies is bounded by the number of equations (we will see in the following that this measure is decreasing), hence by $|E_0|$. After every rewriting sequence we get a set of equations E with a cycle, or a solution σ with a small component. Therefore, there is a total number of at most $|E_0|^2$ rewriting steps. Finally, after every rewriting step we get a simplified set of equations. Therefore, we cannot perform more simplification steps than cycle elimination plus small component elimination steps, hence not more than $|E_0|$.

(ii) *Number of equations, variables, and occurrences of variables*: Simplifications preserve or decrease all these parameters, as well as the partial instantiations performed by the cycle and small component transformations, according to Lemma 7.2. Rewritings are composed by a partial instantiation followed by a simplification; therefore they also preserve or decrease these parameters.

(iii) *Size of the equations,* $|E|$: The size of E is preserved or decreases with simplifications, cycles, and small component steps. However, it can be increased in the number of occurrences of variables at every rewriting step (see Lemma 7.4). Since there are no more than $|E_0|^2$ rewriting steps and the increase is bounded by $|E_0|$, we have $|E| \leq |E_0| + |E_0|^3$ along the execution of the algorithm.

(iv) *Exponent of periodicity,* $\mathrm{eop}(\sigma)$: According to Lemma 7.2, the exponent of periodicity of the solution, after a partial instantiation like the ones we perform in the cycle, small component and rewriting steps, is preserved or decreases. Since by Lemma 2.4, $\mathrm{eop}(\sigma_0) \leq 2^{\alpha\,|E_0|}$ for the initial minimal solution σ_0, this bound holds along the execution of the algorithm.

(v) *Depth of the grammar,* $depth(G)$: There are the following possibilities:

$$\begin{array}{lcll}
depth(G') & = & depth(G) + \mathcal{O}(\log|E|) & \text{(simplification, Lemma 6.2),}\\
depth(G') & = & depth(G) & \text{(small component, Lemma 6.6),}\\
depth(G') & = & depth(G) + \mathcal{O}(\log \#Eq(E) & \\
 & & \quad + \log \mathrm{eop}(\sigma)) & \text{(cycle, Lemma 6.12),}\\
depth(G') & = & depth(G) & \text{(rewriting, Lemma 7.4).}
\end{array}$$

Since $\log \mathrm{eop}(\sigma) = \mathcal{O}(|E_0|)$ and the number of cycle steps as well as of simplifications is at most $|E_0|$, an upper bound is $depth(G) = depth(G_0) + \mathcal{O}(|E_0|^2)$.

(vi) *Size of the grammar,* $|G|$: There are the following possibilities:

$$\begin{array}{lcll}
|G'| & = & |G| + \mathcal{O}(|E|(depth(G) + \log|E|)) & \text{(simplification, Lemma 6.2),}\\
|G'| & \leq & |G| + depth(G) & \text{(small component, Lemma 6.6),}\\
|G'| & = & |G| + \mathcal{O}(depth(G) + \#Eq(E) & \\
 & & \quad + \log \mathrm{eop}(\sigma)) & \text{(cycles, Lemma 6.12),}\\
|G'| & \leq & |G| + \#Eq(E)\ depth(G) & \text{(rewriting, Lemma 7.4).}
\end{array}$$

We know that the maximal number of rewriting steps is $|E_0|^2$ and the maximal number of small component, simplification, and cycle steps is at most $|E_0|$. We also have $\log \mathrm{eop}(\sigma) = \mathcal{O}(|E_0|)$, $|E| = \mathcal{O}(|E_0|^3)$, and $\#Eq(E) = \mathcal{O}(|E_0|)$. Together with the upper bound on $depth(G)$, this gives an upper bound (simplification is responsible for the dominating terms):

$$\begin{array}{rcl}
|G'| & = & |G_0| + |E_0|\,\mathcal{O}(|E_0|^3(depth(G) + \log|E_0|^3)),\\
 & & \quad + |E_0|\,depth(G),\\
 & & \quad + |E_0|\,\mathcal{O}(depth(G) + |E_0|),\\
 & & \quad + 6|E_0|^2\,|E_0|\ depth(G),\\
 & = & |G_0| + \mathcal{O}(|E_0|^4\,depth(G_0) + |E_0|^6).
\end{array}$$

(vii) *Size of the compacted solution,* $|\rho|$: We have to represent the instantiation of at most $|E_0|$ variables, where the size of each instantiation is bounded by the number of rewriting steps. This gives $\mathcal{O}(|E_0|^3)$.

All the transformations are sound according to Lemma 7.2. Now, if the compacted equations are not simplified, we can always simplify them. If they are simplified, either there is a cycle, or a solution with small components, or we can rewrite the equations. □

8. Main results and some remarks. Theorem 7.7 states that, given a compact representation $\langle E, G\rangle$ of a set of equations, we can build a new singleton grammar G' of polynomial size defining all components of the compact representation of a size-minimal solution. The final step is to use Plandowski's theorem (Theorem 4.2) to check that the polynomial-sized guessed substitution is really a unifier.

MAIN THEOREM 8.1. *Solvability of compact representations of basic MSOU problems is NP-complete.*

Proof. Theorem 7.7 shows that for every compact representation $\langle E, G\rangle$ of a basic MSOU problem and every size-minimal solution σ, there is compacted solution $\langle \rho, G'\rangle$ that represents σ, where G' is a singleton grammar of polynomial size in $|E| + |G|$ and $|\rho|$ is also polynomial in $|E|$.

Thus we can guess a polynomial-sized singleton grammar G' and a compacted solution ρ as above, and then test whether $\langle \rho, G'\rangle$ is a solution of $\langle E, G\rangle$. We can replace every variable occurrence in E by its instantiation in ρ, then normalize both sides of each equation $s_i \stackrel{?}{=} t_i$, to obtain $s'_i \stackrel{?}{=} t'_i$, and, finally, extend G' to obtain G'' by Lemma 4.4, generating $s'_1 \# \cdots \# s'_n$ and $t'_1 \# \cdots \# t'_n$, where $\#$ is a new constant symbol. Then, the test for solvability is an equality test w.r.t. the singleton grammar G'', which can be performed in polynomial time by Plandowski's theorem (see Theorem 4.2). This shows that the problem is in NP. Together with the NP-hardness of the problem, which was proved in [22], this leads us to conclude that the problem is NP-complete. □

COROLLARY 8.2. *Monadic second-order unification is NP-complete.*

Proof. The proof follows from Theorem 8.1 and Proposition 3.1. □

COROLLARY 8.3. *Monadic second-order matching is NP-complete.*

Proof. The proof follows from Theorems 2.6 and 8.1. □

Remark 1. Theorem 7.7 clarifies the increase of the size of the grammar representing a size-minimal solution of some compacted equations, after instantiating N variables. This theorem fixes the increase w.r.t. the size of the equations, the logarithm of the upper bound on the exponent of periodicity, and the *depth* of the grammar. The question is then, *Could we avoid the use of the depth of the grammar?* The answer is no. For instance, Lemma 4.6 says that, if we want to define a prefix of some word defined by a grammar G, in the worst case, we can keep the depth, but we may need to increase the size of G' as $|G'| \leq |G| + \mathit{depth}(G)$. If we use only the size of the grammar to characterize it, then in the worst case we may be forced to duplicate the size of the grammar $|G'| \leq 2\,|G|$. Each time that we instantiate a variable, it can be necessary to define a new prefix; therefore, in the worst case, the size of the resulting grammar would be 2^N, being $N \leq |E|$ the number of variables.

The combined use of size and depth allows us to keep track of balancing conditions of singleton grammars as trees and also to provide tighter measures.

Remark 2. Our method computes a compact representation of a size-minimal solution. This means that every solvable MSOU problem has at least one solution that can be represented by a polynomial-sized grammar. Our method can easily be extended to compute a compact representation of any solution; however, there is no longer any size-bound. If one is interested in representing all solutions, then our method does not help, since singleton grammars do not support the representation of infinite sets of words; e.g., the representation of $\{(a\,b)^n \mid n \in \mathbb{N}\}$ is not possible. Note that there is already an investigation of a representation of sets of solutions using words with exponents for MSOU (see [2]).

9. Conclusions. In this paper we proved in Corollary 8.2 that monadic second-order unification (MSOU) is in NP using a result of Plandowski about context-free grammars [17, 18] and the exponential bound on the exponent of periodicity [23, 22]. These results, together with the NP-hardness of the problem [22], prove its NP-completeness. As we mention in the introduction, MSOU is a specialization of bounded second-order unification (BSOU) [22], a variant of second-order unification, where instantiations of second-order variables can use their argument a bounded number of times. During revision of this paper we were able to apply variants of this method to prove that BSOU [11] and stratified context unification [12] are also NP-complete.

Acknowledgment. We acknowledge the meticulous reading and helpful comments of the anonymous referees, which helped us to improve the presentation of the paper.

REFERENCES

[1] G. Dowek, *Higher-order unification and matching*, in Handbook of Automated Reasoning, Vol. II, A. Robinson and A. Voronkov, eds., Elsevier Science, Amsterdam, 2001, pp. 1009–1062.

[2] W. M. Farmer, *A unification algorithm for second-order monadic terms*, Ann. Pure Appl. Logic, 39 (1988), pp. 131–174.

[3] W. M. Farmer, *Simple second-order languages for which unification is undecidable*, Theoret. Comput. Sci., 87 (1991), pp. 173–214.

[4] M. R. Garey and D. S. Johnson, *"Computers and Intractability": A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[5] W. D. Goldfarb, *The undecidability of the second-order unification problem*, Theoret. Comput. Sci., 13 (1981), pp. 225–230.

[6] G. Huet, *A unification algorithm for typed λ-calculus*, Theoret. Comput. Sci., 1 (1975), pp. 27–57.

[7] M. Karpinski, W. Rytter, and A. Shinohara, *An efficient pattern-matching algorithm for strings with short descriptions*, Nordic J. Comput., 4 (1997), pp. 172–186.

[8] A. Kościelski and L. Pacholski, *Complexity of Makanin's algorithm*, J. ACM, 43 (1996), pp. 670–684.

[9] J. Levy, *Decidable and undecidable second-order unification problems*, in Proceedings of the 9th Annual International Conference on Rewriting Techniques and Applications (RTA'98), Lecture Notes in Comput. Sci. 1379, Springer-Verlag, Berlin, 1998, pp. 47–60.

[10] J. Levy, M. Schmidt-Schauß, and M. Villaret, *Monadic second-order unification is NP-complete*, in Proceedings of the 15th Annual International Conference on Rewriting Techniques and Applications (RTA'04), Lecture Notes in Comput. Sci. 3091, Springer-Verlag, Berlin, 2004, pp. 55–69.

[11] J. Levy, M. Schmidt-Schauß, and M. Villaret, *Bounded second-order unification is NP-complete*, in Proceedings of the 17th Annual International Conference on Rewriting Techniques and Applications (RTA'06), Lecture Notes in Comput. Sci. 4098, Springer-Verlag, Berlin, 2006, pp. 400–414.

[12] J. Levy, M. Schmidt-Schauß, and M. Villaret, *Stratified context unification is NP-complete*, in Proceedings of the 3rd Annual International Joint Conference on Automated Reasoning (IJCAR'06), Lecture Notes in Comput. Sci. 4130, Springer-Verlag, Berlin, 2006, pp. 82–96.

[13] J. Levy and M. Veanes, *On the undecidability of second-order unification*, Inform. and Comput., 159 (2000), pp. 125–150.

[14] J. Levy and M. Villaret, *Currying second-order unification problems*, in Proceedings of the 13th Annual International Conference on Rewriting Techniques and Applications (RTA'02), Lecture Notes in Comput. Sci. 2378, Springer-Verlag, Berlin, 2002, pp. 326–339.

[15] Y. Lifshits, *Solving Classical String Problems on Compressed Texts*, The Computing Research Repository (CoRR); available online from http://www.avxiv.orgabs/cs/0604058 (2006).

[16] G. S. Makanin, *The problem of solvability of equations in a free semigroup*, Sb. Math. USSR, 32 (1977), pp. 129–198.

[17] W. Plandowski, *Testing equivalence of morphisms on context-free languages*, in Proceedings of the Second Annual European Symposium on Algorithms (ESA'94), Lecture Notes in Comput. Sci. 855, Springer-Verlag, London, 1994, pp. 460–470.

[18] W. Plandowski, *The Complexity of the Morphism Equivalence Problem for Context-Free Languages*, Ph.D. thesis, Department of Mathematics, Informatics and Mechanics, Warsaw University, Warsaw, Poland, 1995.

[19] W. Plandowski, *Satisfiability of word equations with constants is in PSPACE*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS'99), pp. 495–500.

[20] W. Plandowski, *Satisfiability of word equations with constants is in PSPACE*, J. ACM, 51 (2004), pp. 483–496.

[21] M. Schmidt-Schauß, *Stratified context unification is in PSPACE*, in Proceedings of the 15th International Workshop in Computer Science Logic (CSL'01), Lecture Notes in Comput. Sci. 2142, Springer-Verlag, Berlin, 2001, pp. 498–512.

[22] M. Schmidt-Schauß, *Decidability of bounded second order unification*, Inform. and Comput., 188 (2004), pp. 143–178.

[23] M. Schmidt-Schauß and K. U. Schulz, *On the exponent of periodicity of minimal solutions of context equations*, in Proceedings of the 9th Annual International Conference on Rewriting Techniques and Applications (RTA'98), Lecture Notes in Comput. Sci. 1379, Springer-Verlag, Berlin, 1998, pp. 61–75.

[24] A. P. Zhezherun, *Decidability of the unification problem for second-order languages with unary function symbols*, Kibernetika (Kiev), 5 (1979), pp. 120–125 (in Russian); Cybernetics, 15 (1980), pp. 735–741 (in English).

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1141–1156

THE SPECTRAL METHOD FOR GENERAL MIXTURE MODELS*

RAVINDRAN KANNAN†, HADI SALMASIAN‡, AND SANTOSH VEMPALA§

Abstract. We present an algorithm for learning a mixture of distributions based on spectral projection. We prove a general property of spectral projection for arbitrary mixtures and show that the resulting algorithm is efficient when the components of the mixture are logconcave distributions in $\Re^n$ whose means are separated. The separation required grows with k, the number of components, and $\log n$. This is the first result demonstrating the benefit of spectral projection for *general* Gaussians and widens the scope of this method. It improves substantially on previous results, which focus either on the special case of spherical Gaussians or require a separation that has a considerably larger dependence on n.

Key words. logconcave distributions, mixture models, principal component analysis, singular value decomposition

AMS subject classifications. 62H30, 68Q32, 68T05

DOI. 10.1137/S0097539704445925

1. Introduction.

Mixture models are widely used for statistical estimation, unsupervised concept learning, and text and image classification [12, 17]. Roughly speaking, a finite mixture model is a weighted combination of a finite number of distributions of a known type. More precisely, the problem of learning or estimating a mixture model is formulated as follows. We assume that we get samples from a distribution F on $\Re^n$ which is a mixture (convex combination) of unknown distributions $F_1, F_2, \ldots, F_k$, with (unknown) mixing weights $w_1, w_2, \ldots, w_k > 0$, i.e., $F = \sum_{i=1}^k w_i F_i$, where $\sum_{i=1}^k w_i = 1$. The goal is to (a) classify the sample points according to the underlying distributions and (b) estimate essential parameters of the components such as the mean and covariance matrix of each component. This problem has been widely studied, particularly for the special case when each F_i is a Gaussian.

One algorithm that is often used is the expectation-maximization (EM) algorithm [5, 21]. It is quite general but does not have guarantees on efficiency and could even converge to an incorrect or suboptimal classification. A second known technique, from statistics, projects the sample points to a random low-dimensional subspace and then tries to find the right classification by exploiting the low dimensionality and exhaustively examining all possible classifications. The trouble is that two different densities may overlap after projection—the means of the projected densities may coincide (or get closer), making it hard to separate the samples.

In this paper, we investigate a learning method which is based on principal component analysis (PCA). The idea of this method is known as spectral projection, i.e., representation of data in the subspace spanned by its top k principal components. We present our results following a discussion of the relevant literature.

*Received by the editors September 6, 2004; accepted for publication (in revised form) March 23, 2008; published electronically July 2, 2008. This research was supported by NSF awards ITR-0312354 and ITR-0312339 and the Sloan Foundation.

http://www.siam.org/journals/sicomp/38-3/44592.html

†Department of Computer Science, Yale University, New Haven, CT 06511 (kannan@cs.yale.edu).

‡Department of Mathematics and Statistics, University of Windsor, Windsor, ON N9B 3P4, Canada (hs79@uwindsor.ca).

§College of Computing, Georgia Tech, Atlanta, GA 30332, and Department of Mathematics, MIT, Cambridge, MA 02139 (vempala@cc.gatech.edu).

1.1. Recent theoretical work. There has been progress in recent years in finding algorithms with rigorous theoretical guarantees [3, 2, 4, 19], mostly for the important special case of learning mixtures of Gaussians. These algorithms assume a separation between the means of each pair of component distributions which depends on the variances of the two distributions and also on n and k. For a component F_i of the mixture let μ_i denote its mean and σ_i denote the maximum standard deviation along any direction in $\Re^n$ (see section 1.3). In order for the classification problem to have a well-defined (unique) solution with high probability, any two components i, j must be separated by $\sigma_i + \sigma_j$ times a logarithmic factor; if the separation is smaller than this, then the distributions may "overlap" significantly; namely, some of the samples have a good chance of coming from more than one component. Dasgupta [3] showed that if each mixing weight is $\Omega(1/k)$ and the variances are within a bounded range, then a separation of (the Ω^* notation suppresses logarithmic terms and error parameters)

$$|\mu_i - \mu_j| = (\sigma_i + \sigma_j)\Omega^*(n^{1/2})$$

is enough to efficiently learn the mixture.

Shortly thereafter, this result was improved by Dasgupta and Schulman [4] and Arora and Kannan [2], who reduced the separation required to

$$|\mu_i - \mu_j| = (\sigma_i + \sigma_j)\Omega^*(n^{1/4}).$$

In [4], the algorithm used is a variant of EM (and requires some technical assumptions on the variances), while the result of [2] works for general Gaussians using distance-based classification. The idea is that at this separation, it is possible to examine just the pairwise distances of the sample points and infer the right classification with high probability.

The dependence on n is critical; typically n represents the number of attributes and is much larger than k, the size of the model. Further, the underlying method used in these papers, namely, distance-based classification, inherently needs such a large separation that grows with n [2].

In [19], a spectral algorithm was used for the special case of spherical Gaussians, and the separation required was reduced to

$$|\mu_i - \mu_j| = (\sigma_i + \sigma_j)\Omega^*(k^{1/4}).$$

Since in several applications k is a constant which is much less than n, this result is a substantial improvement for the spherical case. The algorithm uses a projection of the sample onto the subspace spanned by the top k singular vectors of the distribution (i.e., the singular vectors of a matrix, each of whose rows is one of the independent and identically distributed (i.i.d) samples drawn according to the mixture), also called the SVD subspace. The idea there is that the SVD subspace of a mixture of spherical Gaussians *contains* the means of the k components. Hence, after projection onto this subspace the separation between the means is preserved. On the other hand, each component is still a Gaussian and the dimension is only k, and so the separation required is only a function of k. Further, the SVD subspace computed from a random sample is "close" to the means, and this is used in the algorithm.

1.2. New results. Given the success of the spectral method for spherical Gaussians, a natural question is whether it can be used for more general distributions, in particular for nonspherical Gaussians. At first sight, the method does not seem to

be applicable. The property that the SVD subspace of the distribution contains the means is clearly false for nonspherical Gaussians; e.g., see Figure 1. In fact, the SVD subspace can be orthogonal to the one spanned by the means, and so using spectral methods might seem hopeless.

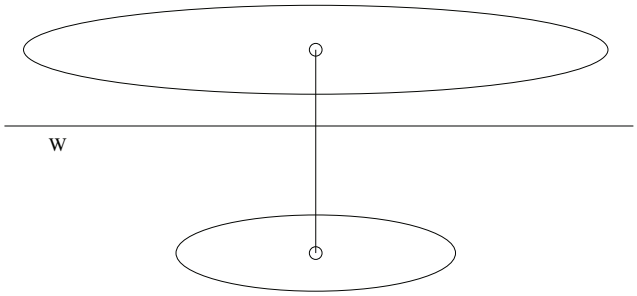


FIG. 1. *The SVD subspace W; the plane that minimizes the average squared distance and is represented by the horizontal line might miss the means of the components.*

The key insight of this paper is that while according to this (misleading) example the SVD subspace may not contain the means, it is always *close* (in an average sense) to the means of the distributions (Theorem 1). As a result, upon projection onto this subspace, the intermean distances are approximately preserved "on average." Moreover, this property is true for a mixture of *arbitrary* distributions.

It is then a reasonable idea to project the sample to the SVD subspace to reduce the dimensionality. To identify individual components in this subspace, we need them to remain nonoverlapping. If the mixture is arbitrary, then even though the means are separated on average, the samples could intermingle. To overcome this, we assume that the component distributions are logconcave.

A function $f : \Re^n \to \Re_+$ is logconcave if its logarithm is concave; i.e., for any two points $x, y \in \Re^n$ and any $\lambda \in [0, 1]$,

$$f(\lambda x + (1 - \lambda)y) \geq f(x)^{\lambda} f(y)^{1-\lambda}.$$

These functions have many useful properties; e.g., the product, minimum, and convolution of two logconcave functions are also logconcave [6, 11, 15]. Logconcave densities are a powerful generalization of Gaussians. Besides Gaussians, many other common probability measures, such as exponential families and the uniform measure over a convex set, are logconcave. So, for example, one component of the mixture could be a Gaussian while another is the uniform distribution over a cube. The following properties make these distributions suitable for our purpose: (a) the projection of a logconcave distribution remains logconcave; (b) the distance of a random point from the mean has an exponentially decreasing distribution.

In section 3, we give an iterative spectral algorithm that identifies one component of the mixture in each iteration. It should be emphasized that there are many possible alternatives for identifying the components *after* projection (e.g., the EM algorithm), and we expect they will also benefit from the enhancement provided by projection. For the postprojection algorithm presented here, we assume that each mixing weight is at least ε and the pairwise separation satisfies

$$|\mu_i - \mu_j| = (\sigma_i + \sigma_j)\Omega^*(k^{\frac{3}{2}}/\varepsilon^2).$$

More precisely, our algorithm requires only a lower bound ε, a probability of error δ, an upper bound k, and a sample set from an n-dimensional mixture distribution of size $\Omega(\frac{nk}{\varepsilon^3}\log^5(nk/\delta))$ which satisfies the given separation, and it classifies all but a fixed number with probability at least $1-\delta$. For a precise statement of our main result together with the exact form of the required separation, see Theorem 3. It is easy to see that it requires time polynomial in $n, \varepsilon, \log(\frac{1}{\delta})$. After classification, the means and covariance matrices of the components can be estimated efficiently. For the special case of Gaussians, $O(n\log^3(n/\delta)/\varepsilon^2)$ samples suffice. Table 1 presents a comparison of algorithms for learning mixtures (logarithmic terms are suppressed).

TABLE 1
Comparison.

Authors	Separation	Assumptions	Method
Dasgupta [3]	$n^{\frac{1}{2}}$	Gaussians, bounded variances, and $w_i = \Omega(1/k)$	random projection
Dasgupta and Schulman [4]	$n^{\frac{1}{4}}$	spherical Gaussians	EM + distances
Arora and Kannan [2]	$n^{\frac{1}{4}}$	Gaussians	distances
Vempala and Wang [19]	$k^{\frac{1}{4}}$	spherical Gaussians	spectral projection
This paper	$\frac{k^{\frac{3}{2}}}{\varepsilon^2}$	logconcave distributions	spectral projection

1.3. Notation. This section introduces the notation used throughout the paper. For readers' convenience, in Table 2 we provide a quick reference to some of our notation. A mixture F has k components $F_1, \ldots, F_k$. We denote their mixing weights by $w_1, \ldots, w_k$ and their means by $\mu_1, \ldots \mu_k$. The maximum variance of F_i in any direction is denoted by σ_i^2. For any subspace W, we denote the maximum variance of F_i along any direction in W by $\sigma_{i,W}^2$. Namely, we set

$$\sigma_i^2 = \max_{v\in\Re^n, ||v||=1} \left\{ \int_{\Re^n} |(x-\mu_i)\cdot v|^2 F_i(x)dx \right\}$$

and

$$\sigma_{i,W}^2 = \max_{v\in W, ||v||=1} \left\{ \int_{\Re^n} |(x-\mu_i)\cdot v|^2 F_i(x)dx \right\}.$$

Let S be a set of i.i.d. samples from F. One can think of S as being picked as follows: first, i is picked from $\{1, 2, \ldots, k\}$ with probability w_i (unknown to the algorithm); then a sample is picked from F_i. We can partition S as

$$S = S_1 \cup S_2 \cup \cdots \cup S_k,$$

where each S_i is from F_i (note: this partition of S is unknown to the algorithm). For each i, we denote by μ_i^S the sample mean, i.e.,

$$\mu_i^S = \frac{1}{|S_i|}\sum_{x\in S_i} x.$$

For a subspace V and a vector x, we write $d(x, V)$ for the orthogonal distance of x from V.

TABLE 2
Notation.

Symbol	Appears for the first time in:
$d(x, W)$	section 1.3
μ_i, μ_i^S	section 1.3
$\hat{\mu}_i, \hat{\mu}_i^S$	section 4.3
$\sigma_i^2, \hat{\sigma}_i^2$	section 1.3
$\hat{\sigma}_{i,W}^2, \hat{\sigma}_{i,W}^2(S)$	section 1.3
$\sigma(p)^2$	section 3
$T(p)$	ALGORITHM
A_i, A_i', A_i''	section 3

For any set of points S, we can form a matrix A whose rows are the points in S. The subspace spanned by the top k right singular vectors of A will be called the *SVD subspace* of S. The SVD subspace can be found efficiently (i.e., in polynomial time). For more on properties and applications of SVD subspaces, see [8].

For any subspace W, we denote the maximum variance of a set of sample points in $S = S_1 \cup \cdots \cup S_k$ which belong to S_i along any direction in W by $\hat{\sigma}_{i,W}^2(S)$. Namely, we set

$$\hat{\sigma}_{i,W}^2(S) = \max_{v \in \Re^n, ||v||=1} \left\{ \frac{1}{|S_i|} \sum_{x \in S_i} |(x - \mu_i^S) \cdot v|^2 \right\}.$$

If there is no ambiguity about the set S or the space W, we use a simplified notation such as $\hat{\sigma}_{i,W}^2$ or $\hat{\sigma}_i^2$.

2. The SVD subspace. In this section, we prove an important property of spectral projection. The theorem says that the SVD subspace of a sample is close to the means of the samples from each component of the mixture, where "close" is in terms of the sample variances. Note that the theorem holds for *any* mixture, but in the analysis of our algorithm, we will apply it only to mixtures of logconcave distributions. We prove that

$$\sum_{i=1}^{k} w_i d(\mu_i, W)^2 \leq k \sum_{i=1}^{k} w_i \sigma_i^2. \tag{1}$$

Here W is the SVD subspace of the entire distribution (subspace spanned by the top k principal components of the distribution). In Theorem 1 we state and prove a variation of this inequality for samples, but it can also be proved in a similar fashion.

THEOREM 1. *Let $S = S_1 \cup S_2 \cup \cdots \cup S_k$ be a sample from a mixture F with k components such that S_i is from the ith component F_i, and let W be the SVD subspace of S (see section 1.3). For each i, let μ_i^S be the mean of S_i and $\hat{\sigma}_{i,W}^2(S)$ be the maximum variance of S_i along any direction in W. Then*

$$\sum_{i=1}^{k} |S_i| d(\mu_i^S, W)^2 \leq k \sum_{i=1}^{k} |S_i| \hat{\sigma}_{i,W}^2(S).$$

Proof. We begin by stating a general lemma which follows immediately from the Pythagorean theorem.

LEMMA 2. *Given $p, p_1, \ldots, p_K \in \Re^n$, if $\mu_P = \frac{1}{K}\sum_{i=1}^K p_i$, then*

$$\sum_{i=1}^K |p_i - p|^2 = K|p - \mu_P|^2 + \sum_{i=1}^K |p_i - \mu_P|^2.$$

Lemma 2 is used in the proof of the following theorem.

Let M be the span of $\mu_1^S, \mu_2^S, \ldots, \mu_k^S$. For $x \in \Re^n$, write $\pi_M(x)$ for the projection of x onto M and $\pi_W(x)$ for the projection of x onto W.

Using Lemma 2 and the facts that μ_i^S is the average of $x \in S_i$ and $\mu_i^S \in M$, we write

$$\begin{aligned}\sum_{x\in S} |\pi_M(x)|^2 &= \sum_{i=1}^k \sum_{x\in S_i} |\pi_M(x) - \mu_i^S|^2 + \sum_{i=1}^k |S_i||\mu_i^S|^2 \\ &\geq \sum_{i=1}^k |S_i||\mu_i^S|^2 \\ &= \sum_{i=1}^k |S_i||\pi_W(\mu_i^S)|^2 + \sum_{i=1}^k |S_i| d(\mu_i^S, W)^2. \qquad (2)\end{aligned}$$

Let $\vec{e}_1, \ldots, \vec{e}_k$ be an orthonormal basis for W. Using Lemma 2 and the fact that the variance of S_i along any direction in the k-dimensional subspace W is at most $\hat{\sigma}_{i,W}^2(S)$, we have

$$\begin{aligned}\sum_{x\in S} |\pi_W(x)|^2 &= \sum_{i=1}^k \sum_{x\in S_i} |\pi_W(x - \mu_i^S)|^2 + \sum_{i=1}^k |S_i||\pi_W(\mu_i^S)|^2 \\ &\leq \sum_{i=1}^k \sum_{j=1}^k \sum_{x\in S_i} |\pi_W(x - \mu_i^S)\cdot \vec{e}_j|^2 + \sum_{i=1}^k |S_i||\pi_W(\mu_i^S)|^2 \\ &\leq k\sum_{i=1}^k |S_i|\hat{\sigma}_{i,W}^2(S) + \sum_{i=1}^k |S_i||\pi_W(\mu_i^S)|^2. \qquad (3)\end{aligned}$$

It is well known that the SVD subspace maximizes the sum of squared projections among all subspaces of rank at most k (alternatively, it minimizes the sum of squared distances to the subspace; see, e.g., [8]). From this, we get

$$\sum_{x\in S} |\pi_W(x)|^2 \geq \sum_{x\in S} |\pi_M(x)|^2.$$

Using this, the right-hand side (RHS) of (3) is at least the RHS of (2), and the theorem follows. ∎

Although we will apply Theorem 1 only for logconcave component distributions, it suggests a benefit for spectral projection more generally. Inequality (1) puts a lower bound on the average squared distance between component means after projection; if the means are well separated to begin with, they continue to be, in an average sense. On the other hand, the distance of a point from the mean of its distribution can shrink only upon projection, thus magnifying the ratio of intercomponent distance to intracomponent distance. This aspect is studied further along with empirical results in [20].

3. An iterative spectral algorithm. In this section, we describe the algorithm. It follows the method suggested by Theorem 1, namely, to project onto the SVD subspace and to try to identify components in that subspace. However, since pairwise distances are preserved only in an average sense, it is possible that some means are very close to each other in the projected subspace and we cannot separate the corresponding samples. To get around this, we will show that all "large" components remain well separated from the rest and there is at least one large component. We identify this component, filter it from the sample, and repeat. For technical reasons (see below), the samples used to compute the SVD are discarded. The input to the algorithm below is a set of N i.i.d. samples, a weight lower bound $0 < \varepsilon < 1$, a maximum probability of error δ, a lower bound k for the number of distributions, and a parameter $N_0 < N$.

ALGORITHM.

Repeat while there are samples left:

1. For a subset S of size N_0, find the k-dimensional SVD subspace W.
2. Discard S and project the rest, T, to the subspace W.
3. For each projected point p:
   --- Find the closest $\varepsilon N/2$ points. Let this set be $T(p)$ with mean $\mu(p)$.
   --- Form the matrix $A(p)$ whose rows are x-$\mu(p)$ for each x in $T(p)$. Compute the largest singular value $\sigma(p)$ of $A(p)$ (note: this is the maximum standard deviation of $T(p)$ over all directions in W).
4. Find a point p_0 for which $\sigma(p_0)$ is maximum. Let T_0 be the set of all points of T whose projection to W is within distance $\frac{2^8\sqrt{k}\log(\frac{Nk}{\delta})}{\varepsilon}\sigma(p)$ of p_0.
5. Label T_0 as one component; estimate its mean and covariance matrix.
6. Delete T_0 from T.

In step 3 of the algorithm, for any point p, the top singular value $\sigma(p)$ of $A(p)$ can also be expressed as follows:

$$\sigma(p)^2 = \max_{v \in W, |v|=1} \frac{1}{|T(p)|} \sum_{q \in T(p)} |q \cdot v|^2 - \left(\frac{1}{|T(p)|} \sum_{q \in T(p)} q \cdot v \right)^2.$$

This value is an estimate of the maximum variance of the entire subsample of the component to which p belongs.

There is a technical issue concerning independence. If we use the entire sample to compute the SVD subspace W, then the sample is not independent from W. So we use a subset S to compute the SVD subspace in each iteration and discard it. The rest of the sample, i.e., the part not used for SVD computation, is classified correctly with high probability. The size of the subset S in each iteration is N_0, where

$$N_0 = 100 \frac{n}{\varepsilon^2} \log^5 \frac{nk}{\delta}.$$

Let $0 < \varepsilon, \delta < 1$. Our main result is the following guarantee for the proposed algorithm.

THEOREM 3. *Suppose we have N i.i.d. samples from a mixture F of k logconcave distributions with mixing weights at least ε and the means of the components separated as follows:*

$$\forall\, i,j \quad |\mu_i - \mu_j| \geq 2^{11}(\sigma_i + \sigma_j)\left(\frac{k^{\frac{3}{2}}}{\varepsilon^2}\right)\log^2\left(\frac{Nk}{\delta}\right).$$

There exists an absolute constant C such that if $N > C\frac{kN_0}{\varepsilon}$, then the iterative spectral algorithm correctly classifies $N - kN_0$ samples with probability at least $1-\delta$ (a subset of kN_0 samples is used by the algorithm and discarded).

Remark. A crude upper bound of $C < 10^6$ can be obtained quite easily by tracing through the proof of Theorem 3. More careful estimates should yield a much smaller upper bound.

We will prove Theorem 3 in the next section. The following corollary of Theorem 3 provides a second guarantee for the algorithm in terms of estimating the means and covariances of the distributions.

COROLLARY 4. *Let $0 < \eta < 1$, and suppose we are given N i.i.d. samples from a mixture F of k logconcave distributions in $\Re^n$, with mixing weights at least ε and the means separated as stated in Theorem 3. For any $1 \leq i \leq k$, let μ_i be the mean and $A_i = \mathsf{E}_{F_i}\left((x-\mu_i)(x-\mu_i)^T\right)$ be the covariance matrix of F_i. If $N = \Omega^*(\frac{nk^2}{\delta^2\eta^2\varepsilon^3})$, then using the iterative spectral algorithm we can find approximations $\mu_1', \ldots, \mu_k'$ to the means and $A_1', \ldots, A_k'$ to the covariance matrices of the components $F_1, \ldots, F_k$ of the sample such that with probability at least $1-\delta$, for $1 \leq i \leq k$,*

$$|\mu_i - \mu_i'| \leq \eta\sigma_i \quad \text{and} \quad ||A_i - A_i'|| \leq \eta\sigma_i^2,$$

where $||\cdot||$ is the spectral norm of the matrix.

The proof of Corollary 4 is given in section 4.4.

4. Analysis.

4.1. Preliminaries. We begin with some properties of logconcave distributions, paraphrased from [13]. The proof of the first uses a theorem from [16] (see also Proposition 2.11 of [7]).

LEMMA 5. *Let $0 < \eta < 1$ and $y_1, \ldots, y_m$ be i.i.d. samples from a logconcave distribution G in $\Re^n$ whose mean is the origin. There is an absolute constant C such that for*

$$m > C\frac{n}{\eta^2}\log^5\left(\frac{n}{\eta\delta}\right)$$

with probability at least $1-\delta$, for any vector $v \in R^n$,

$$(1-\eta)\mathsf{E}_G((v^Ty)^2) \leq \frac{1}{m}\sum_{i=1}^{m}(v^Ty_i)^2 \leq (1+\eta)\mathsf{E}_G((v^Ty)^2).$$

The next lemma is an adaptation of Lemma 5.7 of [13].

LEMMA 6. *Let F be any logconcave distribution in $\Re^n$ with mean μ and second moment $\mathsf{E}_F(|X-\mu|^2) = R^2$. There is an absolute constant c such that, for any $t > 1$,*

$$\Pr(|X-\mu| > tR) < e^{1-t}.$$

Lemma 5.5 of [13] implies the following lemma.

LEMMA 7. *Let $f : \Re \to \Re_+$ be a logconcave density function with variance σ^2. Then*

$$\max_{\Re} f(x) \le \frac{1}{\sigma}.$$

4.2. Sample properties. Assume that $m > N_0$ and T is a set of m elements consisting of i.i.d. samples generated by the mixture F. Then there is a partition T as $T = T_1 \cup T_2 \cup \cdots \cup T_k$, where T_i is the set of samples from F_i.

LEMMA 8. *With probability at least $1 - \delta/4k$, for every $i \in \{1, 2, \ldots, k\}$ we have the following:*

(a) $w_i|T| - \frac{\varepsilon}{4}|T| \le |T_i| \le w_i|T| + \frac{\varepsilon}{4}|T|$.

(b) $|\mu_i - \mu_i^T| \le \frac{\sigma_i}{4}$.

(c) *For any subspace W, $\frac{7}{8}\sigma_{i,W} \le \hat{\sigma}^2_{i,W}(T) \le \frac{8}{7}\sigma^2_{i,W}$.*

Proof.

(a) A point $x \in T$ belongs to T_i with probability w_i. If $T = \{x_1, \ldots, x_m\}$, define a random variable $Y_j, 1 \le j \le m$, as follows:

$$Y_j = \begin{cases} 1 & \text{if } x_j \in T_i, \\ 0 & \text{otherwise.} \end{cases}$$

The Y_j's are Bernoulli variables and $\mathrm{E}(Y_j) = w_i$. Therefore by Chernoff's bound (see Theorem 4.1 of [14]) we have

$$\Pr\left(\frac{Y_1 + \cdots + Y_m}{m} - w_i > \frac{\varepsilon}{4}\right) < \left(\frac{e^{\frac{\varepsilon}{4w_i}}}{(1 + \frac{\varepsilon}{4w_i})^{(1+\frac{\varepsilon}{4w_i})}}\right)^{mw_i}. \tag{4}$$

Since $w_i \ge \varepsilon$, $\varepsilon = 4tw_i$ for some $t \le \frac{1}{4}$. The RHS of (4) equals $(\frac{e^t}{(1+t)^{(1+t)}})^{mw_i}$, and one can check that for $t \le \frac{1}{4}$ we have $\frac{e^t}{(1+t)^{(1+t)}} < e^{-\frac{t^2}{4}}$. Similarly we have

$$\Pr\left(\frac{Y_1 + \cdots + Y_m}{m} - w_i < -\frac{\varepsilon}{4}\right) < (e^{-\frac{t^2}{4}})^{mw_i}.$$

It follows that Lemma 8(a) fails for distribution F_i with probability at most $2e^{-\frac{t^2}{4}mw_i}$, which is equal to $2e^{-\frac{m\varepsilon^2}{64w_i}}$. Now Lemma 8(a) follows immediately from $m \ge N_0$.

(b) For any fixed $|T_i|$, the random variable $\mu_i^T = \frac{1}{|T_i|}\sum_{x \in T_i} x$ is a convolution of logconcave distributions and hence is also logconcave. Its variance is bounded from above by $n\sigma_i^2/|T_i|$. We apply Lemma 6 to the random variable μ_i^T. Now Lemma 8(b) follows from Lemma 8(a).

(c) The proof follows immediately from Lemma 5. ∎

In our proof, we would like to apply Theorem 1. However, the theorem holds for the sample S that is used to compute the SVD subspace. The next lemma derives a similar bound for an independent sample T that is not used in the SVD computation.

LEMMA 9. *Let W be the SVD subspace obtained in the algorithm, and suppose $T = T_1 \cup \cdots \cup T_k$ is the set of sample points not used for the SVD computation in the algorithm. Then we have*

$$\sum_{i=1}^{k} |T_i| d(\mu_i^S, W)^2 \le 2k \sum_{i=1}^{k} |T_i| \hat{\sigma}_i^2, \tag{5}$$

where $\hat{\sigma}_i^2 = \hat{\sigma}_{i,W}^2(T)$ is the maximum variance of T_i along any direction in W.

Proof. First, we apply Theorem 1 to S. Then, using Lemma 8(a), we can relate $|T_i|$ to $|S_i|$, and we have

$$\sum_{i=1}^{k} |T_i| d(\mu_i^S, W) \le \frac{3}{2} k \sum_{i=1}^{k} |T_i| \hat{\sigma}_{i,W}^2(S).$$

Next, Lemma 5 implies that

$$\hat{\sigma}_{i,W}^2(S) \le \frac{7}{6} \sigma_{i,W}^2.$$

Finally, we use the lower bound in Lemma 8(c) to get the desired inequality. ∎

4.3. Proof of Theorem 3. We will prove the following claim: With probability at least $1 - \frac{\delta}{2k}$, the algorithm identifies one component exactly in any one iteration. We will prove the claim for the first iteration, and it will follow inductively for all subsequent iterations. From now on, we assume that Lemma 8 holds. Note that this happens with probability at least $1 - \frac{\delta}{4k}$.

Let $T = T_1 \cup T_2 \cup \cdots \cup T_k$ be the partition of the current sample T according to the components F_i. For each i, recall that μ_i^T is the sample mean, and define $\hat{\mu}_i^T$ to be the projection of μ_i^T onto the subspace spanned by W. Similarly, $\hat{\mu}_i^S$ and $\hat{\mu}_i$ are the projections of μ_i^S and μ_i. For convenience, we write $\hat{\sigma}_{i,W}(T)^2$ as $\hat{\sigma}_i^2$. Let

$$\alpha = 2^{11} \frac{k^{\frac{3}{2}}}{\varepsilon^2} \log^2\left(\frac{Nk}{\delta}\right) \quad \text{and} \quad \beta = \frac{\varepsilon^3}{2^{14} k \log^2(\frac{Nk}{\delta})}.$$

We say that a component F_r is *large* if the following condition holds:

$$|T_r| \hat{\sigma}_r^2 \ge \beta \max_i |T_i| \hat{\sigma}_i^2. \tag{6}$$

Recall that $|T| > N_0$. The proof of Theorem 3 is based on the next two lemmas.

LEMMA 10. *For any large component F_r, for every $i \ne r$,*

$$|\hat{\mu}_i^T - \hat{\mu}_r^T| > \frac{\alpha}{5}(\sigma_i + \sigma_r).$$

Proof. For any $1 \le j \le r$, let $d_j = d(\mu_j^S, W)$. By (5),

$$|T_r| d_r^2 \le 2k \sum_i |T_i| \hat{\sigma}_i^2 \le \frac{2k^2}{\beta} |T_r| \hat{\sigma}_r^2. \tag{7}$$

Thus,

$$d_r^2 \le \frac{2k^2}{\beta} \hat{\sigma}_r^2 \le \frac{\alpha^2}{16} \hat{\sigma}_r^2. \tag{8}$$

Next, let

$$R = \left\{ i \ne r \,:\, |\hat{\mu}_i^S - \hat{\mu}_r^S| \le \frac{\alpha}{4}(\sigma_i + \sigma_r) \right\}.$$

Then we have

$$\begin{aligned} d_i = d(\mu_i^S, W) &= |\mu_i^S - \hat{\mu}_i^S| \\ &= |(\mu_i^S - \mu_i) + (\mu_i - \mu_r) + (\mu_r - \mu_r^S) + (\mu_r^S - \hat{\mu}_r^S) + (\hat{\mu}_r^S - \hat{\mu}_i^S)| \end{aligned}$$

and, by the triangle inequality,

$$\begin{aligned} d_i &\geq |\mu_i - \mu_r| - |(\mu_i^S - \mu_i) + (\mu_r - \mu_r^S) + (\mu_r^S - \hat{\mu}_r^S) + (\hat{\mu}_r^S - \hat{\mu}_i^S)| \\ &\geq |\mu_i - \mu_r| - |\mu_i - \mu_i^S| - |\mu_r - \mu_r^S| - d_r - |\hat{\mu}_i^S - \hat{\mu}_r^S|. \end{aligned}$$

Next, we use the inequality above to show that

$$d_i \geq \frac{\alpha}{3}\sigma_r. \tag{9}$$

To this end, we need inequalities given in (10), (11), and (12). By the separation assumed in Theorem 3, we have

$$|\mu_i - \mu_r| \geq \alpha(\sigma_i + \sigma_r). \tag{10}$$

From Lemma 8(b) we have

$$|\mu_i - \mu_i^S| \leq \frac{\sigma_i}{4} \qquad \text{and} \qquad |\mu_r - \mu_r^S| \leq \frac{\sigma_r}{4}. \tag{11}$$

Moreover, from (8) and Lemma 8(c) it follows that

$$d_r \leq \frac{\alpha}{4}\hat{\sigma}_r \leq \frac{2\alpha}{7}\sigma_{r,W} \leq \frac{2\alpha}{7}\sigma_r. \tag{12}$$

From these facts (9) follows immediately.

Next, from Lemma 8(c) and (9) it follows that

$$d_i \geq \frac{\alpha}{4}\hat{\sigma}_r.$$

Therefore, using (7) and Lemma 9,

$$\begin{aligned} \frac{2k^2}{\beta}|T_r|\hat{\sigma}_r^2 \geq 2k\sum_{i=1}^{k}|T_i|\hat{\sigma}_i^2 &\geq \sum_{i=1}^{k}|T_i|d_i^2 \\ &\geq \sum_{i\in R}|T_i|d_i^2 \geq \sum_{i\in R}|T_i|\frac{\alpha^2}{16}\hat{\sigma}_r^2. \end{aligned}$$

As a result,

$$\sum_{i\in R}|T_i| \leq \frac{32k^2}{\alpha^2\beta}|T_r| < \frac{\varepsilon}{2}|T|.$$

However, since each $|T_i| \geq \frac{\varepsilon}{2}|T|$ (by Lemma 8(a)), this implies that R is empty. Consequently, for any $i \neq r$ we have

$$|\hat{\mu}_i^S - \hat{\mu}_r^S| > \frac{\alpha}{4}(\sigma_i + \sigma_r). \tag{13}$$

To complete the proof of the lemma, we note that by Lemma 8(b), for any $1 \leq j \leq r$,

$$|\hat{\mu}_j^T - \hat{\mu}_j^S| \leq |\mu_j^T - \mu_j^S| \leq |\mu_j^T - \mu_j| + |\mu_j - \mu_j^S| \leq \frac{\sigma_j}{2}.$$

Therefore, by (13) and the triangle inequality,

$$\begin{aligned}|\hat{\mu}_i^T - \hat{\mu}_r^T| &= |(\hat{\mu}_i^T - \hat{\mu}_i^S) + (\hat{\mu}_i^S - \hat{\mu}_r^S) + (\hat{\mu}_r^S - \hat{\mu}_r^T)| \\ &\geq |\hat{\mu}_i^S - \hat{\mu}_r^S| - |\hat{\mu}_r^S - \hat{\mu}_r^T| - |\hat{\mu}_i^S - \hat{\mu}_i^T| \\ &\geq \frac{\alpha}{4}(\sigma_i + \sigma_r) - \frac{\sigma_i}{2} - \frac{\sigma_r}{2} \\ &> \frac{\alpha}{5}(\sigma_i + \sigma_r). \qquad \square\end{aligned}$$

LEMMA 11. *Let $p \in T_i$. With probability at least $1 - \frac{\delta}{4k}$,*

$$\sigma(p)^2 \leq 16k\hat{\sigma}_i^2 \log^2\left(\frac{Nk}{\delta}\right).$$

Further, if i is a large component, then

$$\sigma(p)^2 \geq \frac{w_i^2}{512}\hat{\sigma}_i^2.$$

Proof. Consider the samples in T_i after projection onto W. These points are i.i.d. samples from a logconcave distribution with maximum variance at most $k\sigma_i^2$. Fix a (projected) sample point p. By Lemma 6,

$$\Pr\left(|p - \hat{\mu}_i| > \frac{3}{2}\sqrt{k}\sigma_{i,W}\log\left(\frac{Nk}{\delta}\right)\right) < \frac{\delta}{8kN}. \tag{14}$$

Therefore, with probability at least $1 - \frac{\delta}{8k}$, every projected sample point from any F_i lies within distance at most $\frac{3}{2}\sqrt{k}\sigma_{i,W}\log(\frac{Nk}{\delta})$ of $\hat{\mu}_i$. Consequently, with probability at least $1 - \frac{\delta}{8k}$, any pair of projected samples from T_i are within $3\sqrt{k}\sigma_{i,W}\log(\frac{Nk}{\delta})$ of each other. Since by Lemma 8(a) $|T_i| > \frac{N\varepsilon}{2}$, it follows that elements of $T(p)$ lie within a ball of radius at most $3\sqrt{k}\sigma_{i,W}\log(\frac{Nk}{\delta})$ centered at p. From the definition of $\sigma(p)$, Lemma 2, and Lemma 8(c) it follows that

$$\sigma(p)^2 \leq \frac{1}{|T(p)|}\sum_{x \in T(p)}|x - p|^2 \leq 9k\sigma_{i,W}^2 \log^2\left(\frac{Nk}{\delta}\right) \leq 16k\hat{\sigma}_i^2 \log^2\left(\frac{Nk}{\delta}\right).$$

For the second inequality, note that by Lemma 10 the set $T(p)$ of samples used to compute $\sigma(p)$ are all from T_i. If v is the direction in W for which the distribution F_i has maximum variance, then Lemma 7 implies that for

$$H = \left\{x \in \Re^n : \mu^{T(p)} \cdot v - \frac{\varepsilon}{8}\sigma_{i,W} \leq v.x \leq \mu^{T(p)} \cdot v + \frac{\varepsilon}{8}\sigma_{i,W}\right\}$$

we have $F_i(H) \leq \frac{1}{\sigma_{i,W}} \times 2\frac{\varepsilon\sigma_{i,W}}{8} = \frac{\varepsilon}{4}$.

Now we apply a result which follows from VC-dimension techniques (see [10]). To make the paper self-contained, we will give a proof of this result in section 4.5. For any interval I along the direction v, let $H_I = \{x \in \Re^n : x \cdot v \in I\}$.

LEMMA 12. *Suppose $T = T_1 \cup T_2 \cup \cdots \cup T_k$ is a set of i.i.d. samples generated from the mixture F such that $|T| > N_0$. Fix $i \in \{1, \ldots, r\}$, and let I be any interval along the direction of v such that $F_i(H_I) \leq \frac{\varepsilon}{4}$. With probability at least $1 - \frac{\delta}{8k}$, we have*

$$\left|\frac{|T_i \cap H_I|}{|T_i|} - F_i(H_I)\right| \leq \frac{\varepsilon}{8}.$$

We now complete the proof of Lemma 11. From Lemma 12 we have

$$|T(p) \cap H| \leq \frac{3\varepsilon}{8}|T_i| \leq \frac{3}{4} \times \frac{\varepsilon|T_i|}{2} \leq \frac{3|T(p)|}{4}.$$

This means that at least $\frac{|T(p)|}{4}$ samples in $T(p)$ are out of the strip H; i.e., they are at least as far as $\frac{\varepsilon}{8}\sigma_{i,W}$ apart from $\mu^{T(p)}$ in the direction of v. Hence, using Lemma 8(c),

$$\begin{aligned}\sigma(p)^2 &\geq \frac{1}{|T(p)|}\left(\frac{1}{||v||^2}\sum_{x\in T(p)}(x\cdot v - \mu^{T(p)}\cdot v)^2\right)\\ &\geq \frac{1}{|T(p)|}\times\frac{|T(p)|}{4}\times\left(\frac{\varepsilon}{8}\sigma_{i,W}\right)^2\\ &\geq \frac{\varepsilon^2}{256}\sigma_{i,W}^2 \geq \frac{\varepsilon^2\hat{\sigma}_i^2}{512},\end{aligned}$$

which completes the proof. □

We continue with the proof of Theorem 3. First, note that from Lemma 8(a) it follows that in each iteration the size of the set of samples is at least $2N_0$. The algorithm uses N_0 samples for SVD computation, and the rest are classified. Since there are at least N_0 samples left, all of the necessary lemmas will be valid.

Suppose the point p_0 which maximizes $\sigma(p)$ in step 4 of the algorithm is actually from a large component F_r (i.e., one which satisfies (6)). By Lemma 6 and the first part of Lemma 11, we have

$$\frac{\alpha}{8}\sigma_r > 2^8\frac{\sqrt{k}}{\varepsilon}\sigma(p)\log\left(\frac{Nk}{\delta}\right) > 3\sqrt{k}\sigma_{r,W}\log\left(\frac{Nk}{\delta}\right). \tag{15}$$

It was shown that from (14) it follows that with probability $1-\frac{\delta}{8k}$, any pair of projected samples onto W from F_i are within $3\sqrt{k}\sigma_{i,W}\log(\frac{Nk}{\delta})$ of each other. This, together with (15), means that the output of an iteration contains T_r entirely. Moreover, from Lemma 10 and the above consequence of (14) it follows that the distance between any projected samples of F_r and F_i, $i \neq r$, is at least $\frac{\alpha}{8}\sigma_r$. This, together with (15), implies that the output of the iteration does not contain any samples from any F_i, $i \neq r$. Therefore the output of the iteration exactly classifies samples from one distribution.

Next, we will show that the point p_0 in step 4 of the algorithm must indeed be from a large component. Let r be the component for which $|T_r|\hat{\sigma}_r^2$ is maximum. Take any $p \in T_i$ for an i which is not large, i.e.,

$$|T_i|\hat{\sigma}_i^2 < \beta|T_r|\hat{\sigma}_r^2. \tag{16}$$

Therefore,

$$\hat{\sigma}_i^2 \leq \beta\frac{|T_r|}{|T_i|}\hat{\sigma}_r^2 \leq \frac{2\beta}{\varepsilon}\hat{\sigma}_r^2.$$

By Lemma 11,

$$\sigma(p)^2 < 16k\hat{\sigma}_i^2\log^2\left(\frac{Nk}{\delta}\right) \leq \frac{32k\beta}{\varepsilon}\hat{\sigma}_r^2\log^2\left(\frac{Nk}{\delta}\right) = \frac{\varepsilon^2}{512}\hat{\sigma}_r^2.$$

On the other hand, for any point $q \in T_r$,

$$\sigma(q)^2 \geq \frac{\varepsilon^2}{512}\hat{\sigma}_r^2 > \sigma(p)^2.$$

Hence the point p_0 chosen in step 4 of the algorithm will be from a large component. This completes the proof of Theorem 3.

4.4. Proof of Corollary 4. Assume $N > \frac{C'k^2}{\varepsilon^3\eta^2\delta^2}\log^5(\frac{nk}{\delta\eta})$, where C' is a large enough constant. We use our spectral algorithm to obtain a classification of $N - kN_0$ samples such as $T = T_1 \cup \cdots \cup T_k$. By Theorem 3, with probability at least $1-\delta$ the algorithm classifies the samples correctly. For any $1 \leq i \leq k$, set

$$\mu_i' = \mu_i^T = \frac{1}{|T_i|}\sum_{x\in T_i} x \quad \text{and} \quad A_i' = \frac{1}{|T_i|}\sum_{x\in T_i}(x-\mu_i')(x-\mu_i')^T.$$

From Lemma 8(a) it follows that with probability at least $1-\delta$, for every $1 \leq i \leq k$ we have $|T_i| > \frac{4n}{\eta^2}\log^2(\frac{2k}{\delta})$. Now the samples in T_i are i.i.d. and all generated from F_i. With a method analogous to the proof of Lemma 8(b) given below, we can prove that for any $1 \leq i \leq k$, with probability at least $1 - \frac{\delta}{2k}$ we have $|\mu_i - \mu_i'| < \frac{\eta}{2}\sigma_i$.

Finally, let G_i be the shifted logconcave distribution given by

$$G_i(x) = F_i(x + \mu_i).$$

Clearly the mean of G_i is located at the origin. Set

$$A_i'' = \frac{1}{|T_i|}\sum_{x\in T_i}(x-\mu_i)(x-\mu_i)^T.$$

Since $|T_i| = \Omega\Big(\frac{n}{(\frac{\delta^2\eta^2}{4k^2})}\log^5(\frac{n}{(\frac{\delta^2\eta^2}{4k^2})})\Big)$, Lemma 5 implies that for any $1 \leq i \leq k$, with probability at least $1 - \frac{\delta}{2k}$ for any vector $v \in \Re^n$ we have

$$\begin{aligned}|v^T A_i'' v| &= \left|\left(\frac{1}{|T_i|}\sum_{y\in T_i-\mu_i}(v^T y)^2\right) - \mathsf{E}_{G_i}((v^T y)^2)\right| \\ &\leq \frac{\eta}{2}\mathsf{E}_{G_i}((v^T y)^2) = \frac{\eta}{2}v^T A_i v,\end{aligned}$$

which implies that $||\sum_{x\in T_i}(x-\mu_i)(x-\mu_i)^T|| \leq \frac{\eta}{2}||A_i|| \leq \frac{\eta}{2}\sigma_i^2$. Next, a simple calculation shows that

$$A_i' = A_i'' - (\mu_i - \mu_i')(\mu_i - \mu_i')^T,$$

which, together with $|\mu_i - \mu_i'| < \frac{\eta}{2}\sigma_i$, implies that $||A_i' - A_i|| < \eta\sigma_i^2$.

4.5. Proof of Lemma 12. In this section we prove Lemma 12. Our notation will be the same as in the statement of the lemma. Let $\mathcal{L}_v$ be the line in the direction of v. Let $t = \lceil\frac{30}{\varepsilon}\rceil$. Choose a sequence of closed intervals $I_1, \ldots, I_t$ on $\mathcal{L}_v$ such that the following hold:

- For any $1 \leq j \leq t$, we have $F(H_{I_j}) = \frac{1}{t}$.
- $\mathcal{L}_v = I_1 \cup \cdots \cup I_t$.
- The interiors of $I_1, \ldots, I_t$ are disjoint.

For any $1 \le j \le t$, let r_j be the number of samples in T_i which belong to H_{I_j}. Using Chernoff's inequality one can see that

$$\Pr\left(\left|\frac{r_j}{|T_i|} - \frac{1}{t}\right| > \frac{1}{100t}\right) < 2e^{-\frac{|T_i|}{10^4 t}}.$$

From Lemma 8(a) it follows that with probability at least $1 - \frac{\delta}{8k}$, for any j we have $|\frac{r_j}{|T_i|} - \frac{1}{t}| < \frac{1}{100t}$. Next, we show that when this holds, the inequality of the statement of the lemma holds as well.

Let I be an interval such that $F_i(H_I) = l \le \frac{\varepsilon}{4}$. Suppose a, b are chosen such that $I_{a+1} \cup \cdots \cup I_{b-1} \subset I \subset I_a \cup \cdots \cup I_b$. Then $(b-a+1) > lt$ and $b-a-1 < lt$. Therefore

$$\begin{aligned}\frac{|T_i \cap H_I|}{|T_i|} &< \frac{r_a + \cdots + r_b}{|T_i|} < (b-a+1)\left(\frac{1}{t} + \frac{1}{100t}\right) < (lt+2)\left(\frac{101}{100t}\right)\\ &= l + \frac{l}{100} + \frac{101}{50t} < l + \frac{\varepsilon}{8}\end{aligned}$$

and, similarly,

$$\begin{aligned}\frac{|T_i \cap H_I|}{|T_i|} &> \frac{r_{a+1} + \cdots + r_{b-1}}{|T_i|} > (b-a-1)\left(\frac{1}{t} - \frac{1}{100t}\right) > (lt-2)\left(\frac{99}{100t}\right)\\ &= l - \frac{1}{100t} - \frac{99}{50t} > l - \frac{\varepsilon}{8}.\end{aligned}$$

5. Concluding remarks. In this paper we showed that assuming a mild separation for the means a PCA-based learning algorithm provably classifies samples from a mixture of logconcave distributions.

From the example in Figure 1, it is not hard to see that spectral projection requires a separation between means that grows with the largest variance of individual components. Following the preliminary version of our results [18, 9], Achlioptas and McSherry [1] have improved the polynomial dependence on k and ε using a more sophisticated algorithm *after* projection. It remains an open problem to learn (nonspherical) Gaussians at smaller separation. The near disjointness of Gaussian components (e.g., total variation distance nearly 1 for two Gaussians), and thus their learnability, is implied by the assumption that the distance between two means is of the order of the variance *in the direction of* the line joining the means. Spectral projection fails at such a small separation, and a different technique will have to be used.

On the other hand, the main technique used in this paper is fairly easy to implement and commonly used in practice for many applications. In [20], Vempala and Wang present empirical evidence and propose an explanation for why the method is effective for real data where the assumption that the components are Gaussian (or logconcave) might not be valid.

Finally, most guarantees for spectral methods assume that the data is generated from some restricted model such as a random model. Our algorithm is also for "random" data, but the distributions considered are more general. Spectral projection seem to be well suited for such models, and our result can be viewed as further evidence of this.

Acknowledgment. We thank the referees for patiently reading the original version of the paper and making several useful suggestions.

REFERENCES

[1] D. Achlioptas and F. McSherry, *On spectral learning of mixtures of distributions*, in Learning Theory, Springer, Berlin, 2005, pp. 458–469.
[2] S. Arora and R. Kannan, *Learning mixtures of arbitrary Gaussians*, Ann. Appl. Probab., 15 (2005), pp. 69–92.
[3] S. Dasgupta, *Learning mixtures of Gaussians*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 634–644.
[4] S. Dasgupta and L. Schulman, *A two-round variant of EM for Gaussian mixtures*, in Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, 2000, pp. 152–159.
[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm. With discussion*, J. Roy. Statist. Soc. Ser. B, 39 (1977), pp. 1–38.
[6] A. Dinghas, *Über eine Klasse superadditiver Mengenfunktionale von Brunn–Minkowski–Lusternik-schem Typus*, Math. Z., 68 (1957), pp. 111–125.
[7] A. A. Giannopoulos and V. D. Milman, *Concentration property on probability spaces*, Adv. Math., 156 (2000), pp. 77–106.
[8] G. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1989.
[9] R. Kannan, H. Salmasian, and S. Vempala, *The spectral method for general mixture models*, in Learning Theory, Lecture Notes in Comput. Sci. 3559, Springer, Berlin, 2005, pp. 444–457.
[10] M. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*, MIT Press, Cambridge, MA, 1994.
[11] L. Leindler, *On a certain converse of Hölder's Inequality* II, Acta Sci. Math. (Szeged), 33 (1972), pp. 217–223.
[12] B. Lindsay, *Mixture Models: Theory, Geometry and Applications*, American Statistical Association, Alexandria, VA, 1995.
[13] L. Lovász and S. Vempala, *The geometry of logconcave functions and sampling algorithms*, Random Structures Algorithms, 30 (2007), pp. 307–358.
[14] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[15] A. Prékopa, *Logarithmic concave measures and functions*, Acta Sci. Math. (Szeged), 34 (1973), pp. 335–343.
[16] M. Rudelson, *Random vectors in the isotropic position*, J. Funct. Anal., 164 (1999), pp. 60–72.
[17] D. M. Titterington, A. F. M. Smith, and U. E. Makov, *Statistical Analysis of Finite Mixture Distributions*, Wiley, Chichester, UK, 1985.
[18] S. Vempala, *On the Spectral Method for Mixture Models*, IMA Workshop on Data Analysis and Optimization, 2003; http://www.ima.umn.edu/talks/workshops/5-6-9.2003/vempala/vempala.html.
[19] S. Vempala and G. Wang, *A spectral algorithm for learning mixtures of distributions*, J. Comput. System Sci., 68 (2004), pp. 841–860.
[20] S. Vempala and G. Wang, *The benefit of spectral projection for document clustering*, in Proceedings of the 3rd Annual Workshop on Clustering High Dimensional Data and its Applications, SIAM International Conference on Data Mining, 2005.
[21] C. F. J. Wu, *On the convergence properties of the EM algorithm*, Ann. Statist., 11 (1983), pp. 95–103.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1157–1174

IMPROVED APPROXIMATION ALGORITHMS FOR BROADCAST SCHEDULING*

NIKHIL BANSAL†, DON COPPERSMITH‡, AND MAXIM SVIRIDENKO†

Abstract. We consider scheduling policies in a client-server system where the server delivers data by broadcasting it to the users. In thesimplest model of the problem, there is a single server that holds n pages of unit size. Multiple requests for these pages arrive over time. At each time slot the server broadcasts exactly one page which satisfies all of the outstanding requests for this page at that time. We consider the problem of minimizing the average response time of requests, where the response time of the request is the duration since the request is placed until the time it is satisfied. For the offline version of this problem we give an algorithm with an approximation ratio of $O(\log^2(n)/\log\log(n))$. More generally, for any $\epsilon > 0$, the algorithm achieves an average response time of $(2+\epsilon)\cdot \mathrm{OPT} + O(\log n \cdot \log_{(1+\epsilon)} n)$, which is useful when the optimum value is large. This substantially improves the previously best known approximation factor of $O(\sqrt{n})$ for the problem [N. Bansal, M. Charikar, S. Khanna, and J. Naor, *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Vancouver, British Columbia, ACM, New York, SIAM, Philadelphia, 2005, pp. 215–221]. Our result is based on iteratively relaxing and rounding an auxiliary linear program derived from a natural linear programming relaxation of the problem.

Key words. LP rounding, approximation algorithm, broadcast scheduling

AMS subject classifications. 68W25, 68M20, 68Q25, 68W40, 90B35, 90C59

DOI. 10.1137/060674417

1. Introduction.

In a broadcast data dissemination system, we have a collection of data items at a broadcast server (typically a satellite). Users submit requests for these data items at various times and the server continuously transmits the data items. Whenever the server broadcasts a data item, it simultaneously satisfies all users waiting for that item. Broadcast systems have received a lot of attention recently. These systems exploit the fact that most requests are for a small common set of objects and they scale very well with increasing demand and number of users. Data broadcasting is actually being used in many commercial systems such as Intel Intercast system, Hughes DirecPC system [10], and the Airmedia system [1] to increase the bandwidth of the system. In fact broadcast is not unique to computer systems. There are several radio music channels where listeners submit the requests for songs, and then these songs eventually get played.

There has been a lot of research interest in various aspects of broadcast systems. In this paper we consider scheduling algorithms to improve the quality of service perceived by the users of the system. We focus on the average response time, which is one of the most commonly used measures of quality of service, defined as the average time a user waits until his request is satisfied.

Problem formulation. The setting and problem we study in this paper is formalized as follows: There is a collection of pages $P = \{1, \ldots, n\}$. Time is slotted and any page can be broadcast in a *single* time slot. At any time t, the broadcast server

*Received by the editors November 9, 2006; accepted for publication (in revised form) March 6, 2008; published electronically July 16, 2008.
http://www.siam.org/journals/sicomp/38-3/67441.html
†IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (nikhil@us.ibm.com, sviri@us.ibm.com).
‡IDA Center for Communications Research, Princeton, NJ 08540 (dcopper@idaccr.org). This work was done while this author was at IBM T.J. Watson Research Center.

receives $n_p(t)$ requests for page $p \in P$. We say that a request ρ for page p that arrives at time t is satisfied at time $c_p(t)$, if $c_p(t)$ is the first time after t when page p is transmitted by the broadcast server. The *response time* of the request ρ is defined to be the time that elapses from its arrival till the time it is satisfied, i.e., $c_p(t) - t$. We assume that request ρ arrives in the end of the time slot t and, therefore, cannot be satisfied in the time slot in which it arrived; i.e., the response time for any request is at least 1. Let T denote the last time when any request arrives.

We consider the average response time minimization problem, where we want to find a broadcast schedule that minimizes the average response time, defined to be $(\sum_{p\in P}\sum_{t=1}^{T} n_p(t)(c_p(t)-t))/(\sum_{p\in P}\sum_{t=1}^{T} n_p(t))$. In this paper we study the offline problem, where the request sequence is known in advance to the scheduling algorithm.

The problem can also be viewed as the response time minimization version of the minimum latency problem (aka traveling repairman problem) on a uniform metric space. In the classical minimum latency problem [5, 8, 16], we wish to find a tour on n points in a metric space which minimizes the average time a vertex is visited. This can be viewed as a request arriving at each vertex at time 0, and the goal is to satisfy the average response time of the requests. One can consider a stronger variant of the minimum latency problem when requests at vertices arrive over time, and, moreover, several requests might arrive at a vertex at different times. By associating each vertex with a page, and viewing every visit to a vertex as a transmission of the corresponding page, it can be seen that the problem considered in this paper is identical to the response time version of the latency problem for the uniform metric space.

Previous work. The average response time problem was shown to be NP hard by Erlebach and Hall [13]. Most of the previous algorithmic work has focused on resource augmentation where the server is given extra speed compared to the optimal algorithm. These results compare k-speed approximation algorithm against the performance of an optimal 1-speed algorithm, where a k-speed algorithm is one that allows a server to broadcast k pages in each time slot. Kalyanasundaram, Pruhs, and Velauthapillai [17] gave the first $\frac{1}{\alpha}$-speed, $\frac{1}{1-2\alpha}$-approximation algorithm for any fixed α, for $0 \le \alpha \le 1/3$. This guarantee was improved in a sequence of papers [14, 13, 15]. Gandhi et al. [14] gave a $\frac{1}{\alpha}$-speed, $\frac{1}{1-\alpha}$-approximation algorithm for any $\alpha \in (0, 1/2]$. Erlebach and Hall [13] gave a 6-speed 1-approximation algorithm for the problem, which was improved to a 4-speed, 1-approximation algorithm by [14]. Then, Gandhi et al. [15] gave a 2-speed, 1-approximation. Subsequently, Bansal et al. [2] gave an algorithm that achieves a constant approximation ratio for arbitrarily small extra speed up factor. Their algorithm achieves an additive approximation ratio of $O(1/\epsilon)$ with $(1+\epsilon)$-speed.[1]

When no extra speed is allowed, the problem seems to be considerably harder. Note that repeatedly transmitting the pages in the cyclic order $1, \ldots, n$ is an $O(n)$ approximation, as every request has a response time of at most n in the schedule above, and at least 1 in any schedule. Prior to our work, the only result with a guarantee better than the naive $O(n)$ was due to Bansal et al. [2]. Their algorithm produces a solution with an average response time of $\mathrm{OPT} + O(\sqrt{n})$, where OPT is the optimum average response time. We note that this is an additive $O(\sqrt{n})$ approximation, hence it directly implies an $O(\sqrt{n})$ multiplicative approximation (since OPT is at least 1). Alternately, the result can be viewed as an $O(\max(1, \sqrt{n}/\mathrm{OPT}))$ multiplicative

[1] Here $(1+\epsilon)$-speed means that the algorithm is allowed to transmit one extra page every $1/\epsilon$ time steps.

approximation. Furthermore, for any $\epsilon > 0$, the same algorithm achieves the $(1+\epsilon)$-speed, $O(1/\epsilon)$-additive approximation guarantee mentioned earlier.

In the online setting, a lower bound of $\Omega(\sqrt{n})$ without speedup and a lower bound of $\Omega(1/\epsilon)$ with a speedup factor of $(1+\epsilon)$ on the competitive ratio of any randomized online algorithm is known [2]. In [17, Lemma 7], an $\Omega(n)$ lower bound on the competitive ratio of deterministic algorithms is given. Edmonds and Pruhs [11] gave a $(4+\epsilon)$-speed, $O(1+1/\epsilon)$-competitive online algorithm. Later, they [12] showed that a natural algorithm, Longest Wait First, is 6-speed, $O(1)$-competitive. Recently, Robert and Schabanel have extended these results to the setting where jobs can have dependencies [19].

Other measures have also been studied in the broadcast setting. Bartal and Muthukrishnan [4] considered the problem of minimizing the maximum response time of a request and gave an $O(1)$-competitive algorithm. Charikar and Khuller [7] consider a generalization of the above problem where the goal is to minimize the maximum response time of a certain fraction of the requests. The profit maximization version of the problem has also been studied. Here each request also has a deadline and a profit which is obtained if this request is satisfied before its deadline. This problem was first studied by Bar-Noy et al. [3] (in a more general setting), who gave a 1/2-approximation algorithm for it. Later, Gandhi et al. [15] designed a 3/4-approximation algorithm for the problem by using the dependent randomized rounding technique. Recently, Chang et al. [6] showed that this problem is NP-hard even in the special case when all profits are equal to 1 (referred to as the throughput maximization problem).

Our results. Our main result is an $O(\log^2 n/\log\log n)$-approximation algorithm for minimizing average response time in the absence of extra speed. In fact the bound above follows from a more general guarantee of $(2+\gamma)\cdot \mathrm{OPT} + O(\log_{1+\gamma} n \cdot \log n)$, where $\gamma > 0$ is an arbitrary positive parameter and OPT denotes the value of the optimum solution. The bound $O(\log^2 n/\log\log n)$ follows by choosing $\gamma = \Theta(\log n)$. Setting γ arbitrarily close to 0 implies a guarantee of $(2+\gamma)\mathrm{OPT} + O(\log^2 n/\gamma)$, which could be more useful in cases when OPT is large. For example, this implies an $O(1)$ approximation when OPT is $\Omega(\log^2 n)$.

Organization. We begin by defining a natural integer programming formulation in section 1 and discuss some properties of its linear programming (LP) relaxation. Section 2 describes the high level idea of our algorithm, and we present the algorithm and its analysis in section 3. The analysis is split into two parts. For simplicity, we first show a guarantee of $3\cdot \mathrm{OPT} + O(\log^2(T+n))$, where T is the time horizon. Then, in section 3.5 we show how to remove the dependence on T, and show how to refine the guarantee further to obtain an $O(\log^2 n/\log\log n)$ approximation. Finally, in section 4 we give an example of a solution to the LP relaxation where every local rounding procedure (defined below) incurs a large gap. This essentially implies that techniques used prior to our work are unlikely to yield an approximation ratio better than $\Theta(\sqrt{n})$.

Preliminaries. We begin by considering an LP relaxation of a natural time-indexed integer linear program (ILP) for the response time minimization problem. This formulation is also the starting point of all previously known approximation algorithms for this problem [17, 14, 15, 2].[2]

Consider the following time-indexed integer programming formulation: For each page $p = 1, \ldots, n$ and each time t', there is a variable $y_{pt'}$ which indicates whether

[2]Strictly speaking, [17] considered a different LP formulation. However, it is equivalent [18] to the one considered in this paper.

page p was transmitted at time t'. In particular, $y_{pt'}$ is 1 if p is broadcast at time t' and 0 otherwise. We have another set of variables $x_{ptt'}$ indexed by a page p and two times t and t' such that $t' > t$. These are used to model the response of a request. In particular, $x_{ptt'} = 1$ if a request for page p that arrived at time t is satisfied at time $t' > t$ and 0 otherwise. Let n_{pt} denote the number of requests for page p that arrive at time t. The following integer program is an exact formulation of the problem:

$$\min \sum_{p} \sum_{t} \sum_{t'=t+1}^{T+n} (t' - t) \cdot n_{pt} \cdot x_{ptt'} \tag{1}$$

$$\text{subject to} \quad \sum_{p} y_{pt'} \leq 1 \quad \forall t', \tag{2}$$

$$\sum_{t'=t+1}^{T+n} x_{ptt'} \geq 1 \quad \forall p, t, \tag{3}$$

$$x_{ptt'} \leq y_{pt'} \quad \forall p, t, t' > t, \tag{4}$$

$$x_{ptt'} \in \{0, 1\} \quad \forall p, t, t', \tag{5}$$

$$y_{pt'} \in \{0, 1\} \quad \forall p, t'. \tag{6}$$

Here T denotes the last time when any request arrives. Observe that it suffices to define variables only until time $t = T + n$, as all of the requests can be satisfied by transmitting page p at time $T + p$ for $p = 1, \ldots, n$. The set of constraints (2) ensures that at most one page is transmitted in each time slot. The set of constraints (3) ensures that each request must be satisfied and, finally, the set of constraints (4) ensures that a request for page p can be satisfied at time t only if p is transmitted at time t. A request that arrives at time t and is satisfied at time t' contributes $(t' - t)$ to the objective function. Without loss of generality, it can be assumed that in any optimum solution the constraints (2) are satisfied with an equality and that for every p and t, the variable $x_{ptt'}$ is equal to 1 at the earliest time $t' > t$ such that $y_{pt'} = 1$.

As stated the size of the formulation is polynomial in the time horizon T which could be arbitrarily large. However, we can assume that the input size of the problem is at least T/n, since if there is a period of n consecutive time units when no page is requested, then we can split the problem into two disjoint instances.

Consider the linear program obtained by relaxing the integrality constraints on $x_{ptt'}$ and y_{pt}. This fractional relaxation may be viewed as broadcasting pages fractionally at each unit of time such that total fraction of all the pages broadcast in any unit of time is 1. A request for a page p arriving at a time t is considered completely satisfied at time t' if t' is the earliest time such that the total amount of page p broadcast during the interval $(t, t']$ is at least 1.

Let (x^*, y^*) be some optimum solution to the fractional relaxation of the program (1)–(6). It is easily seen that the value of variables (x^*) are completely determined by the values of variables (y^*). Since each page needs one unit of transmissions to be satisfied, we can assume that for each p and t, the variable $x^*_{ptt'}$ is greedily set equal to $y^*_{pt'}$ starting from $t' = t + 1$ until one unit of page p has been transmitted. Equivalently, for each page p, time t, and any time $\tau > t$,

$$\sum_{t'=\tau}^{\infty} x_{ptt'} = \max \left\{ 0, \left(1 - \sum_{t''=t+1}^{\tau-1} y_{pt''} \right) \right\}. \tag{7}$$

For each page p and time t, the solution (x^*, y^*) determines a (fractional) response time $r(p,t)$ for the request for p that arrives at time t. In particular, $r(p,t) = \sum_{t'>t}(t'-t)x^*_{ptt'}$. As $(t'-t)$ can be written as $\sum_{\tau=t+1}^{t'} 1$, we can rewrite $r(p,t) = \sum_{t'>t}\sum_{\tau=t+1}^{t'} x^*_{ptt'}$. By interchanging the order of summation, we get that $r(p,t) = \sum_{\tau>t}\sum_{t'\geq\tau} x^*_{ptt'}$. By (7), it follows that for each page p and time t we have

$$r(p,t) = \sum_{\tau>t} \max\left\{0, 1 - \sum_{t''=t+1}^{\tau-1} y_{pt''}\right\}. \tag{8}$$

The following is an extremely useful view of response times that we will use repeatedly.

LEMMA 1. *Consider a request for a page p that arrives at time t. For $\alpha \in (0,1]$, let $t(\alpha,p)$ denote the earliest time after t such that the cumulative amount of page p broadcast by the LP during the interval $[t+1, t(\alpha,p)]$ is at least α. Then $r(p,t) = \int_0^1 (t(\alpha,p)-t)d\alpha$.*

Equivalently, if we choose α uniformly at random in $(0,1]$ and transmit page p at time $t(\alpha,p)$, then the expected response time for this request is equal to the LP cost for this request.

Proof. Consider some time $\tau > t$. Since α is chosen uniformly at random in $(0,1]$, the probability that $t(\alpha,p) \geq \tau$ is exactly equal to the probability that $\alpha > \sum_{t''=t+1}^{\tau-1} y_{pt''}$, which is exactly equal to $\max\{0, 1 - \sum_{t''=t+1}^{\tau-1} y_{pt''}\}$. Now,

$$\begin{aligned}\int_0^1 (t(\alpha,p)-t)d\alpha &= \mathbb{E}_\alpha[t(\alpha,p)-t] \\ &= \sum_{\tau>t} \Pr[t(\alpha,p) \geq \tau] \\ &= \sum_{\tau>t} \max\left\{0, 1 - \sum_{t''=t+1}^{\tau-1} y_{pt''}\right\}\end{aligned}$$

which is exactly equal to $r(p,t)$ by (8), and hence the result follows. ∎

2. Algorithm overview and techniques. The algorithm begins by solving the LP relaxation of (1)–(6). Our high level approach is the same as the one introduced in [2]. We first produce a *tentative* schedule which has a good response time, but can violate the "capacity" constraints (2), by transmitting more than one page during certain time slots. However, this violation will be bounded as explained in condition 2 below. In particular, suppose this tentative schedule satisfies the following properties:

1. The total response time for this schedule is at most $c = O(1)$ times the cost of the LP relaxation.
2. The capacity constraints are satisfied approximately in the following sense. For any interval of time $(t,t']$, the total number of pages broadcast by the tentative schedule during $(t,t']$ is at most $t'-t+b$, for some b independent on the time interval $(t,t']$. We refer to this b as the *backlog* of the schedule.

In this case, the tentative schedule can be transformed into a valid schedule as follows: We transmit pages in the same order as the tentative schedule while ensuring that no page is transmitted at an earlier time than in the tentative schedule. It is not hard to see that the backlog property ensures that no page is transmitted more than b steps later than in the tentative schedule (see Lemma 8 for a formal proof). Thus, this

produces an integral solution with average response time $c \cdot \mathrm{OPT} + b$. The $O(\sqrt{n})$-approximation algorithm of [2] was based on obtaining a tentative solution with cost equal to the LP cost (i.e., $c = 1$) and backlog $O(\sqrt{n})$.

In this paper we will give a procedure to obtain a tentative schedule with $c = 2+\gamma$ and $b = O(\log_{1+\gamma}(T+n) \cdot \log(T+n))$, for any arbitrary $\gamma > 0$. By Lemma 11, this will imply the desired approximation guarantee of $(2+\gamma)\mathrm{OPT} + O(\log_{1+\gamma} n \cdot \log n)$. Our improved approximation is based on two new ideas: First, we relax a locality requirement (explained below) in the rounding procedure. Prior to our work, all algorithms were *local* in the following sense: Given a solution to the LP formulation, they produced a schedule (or a tentative schedule) that ensured that for every interval, if the LP solution transmits more than one unit of page p during this interval, then the rounded solution has at least one transmission of p during this interval. The main reason for enforcing this locality was that the response time for each request can be charged directly to its cost in the LP solution, which makes the analysis relatively simple. Interestingly, in section 4 we show that relaxing this locality requirement seems necessary to obtain approximation ratios better than $\Theta(\sqrt{n})$. In particular, we show that there exist LP solutions such that any rounding procedure that is local must have backlog at least $\Omega(\sqrt{n})$.

In our rounding procedure we relax the locality requirement so that it does not necessarily hold for all time intervals. In particular, for each page p we partition the time horizon, $1, \ldots, T+n$, into intervals $B(p,i)$ called blocks, where $B(p,i)$ refers to the ith block for page p. These intervals are small in the sense that the cumulative amount of page p transmitted by the LP during $B(p,i)$ is $O(\log(T+n))$. We require only that the rounding be local within each block $B(p,i)$. While this could lead to some requests (that lie at the interface of two blocks) to pay much more than their LP costs, our technique for constructing $B(p,i)$ ensures that the response times of these requests are not too much more.

The second part of the algorithm is to give a scheme to choose local schedules for each block $B(p,i)$ such that when the tentative schedule is constructed by merging these local schedules for all the pages, it satisfies the following two properties. First, the cost of the tentative schedule is bounded by a constant times the optimum, and second, the backlog of the tentative schedule is bounded by $O(\log^2(T+n))$. To do this, we solve a sequence of linear programs iteratively. We begin by defining a (different) linear program (LP') where the variables correspond to the possible local schedules that can be chosen for each block $B(p,i)$. LP' has the property that there is a feasible fractional solution where the backlog is 0 and the response time is only $O(1)$ times the optimum cost. Observe that a solution to LP' can be viewed as assigning a linear combination of local schedules to each block. Our novel step is to show that the number of constraints in LP' can be reduced in such a way that in any basic solution to this reduced linear program, at least a constant fraction of blocks gets assigned exactly one local schedule, and yet the backlog increases only by $O(\log(T+n))$. Since this linear program has fewer constraints, the objective function can only be better. Since at least a constant fraction of blocks gets assigned exactly one schedule, we can remove these blocks to obtain a smaller problem instance. This allows us to apply this procedure iteratively, where we successively relax the constraints. At each iteration, the problem size decreases geometrically, the backlog increases additively by $O(\log(T+n))$, and the objective function does not worsen. After $O(\log(T+n))$ iterations the procedure terminates and we obtain a tentative schedule with backlog $O(\log^2(T+n))$, and cost equal to $O(1)$ times the optimum cost. We now describe the

details.

3. Minimizing average response time. Our algorithm begins by solving the LP (1)–(6). Fix an optimum solution (x^*, y^*) to this LP, and let $r(p,t)$ denote the response time according to the LP solution for a request for page p at time t, that is $r(p,t) = \sum_{t'>t}(t'-t)x^*_{ptt'}$. Let $c(p,t,t')$ denote the cumulative amount of page p transmitted by the LP solution during the time interval $(t,t']$, that is $c(p,t,t') = \sum_{t''=t+1}^{t'} y^*_{pt''}$. In the rest of this paper we refer to the time interval $(t-1,t]$ for $t \geq 1$ as the *time slot* t. We now define the key concept of p-good time slots which will be used to form the blocks for each page.

3.1. Blocks and p-good time slots.

DEFINITION 2 (p-good time slot). *Let $r(p,t)$ denote the response times as determined by the solution to the LP* (1)–(6). *For a page p we say a time slot t is p-good if $r(p,t) \leq 2r(p,\tau)$ for all $\tau < t$ such that $c(p,\tau,t) \leq 1$.*

Intuitively, the LP response time for page p at a p-good time t is not much more than that at any other time τ, where τ is sufficiently close to t in the sense defined above. The following lemma shows that a p-good slots can be found in any interval of time that broadcasts a sufficient amount of page p.

LEMMA 3. *Any time interval $(t,t']$ such that $c(p,t,t') > \log(T+n)$ contains a p-good slot.*

Proof. For the sake of contradiction, suppose that none of the slots in the interval $(t,t']$ are p-good. Since t' is not p-good, there exists a $t_1 < t'$ such that $c(p,t_1,t') \leq 1$ and $r(p,t_1) > 2r(p,t')$. Note that t_1 lies in the interval (t,t') and, hence, is not p-good by our assumption. Thus there exists t_2 such that $c(p,t_2,t_1) \leq 1$ and $r(p,t_2) > 2r(p,t_1)$. Repeating the argument for $k = \log(T+n)$ steps, we obtain a sequence of slots $t < t_k < \cdots < t_2 < t_1 < t'$ such that $c(p,t_k,t') \leq \log(T+n)$ and $r(p,t_k) > 2^k r(p,t') = (T+n)\cdot r(p,t')$, which is impossible as the response time for any request is bounded between 1 and $(T+n)$. ∎

Lemma 3 implies that if t is a p-good slot, and such that $c(p,t,T+n) > \log(T+n)$, then there is another p-good slot $t' > t$ such that $c(p,t,t') \leq \log(T+n)$. Thus, for each p we can form a collection

$$G(p) = \{0 = t(p,0), t(p,1), t(p,2), \ldots, t(p,b_p) = (T+n)\}$$

of time slots such that the $t(p,i)$ is p-good for $1 \leq i \leq b_p - 1$ and $1 \leq c(p,t(p,i-1),t(p,i)) \leq 1 + \log(T+n)$ for all $1 \leq i \leq b_p - 1$. The last interval must be such that $1 \leq c(p,t(p,b_p-1),t(p,b_p)) \leq 2 + \log(T+n)$, and such a collection of slots can be formed by a simple iterative greedy strategy.

Indeed, let $t(p,s)$ be the current p-good point. Let $t(p,s,1)$ be the next p-good point. By Lemma 3 $c(p,t(p,s),t(p,s,1)) \leq \log(T+n)$. If $c(p,t(p,s),t(p,s,1)) \geq 1$, then $t(p,s+1) = t(p,s,1)$. Otherwise, let $t(p,s,2)$ be the next p-good point. By Lemma 3 $c(p,t(p,s,1),t(p,s,2)) \leq \log(T+n)$ and therefore $c(p,t(p,s),t(p,s,2)) \leq 1 + \log(T+n)$. If $c(p,t(p,s),t(p,s,2)) \geq 1$, then $t(p,s+1) = t(p,s,2)$. Otherwise, we define $t(p,s,3)$ and continue until we either define $t(p,s+1)$ or hit the end of the planning horizon. If the last interval produced by this greedy strategy has length smaller than 1, then we merge this interval and the second-last interval (this is why we have slack for the size of the last interval).

We call the time intervals $(0 = t(p,0), t(p,1)], (t(p,1), t(p,2)], \ldots, (t(p,b_p-1), t(p,b_p)]$, blocks for page p. Note that there are b_p blocks for page p. We will use $B(p,i) = (t(p,i-1), t(p,i)]$ to denote the ith block for page p. Let $\mathcal{B}_p$ denote the set

of all blocks for page p, and let $\mathcal{B} = \cup_p \mathcal{B}_p$ denote the set of all blocks. For a block $B(p,i)$, we define its *tail* to be the time slots t such that $c(p,t,t(p,i)) < 1$. That is, the cumulative amount of page p transmitted after time t until the end of the block in which t lies is less than 1. Obviously, there is a tail for each block $B(p,i)$ since $c(p,t(p,i-1),t(p,i)) \geq 1$. Note that if a request for page p arrives during the tail of a block, then it is not satisfied completely within that block by the LP solution.

Let us focus on a particular block, say $B(p,i)$. For $\alpha \in (0,1]$, $l = 0,1,2,\ldots$, and block $B(p,i)$, let $t(p,i,l,\alpha)$ denote the earliest time in $B(p,i)$ when an $l+\alpha$ amount of page p has been broadcast since the start of $B(p,i)$; i.e., an $(l+\alpha)$ amount of page p has been broadcast during $(t(p,i-1),t(p,i,l,\alpha)]$ (the time $t(p,i,l,\alpha)$ is defined only if block $B(p,i)$ transmits at least $l+\alpha$ amount of page p). For a given block $B(p,i)$ and $\alpha \in (0,1]$, let $C(p,i,\alpha)$ denote the set of all time slots $t(p,i,l,\alpha)$ for $l = 0,1,2,\ldots$.

Note that if we transmit page p at slots in $C(p,i,\alpha)$ for some $\alpha \in (0,1]$, then this is a local schedule for $B(p,i)$ (since we transmit page p whenever the LP transmits one unit of page p during any time interval contained in $B(p,i)$). We say that this local schedule is formed by choosing the *offset* α for block $B(p,i)$. We will only be interested in local schedules for blocks that are obtained by choosing some offset $\alpha(p,i)$ for each block $B(p,i)$ and transmitting page p at all time slots in $C(p,i,\alpha(p,i))$. Suppose we arbitrarily choose some offset $\alpha(p,i)$ for each block $B(p,i)$. We claim that the tentative schedule thus obtained satisfies each request. To see this, observe that each request for page p that arrives in $B(p,i)$ for $1 \leq i \leq b_p - 1$ (i.e., except for the last block for page p) is served within $B(p,i)$ or in $B(p,i+1)$. Finally, as there is at least one unit of page p broadcast in the LP solution after the arrival of the last request for page p (see comment after the definition of LP (1)–(6), all requests for page p that arrive during the last block $B(p,b_p)$ are served within $B(p,b_p)$.

The following lemma shows that if we construct a tentative schedule by choosing an offset independently at random for each block, then the expected total response time is not too high and all of the capacity constraints are satisfied at each time step in expectation.

LEMMA 4. *Suppose for each block $B(p,i) \in \mathcal{B}$, we choose the offset $\alpha(p,i)$ independently and uniformly at random in $[0,1]$. Then the tentative schedule thus obtained satisfies the following properties:*

1. *The expected number of pages transmitted at any time step t is exactly* 1.
2. *For each request, its expected response time is at most* 3 *times the cost incurred by it in the LP solution of* (1)–(6).

Proof. Consider a time t and suppose that $t \in B(p,i)$. Since we choose the offset $\alpha(p,i)$ uniformly at random in $(0,1]$, the probability that page p is transmitted at time t is exactly y^*_{pt}.

For any p, the blocks $B(p,i)$ partition the entire time interval $(0,T+n]$, and hence for each time t there is exactly one block for page p that contains t. Thus, the probability that page p is transmitted at time t in the tentative schedule is exactly y^*_{pt}. Summing up over all of the pages, we have that the expected number of pages transmitted at time t is exactly $\sum_p y^*_{pt}$ which is exactly 1 by constraints (2) in the LP. This proves the first part of the lemma.

Consider a particular block $B(p,i)$, and let ρ be a request for page p that arrives during $B(p,i)$. We say that request ρ is *early* if it does not arrive in the tail of $B(p,i)$, or, equivalently, that $c(p,t_\rho,t(p,i)) \geq 1$. Note that ρ is always served within $B(p,i)$ irrespective of the choice of $\alpha(p,i)$. Since $\alpha(p,i)$ is chosen uniformly at random in $(0,1]$, by Lemma 1 the expected response time for an early request is exactly $r(p,t)$.

Thus it suffices to focus on the contribution of requests that are not early. Consider a request ρ for page p that arrives at time $t_\rho \in B(p,i)$ such that $c(p,t_\rho,t(p,i)) < 1$. Since $\alpha(p,i)$ is chosen uniformly at random, with probability $c(p,t_\rho,t(p,i))$ this request is served in $B(p,i)$ and with probability $1-c(p,t_\rho,t(p,i))$ it is served in $B(p,i+1)$. Since $\alpha(p,i+1)$ is chosen uniformly at random in $(0,1]$ and is independent of $\alpha(p,i)$, it follows that conditioned on the event that ρ is served in $B(p,i+1)$, by Lemma 1 its expected response time is $(t(p,i)-t_\rho)+r(p,t(p,i))$. Thus the overall expected response time of ρ is

$$\begin{aligned}
&\left(\sum_{t''=t_\rho+1}^{t(p,i)} (t''-t_\rho)\cdot x^*_{pt_\rho t''}\right) + (1-c(p,t_\rho,t(p,i)))\cdot(t(p,i)-t_\rho+r(p,t(p,i)))\\
&= \left(\sum_{t''=t_\rho+1}^{t(p,i)} (t''-t_\rho)\cdot x^*_{pt_\rho t''}\right) + \left(\sum_{t''=t(p,i)+1}^{\infty} (t(p,i)-t_\rho)\cdot x^*_{pt_\rho t''}\right)\\
&\quad +(1-c(p,t_\rho,t(p,i)))\cdot r(p,t(p,i))\\
&\le \left(\sum_{t''=t_\rho+1}^{t(p,i)} (t''-t_\rho)\cdot x^*_{pt_\rho t''}\right) + \left(\sum_{t''=t(p,i)+1}^{\infty} (t''-t_\rho)\cdot x^*_{pt_\rho t''}\right)\\
&\quad +(1-c(p,t_\rho,t(p,i)))\cdot r(p,t(p,i))\\
&\le \left(\sum_{t''=t_\rho+1}^{\infty} (t''-t_\rho)\right)\cdot x^*_{pt_\rho t''} + r(p,t(p,i)) \le r(p,t_\rho)+2r(p,t_\rho) = 3r(p,t_\rho).
\end{aligned}$$

The first equality follows by observing that

$$\left(\sum_{t''=t(p,i)+1}^{\infty} x^*_{pt_\rho t''}\right) = 1-c(p,t_\rho,t(p,i)).$$

The final step follows as $t(p,i)$ is a p-good time slot; hence by definition $r(p,t(p,i)) \le 2r(p,t_\rho)$. □

The following lemma will allow us to consider discrete choices for the offsets $\alpha(p,i)$.

LEMMA 5. *Let $\varepsilon > 0$ be an arbitrary precision parameter. We can assume that $x_{ptt'}$ and y_{pt} are integral multiples of $\delta = \varepsilon/(T+n)^2$. This adds at most ε to the response time of each request.*

Proof. Given an arbitrary LP solution, we simply round down the values of y_{pt} to the closest multiple of δ and modify $x_{ptt'}$ accordingly. We also transmit $\delta\cdot T \le 1/n$ units of each page p at time $T+n+1$ to ensure that each request remains completely satisfied.

Observe that each $x_{ptt'}$ is reduced by at most δ. As the response time for a request for page p at time t is $\sum_{t'>t}(t'-t)x_{ptt'}$, the rounding adds at most $T\cdot T\cdot\delta \le \varepsilon$ to the response time of each request. □

Thus we can assume that all of the offsets $\alpha(p,i)$ are integral multiples of δ for $\delta = \varepsilon/(T+n)^2$. Henceforth, we will use $B(p,i,j)$ to denote the time slots in $C(p,i,\alpha=\delta j)$ (i.e., those obtained by choosing $\alpha(p,i)=\delta j$), where $j=0,\ldots,1/\delta-1$. We will call the set of time slots $B(p,i,j)$ a *block-offset*.

3.2. Auxiliary LP. Recall that our goal is to choose exactly one offset for each block in such a way that the total response time of the tentative schedule thus obtained is not too high, and the backlog is small. For this purpose we define the LP (9)–(12).

We have variables z_{pij} that correspond to choosing the offset j for block $B(p,i)$, or equivalently transmitting the page p at times in $B(p,i,j)$ during $B(p,i)$. The parameters $R(B(p,i,j))$ and $w(B(p,i,j),t)$ are defined below.

$$\min \sum_p \sum_i \sum_j R(B(p,i,j)) \cdot z_{pij} \tag{9}$$

$$\text{subject to} \quad \sum_j z_{pij} = 1 \quad \forall p,i, \tag{10}$$

$$\sum_p \sum_i \sum_j w(B(p,i,j),t) \cdot z_{pij} = 1 \quad \forall t, \tag{11}$$

$$z_{pij} \geq 0 \quad \forall p,i,j. \tag{12}$$

Here $w(B(p,i,j),t)$ is an indicator function: $w(B(p,i,j),t)$ is 1 if $t \in B(p,i,j)$ and 0 otherwise. Observe that the constraints (10) require that there is exactly one unit of offsets chosen for each block $B(p,i)$, and the constraints (11) require that for each time slot t the total amount of pages transmitted is exactly 1. Thus, this LP can be viewed as choosing a convex combination of local schedules for each block $B(p,i)$ such that certain global constraints are satisfied.

The objective function of minimizing the total response time is expressed in terms of the variables z_{pij} as follows: For each block $B(p,i)$ we associate a *block-offset response time* $R(B(p,i,j))$ which essentially accounts for the contribution of the block-offset $B(p,i,j)$ to the total response time. Observe that choosing an offset for block $B(p,i)$ can affect the response time of requests for page p that arrive in $B(p,i)$ and possibly the requests that arrive during the tail of $B(p,i-1)$. The block-offset response time $R(B(p,i,j))$ is computed as follows:

1. Let t' denote the earliest time in $B(p,i,j)$. Each request for page p in the tail of the previous block $B(p,i-1)$ contributes $t' - t(p,i-1)$ to $R(B(p,i,j))$. Note that this is the amount of time (restricted to time slots in $B(p,i)$) that any request in $B(p,i-1)$ might possibly have to wait until it is satisfied.
2. For a request ρ for page p that arrives at time t where $t \in B(p,i)$, we do the following. Let t' denote the earliest time such that $t' > t$ and $t' \in B(p,i,j)$. If such a t' does not exist, then we set $t' = t(p,i)$. Then the request ρ contributes exactly $t' - t$ to $R(B(p,i,j))$. Note that this quantity is the contribution to the response time of ρ restricted to the time slots in $B(p,i)$.

This definition of $R(B(p,i,j))$ ensures that for any tentative schedule obtained by choosing one offset for each block, i.e., for any setting of z_{pij} to 0 or 1 subject to the constraints (10), the total response time of this tentative schedule is no more than $\sum_{p,i,j} R(B(p,i,j)) \cdot z_{pij}$.

We next observe that the linear program above has a good fractional solution. In particular, Lemma 4 implies the following about the linear program defined by LP (9)–(12).

LEMMA 6. *There is a feasible solution to the LP* (9)–(12) *with cost no more than* 3 *times the cost of the optimum value of the LP* (1)–(6).

Proof. Consider the solution where $z_{pij} = \delta$ for all p,i and $0 \leq j \leq 1/\delta - 1$. This corresponds to choosing an offset uniformly at random for each block $B(p,i)$. Lemma 4 (part 1) implies that the constraints (11) are satisfied.

We now show that the cost of this solution is no more than 3 times the cost of LP (1)–(6). Consider an early request in $B(p,i)$. Since this request is completely served within $B(p,i)$ by Lemma 1, the contribution of this request to the objective function is exactly its response time. For a request for page p that arrives at time t in the tail of $B(p,i-1)$, its contribution to the objective function corresponding to $\sum_j z_{p,i-1,j} \cdot R(B(p,i-1,j))$ is exactly $\sum_{t''=t+1}^{t(p,i)}(t''-t)x^*_{ptt''} + (1-c(p,t,t(p,i))) \cdot (t(p,i)-t)$ which is at most $r(p,t)$. Similarly, the contribution to $\sum_j z_{p,i,j} \cdot R(B(p,i,j))$ is exactly $r(p,t(p,i-1))$ (by Lemma 1), which is at most $2r(p,t)$ by the definition of a p-good point and as t lies in tail of $B(p,i-1)$. □

Observe that a solution to the LP (9)–(12) satisfies the capacity constraints (11) exactly. Hence if this LP had an integral optimum solution, then the backlog of the tentative schedule implied by this solution would be 0; we would then have an exact schedule with cost at most 3 times the optimum. However, this LP in general could have a fractional optimum solutions. The rest of the algorithm will deal with obtaining an integral solution to the above LP by successively relaxing the capacity constraints (11) but still ensuring that the quality of the solution remains reasonably good.

3.3. The algorithm. The idea for our algorithm is the following: Suppose we relax the capacity constraints (11) in the auxiliary LP such that we require them only to hold for time intervals of integer size b' rather than for each time unit. That is, we require only that intervals of size b' contain exactly b' pages, but we do not care how these pages are transmitted during an interval. An important observation is that if this relaxed LP had an integral optimum solution, it would give a tentative schedule with backlog $2b'$ and cost at most 3 times the optimum (and we would obtain an approximation guarantee of $3 \cdot \mathrm{OPT} + 2b'$). Since this is too strong to hope for, we will show something weaker. We will show that for $b' = O(\log(T+n))$, there is an optimum solution to this LP where more than half of the blocks $B(p,i)$ have some z_{pij} set integrally to 1. Armed with this result, we will define a smaller problem instance by removing the blocks $B(p,i)$ that have some $z_{pij} = 1$. We then apply this procedure to this smaller problem by redefining the intervals and the LP suitably and relaxing the capacity constraints of the type (11). We repeat this for $O(\log(T+n))$ steps, until we obtain a problem where only a constant number of blocks remain which can be easily solved by trying all possibilities. At a high level, we add $O(\log(T+n))$ to the backlog at each iteration of this process, which will imply that in the end we obtain a tentative schedule with cost at most 3 times the optimum and backlog of $O(\log^2(T+n))$. We now make these arguments precise.

Before we can describe the algorithm to compute the tentative schedule formally, we need some notation. Let $I = (t_1, t_2]$ be a collection of time slots $t_1+1, \ldots, t_2$. We refer to I as an interval. The size of I denoted by $\mathrm{Size}(I)$ is defined as $t_2 - t_1$. The weight of an interval with respect to $B(p,i,j)$, which we denote by $w(B(p,i,j),I)$, is the cardinality of the set $B(p,i,j) \cap I$. That is, $w(B(p,i,j),I)$ is the number of time slots in the interval I that belong to $B(p,i,j)$.

Our algorithm will solve a sequence of LP's. At step k, some variables z_{pij} that were fractional (not 0 or 1) at the end of step $k-1$ get assigned to 1. A partial solution is an assignment where some z_{pij} are set to 1. For a partial solution obtained at the end of step k, and an interval I, let $\mathrm{Used}(I,k)$ denote the number of time slots in this interval used up by z_{pij} that are assigned integrally to 1, i.e., $\mathrm{Used}(I,k) = \sum_{p,i,j} w(B(p,i,j),I)$ such that $z_{pij} = 1$. We will use $\mathrm{Free}(I,k)$ to denote $\mathrm{Size}(I) - \mathrm{Used}(I,k)$.

We now describe the algorithm to compute the tentative schedule.

1. Initialize: We divide the time horizon from $1, \ldots, T+n$ into consecutive intervals of size $5\log(T+n)$. We call this collection of intervals $\mathcal{I}_0$. For all $I \in \mathcal{I}_0$, we define $\text{Used}(I,0) = 0$ and $\text{Free}(I,0) = \text{Size}(I) - \text{Used}(I,0) = \text{Size}(I)$. Let $\tilde{\mathcal{B}}_0$ be the set of all blocks $B(p,i)$, and let $\mathcal{S}_0 = \emptyset$.
2. Repeat the following for $k = 1, \ldots,$.
   - Consider the following linear program defined iteratively based on $\tilde{\mathcal{B}}_{k-1}$, $\mathcal{I}_{k-1}$, and $\text{Free}(I, k-1)$:

$$\min \sum_{B(p,i)\in\tilde{\mathcal{B}}_{k-1}} \sum_{j=0}^{1/\delta-1} R(B(p,i,j)) \cdot z_{pij} \tag{13}$$

$$\text{subject to} \quad \sum_{j=0}^{1/\delta-1} z_{pij} = 1 \qquad \forall p, i : B(p,i) \in \tilde{\mathcal{B}}_{k-1}, \tag{14}$$

$$\sum_{p\in P} \sum_{i=1}^{b_p} \sum_{j=0}^{1/\delta-1} w(B(p,i,j), I) \cdot z_{pij} = \text{Free}(I, k-1) \qquad \forall I \in \mathcal{I}_{k-1}, \tag{15}$$

$$z_{pij} \geq 0 \qquad \forall p, i, j. \tag{16}$$

     Note that for $k = 1$ this LP is similar to the LP defined by (9)–(12) except that the constraints in (11) are relaxed to hold only for intervals of size $5\log(T+n)$ rather than for each time unit.
   - Solve this LP and consider some basic solution. Let $\mathcal{P}$ denote the set of blocks $B(p,i)$ such that $z_{pij} = 1$ for some j. Let $\mathcal{S}$ denote the set of block-offset pairs $B(p,i,j)$ such that $z_{pij} = 1$.
   - Set $\tilde{\mathcal{B}}_k = \tilde{\mathcal{B}}_{k-1} \setminus \mathcal{P}$. These are precisely the blocks $B(p,i)$ for which z_{pij} is not equal to 1 for any j at the end of step k. Set $\mathcal{S}_k = \mathcal{S}_{k-1} \cup \mathcal{S}$. These are precisely the variables z_{pij} that are integrally set to 1 thus far by the end of step k. For each interval $I \in \mathcal{I}_{k-1}$, recompute

$$\text{Used}(I,k) = \text{Used}(I,k-1) + \sum_{p,i,j:B(p,i,j)\in\mathcal{S}} w(B(p,i,j), I).$$

     Note that $\sum_{p,i,j:B(p,i,j)\in\mathcal{S}} w(B(p,i,j), I)$ is exactly the number of pages that are assigned to be transmitted during interval I in step k. Set $\text{Free}(I,k) = \text{Size}(I) - \text{Used}(I,k)$. Essentially, $\text{Free}(I,k)$ denotes the number of free time slots in interval I at the end of step k.
   - Finally, we compute the set of intervals $\mathcal{I}_k$ by merging the intervals in $\mathcal{I}_{k-1}$ as follows: Initially $\mathcal{I}_k = \emptyset$. Starting from the leftmost interval in $\mathcal{I}_{k-1}$, merge intervals $I_1, I_2, \ldots, I_l \in \mathcal{I}_{k-1}$ greedily to form I until $\text{Free}(I_1,k) + \text{Free}(I_2,k) + \cdots +, \text{Free}(I_l,k)$ first exceeds $5\log(T+n)$. We set $\text{Free}(I,k) = \text{Free}(I_1,k) + \text{Free}(I_2,k) + \cdots +, \text{Free}(I_l,k)$ and $\text{Used}(I,k) = \text{Used}(I_1,k) + \text{Used}(I_2,k) + \cdots +, \text{Used}(I_l,k)$. By construction, we have that $5\log(T+n) \leq \text{Free}(I,k) \leq 10\log(T+n)$. Add I to $\mathcal{I}_k$ and remove $I_1, \ldots, I_l$ from $\mathcal{I}_{k-1}$ and repeat the process until the total free space in the intervals in $\mathcal{I}_{k-1}$ is less than $5\log(T+n)$; hence, we cannot form new intervals. In this case we just merge all of the remaining intervals in $\mathcal{I}_{k-1}$ into one interval and add this final interval to $\mathcal{I}_k$.

- If $|\mathcal{I}_k| = 1$, then the algorithm makes one more iteration and then stops. On this last iteration there is just one constraint of type (15) in the relaxed auxiliary LP. The optimal solution is integral and very easy to define. We choose the best offset for every remaining block; i.e., we define $z_{pij} = 1$ if $R(B(p,i,j)) = \min_s R(B(p,i,s))$ for block $B(p,i)$.

3.4. Analysis.

LEMMA 7. *At each iteration of step* 2 *in the previous algorithm, the number of blocks* $B(p,i)$ *that do not have any* z_{pij} *set to* 1 *decreases by a constant factor. In particular*

$$|\tilde{\mathcal{B}}_k| \leq 0.6 \cdot |\tilde{\mathcal{B}}_{k-1}| + 1.$$

Proof. The total number of nontrivial constraints (of types (14) and (15)) in the LP at step k is $|\mathcal{I}_{k-1}| + |\tilde{\mathcal{B}}_{k-1}|$. Consider a basic optimal solution of the LP at stage k. Let f_k be the number of nonzero variables that are set fractionally (strictly between 0 and 1), and let g_k denote the number of variables set to 1. Then, since we have a basic solution, we have that $f_k + g_k \leq |\mathcal{I}_{k-1}| + |\tilde{\mathcal{B}}_{k-1}|$. Now, consider the constraints of type (14); if in some block $B(p,i)$ there is no z_{pij} that is set to 1, then there must be at least 2 variables z_{pij} set fractionally, which implies that $f_k/2 + g_k \geq |\tilde{\mathcal{B}}_{k-1}|$. Combining these two facts implies that $g_k \geq |\tilde{\mathcal{B}}_{k-1}| - |\mathcal{I}_{k-1}|$. By definition, as $|\tilde{\mathcal{B}}_k| = |\tilde{\mathcal{B}}_{k-1}| - g_k$, this implies that $|\tilde{\mathcal{B}}_k| \leq |\mathcal{I}_{k-1}|$.

We now upper bound $|\mathcal{I}_{k-1}|$. Let Free_{k-1} denote the total free space at the end of iteration $k-1$, that is, $\sum_{I \in \mathcal{I}_{k-1}} \text{Free}(I, k-1)$. Since each interval, except probably the last one, has at least $5 \log(T+n)$ free spaces, we have that $|\mathcal{I}_{k-1}| \leq \lceil \text{Free}_{k-1}/(5 \log(T+n)) \rceil$. As for any block-offset $B(p,i,j)$ and interval I, the number of time slots $w(B(p,i,j), I)$ is at most $\log(T+n) + 2$, it follows from the constraints (14) and (15) that $\text{Free}_{k-1} \leq (\log(T+n) + 2)|\tilde{\mathcal{B}}_{k-1}| \leq 3 \log(T+n)|\tilde{\mathcal{B}}_{k-1}|$. This implies that

$$|\mathcal{I}_{k-1}| \leq \lceil 0.6 \cdot |\tilde{\mathcal{B}}_{k-1}| \rceil \leq 0.6 \cdot |\tilde{\mathcal{B}}_{k-1}| + 1. \qquad \square$$

As $|\tilde{\mathcal{B}}_0| \leq T+n$, by Lemma 7 we have that the algorithm stops after $\log(T+n) + \Theta(1)$ iterations. When our algorithm ends we obtain an assignment of zero-one values to variables z_{pij}. Since in every step of our algorithm we relaxed the LP from the previous step, the cost of this final integral solution is upper bounded by the optimal value of LP (9)–(12), which is at most 3 times the optimal value of LP (1)–(6) by Lemma 6. This solution also provides us with an integral tentative schedule since it gives us an assignment of pages to the time slots.

To actually obtain a proper schedule from this tentative schedule, we look at the pages transmitted in the tentative schedule at time 0 and greedily assign it to the next free slot after time t. Formally, we can view the process of constructing the feasible schedule from the tentative schedule as follows: There is a queue Q. Whenever a page p is tentatively scheduled at time t, we add p to the tail of Q at time t. At every time step, if Q is nonempty, we broadcast the page at the head of Q.

To complete the proof, we show that no page is delayed more than $O(\log^2(T+n))$ than its position in the tentative schedule. Thus it suffices to show that the queue length $Q(t)$ at time t in the above description is always bounded by $O(\log^2(T+n))$ at all times t.

LEMMA 8. *Let* $\text{Used}(t_1, t_2)$ *denote the pages transmitted during* $(t_1, t_2]$ *in the tentative schedule. The maximum queue length at any time is bounded by* $\max_{t_1 < t_2} (\text{Used}(t_1, t_2) - (t_2 - t_1))$.

Proof. Let t_2 be the time when the backlog in the queue is maximum, and let b denote this backlog. Consider the last time t_1 was before t_2 the queue was empty. Since t_1 was the last time when the queue was empty, it must be the case that exactly $t_2 - t_1$ pages were transmitted during the interval $(t_1, t_2]$, and hence b is exactly $\text{Used}(t_1, t_2) + t_2 - t_1$. This implies the desired result. ∎

LEMMA 9. *For every* t_1, t_2, $\text{Used}(t_1, t_2) - (t_2 - t_1) \leq 20 \log^2(T+n) + O(\log(T+n))$.

Proof. Consider the time interval $(t_1, t_2]$. If $\Lambda \leq \log(T+n) + \Theta(1)$ is the number of iterations of our algorithm, then there are at most 2Λ intervals $[a, b]$ generated by the algorithm which strictly overlap with t_1, i.e., either $a < t_1 < b$ or strictly overlap t_2, i.e., $a < t_2 < b$. The total number of pages assigned to these intervals by the tentative schedule is at most $2\Lambda \cdot 10 \log n \leq 20 \log^2(T+n) + O(\log(T+n))$.

All other intervals generated by our algorithm do not strictly overlap with t_1 and t_2. They are either completely inside $(t_1, t_2]$ or else lie completely outside $(t_1, t_2]$. We claim that by the constraints (15), the total number of pages assigned to the intervals completely contained in $(t_1, t_2]$ is upper bounded by $t_2 - t_1$. This follows from the fact that on each iteration we are allowed to use only time slots which were not occupied by the integral assignments from previous iterations and the total number of pages transmitted in every interval on each iteration is exactly the length of this interval minus the amount of free space which could be used on the next iteration. The lemma thus follows. ∎

Thus we have the following theorem.

THEOREM 10. *The above algorithm produces a broadcast schedule with average response time at most* $3 \cdot OPT + O(\log^2(T+n))$, *where* OPT *denotes the average response time of the optimum schedule.*

3.5. Improving the approximation guarantee further. We first remove the dependence on T in the approximation ratio. We do this by showing that at the expense of a small loss (a factor of $(1 + o(1))$) in the approximation ratio, the time horizon T can be assumed to be polynomially bounded in n.

LEMMA 11. *We are given an instance* I *of the broadcast scheduling problem with planning horizon of length* T *and* n *pages. Then we can define instances* $I_1, \ldots, I_s$ *such that each* I_i *has a time horizon of length at most* $2n^5$, *and the solutions to these satisfy the following properties:*

1. *Let* LP^* *and* LP_i *denote the optimal value to the LP* (1)–(6) *for* I *and* I_i, *respectively. Then,* $\sum_{i=1}^{s} LP_i \leq (1 + o(1)) LP^*$.
2. *Given any integral solutions* $IP_1, \ldots, IP_s$ *for* $I_1, \ldots, I_s$, *respectively, we can obtain another solution for* I *with cost* $(1 + o(1))(\sum_{i=1}^{s} IP_i)$.

It is easily seen that this lemma allows us to assume that $T = \text{poly}(n)$. In particular, we can apply this decomposition to the original instance I, obtain an $O(\log n)$ approximation to each of these instances, and then use the second property to obtain an integral solution to I.

Proof of Lemma 11. We describe a randomized decomposition procedure which satisfies the properties in expectation. Once we describe this procedure it will be immediately clear how this can be derandomized in polynomial time. We first split the original time interval $[1, T]$ into time intervals $T_1, \ldots, T_s$ as follows. Choose an integer $k = 1, \ldots, n^5$ uniformly at random, and let $T_1 = [1, k + n^5]$, and $T_i = (k + (i-1)n^5, k + in^5]$ for $i = 2, \ldots, s$. Let $V_i \subseteq T_i$ denote the interval consisting of the last n time steps of T_i for $i = 1, \ldots, s$. The instance I_i consists of two types of requests in I: those that arrive during $T_i \setminus V_i$ and those that arrive during V_i. We keep the requests

that arrive during $T_i \setminus V_i$ unchanged and move those that arrive during V_i to I_{i+1} by changing their arrival time to the beginning of T_{i+1} (thus I_i contains the requests that arrive during V_{i-1}). This completes the description of the instances. Note that there are only $O(n^5)$ possible decompositions, and hence the procedure can be easily derandomized by trying out all choices of k.

Consider an optimal solution (x, y) to LP (1)–(6) for I. Suppose we choose this solution for each I_i. Then response times for requests that arrive during $T_i \setminus V_i$ remain unchanged and it can only increase for those that arrive during V_i. To handle this, we modify the solution (x, y) as follows. Consider the first n^2 time slots of T_{i+1}, since there are n pages there must be at least some page p_{i+1} such that the amount of p_{i+1} transmitted by the LP during these slots is at least n. We remove $n-1$ units of page p_{i+1} from these slots and, for every page other than p_{i+1}, arbitrarily assign one unit of this page fractionally to the free space in the first n^2 slots. This can increase the response time of requests that arrive during T_{i+1} by at most n^2. Moreover, the response time of requests that arrive during V_i is at most $n^2 + n$. Since a request has a probability of at most $n/n^5 = 1/n^4$, over the random choices of k, of arriving in some V_i, the expected increase in the total response time of requests in V_i is at most $(n^2 + n)m/n^4 = o(m)$, where m is the total number of requests in the instance. Thus the average response time increases by at most $o(1)$. Similarly, for requests in the first n^2 slots of T_{i+1} the average response time increases by at most $n^2 m/n^3 = o(m)$ in expectation. This implies the first part of the lemma.

We now prove the second part of the lemma. Assume we are given an integral solution for each I_i. Since all requests in I_i arrive during $T_i \setminus V_i$ and $|V_i| = n$, we can assume that all the requests in I_i are satisfied during T_i. Consider the integral solution for I obtained by concatenating all of the schedules for I_i. When we consider this schedule for the instance I, the only difference is that the response time for requests that arrived during V_i can have an additional response time of n, since these requests were moved at most n steps in the future to obtain the instances I_i. Again, over the random choices of k, the probability that a request lies in some V_i is $1/n^4$ and hence the expected increase in the average response time is at most $n/n^4 = 1/n^3$. This proves the desired result. ∎

We next show how to obtain a more refined balance between the multiplicative and additive term in Theorem 10.

LEMMA 12. *For any $\gamma > 0$, there is algorithm that achieves an approximation guarantee of $(2+\gamma)OPT + O(\log_{1+\gamma}(T+n) \cdot \log(T+n))$ for minimizing the average response time. Choosing γ close to 0 implies a guarantee of $(2+\gamma) \cdot OPT + O(\log^2(T+n)/\gamma)$.*

Proof. For a fixed $\gamma > 0$, we modify the definition (see Definition 2) of a p-good point such that we call a time t to be p-good if $r(p,t) \leq (1+\gamma) r(p,\tau)$ for all τ, t such that $c(p,\tau,t) < 1$. With this modification, imitating Lemma 3, we can form blocks $B(p,i)$, where the amount of page p transmitted in a block is at most $\log_{1+\gamma}(T+n) + 2$. Moreover, Lemma 4 now gives us that the expected response time of the tentative schedule obtained is at most $(2+\gamma)$ times the optimum cost. Now repeat the algorithm in section 3.3 with intervals of size $5\log_{1+\gamma}(T+n)$ (instead of intervals of size $5\log(T+n)$). Again we have that the number of blocks reduces by more than a factor of half at each iteration while adding $O(\log_{1+\gamma}(T+n))$ to the backlog. Thus, there are $O(\log(T+n))$ iterations of step 2 of the algorithm, and it follows directly that the backlog of the tentative schedule thus constructed is at most $O(\log_{1+\gamma}(T+n) \cdot \log(T+n))$, which implies the desired guarantee. ∎

By Theorem 10, Lemma 11, and setting $\gamma = \log(T+n)$ in Lemma 12, we have the following theorem.

THEOREM 13. *There is a polynomial time algorithm that produces a broadcast schedule with average response time at most* $O((\log^2 n/\log\log n)OPT)$*, where OPT denotes the average response time of the optimum schedule.*

4. Bad instance for local rounding procedures. We show that any way of rounding the ILP (1)–(6) that gives a tentative schedule that is local is unlikely to achieve an approximation ratio better than $O(\sqrt{n})$. Recall that a local rounding procedure ensures that there is a transmission of page p in every time interval $[t_1, t_2]$, where the LP transmits at least one unit of page p. One consequence of having a local schedule is that if a page p is broadcast at time t, then the next broadcast of page p must be at some time t' such that the cumulative amount of page p transmitted by the LP solution during $(t, t']$ is no more than 1. We give an example of an LP solution for which every tentative schedule that is local has a backlog of $\Omega(\sqrt{n})$ at some time. This will imply that algorithmic techniques based on local tentative schedules are unlikely to yield an approximation guarantee better than $\Theta(\sqrt{n})$.

We construct a half integral LP solution as follows: Let H be the Hadamard matrix of order n, and J be the matrix of order n with all entries equal to 1. Consider the matrix $A = \frac{1}{2}(H+J)$. The matrix A is a $\{0,1\}$ matrix where each row A_i (except for the one with all 1's) contains exactly $n/2$ 1's. A well-known property (see, for example, [9, p. 17]) of A is that for any vector $x \in \{-1,+1\}^n$, there is a row with a discrepancy of at least $\sqrt{n}/2$. That is, for each vector $x \in \{-1,+1\}^n$, $|A_i x| \geq \sqrt{n}/2$ for some $1 \leq i \leq n$.

We view these A_i as subsets of $\{1,\ldots,n\}$. We also assume that n is a multiple of 4. Let n_i denote the number of $1's$ in A_i. Let $S_i = n_1 + \cdots + n_i$, and let $S_0 = 0$. The LP schedule is constructed as follows: The schedule transmits, in any particular order, 1/2 unit of each page in A_i during the interval $(S_{i-1}, S_{i-1} + n_i/2]$, and again during the interval $(S_{i-1} + n_i/2, S_i]$. For $j \in A_i$, we denote the time when it is transmitted (half unit) during $(S_{i-1}, S_{i-1} + n_i/2]$ its *odd* slot and its transmission during $(S_{i-1} + n_i/2, S_i]$ its *even* slot.

Now, any local tentative schedule requires that if a page is transmitted in its odd slot at some time t, then the next transmission of this page must be no later than next odd slot for this page, and similarly for even slots. We first note that it suffices to consider *strictly* local schedules where each page in transmitted in the tentative schedule only during the odd slots or only during the even slots. Indeed, if some page is transmitted in both an odd slot and the adjacent even slot, then the number of transmissions of the page in the integral solution will be strictly more than those in the LP solution, leading to a backlog of at least 1. Repeating the instance k times (with completely disjoint set of pages) will imply a backlog of k. Thus it suffices to consider strictly local schedules.

A strict local schedule associates a vector $x \in \{-1,+1\}^n$, where the ith entry is -1 if page i is transmitted during odd slots and is 1 otherwise. As the number of pages transmitted by the tentative schedule during $(S_{i-1}, S_i]$ is exactly n_i, it is easy to see that $|A_i x|$ is exactly equal to the backlog at time $S_{i-1} + n_i/2$ or S_i, which implies the desired claim for strict tentative schedules.

5. Concluding remarks. In the offline setting, nothing more than NP-completeness is known for the problem. It would also be very interesting to construct nontrivial integrality gaps for the LP relaxation considered in this paper. Currently, the best

known integrality gap for the LP relaxation is 1.027 [2], and hence it is possible that an $O(1)$ approximation for the problem can be obtained using this LP.

Another very interesting problem would be to give an $O(1+\epsilon)$-speed, $O(1)$-competitive algorithm for the online version of the problem. Currently, no $O(1)$ competitive algorithm is known for the problem that uses a speed up of less than 4.

Acknowledgments. We thank Moses Charikar, Sanjeev Khanna, Tomasz Nowicki, Tracy Kimbrel, Seffi Naor, Kirk Pruhs, Baruch Schieber, and Grzegorz Swirszcz for helpful discussions.

REFERENCES

[1] Airmedia website, http://www.airmedia.com.
[2] N. Bansal, M. Charikar, S. Khanna, and J. Naor, *Approximating the average response time in broadcast scheduling*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, British Columbia, ACM, New York, SIAM, Philadelphia, 2005, pp. 215–221.
[3] A. Bar-noy, S. Guha, Y. Katz, J. Naor, B. Schieber, and H. Shachnai, *Throughput maximization of real-time scheduling with batching*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, SIAM, Philadelphia, 2002, pp. 742–751.
[4] Y. Bartal and S. Muthukrishnan, *Minimizing maximum response time in scheduling broadcasts*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, SIAM, Philadelphia, 2000, pp. 558–559.
[5] A. Blum, P. Chalasani, D. Coppersmith, W. R. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proceedings of the 26th Annual Symposium on Theory of Computing (STOC), Montreal, Quebec, 1994, pp. 163–171.
[6] J. Chang, T. Erlebach, R. Gailis, and S. Khuller, *Broadcast scheduling: Algorithms and complexity*, in Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, SIAM, Philadelphia, 2008, pp. 473–482.
[7] M. Charikar and S. Khuller, *A robust maximum completion time measure for scheduling*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, Miami, FL, ACM, New York, SIAM, Philadelphia, 2006, pp. 324–333.
[8] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, *Paths, trees, and minimum latency tours*, in Proceedings of the 44th Annual Symposium Foundations of Computer Science (FOCS), Cambridge, MA, 2003, pp. 36–45.
[9] B. Chazelle, *The Discrepancy Method: Randomness and Complexity*, 1st ed., Cambridge University Press, Cambridge, 2000.
[10] DirecPC website, http://www.direcpc.com.
[11] J. Edmonds and K. Pruhs, *Broadcast scheduling: When fairness is fine*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, SIAM, Philadelphia, 2002, pp. 421–430.
[12] J. Edmonds and K. Pruhs, *A maiden analysis of longest wait first*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, ACM, New York, SIAM, Philadelphia, 2004, pp. 818–827.
[13] T. Erlebach and A. Hall, *NP-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, SIAM, Philadelphia, 2002, pp. 194–202.
[14] R. Gandhi, S. Khuller, Y. Kim, and Y-C. Wan, *Approximation algorithms for broadcast scheduling*, in Proceedings of the 9th Annual Conference on Integer Programming and Combinatorial Optimization (IPCO), Cambridge, MA, 2002.
[15] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, *Dependent rounding in bipartite graphs*, J. ACM, 53 (2006), pp. 324–360.
[16] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Math. Programming, 82 (1998), pp. 111–124.
[17] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai, *Scheduling broadcasts in wireless networks*, in Proceedings of the 8th Annual European Symposium on Algorithms,

Saarbrucken, Germany, 2000, pp. 290–301.

[18] S. Khuller and Y. Kim, *Equivalence of two linear programming relaxations for broadcast scheduling*, Oper. Res. Lett., 32 (2004), pp. 473–478.

[19] J. Robert and N. Schabanel, *Pull-based data broadcast with dependencies: Be fair to users, not to items*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, ACM, New York, SIAM, Philadelphia, 2007, pp. 238–247.

SIAM J. COMPUT.
Vol. 38, No. 3, pp. 1175–1206

GEOMETRIC COMPLEXITY THEORY II: TOWARDS EXPLICIT OBSTRUCTIONS FOR EMBEDDINGS AMONG CLASS VARIETIES*

KETAN D. MULMULEY† AND MILIND SOHONI‡

Dedicated to Sri Ramakrishna

Abstract. In [K. D. Mulmuley and M. Sohoni, *SIAM J. Comput.*, 31 (2001), pp. 496–526], henceforth referred to as Part I, we suggested an approach to the P vs. NP and related lower bound problems in complexity theory through geometric invariant theory. In particular, it reduces the arithmetic (characteristic zero) version of the $NP \not\subseteq P$ conjecture to the problem of showing that a variety associated with the complexity class NP cannot be embedded in a variety associated with the complexity class P. We shall call these *class varieties* associated with the complexity classes P and NP. This paper develops this approach further, reducing these lower bound problems—which are all nonexistence problems—to some existence problems: specifically to proving the existence of *obstructions* to such embeddings among class varieties. It gives two results towards explicit construction of such obstructions. The first result is a generalization of the Borel–Weil theorem to a class of orbit closures, which include class varieties. The second result is a weaker form of a conjectured analogue of the second fundamental theorem of invariant theory for the class variety associated with the complexity class NC. These results indicate that the fundamental lower bound problems in complexity theory are, in turn, intimately linked with explicit construction problems in algebraic geometry and representation theory. The results here were announced in [K. D. Mulmuley and M. Sohoni, in *Advances in Algebra and Geometry (Hyderabad,* 2001*)*, Hindustan Book Agency, New Delhi, India, 2003, pp. 239–261].

Key words. computational complexity, algebraic geometry, representation theory, geometric invariant theory

AMS subject classifications. 20G05, 03D15, 70G55

DOI. 10.1137/080718115

1. Main results.

We shall now state the results precisely. For the sake of completeness, we recall in section 2 the main results from Part I of this paper [26]. The rest of this paper is self-contained. All groups in this paper are algebraic, and the base field is $\mathbb{C}$.

Let G be a connected, reductive group, V its (finite dimensional) linear representation, and $P(V)$ the corresponding projective space. Let $\Delta_V[v]$ denote the projective closure of the G-orbit of v in $P(V)$. It is an almost-homogeneous space in the terminology of [1]. Let $R_V[v]$ be its homogeneous coordinate ring, $I_V[v]$ its ideal, and $R_V[v]_d$ the degree d component of $R_V[v]$.

In Part I, we reduced arithmetic (characteristic zero) implications of the lower bound problems in complexity theory, such as P vs. NP and NC vs. $P^{\#P}$, to instances of the following problem (section 2).

PROBLEM 1.1 (the orbit closure problem). *Given explicit points $f, g \in P(V)$, does $f \in \Delta_V[g]$? Equivalently, is $\Delta_V[f] \subseteq \Delta_V[g]$?*

*Received by the editors January 24, 2005; accepted for publication (in revised form) January 10, 2008; published electronically July 16, 2008.

http://www.siam.org/journals/sicomp/38-3/71811.html

†Department of Computer Science, University of Chicago, Chicago, IL 60637 (mulmuley@cs.chicago.edu). The work of this author was supported by NSF grant CCR 9800042 and, in part, by a Guggenheim Fellowship. Part of this work was done while this author was visiting Indian Institute of Technology, Mumbai.

‡Department of Computer Science and Engineering, Indian Institute of Technology, Mumbai 400076, India (sohoni@cse.iitb.ernet.in).

The goal is to show that this is not the case in the problems under consideration.

The f's and g's here depend on the complexity classes in the lower bound problem under consideration. In the context of the P vs. NP problem, the point g will correspond to a judiciously chosen P-complete problem, and f to a judiciously chosen NP-complete problem. We call $\Delta_V[g]$ and $\Delta_V[f]$ the *class varieties* associated with the complexity classes P and NP (this terminology was not used in Part I). The orbit closure problem in this context is to show that the class variety associated with NP cannot be embedded in a class variety associated with P. We have oversimplified the story here. There is not just one class variety associated with a given complexity class, but a sequence of class varieties depending on the parameters of the lower bound problem under consideration. In the context of the P vs. NP problem, the goal is to show that a class variety for NP associated with a given set of parameters cannot be embedded in the class variety for P associated with the same set of parameters. This would imply that $P \neq NP$ in characteristic zero.

Class variety for the complexity class NC. We give an example of a class variety associated with the complexity class NC, the class of problems with efficient parallel algorithms. This occurs in the context of the NC vs. $P^{\#P}$ problem (section 2.1). Here we let g be the determinant function, which is a complete function for this class. Specifically, let Y be an $m \times m$ variable matrix, which can also be thought of as a variable l-vector, $l = m^2$. Let $V = \mathrm{Sym}^m(Y)$ be the space of homogeneous forms of degree m in the l variable entries of Y, with the natural action of $G = SL(Y) = SL_l(\mathbb{C})$. Let $g = \det(Y) \in P(V)$ be the determinant form, considered as a point in the projective space. Then $\Delta_V[g]$, the orbit closure of the determinant function, is the class variety associated with NC. This is a basic example of a class variety, which the reader may wish to keep in mind throughout this paper.

For arbitrary f and g, Problem 1.1 is hopeless. But f and g in the preceding lower bound problems can be chosen judiciously, like the determinant function, to have some special properties (cf. section 2 and Part I). To state these properties, we need a few definitions.

Given a point $v \in P(V)$, let $\hat{v} \in V$ denote a nonzero point on the line representing v; the exact choice of $\hat{v}$ will not matter. Let $G_v, G_{\hat{v}} \subseteq G$ denote the stabilizers of v and $\hat{v}$, respectively. We say that v is *characterized* by its stabilizer if $V^{G_{\hat{v}}}$, the set of points in V stabilized by $G_{\hat{v}}$, is equal to $\mathbb{C}v$, the line in V corresponding to v.

Following Mumford, Fogarty, and Kirwan [27] and Kempf [10], we say that v is *stable* if the orbit $G\hat{v} \subseteq V$ is closed, and *semistable* if the closure of this orbit does not contain zero. We say v belongs to the *null cone* if all homogeneous G-invariants of positive degree vanish at $\hat{v}$. We also define a more general notion of *partial stability* which also applies to points in the null cone. A stable point is also partially stable by definition. Now suppose v is not stable. Let S be any closed G-invariant subset of V not containing $\hat{v}$ and meeting the boundary of the orbit $G\hat{v}$. Kempf [10] associates with v and S a canonical parabolic subgroup $P = P[S, v] \subseteq G$, called its canonical destabilizing flag. Let L be its semisimple Levi subgroup. We say that v is *partially stable* with defect zero or, more specifically, (L, P)-stable if (1) the unipotent radical U of P is contained in G_v and (2) v is stable with respect to the restricted action of L on V. A more general notion of partial stability allowing nonzero defect is given later (Definition 7.1).

We say that v is *excellent* if

1. it is stable or partially stable with defect zero, and
2. it is characterized by its stabilizer.

If V is an irreducible representation $V_\lambda(G)$ of G, corresponding to a dominant weight λ, then the point in $P(V)$ corresponding to the highest weight vector of G is excellent. This is the simplest example of an excellent point. In this case, the stabilizer G_v is a parabolic subgroup $P = P_\lambda$ of G, and the orbit $Gv \cong G/P$ is closed. Hence $\Delta_V[v] \cong G/P$. The algebraic geometry of G/P has been intensively studied in the literature and is well understood by now; cf. [5, 13] for surveys.

For the lower bound problems under consideration, the points f and g can be chosen so that they are either excellent or almost excellent; the meaning of almost excellent is stated in section 2. For example, the determinant function above is excellent. In this paper, we shall develop an approach to the orbit closure problem specifically for such f and g. The goal is to understand the orbit closure problem by systematically extending the results for G/P to the (almost) excellent points that arise in this approach.

A natural approach to the orbit closure problem is the following. If f lies in $\Delta_V[g]$, then the embedding $\Delta_V[f] \hookrightarrow \Delta_V[g]$ is G-equivariant. This gives a degree preserving G-equivariant surjection from $R_V[g]$ to $R_V[f]$. Hence, if S is any irreducible representation of G, its multiplicity in $R_V[g]_d$ is greater than or equal to its multiplicity in $R_V[f]_d$, for all d.

DEFINITION 1.2. *We say that S is an* obstruction *for the pair (f, g) if, for some d,*

1. *it occurs in (a complete G-decomposition of) $R_V[f]_d$,*
2. *but not in $R_V[g]_d$.*

Existence of such an S implies that f cannot lie in $\Delta_V[g]$. In a lower bound problem, this S can be considered to be a "witness" to the computational hardness of f.

If S occurs in $R_V[g]_d$, then it is easy to show (Proposition 4.2) that its dual S^* contains a G_g-module isomorphic to $(\mathbb{C}g)^d$, the dth tensor power of $\mathbb{C}g$. Hence we are led to the following definition.

DEFINITION 1.3. *We say that S is a* strong obstruction *if, for some d,*

1. *it occurs in $R_V[f]_d$,*
2. *but its dual S^* does not contain a G_g-module isomorphic to $(\mathbb{C}g)^d$.*

A strong obstruction is also an obstruction.

For the (f, g)'s in the lower bound problems under consideration, strong obstructions are conjectured to exist in plenty (section 3). But to prove their existence it is necessary to construct them more or less explicitly. Otherwise, the proof technique cannot cross the natural proof barrier formulated in [28] that any technique for proving the $P \neq NP$ conjecture must cross. Explicit constructions have been used in the theory of computing earlier in different contexts. For example, explicit expanders, needed for efficient pseudorandom generation, have been constructed by Margulis [20], and Lubotzky, Phillips, and Sarnak [16]. The essential difference from the situation here is that proving existence of expanders is easy, whereas proving existence of obstructions is itself the main problem.

Hence, we are led to the following problem.

PROBLEM 1.4 (explicit construction of obstructions). *Given f and g as in Problem* 1.1*, explicitly construct a (strong) obstruction for the embedding $\Delta_V[f] \hookrightarrow \Delta_V[g]$.*

In the orbit closure problems under consideration, $H = G_g$ turns out to be a reductive subgroup of G. Hence, to solve Problem 1.4, we have to solve the following problems first.

PROBLEM 1.5 (subgroup restriction problem). *Let H be a reductive subgroup of*

a connected, reductive group G. Find an explicit decomposition of a given irreducible G-representation S as an H-module.

This arises in the context of the second condition in Definition 1.3 (with S^* in place of S).

Problem 1.5, with H equal to the stabilizer of the determinant function considered earlier, turns out to be equivalent to the Kronecker problem of finding an explicit decomposition of the tensor product of two irreducible representations of the symmetric group; cf. section 2. This is an outstanding problem in the representation theory of the symmetric group [19, 5]. Other specific instances of Problem 1.5 that arise in the lower bound problems under consideration (cf. section 2) include the well-known plethysm problem [19, 5], which is an outstanding problem in the representation theory of $GL_n(\mathbb{C})$.

PROBLEM 1.6 (problem in geometric invariant theory). *Let $v \in P(V)$ be an (almost) excellent point. Find an explicit decomposition of $R_V[v]_d$, for a given d, as a G-module.*

This is needed in the context of both conditions in Definition 1.2. For this, it is desirable to solve the following problem first.

PROBLEM 1.7 (SFT problem). *Let $v \in P(V)$ be an (almost) excellent point. Find an explicit set of generators for the ideal $I_V[v]$ of $\Delta_V[v]$ with good representation theoretic properties. (The short form SFT is explained below.)*

Problems 1.6 and 1.7 are intractable for general v's. Hence, specialization to almost excellent v's is necessary. Some additional reasonable restrictions may be necessary in these problems.

When $V = V_\lambda(G)$, v is the point corresponding to the highest weight vector of $V_\lambda(G)$, and $\Delta_V[v] \cong G/P$, the second fundamental theorem (SFT) of invariant theory for G/P [13] answers Problem 1.7. By the Borel–Weil theorem for G/P [5], $R_V[v]_d = V_{d\lambda}(G)^*$. This answers Problem 1.6.

What is desired is a generalization of these results for G/P to the class varieties $\Delta_V[v]$, for the (almost) excellent v's under consideration. Before we go any further, let us point out the main difference between G/P and the class varieties:

1. Luna and Vust [18] have assigned a complexity to orbit closures, which measures the complexity of their algebraic geometry. All orbit closures whose algebraic geometry has been well understood have low Luna–Vust complexity—close to zero. For example, the Luna–Vust complexity of G/P is zero. In contrast, the Luna–Vust complexity of a class variety can be polynomial in the number of parameters in the lower bound problem under consideration.
2. The analogue of the subgroup restriction problem (Problem 1.5), with H being the parabolic stabilizer P of the highest weight vector in $V_\lambda(G)$, is trivial. In contrast, the instances of Problem 1.5 in the context of the class varieties include the nontrivial Kronecker and plethysm problems.

This indicates that the algebraic geometry of class varieties is substantially more complex than that of G/P. For this reason, we cannot expect a full solution to Problems 1.6 and 1.7 until the outstanding Problem 1.5 in representation theory is resolved. Rather, our goal is to connect Problems 1.6 and 1.7 with the "easier" Problem 1.5 for the almost excellent v's under consideration. We prove two results in this direction.

Let us begin by considering a weaker form of Problem 1.6; i.e., we only ask which G-modules can occur in $R_V[v]$, without worrying about $R_V[v]_d$ for a specific d. This is addressed by the following result.

We call a G-module $V_\lambda(G)$ $G_{\hat{v}}$-admissible if it contains a $G_{\hat{v}}$-invariant (cf. Definition 4.1).

THEOREM 1.1 (Borel–Weil for orbit closures of partially stable points). *Let V be a (finite dimensional) linear representation of a connected, reductive G.*

(a) *If $v \in P(V)$ is stable, an irreducible G-module $V_\lambda(G)$ with weight λ can occur in $R_V[v]$ iff $V_\lambda(G)$ is $G_{\hat{v}}$-admissible.*

(b) *Suppose v is partially stable with defect zero, specifically (L, P)-stable, as defined above. Let $S_V[v]$ be the homogeneous coordinate ring of the projective closure in $P(V)$ of the L orbit of v. Then the G-module structure of $R_V[v]$ is completely determined by the L-module structure of $S_V[v]$. A weaker statement holds for a partially stable point of nonzero defect as defined in section 7.*

A precise statement of (b) is given in section 8. We actually prove a stronger result (Theorem 8.2) that specializes to the Borel–Weil theorem [11] when v corresponds to the highest weight vector of an irreducible representation $V = V_\lambda(G)$ of a semisimple G.

When the defect is nonzero, Theorem 1.1(b) does not tell precisely which irreducible G-modules occur in $R_V[v]$ if we only know which irreducible L-modules occur in $S_V[v]$ as a whole. But it gives good information on this and also on which irreducible G-modules occur in $R_V[v]_d$, for a given d, provided we know precisely which irreducible L-modules occur in every degree d-component $S_V[v]_d$; this is Problem 1.6 for a stable v, with L playing the role of G.

Now we turn to the actual Problem 1.6. For this, we have to understand Problem 1.7 first. We turn to this problem next.

Let v be an excellent point. We associate with it a representation-theoretic data $\Pi_v = \cup_d \Pi_v(d)$ (cf. Definitions 6.1 and 10.1). If v is stable, $\Pi_v(d)$ is just the set of all irreducible G-submodules of $\mathbb{C}[V]$ whose duals do not contain a G_v-submodule isomorphic $(\mathbb{C}v)^d$.

Then the ideal $I_V[v]$ contains all modules in Π_v (Proposition 4.2). Let $X(\Pi_v)$ be the variety (scheme) defined by the ideal generated by the modules in Π_v. It follows that $\Delta_V[v] \subseteq X(\Pi_v)$.

Now we ask the following question.

QUESTION 1.8. *Suppose v is excellent. Is $X(\Pi_v) = \Delta_V[v]$ as a variety or, more strongly, as a scheme?*

The scheme theoretic equality means that the ideal $I_V[v]$ of $\Delta_V[v]$ is generated by the modules in Π_v.

If v is stable, then G_v is reductive [2, 24]. Hence, the G-modules contained in Π_v are precisely determined once we know the answer to Problem 1.5, with $H = G_v$. This turns out to be so even for the partially stable v's that arise in the lower bound problems, by letting H be the reductive part of G_v. Hence, if the answer to Question 1.8 is yes, the algebraic geometry of $\Delta_V[v]$ is completely determined by the representation theory of the pair (G_v, G), and hence, Problems 1.6 and 1.7 are intimately related to Problem 1.5. Clearly, this can happen only for very special v's. The answer need not be yes even for a general excellent v.

When v corresponds to the highest weight vector of $V_\lambda(G)$, so that $\Delta_V[v] = G/P$, the answer to Question 1.8 is yes. This follows from the SFT for G/P [13] (cf. section 10.1).

We conjecture that the situation is similar for the class variety associated with the complexity class NC.

CONJECTURE 1.9 (SFT for the orbit closure of the determinant). *Let $\Delta_V[v]$ be*

the class variety associated with the complexity class NC—the orbit closure of the determinant function.

Then $X(\Pi_v) = \Delta_V[v]$ as a variety. (It would be interesting to know if this is so even as a scheme.)

This conjecture is expected because of the very special nature of the determinant function. We have already remarked that it is excellent. Furthermore, its stabilizer has an additional conjectural property called G-separability (Definition 6.3). For analogous conjectures for other almost excellent class varieties, one has to address complications caused by almost excellence instead of full excellence. This is possible and will be done elsewhere.

The following general result implies a weaker form of Conjecture 1.9 when v is the determinant function.

THEOREM 1.2 (SFT for the orbit of an excellent point). *Suppose V is a linear representation of a connected, reductive group G, and $v \in P(V)$ is an excellent point.*

(a) *Suppose v is stable. Furthermore, assume that the stabilizer $G_{\hat{v}}$ is G-separable (cf. Definition 6.3). Then the orbit $Gv \subseteq P(V)$ is determined by the representation-theoretic data Π_v within some G-invariant neighborhood U, i.e.,*

$$Gv = \Delta_V[v] \cap U = X(\Pi_v) \cap U,$$

as schemes.

(b) *A generalized result also holds for the G-orbit of a partially stable, excellent point with defect zero.*

This follows from a stronger result proved in section 6 (stable case) and section 11 (partially stable case).

When v corresponds to the highest weight vector in $V_\lambda(G)$, Theorem 1.2(b), after some strengthening (cf. section 10.1), becomes the second fundamental theorem for G/P [13]—hence the terminology.

The rest of this paper is organized as follows. In section 2, we describe how the orbit closure problem arises in complexity theory, and we summarize the relevant results from Part I. In section 3 we describe why obstructions should exist in the context of the orbit closure problems under consideration. In section 4 we prove some basic propositions based on the notion of admissibility. The stable case of Theorem 1.1 is proved in section 5. The stable case of Theorem 1.2 is proved in section 6. The stable cases illustrate the main ideas in this paper. The notion of partial stability is introduced in section 7. The partially stable case of Theorem 1.1 is proved in section 8. Its specialization in the context of complexity theory is given in section 9. The partially stable case of Theorem 1.2 is proved in section 11. Conjectural G-separability of the stabilizer of the determinant is proved in section 12 for a special case.

Notation. We let G denote a connected reductive group. An irreducible G-representation with highest weight λ will be denoted by $V_\lambda(G)$. We say that $V_\lambda(G)$ occurs in a G-module M, or that M contains $V_\lambda(G)$, if a complete decomposition of M into G-irreducibles contains a copy of $V_\lambda(G)$. We denote the dual of M by M^*. We always denote a Levi-decomposition of a parabolic subgroup $P \subseteq G$ in the form $P = TLU = KU$, where T is a torus, L is a semisimple Levi subgroup, $K = TL$ is a reductive Levi subgroup, and U is the unipotent radical. The root system of K is a subsystem of that of G. Hence a dominant weight of G can be assumed to be a dominant weight of K by restriction.

2. The orbit closure problem. In this section we describe the orbit closure problem that arises in complexity theory and the related results; cf. Part I for details and proofs.

Let $Y = [y_0, \cdots, y_{l-1}]$ denote a variable l-vector. For $k < l$, let $X = [y_1, \cdots, y_k]$ and $\bar{X} = [y_0, \cdots, y_k]$ be its subvectors of size k and $k+1$. Let $V = \mathrm{Sym}^m(Y) = \mathrm{Sym}^m((\mathbb{C}^l)^*)$ be the space of homogeneous forms of degree m in the l variable-entries of Y, with the natural action of $G = SL(Y) = SL_l(\mathbb{C})$, and $\hat{G} = GL(Y) = GL_l(\mathbb{C})$.

Let $W = \mathrm{Sym}^n(X)$, $n < m$, be the representation of $GL(X) = GL_k(\mathbb{C})$. We have a natural embedding $\phi : W \to V$, which maps any $w \in W$ to $y^{m-n}w$, where $y = y_0$ is used as the homogenizing variable. The image $\phi(W)$ is contained in $\bar{W} = \mathrm{Sym}^m(\bar{X})$, a representation of $GL(\bar{X}) = GL_{k+1}(\mathbb{C})$.

DEFINITION 2.1. *We say that $f = \phi(h)$ is partially stable with respect to the action of G (and also $\hat{G}$) if $h \in P(W)$ is stable with respect to the action of $SL_k(\mathbb{C})$.*

These are the only kinds of partially stable points that arise in the context of complexity theory. If the reader wishes, he may confine himself to only these kinds. When we introduce a more general definition of partial stability (section 7), it will turn out that f is partially stable with defect one. In contrast, the (L, P)-stable points in the introduction will turn out to be partially stable points with defect zero. Note that f in Definition 2.1 belongs to the null cone of the G-action—this follows easily from the Hilbert–Mumford criterion [27].

The orbit closure problems (Problem 1.1) that arise in complexity theory (cf. Part I) have the following form.

PROBLEM 2.2. *Given fixed forms $g \in P(V)$ and $h \in P(W) \overset{\phi}{\hookrightarrow} P(V)$, does $f = \phi(h)$ belong to $\Delta_V[g]$? That is, is $\Delta_V[f] \subseteq \Delta_V[g]$?*

The goal is to show that the specific f does not belong to $\Delta_V[g]$. The specific f and g depend on the lower bound problem under consideration and will be either excellent (cf. section 1) or almost excellent. The latter means that (1) the defect of partial stability may not be zero, but will be small, and (2) the point may not be fully characterized by the stabilizer, but almost (as explained in Part I).

The following are two instances of the orbit closure problem that arise in complexity theory.

2.1. Arithmetic version of the NC vs. $P^{\#P}$ conjecture. In concrete terms, this says that the permanent of an $n \times n$ matrix cannot be computed by an integral circuit of depth $\log^c n$, for any constant $c > 0$ [31].

The class varieties in this context are as follows. Let Y be an $m \times m$ variable matrix, which can also be thought of as a variable l-vector, $l = m^2$. Let X be its, say, principal bottom-right $n \times n$ submatrix, $n < m$, which can be thought of as a variable k-vector, $k = n^2$. We use any entry y of Y not in X as the homogenizing variable for embedding $W = \mathrm{Sym}^n(X)$ in $V = \mathrm{Sym}^m(Y)$. Let $g = \det(Y) \in P(V)$ be the determinant form (which will also be considered as a point in the projective space), and $f = \phi(h)$, where $h = \mathrm{perm}(X) \in P(W)$. Then $\Delta_V[g]$ is the class variety associated with NC, and $\Delta_V[f]$ is the class variety associated with $P^{\#P}$. These depend on the lower bound parameters n and m. If we wish to make these implicit, we should write $\Delta_V[f, n, m]$ and $\Delta_V[g, m]$ instead of $\Delta_V[f]$ and $\Delta_V[g]$.

It is conjectured in Part I that, if $m = 2^{O(\mathrm{polylog}_n)}$ and $n \to \infty$, then $f \notin \Delta_V[g]$; i.e., the class variety $\Delta_V[f, n, m]$ cannot be embedded in the class variety $\Delta_V[g, m]$. This implies the arithmetic form of the $NC \neq P^{\#P}$ conjecture.

The following result provides the connection with geometric invariant theory.

THEOREM 2.1 (cf. Part I). *The point $h = perm(X) \in P(W)$ is stable with respect to the action of $SL(X) = SL_k(\mathbb{C})$ on $P(W)$ (thinking of X as a k-vector). Hence the point $f = \phi(h) \in P(V)$ is partially stable (Definition 2.1) with respect to the action of $G = SL(Y) = SL_l(\mathbb{C})$, as well as $\hat{G} = GL(Y) = GL_l(\mathbb{C})$.*

Similarly, $g = \det(Y) \in P(V)$ is stable with respect to the action of G on $P(V)$, thinking of Y as an l-vector on which $SL_l(\mathbb{C})$ acts in the usual way.

Moreover, both $\mathrm{perm}(X) \in P(W)$ and $\det(Y) \in P(V)$ are characterized by their stabilizers. Hence, both h and g are excellent. But, in contrast, $f = \phi(h)$ is only almost excellent—because its defect of partial stability is one.

The stabilizer of $\det(Y)$ in $G = SL_{m^2}(\mathbb{C})$ consists of linear transformations of the form $Y \to AY^*B^{-1}$, thinking of Y as an $m \times m$ matrix, where Y^* is either Y or Y^T, $A, B \in GL_m(\mathbb{C})$. The stabilizer of $\mathrm{perm}(X)$ in $SL_{n^2}(\mathbb{C})$ is generated [22] by linear transformations of the form $X \to \lambda X \mu^{-1}$, thinking of X as an $n \times n$ matrix, where λ and μ are either diagonal or permutation matrices.

Let $H \subseteq G = SL_{m^2}(\mathbb{C})$ be the stabilizer of $\det(Y)$. Since $SL_m(\mathbb{C}) \times SL_m(\mathbb{C})$ is a subgroup of H, the subgroup restriction problem (Problem 1.5) in this context becomes the following problem.

PROBLEM 2.3 (Kronecker problem). *Given a partition λ of height at most m^2, find an explicit decomposition of $V_\lambda(G)$ as an $SL_m(\mathbb{C}) \times SL_m(\mathbb{C})$-module:*

$$V_\lambda(G) = \oplus_{\alpha,\beta} k^\lambda_{\alpha,\beta} V_\alpha(SL_m(\mathbb{C})) \otimes V_\beta(SL_m(\mathbb{C})),$$

where α, β range over partitions of height at most m.

The coefficients $k^\lambda_{\alpha,\beta}$'s here are the same as the Kronecker coefficients that arise in the internal product of Schur functions. The problem of decomposing the tensor product of two irreducible representations of the symmetric group S_m can be reduced to this problem [5]. This is one of the outstanding problems in the representation theory of symmetric groups.

2.2. Arithmetic (nonuniform) version of the $P \neq NP$ conjecture. This is a version of the usual $P \neq NP$ conjecture (the nonuniform version), which does not involve problems of positive characteristic and, hence, is addressed first.

Now h, g in the orbit closure problem (Problem 2.2) correspond to some integral functions that are NP-complete and P-complete, respectively. These functions have to be chosen judiciously, because most functions that arise in complexity theory, e.g., the one associated with the travelling salesman problem, do not have a nice stabilizer, as required in our approach. For a detailed definition of h and g, see Part I. We shall call $\Delta_V[f]$, $f = \phi(h)$, and $\Delta[g]$ for the specific h and g here the *class varieties* associated with the complexity classes NP and P. The conjecture that $NP \not\subseteq P$ in characteristic zero is then reduced to the problem of showing that the class variety $\Delta_V[f]$ associated with NP cannot be embedded in the class variety $\Delta_V[g]$ associated with P, for the parameters of the lower bound problem under consideration.

The following is an analogue of Theorem 2.1 in this context.

THEOREM 2.2. *The point $h \in P(W)$, for a suitable W, which corresponds to an NP-complete function as in* [26]*, is stable with respect to the action of $SL(W)$ on $P(W)$. Hence, the point $f = \phi(h)$ is partially stable.*

The h here is not completely characterized by its stabilizer, but almost so; cf. Part I. Hence it is almost excellent. The subgroup restriction problem, Problem 1.5, that arises for the stabilizer of h is essentially the well-known plethysm problem [5] in the theory of symmetric functions.

3. Why should obstructions exist? Before we go any further, we have to argue why obstructions should exist for the pairs (f, g) that arise in the lower bound problems under consideration.

Let us begin with an observation that for an orbit closure problem that arises in complexity theory, an obstruction for the pair (f, g) cannot exist if l is sufficiently larger than k. For example, let $(f, g) = (\phi(h), g)$, where $h = \text{perm}(X)$ and $g = \det(Y)$, as in section 2.1. Then there cannot be any obstruction for $m > n!$ or, for that matter, $m > 2^{cn}$ for a large enough constant c. This is because $\text{perm}(X)$ has a formula of size 2^{cn} for a large enough $c > 0$ [22] (the usual formula is of size $n!$), and hence $f \in \Delta_V[g]$, for $m > 2^{cn}$ (cf. Part I).

At the other extreme, when $l = k$, so that f is a stable point of V, it follows from the étale slice theorem [27, 17] that, if $f \in \Delta_V[g]$, then some conjugate of the stabilizer of f must be contained in the stabilizer of g (cf. Part I). This will not happen for our judiciously chosen f and g. For example, when f and g are the permanent and the determinant and $m = n$—in fact, in this case there are infinitely many obstructions to this containment (cf. Part I).

The goal is to understand the transition between these two extremes.

First, let us consider the arithmetic implication of the $P^{\#P} \neq NC$ conjecture. Let $g = \det(Y)$, $f = \phi(h)$, and $h = \text{perm}(X)$ as in section 2.1.

PROPOSITION 3.1. *Suppose $h = perm(X)$ cannot be approximated infinitesimally closely by a circuit of depth $O(\log^c n)$, where $c > 0$ is a constant, and $n \to \infty$. Suppose $X(\Pi_g) = \Delta_V[g]$ as varieties (cf. Conjecture* 1.9*). Then there exists a strong obstruction for the pair (f, g), for $m \leq 2^{\log^{c/2} n}$.*

Proof. It is proved in Part I that the hypothesis implies that $f \notin \Delta_V[g]$ if $m \leq 2^{\log^{c/2} n}$. Assuming $X(\Pi_g) = \Delta_V[g]$, this means $f \notin X(\Pi_g)$. Hence there exists a G-module $S \in \Pi_g$ which does not vanish on f, and hence on its orbit. So S occurs in $R_V[f]$. By the definition of Π_g, the dual S^* does not contain a G_g-module isomorphic to $(\mathbb{C}g)^d$. Hence S is a strong obstruction for the pair (f, g). □

Since $\text{perm}(X)$ is $\#P$-complete [31], it is not expected to have infinitesimally close approximations by circuits of $O(log^c(n))$ depth, for any constant $c > 0$. Hence, Proposition 3.1 leads to the following conjecture.

CONJECTURE 3.2. *There exist (infinitely many) strong obstructions for $(f, g) = (\phi(h), g)$, $g = \det(Y)$, $h = perm(X)$ if $m = 2^{\log^c n}$, c is a constant, and $n \to \infty$.*

In turn, this conjecture implies $f \notin \Delta_V[g]$ and, hence, the arithmetic implication of the $P^{\#P} \neq NC$ conjecture (section 2.1).

In the same vein, we also make the next conjecture.

CONJECTURE 3.3. *There exist (infinitely many) obstructions for $(f, g) = (\phi(h), g)$ that occur in the context of the P vs. NP problem if $m = poly(n)$ and $n \to \infty$, where n denotes the input size parameter and m denotes the circuit size parameter in the nonuniform version of the P vs. NP problem.*

This would imply $f \notin \Delta_V[g]$ and, hence, the arithmetic implication of the $P \neq NP$ conjecture in section 2.2. This conjecture is motivated by similar considerations as in Proposition 3.1. The g that occurs in the context of the P vs. NP problem is not fully characterized by its stabilizer. But it is still determined by its stabilizer to a large extent. Hence, similar considerations apply.

4. Admissibility. In this section, we introduce a basic notion of admissibility and study how it influences which G-modules may appear in the homogeneous coordinate ring $R_V[v]$ of the projective-orbit closure of a point $v \in P(V)$. The basic propositions proved here will be useful in the proofs of the main results.

DEFINITION 4.1. *Given a reductive subgroup $H \subseteq G$ and an H-module W, we say that a G-module M is (H, W)-admissible if some irreducible H-submodule of M occurs in W.*

We say that M is H-admissible if it is $(H, 1_H)$-admissible, where 1_H is the trivial H-module, i.e., if it contains a (nonzero) H-invariant.

For general H, not necessarily reductive, we say that M is H-admissible if M^* contains an H-invariant.

If H is reductive, M contains an H-invariant iff M^* does—this follows from Weyl's result on complete reducibility of a reductive group representation—and hence, the second and third statements are then equivalent.

Given a G-module S and a subgroup $H \subseteq G$, not necessarily reductive, we shall say that S has an H-coinvariant if S is H-admissible; i.e., the dual module S^* has an H-invariant (cf. Definition 4.1).

Let $h \in P(V)$ be any point, not necessarily stable. Let $\mathbb{C}h$ be the corresponding line in V. It is one-dimensional, i.e., a character, as a G_h-module, and trivial as a $G_{\hat{h}}$-module. Let $\check{\Delta}[h] = \check{\Delta}_V[h] \subseteq V$ denote the affine cone of the projective-orbit closure $\Delta[h]$. Its coordinate ring $\mathbb{C}[\check{\Delta}[h]]$ coincides with the homogeneous coordinate ring $R[h] = R_V[h]$ of $\Delta[h]$. Since the G-action is degree preserving, each homogeneous component $R[h]_d$ is a finite dimensional G-module.

PROPOSITION 4.2. *Let V be a linear representation of a reductive group G. Let $h \in P(V)$ be any point, not necessarily stable, with stabilizer $G_h \subseteq G$. Let S be any irreducible G-module occurring in $R[h]$—that is, in $R[h]_d$ for some d. Then the dual module S^* must contain a G_h-submodule isomorphic to $(\mathbb{C}h)^d$, and hence both S and S^* are $G_{\hat{h}}$-admissible. In particular, a G-module $S \subseteq \mathbb{C}[V]_d$ not satisfying this constraint belongs to the ideal $I_V[h]$.*

Similarly, given an algebraic subgroup $H \subseteq G$ and an H-module M, let $B = G \times_H M$ be the induced bundle [27] *with base space G/H and fiber M. Let N be any irreducible G-submodule of $\Gamma(G/H, B)$, the space of global sections of B. Then the G-module $Hom(N, M)$ must contain a nonzero H-invariant.*

Proof. Not all functions in S can vanish at $\hat{h}$: Otherwise, they will vanish identically on the G-orbit of $\hat{h}$ in V, and so also on its cone, since the functions in S are homogeneous. But the cone of the affine G-orbit of h is dense in $\check{\Delta}[h]$. Hence, it would follow that the functions in S vanish on $\check{\Delta}[h]$ identically, a contradiction.

Consider the G_h-equivariant map $\phi : S \to ((\mathbb{C}h)^*)^d = (\mathbb{C}h^d)^*$ that maps every function in S to its restriction on the line $\mathbb{C}h$. It follows that this evaluation map is nonzero. Hence the dual map ϕ injects the G_h-module $\mathbb{C}h^d$ into S^*.

The argument extends to the vector bundle B by considering instead the evaluation map $\phi : N \to M$ at the base point $e \in G/H$, which must be nonzero and H-equivariant; i.e., $\phi \in \mathrm{Hom}(N, M)^H$. ∎

5. Admissibility and stability. In this section we shall prove the first statement of Theorem 1.1 concerning stable points.

PROPOSITION 5.1. *Let $h \in P(V)$ be a point such that the stabilizer $H = G_{\hat{h}}$ of $\hat{h} \in V$ is reductive. Then every irreducible G-module occurring in $R[h]$ must be H-admissible; i.e., it must contain a nonzero H-invariant.*

If $G_{\hat{h}}$ is not reductive, this still holds if H is any reductive subgroup of $G_{\hat{h}}$.

Proof. If H is reductive, then Weyl's theorem on complete decomposibility of H-modules into irreducibles implies that the existence of an H-invariant is equivalent to the existence of an H-coinvariant. Hence this follows from Proposition 4.2. ∎

Conversely, we have the following proposition.

PROPOSITION 5.2. *Suppose $h \in P(V)$ is stable. Then every H-admissible, irreducible G-module occurs in $R[h]$.*

Proof. Since h is stable, the stabilizer $H = G_{\hat{h}}$ is reductive [3, 24], and the orbit $G\hat{h} \subseteq V$ is affine and isomorphic to G/H [27]. Moreover, an explicit G-module decomposition of the coordinate ring $\mathbb{C}[G\hat{h}] = \mathbb{C}[G/H]$ can be computed as follows. First, we recall (cf. [30, p. 48]) the algebraic version of the *Peter–Weyl theorem*:

$$\mathbb{C}[G] = \oplus_S S \otimes S^*, \tag{1}$$

where S ranges over all irreducible G-modules and S^* is the dual module. From this it follows that

$$\mathbb{C}[G/H] = \oplus_S S \otimes (S^*)^H, \tag{2}$$

where $(S^*)^H$ denotes the subspace of H-invariants in S^*. Since h is stable, the affine orbit $G\hat{h}$ is closed in V. So it is a closed G-subvariety of the cone $\check{\Delta}[h] \subseteq V$, which is also a G-variety. It follows that there is a G-equivariant surjection from $R[h]$ to $\mathbb{C}[G\hat{h}] = \mathbb{C}[G/H]$. Both $R[h]$ and $\mathbb{C}[G/H]$ have direct sum decompositions into finite dimensional G-modules. It follows that every irreducible G-module that occurs in $\mathbb{C}[G/H]$ must occur in $R[h]$. But by the Peter–Weyl theorem, i.e., (2), the irreducible G-modules that appear within $\mathbb{C}[G/H]$ are precisely the H-admissible ones. □

Proof of Theorem 1.1(a). Since $v \in P(V)$ is stable, $G_{\hat{v}}$ is reductive [3, 24]. Hence this follows from Propositions 5.1 and 5.2. □

6. SFT for the orbit of a stable, excellent point. In this section we prove Theorem 1.2 for stable points. To give its precise statement, we need a few definitions.

We associate with a stable point v representation-theoretic data Π_v and $\Sigma_v \subseteq \Pi_v$ as follows.

DEFINITION 6.1. *Suppose $v \in P(V)$ is stable.*

Let Σ_v be the set of all non–$G_{\hat{v}}$-admissible G-submodules of $\mathbb{C}[V]$—here $G_{\hat{v}}$ is necessarily reductive [3, 24].

Let $\Pi_v = \cup_d \Pi_v(d)$, where $\Pi_v(d)$ is the set of all irreducible G-submodules of $\mathbb{C}[V]$ whose duals do not contain a G_v-submodule isomorphic $(\mathbb{C}v)^d$—the dth tensor power of $\mathbb{C}v$.

Clearly $\Sigma_v \subseteq \Pi_v$. Basis elements (suitably chosen) of the G-submodules of Σ_v will be called *nonadmissible basis elements*.

PROPOSITION 6.2. *If v is stable, the G-modules in the representation-theoretic data Π_v, and hence Σ_v, associated with v are contained in $I_V[v]$.*

This follows from Proposition 4.2.

DEFINITION 6.3. *Given a reductive $H \subseteq G$, we say that a nontrivial, irreducible H-module L which occurs in some G-module is G-separable (from the trivial H-module) if there exists an irreducible non–H-admissible G-module M that contains L; we say it is strongly G-separable if there exist infinitely many such G-modules. We shall say that a subgroup $H \subseteq G$ is G-separable (strongly G-separable) if every nontrivial irreducible H-module which occurs in some G-module is G-separable (resp., strongly G-separable).*

For example, $SL_k(\mathbb{C}) \subseteq SL_n(\mathbb{C})$, $k > n/2$, and a semisimple $H \subseteq H \times H$ (diagonal embedding) are separable (Proposition 12.1). We conjecture that $SL_n(\mathbb{C}) \times SL_n(\mathbb{C}) \subseteq SL(\mathbb{C}^n \otimes \mathbb{C}^n) = SL_{n^2}(\mathbb{C})$ is separable and prove this for $n = 2$ (Proposition 12.6). We also conjecture that the stabilizers of the permanent, the determinant, and other functions that arise in our lower bound applications are G-separable; the stabilizer of

the determinant is very similar to the subgroup $SL_n(\mathbb{C}) \times SL_n(\mathbb{C}) \subseteq SL_{n^2}(\mathbb{C})$ above (cf. section 2.1).

A precise statement of Theorem 1.2 now reads as follows.

THEOREM 6.1. *Suppose V is a linear representation of a connected, reductive group G. Let $v \in P(V)$ be a stable point such that stabilizer $G_{\hat{v}}$ is G-separable (cf. Definition* 6.3*) and characterizes v.*

Then there exists a homogeneous G-invariant $\beta \in \mathbb{C}[V]$ not vanishing at v such that the ideal of Gv as a closed subvariety of the open neighborhood $U = P(V)_\beta = P(V) \setminus \{\beta = 0\}$ is generated by the nonadmissible basis elements in Σ_v—in fact, it is generated by the bases of less than $\operatorname{codim}(Gv, P(V))$ *irreducible, non–$G_{\hat{v}}$-admissible G-submodules of $\mathbb{C}[V]$.*

Remark. Since $\Sigma_v \subseteq \Pi_v$, this statement is slightly stronger than Theorem 1.2.

Theorem 6.1, in turn, follows from the following stronger result.

Let X be a nonsingular, affine G-variety, G a connected reductive group. Given a point $x \in X$, we shall denote by $[x] \subseteq X$ the subvariety consisting of all points in X whose stabilizers contain $H = G_x$, the stabilizer of x. Assume that x is a nonsingular point of $G \cdot [x]$; when the orbit $Gx \subseteq X$ is closed, this is automatically so, because of the étale slice theorem [17] (cf. the proof of Lemma 6.3). We shall denote by N_x (resp., $N_{[x]}$) the H-module that is an H-complement of the tangent space of $G \cdot x$ (resp., $G \cdot [x]$) at x in the total tangent space to X at x; it can be thought of as the "normal" space to $G \cdot x$ (resp., $G \cdot [x]$) at x. $N_{[x]}$ is the H-submodule of N_x consisting of all nontrivial H-submodules of N_x.

Given a G-invariant $\beta \in \mathbb{C}[X]$, we shall denote by $X(\beta)$ the G-variety obtained from X by removing the divisor $\{\beta = 0\}$.

We shall denote the codimension of a subvariety $Y \subseteq X$ by $\operatorname{codim}(Y, X)$. We say that an open subset $U \subseteq X$ is saturated if it is of the form $\psi^{-1}(U')$, where ψ is the projection from X to the quotient X/G and U' is an open subset of X/G.

Theorem 6.1 for stable points in $P(V)$ follows from the following result by letting $X = V$ and $x = \hat{v}$. When $v \in P(V)$ is characterized by the stabilizer $G_{\hat{v}}$, $[x] = \mathbb{C}v$. Passage from V to $P(V)$ is possible because the nonadmissible basis elements are homogeneous.

THEOREM 6.2. *Assume that G is a connected, reductive group, and X an affine, nonsingular, irreducible G-variety X. Let $x \in X$ be a point, with stabilizer $H = G_x$, whose orbit Gx is closed. Suppose every H-module L that appears in $N^*_{[x]}$ is G-separable (Definition* 6.3*). Then, for some G-invariant $\beta \in \mathbb{C}[X]$ not vanishing at x, and non–H-admissible, irreducible G-submodules $P_i \subseteq \mathbb{C}[X]$, $1 \le i \le r$, with $r < \operatorname{codim}(G \cdot [x], X)$, Spec$(\mathbb{C}[X]/J) \cap X(\beta) = G \cdot [x] \cap X(\beta)$, where J denotes the ideal generated by the P_i's.*

(Here we are identifying a variety with the corresponding reduced scheme supported by it.)

Proof. By Proposition 4.2, or rather its proof, the functions in every non–H-admissible P within $\mathbb{C}[X]$ must vanish on $G \cdot [x]$. We need to show that, for some G-invariant β not vanishing at x, the zero set of J within $X(\beta)$ equals $G \cdot [x] \cap X(\beta)$ scheme-theoretically.

Étale slice theorem (see [27, p. 198], [17]). Let x be a point of an affine, smooth, irreducible G-variety X, whose orbit $Gx \subseteq X$ is closed. Then there exists a smooth, affine H-variety $Y \subseteq X$ passing through x and a strongly étale map ψ from $G \times_H Y$ to a G-invariant neighborhood of $G \cdot x$ in X of the form $X(\alpha)$, for some G-invariant $\alpha \in \mathbb{C}[X]$.

Here $Z = G \times_H Y$ denotes the induced G-equivariant fiber bundle, with base G/H and fiber isomorphic to Y [27]. Strong étaleness of ψ means that the map ψ/G from the quotient Z/G to X/G is étale and that the induced natural map from Z to $X \times_{X/G} Z/G$, the G-variety obtained from X by base extension, is a G-isomorphism.

The slice theorem suggests that we prove our theorem in two steps. First, consider the case when X is a fiber bundle of the form $G \times_H Y$, where Y is a smooth affine variety, and then make a transition to the general case. Note that $H = G_x$ is reductive since $Gx \subseteq X$ is closed and hence affine [24].

We shall need the following proposition.

PROPOSITION 6.4. *Let V be a finite-dimensional irreducible G-module, G connected and reductive, with basis coordinate functions $V_1, \ldots, V_s$. Let $g \in V$ be a point with closed, affine orbit $Gg \subseteq V$. Further, let $I(g)$ be the ideal of Gg. Let J be an ideal of $\mathbb{C}[V]$ such that the following hold:*

(i) *The variety of J is precisely the orbit $O = Gg$.*

(ii) *The ideal J is itself G-invariant.*

(iii) *There are elements $w_1, \ldots, w_k \in J$ such that the tangent space TO_g of the orbit O at g consists of precisely the tangent vectors in TV_g annihilated by the differential forms dw_i's.*

Then $J = I(g)$; i.e., Spec$(\mathbb{C}[V]/J) = O$.

Suppose (i) *is replaced by the weaker condition:*

(i)′ *the variety of J contains the orbit $O = G \cdot g$.*

Then there exists a G-invariant neighborhood U_g of the orbit Gg such that the zero set of J restricted to U_g coincides with Gg scheme-theoretically.

Proof. The G-invariance of J and the connectedness of G imply that all associated primes of J must themselves be G-invariant. Since there are no proper G-invariant subsets of O, we conclude that there are no associated primes of J other than $I(g)$. Now (iii) may be used to apply the "Jacobian criterion" (Matsumura [21, Theorem 30.4]) locally. The G-invariance of J shows that (iii) holds at every point $y \in O$. The global assertion then follows. □

Given an H-module M, we denote by $\mathbb{C}[M]$ the H-module $\sum_{i \geq 0} \mathrm{Sym}^i(M^*)$, i.e., the space of polynomial functions on M. Let N denote the tangent space to Y at x; it is an H-module. Now we prove the theorem for the variety $G \times_H N$.

LEMMA 6.3. *The theorem holds when $X = G \times_H N$ and $x = (1_G, 0_N)$ is the base point on its null section G/H, with stabilizer $G_x = H$.*

Proof. In this case N can be identified with the normal space N_x at x to the orbit $G \cdot x = G/H$. Let $N = \sum_R R$ be an H-module decomposition of N into irreducibles. Then we can write $N_x = N_{[x]} + M_x$, where $\bar{N} = N_{[x]}$ is the sum of all nontrivial H-submodules R in this decomposition and M_x is the sum of all trivial H-submodules. The subvariety $G \cdot [x] = G \cdot M_x$, and the codimension of $G \cdot [x]$ is just the dimension of $N_{[x]}$.

For any H-submodule L of N_x, consider the induced bundle $F(L^*) : G \times_H L^* \to G/H$. Let $\mathcal{O}_{F(L^*)}$ be the sheaf of germs of sections of this bundle. Let $H^0(G/H, \mathcal{O}_{F(L^*)})$ be the G-module of its global sections. These global sections are regular functions on $G \times_H L$ that are linear on each fiber. Clearly $H^0(G/H, \mathcal{O}_{F(N^*_{[x]})})$ is a G-submodule of $H^0(G/H, \mathcal{O}_{F(N^*_x)})$, whose elements are regular functions on X linear on each fiber. Since G is connected, we can apply the Jacobian criterion (Proposition 6.4) and the transitivity of G-action. Hence it suffices to show that the sections in the non–H-admissible G-submodules of $H^0(G/H, \mathcal{O}_{F(N^*_{[x]})})$, when restricted to the fiber $N^*_{[x]}$ at x, span $N^*_{[x]}$; clearly the number r of such submodules is less than $\dim(N_{[x]}) = \mathrm{codim}(G \cdot [x], X)$.

Let $N_{[x]} = \oplus_R R$ be an H-module direct sum decomposition of $N_{[x]}$, where each R is an irreducible, nontrivial H-submodule. Then $N^*_{[x]} = \oplus_R R^*$, as an H-module, so we get a natural G-module decomposition

$$H^0(G/H, \mathcal{O}_{F(N^*_{[x]})}) = \oplus_R H^0(G/H, \mathcal{O}_{F(R^*)}).$$

Hence, it suffices to show that for each R in this decomposition, there exists a non–H-admissible G-submodule of $H^0(G/H, \mathcal{O}_{F(R^*)})$, whose sections, when restricted to the fiber R^* at x, span R^*.

Thus, let R be any such nontrivial, irreducible H-submodule in this decomposition and let $L = R^*$ be its dual. By the Peter–Weyl theorem (1)

$$H^0(G/H, \mathcal{O}_{F(L)}) = \oplus_Q Q \otimes Hom(Q, L)^H, \tag{3}$$

where Q ranges over all finite dimensional irreducible G-modules, and $Hom(Q, L)^H$ denotes the vector space of H-equivariant linear maps from Q to L. Thus the G-modules Q that appear in $H^0(G/H, \mathcal{O}_{F(L)})$ are precisely the ones that contain L. By our G-separability assumption, there exists a nonadmissible, irreducible G-module Q_L containing L. By (3), $H^0(G/H, \mathcal{O}_{F(L)})$ contains a copy of Q_L. Fix one such copy; we denote it by Q_L again. The restriction of Q_L to the fiber L of $F(L)$ at x is precisely L. Hence the basis elements of Q_L, when restricted to L, span L. □

For every R that appears in the H-module decomposition of $N_{[x]}$, let $Q_L \subseteq H^0(G/H, \mathcal{O}_{F(N^*_{[x]})})$, $L = R^*$, be a fixed copy as in the proof above. Let Φ be the set of such finitely many Q_Ls, each a non–H-admissible, irreducible G-module of regular functions on $G \times_H N$. The number r of Q_L's in Φ is less than $\operatorname{codim}(G \cdot [x], X)$. Since many R's in the H-module decomposition may be isomorphic, many Q_Ls in Φ may be isomorphic as G-modules. The proof above shows that the following lemma holds.

LEMMA 6.4. *The differentials of the basis elements of the non–H-admissible G-modules Q_L in Φ, when restricted to $N_{[x]}$, span the whole of $N_{[x]}$, and the zero set of the ideal generated by them coincides with $G \cdot [x] = G \cdot M_x$ scheme-theoretically. The number r of modules in Φ is less than codim$(G \cdot [x], G \times_H N)$.*

Now we turn to the general case. By the étale slice theorem, there exists an affine H-variety $Y \subseteq X$ passing through x and a strongly étale map ψ from $G \times_H Y$ to a G-invariant neighborhood of $G \cdot x$ in X of the form $X(\alpha)$, for some G-invariant $\alpha \in \mathbb{C}[X]$ not vanishing at x. Since $\mathbb{C}[X(\alpha)] = \mathbb{C}[X]_\alpha$, the ideal generated by non–H-admissible, irreducible G-submodules of $\mathbb{C}[X]$ within $\mathbb{C}[X(\alpha)]$ coincides with the one generated by non–H-admissible, irreducible G-submodules of $\mathbb{C}[X(\alpha)]$. Hence, in the statement of the theorem, we can replace X by $X(\alpha)$. Strong étaleness of ψ implies [27] that there is an analytic neighborhood $Y_{an} \subseteq N_x$ of x in N_x—called an analytic slice through x—such that $G \times_H Y_{an}$ is G-isomorphic to an analytic G-invariant neighborhood U of the orbit of x. However, there may not be an algebraic slice with this property, and this forces us into the analytic category in what follows. Since $U \simeq G \times_H Y_{an} \subseteq G \times_H N$, each Q_L corresponds to, and can be identified with, a G-module $Q_L(U)$ of analytic functions on U. By Lemma 6.4, the zero set of the $Q_L(U)$'s in Φ within U coincides, as a complex space [8], with $G \cdot [x] \cap U$. Our goal is to show that each $Q_L(U)$ can be approximated very closely within U by an isomorphic G-submodule of $\mathbb{C}[X]$. For this, we shall need the following results from complex function theory.

Cartan–Oka theorem (see [8]). Let A be a Stein space, and let B be its closed analytic subspace. Then every holomorphic function on B extends to a holomorphic function on A.

Let $\mathcal{O}_A$, $\mathcal{O}_B$ be the sheaves of germs of holomorphic functions on A and B, respectively, and let $\mathcal{I}_B$ be the sheaf of ideals of B. Then by Oka's theorem $\mathcal{I}_B$ is coherent, and since A is Stein, its higher cohomology $H^i(A, \mathcal{I}_B)$, $i > 0$, vanishes (Cartan's theorem B). Hence, this result follows from the long exact cohomology sequence associated with the exact sequence of sheaves

$$0 \to \mathcal{I}_B \to \mathcal{O}_A \to \mathcal{O}_B \to 0,$$

where we consider $\mathcal{O}_B$ as a sheaf on A via extension by zero.

We shall denote the ring of holomorphic functions on an analytic variety W by $\mathcal{O}(W)$.

LEMMA 6.5. *Let A be a Stein G-space, and let B be its closed analytic G-subspace, with G a connected reductive group. Let M be a finite dimensional G-submodule of $\mathcal{O}(B)$. Then there exists a G-equivariant extension map $\phi : M \to \mathcal{O}(A)$.*

Here, we say that ϕ is an extension map if, for any $s \in M$, the restriction of $\phi(s)$ to B coincides with s.

Proof. Fix a basis $s_1, \ldots, s_l$ of M. By the Cartan–Oka theorem, each s_i can be extended to a holomorphic function $\hat{s}_i$ on A. Let $\rho : M \to \mathcal{O}(A)$ be a linear map defined by setting $\rho(\sum_i b_i s_i) = \sum_i b_i \hat{s}_i$. Though ρ need not be G-equivariant, it may be converted into a G-equivariant map by Weyl's unitary trick [5]. Specifically, regard $Hom(M, \mathcal{O}(A))$ as a G-module in the natural way. Fix a maximal compact subgroup $E \subseteq G$. Let de denote the left-invariant Haar measure on E, and let

$$\phi = \int_E e(\rho) de.$$

Then ϕ is an E-equivariant extension. Since M is finite dimensional, it follows from the unitary trick that ϕ is G-equivariant as well. □

LEMMA 6.6. *Let W be a linear representation of connected, reductive G, V a linear space with trivial G action, and $D \subseteq V$ a ball around the origin. Let $A = W \times D$. Then the following hold:*

(1) *Any holomorphic function a on A has a unique power series expansion of the form*

$$a(w, v) = \sum_{i,j} a_i^j w^j v^i. \tag{4}$$

Here $i = (i_1, \cdots, i_r)$, $r = \dim(V)$, and $j = (j_1, \cdots, j_q)$, $q = \dim(W)$, are tuples of nonnegative integers, and $v^i = v_1^{i_1} \cdots v_r^{i_r}$ and $w^j = w_1^{j_1} \cdots w_q^{j_q}$, where $v_1, \ldots, v_r$ are the coordinates of V and $w_1, \ldots, w_q$ are the coordinates of W.

(2) *For any $k = (i, d)$, the map $\delta_k : \mathcal{O}(A) \to \mathbb{C}[W \times V] = \mathbb{C}[W] \otimes \mathbb{C}[V]$, which maps a to $\sum_{j : j_1 + \cdots + j_q = d} a_i^j w^j v^i$, is G-equivariant.*

Proof. The first statement follows because A is a proper Reinhardt domain in $W \times V$ (cf. [6, p. 20]).

Each $w_1^{j_1} \cdots w_q^{j_q}$ is a polynomial (regular) function on W and is contained in the finite dimensional G-submodule in $\mathbb{C}[W]$ of homogeneous forms of degree $d = j_1 + \cdots + j_q$. Since the G-action on D is trivial, it follows that each δ_k is G-equivariant. □

Let X be an affine, smooth G-variety, with G a connected reductive group. Let ψ be the projection from X to its quotient X/G. Let x be a point in X with closed orbit $Gx \subseteq X$ and $\bar{x} = \psi(x)$ its projection. Embed the affine variety X/G in a linear

space V, with $\bar{x}$ at its origin. Suppose $U_{\bar{x}}$ is a Stein neighborhood of $\bar{x}$ in X/G such that $U_{\bar{x}} = D \cap X/G$, where $D \subseteq V$ is a ball around $\bar{x}$. Let $U = \psi^{-1}(U_{\bar{x}})$.

LEMMA 6.7. *Let Q be a finite dimensional G-submodule of $\mathcal{O}(U)$. Then there exist G-equivariant linear maps $\rho_k : Q \to \mathbb{C}[X]$ such that any $s \in Q$ admits a power series expansion $s = \sum_{k=0}^{\infty} \rho_k(s)$ that converges everywhere in U.*

Proof. We can embed X G-equivariantly as a closed affine G-subvariety of some linear representation W of G [10, Lemma 1.1]. Let $A = W \times D$, which is Stein. It has a G-action, the action on D being trivial. Let $B \subseteq A$ be the closed analytic G-subspace consisting of points (x, u), with $x \in X$, $u \in U_{\bar{x}}$, and $\psi(x) = u$. It is isomorphic to U. So s corresponds to a holomorphic function on B, which we shall denote by s again. Thus we can regard $Q \subseteq \mathcal{O}(B)$.

By Lemma 6.5, there exists a G-equivariant extension map $\phi : Q \to \mathcal{O}(A)$. Let δ_k be the G-equivariant map of Lemma 6.6 applied to A. Finally, let $\alpha : \mathbb{C}[W \times V] \to \mathbb{C}[X]$ be the G-equivariant restriction map corresponding to the G-equivariant embedding $X \to W \times V$, which maps $x \in X$ to $(x, \psi(x))$. Let $\rho_k = \alpha \circ \delta_k \circ \phi$. Then $s \in Q$ has a G-equivariant power series expansion

$$s = \sum_k \rho_k(s)$$

that converges everywhere in U. ∎

Now we return to the proof of Theorem 6.2. Let ψ be the strongly étale map from $G \times_H Y_{an}$ to a G-invariant neighborhood U of the orbit $G \cdot x$. Here $Y_{an} \subseteq N_x$ is an analytic slice, and U is of the form $\psi^{-1}U_{\bar{x}}$, where $U_{\bar{x}}$ is an analytic neighborhood of $\bar{x} = \psi(x)$. We can assume that $U_{\bar{x}}$ is Stein, of the form $D \cap X/G$ as in Lemma 6.7, for a small enough ball D around $\bar{x}$ in $V \supseteq X/G$. Let $Q_L \in \Phi$ be the finitely many, irreducible, non–H-admissible G-submodules of the ring of regular functions on $G \times_H N$ as in Lemma 6.4; their number r is less than $\operatorname{codim}(G \cdot [x], G \times_H N) = \operatorname{codim}(G \cdot [x], X)$. We shall denote the restriction of Q_L to $G \times_H Y_{an}$ by Q_L again. It corresponds to a G-module of analytic functions on U, which we shall denote by $Q_L(U)$; the analytic functions in $Q_L(U)$, though, may not extend to the whole of X.

Now we come to the crux of the proof. The G-module $Q_L(U)$ is isomorphic to Q_L and, hence, finite dimensional. Hence we may apply Lemma 6.7. Let $\rho_k(L)$ denote the G-equivariant projection from $Q_L(U)$ to $\mathbb{C}[X]$ therein. Let $\tilde{\rho}_k(L) = \sum_{j \le k} \rho_j(L)$. When k is large enough, $\tilde{\rho}_k(L)(Q_L(U))$ will be a good approximation to $Q_L(U)$. Let $Q_L^k \subseteq \mathbb{C}[X]$ be the G-module that is the image of this G-equivariant projection $\tilde{\rho}_k$. Since $Q_L(U) \simeq Q_L$ is irreducible, Q_L^k is either zero or isomorphic to Q_L. When k is large enough, Q_L^k is isomorphic to Q_L—hence it is non–H-admissible and vanishes on $G \cdot [x]$.

Since U is G-isomorphic to $G \times_H Y_{an} \subseteq G \times_H N$, it follows from Lemma 6.4 that the differentials of the basis functions in all the $Q_L(U)$'s in Φ, when restricted to $N_{[x]}$, span the whole of $N^*_{[x]}$. We approximate each $Q_L(U)$ by $Q_L^k \subseteq \mathbb{C}[X]$ for a large enough k. When k is large enough, the differentials of the basis functions in Q_L^k, when restricted to $N_{[x]}$, will also span the whole of $N^*_{[x]}$. But each Q_L^k is a non–H-admissible, irreducible G-submodule of $\mathbb{C}[X]$. Thus it follows that the differentials of the basis functions of the non–H-admissible, irreducible G-submodules Q_L^k of $\mathbb{C}[X]$, for k large enough, span $N^*_{[x]}$. Because of the transitivity of the G-action, the same holds for all points in the orbit of x. Since G is connected, and all Q_L's are G-modules, it now follows from the Jacobian criterion (Proposition 6.4, or rather its proof) and the fact that $U \simeq G \times_H Y_{an}$, that the zero set of the basis functions of these Q_L^k's within U

coincides with $G \cdot [x] \cap U$ scheme-theoretically (i.e., as a complex space [8]). Since Q_L^k's are G-submodules of $\mathbb{C}[X]$, there exists a Zariski-open G-invariant neighborhood $U' \supseteq U$ such that the zero set of Q_L^k's within U' coincides with $G \cdot [x] \cap U'$ scheme-theoretically. It remains to show that U' can be chosen to be of the form $X(\beta)$, for some G-invariant β. The projection $\psi(U')$ into X/G is a constructible [9] set that contains $U_{\bar{x}}$. Hence $\psi(U')$ contains a Zariski-open affine neighborhood of the form $(X/G)_\alpha$ for some G-invariant α not vanishing at x. Its inverse $\psi^{-1}(X/G)_\alpha$ is of the form $X(\alpha)$ and has the required properties. □

Remark. Suppose every H-module that appears in $N^*_{[x]}$ is not G-separable, as assumed in Theorem 6.2. Then one can similarly prove a weaker assertion that, for some G-invariant analytic neighborhood U (as in the proof above) of Gx, $\mathrm{Spec}(\mathbb{C}[X]/J) \cap U$, as a complex space [7], is a subspace of $G \times_H \mathrm{Spec}(I)$, where $\mathrm{Spec}(I)$ is a subscheme of $N_{[x]}$ and $I \subseteq \mathbb{C}[N_{[x]}]$ is the ideal generated by the G-separable H-submodules of $\mathbb{C}[N_{[x]}]$.

7. Partial stability. Let V be a linear representation of G. Let $P = KU$ be a parabolic subgroup of G, and let R be a reductive subgroup of K.

DEFINITION 7.1. *We say that $v \in P(V)$ is (R, P)-stable (*partially stable*) if* (1) *it is stable with respect to the restricted action of R on V, and* (2) *$U \subseteq G_v \subseteq P$.*

Here $U \subseteq G_v$ implies that $U \subseteq G_{\hat{v}}$. The defect $\delta(v)$ of v is defined to be the difference between the ranks of the root systems of R and K. In our applications, the defect will be small—in fact, just one—and R will always be a semisimple Levi subgroup of a parabolic subgroup of K—so that the root system of R will always be a subsystem of that of K.

A stable point of V is (G, G)-stable. A point $v \in P(V)$ is (R, P)-stable iff it is an (R, K)-stable point of $P(Y)$, where $Y = V^U$ is the K-module of U-invariants in V.

Example 1. The simplest example of a partially stable point with defect zero is the point $v = v_\lambda \in P(V)$ that corresponds to the highest weight vector of an irreducible G-representation $V = V_\lambda(G)$. The stabilizer $P = G_v$ is parabolic, and v is clearly (L, P)-stable, where L is a semisimple Levi subgroup of P.

Example 2. Let $f = \phi(h)$ be as in Definition 2.1, with h stable. Then f is (R, P)-stable, with defect one, with respect to the action of G (as well as $\hat{G}$), where: P is a parabolic subgroup of G (resp., $\hat{G}$), whose elements transforms the variables in $\bar{X}$ to their linear combinations, thus preserving an appropriate flag $\mathbb{C}^{k+1} \subseteq \mathbb{C}^l$, and $R \simeq SL_k(\mathbb{C}) \times SL_{l-k-1}$ is naturally embedded in the semisimple Levi subgroup of P isomorphic to $SL_{k+1}(\mathbb{C}) \times SL_{l-k-1}(\mathbb{C})$.

DEFINITION 7.2. *Given dominant weights α and β of R and K, we shall say that $\alpha \lhd_R^K \beta$, or $\beta \rhd_R^K \alpha$, if $V_\alpha(R)$ occurs in $V_\beta(K)$, dropping the superscript or subscript whenever possible.*

In the definition of (R, P)-stability the group R will usually be such that

$$\tilde{L} \subseteq R \subseteq \tilde{K} \subseteq K, \tag{5}$$

for some parabolic subgroup $\tilde{P} = \tilde{T}\tilde{L}\tilde{U} = \tilde{K}\tilde{U}$ of K, as in Example 2. Then, using Littelmann's restriction rule [14], one can determine how any irreducible representation $V_\beta(K)$ explicitly decomposes as a $\tilde{K}$-module (and hence as an R-module). This, in turn, gives an explicit relationship between α and β in Definition 7.2.

In Example 2 above, $K \simeq GL_{1+k}(\mathbb{C}) \times GL_{l-1-k}$ and $R \simeq SL_k(\mathbb{C}) \times SL_{l-1-k}(\mathbb{C})$. In this case, Littelmann's restriction rule reduces to a variant of the well-known Pieri's branching rule [5], which gives an explicit decomposition of $V_\mu(GL_{1+k}(\mathbb{C}))$ as a $GL_k(\mathbb{C})$-module.

For a connected reductive group D, we shall denote by i_D the canonical involution of its dominant weights so that $V_\lambda(D)^* = V_{i_D\lambda}(D)$. Let $v \in P(V)$ be an (R,P)-stable point as above. Let W and Y be, respectively, the smallest K-submodule and R-submodule of V containing $\hat{v}$.

DEFINITION 7.3. *We say that a dominant weight β of G lies over a weight μ of R at v and degree d if*

1. *$V_\mu(R)$ and $V_{\beta'}(K)$ occur in $R_Y[v]_d$ and $R_W[v]_d$, respectively, where $\beta' = i_K(i_G\beta)$, and*
2. *$\mu \lhd_R^K \beta'$.*

We say that a dominant weight β of G lies over a weight μ of R at v if this is so at some d.

This definition does not depend on the choice of a Levi subgroup $K \supseteq R$ of P, because $U \subseteq G_v$. When the defect is zero and R satisfies (5), condition 2 just says that the weight β', restricted to R, is equal to μ. The number of β's lying over μ at a fixed d depends on the defect; it is small if the defect is small.

8. Borel–Weil for a partially stable point. In this section we shall prove Theorem 1.1(b) for partially stable points. Its precise statement is as follows.

THEOREM 8.1. *Suppose v is (R,P)-stable (cf. Definition 7.1). Then $V_\lambda(G)$ can occur in $R_V[v]$ only if λ lies over some $R_{\hat{v}}$-admissible dominant weight μ of R at v (cf. Definition 7.3). Conversely, for every $R_{\hat{v}}$-admissible dominant weight μ of R, $R_V[v]$ contains $V_\lambda(G)$ for some dominant weight λ of G lying over μ at v.*

This will follow from the following stronger result.

Suppose $v \in P(V)$ is partially stable, specifically (R,P)-stable, where $P = TLU = KU$ and $R \subseteq K$.

Let W and Y be, respectively, the smallest K-submodule and R-submodule of V containing $\hat{v}$. Let $\mathcal{O}(d)$ be the twisting sheaf on $P(V)$, and let $\mathcal{O}_{\Delta_V[v]}(d)$, $\mathcal{O}_{\Delta_W[v]}(d)$ be the corresponding invertible sheaves on $\Delta_V[v]$ and $\Delta_W[v]$, respectively. Let $\Gamma(\Delta_V[v], \mathcal{O}_{\Delta_V[v]}(d))$ and $\Gamma(\Delta_W[v], \mathcal{O}_{\Delta_W[v]}(d))$ be the G- and K-modules of their global sections. Clearly $R_V[v]_d \subseteq \Gamma(\Delta_V[v], \mathcal{O}_{\Delta_V[v]}(d))$ for all $d \geq 0$. We have an equality for all $d \geq 0$ iff $\Delta_V[v]$ is projectively normal (cf. Hartshorne [9, p. 126]). Similarly, $R_W[v]_d \subseteq \Gamma(\Delta_W[v], \mathcal{O}_{\Delta_W[v]}(d))$ for all $d \geq 0$, with equality if $\Delta_W[v]$ is projectively normal.

The following result shows that the G-module structure of $R_V[v]$ is ultimately related to the R-module structure of $R_Y[v]$. In turn, we already know which R-modules can occur in $R_Y[v]$ since $v \in P(Y)$ is stable with respect to the action of R (Theorem 1.1(a)).

THEOREM 8.2 (Borel–Weil for partially stable points). *Suppose $v \in P(V)$ is (R,P)-stable as above. Then*

1. *The G-module structure of $R_V[v]$ is equivalent to the K-module structure of $R_W[v]$: Specifically, the multiplicity of a G-module $V_\lambda(G)$ in $R_V[v]_d^*$ is equal to the multiplicity of the K-module $V_\lambda(K)$ in $R_W[v]_d^*$, where λ is regarded as a dominant weight of K by restriction. Moreover, a K-module $V_\alpha(K)$ can occur in $R_W[v]_d^*$ only if α is also a dominant weight of G.*
2. *The multiplicity of $V_\lambda(G)$ in the module $\Gamma(\Delta_V[v], \mathcal{O}_{\Delta_V[v]}(d))^*$ of global sections of $\mathcal{O}_{\Delta_V[v]}(d)$ is less than or equal to the multiplicity of $V_\lambda(K)$ in $\Gamma(\Delta_W[v], \mathcal{O}_{\Delta_W[v]}(d))^*$. If $\Delta_W[v]$ is projectively normal, then the two multiplicities are equal, for all λ and $d \geq 0$, and $\Delta_V[v]$ is also projectively normal.*
3. *A K-module $V_\beta(K)$ can occur in $R_W[v]_d$ only if, for some dominant weight $\alpha \lhd_R^K \beta$ of R, $V_\alpha(R)$ occurs in $R_Y[v]_d$. Conversely, for every R-module $V_\alpha(R)$*

occurring in $R_Y[v]_d$, there exists a dominant weight $\beta \rhd_R^K \alpha$ of K such that $V_\beta(K)$ occurs in $R_W[v]_d$.

4. *Finally, an R-module $V_\mu(R)$ occurs in $R_Y[v]$, i.e., in some $R_Y[v]_d$, iff it is $R_{\hat{v}}$-admissible.*

Remark 1. In the third statement, it is desirable that we have an explicit criterion for deciding if $\alpha \lhd_R^K \beta$. When R satisfies (5), such a criterion is given by Littlemann's rule as pointed out there.

Remark 2. When G is semisimple and simply connected and v corresponds to the highest weight vector in $V = V_\lambda(G)$, $\Delta_V[v] = G/P$ and $\Delta_W[v]$ is just the point v. Hence $\Gamma(\Delta_W[v], \mathcal{O}_{\Delta_W[v]}(d))^* = V_{d\lambda}(K)$, for $d \geq 0$. The second statement now implies that $\Gamma(\Delta_V[v], \mathcal{O}_{\Delta_V[v]}(d))^* = \Gamma(G/P, \mathcal{O}_{G/P}(d))^* = V_{d\lambda}(G)$ for $d \geq 0$, which is the Borel–Weil theorem [11].

We will first prove two propositions. For that we need the following lemma from representation theory.

LEMMA 8.3 (cf. [11, Theorem 5.104]). *Let $V_\lambda(G)$ be an irreducible representation of a connected reductive group G with highest weight λ. Let $P = KU \subseteq G$ be a parabolic subgroup. Then $V_\lambda(G)^U = V_\lambda(K)$; here $V_\lambda(G)^U$ is the subspace of U-invariants in $V_\lambda(G)$.*

Let $z \in P(V)$ be a point whose stabilizer $G_z \subseteq G$ contains U, so that the stabilizer $G_{\hat{z}} \subseteq G$ of $\hat{z} \in V$ also contains U. Let Z be the smallest K-submodule of V containing $\hat{z}$. Let i denote the embedding of Z in V. The following result shows that $R_Z[z]$ and $R_V[z]$ are closely related.

PROPOSITION 8.1. (a) *The multiplicity of an irreducible module $V_\lambda(G)$ in $R_V[z]_d^*$ is equal to the multiplicity of $V_\lambda(K)$ in $R_Z[z]_d^*$. Moreover, $V_\alpha(K)$ can occur in $R_Z[z]_d^*$ only if α is also a dominant weight of G.*

(b) *The multiplicity of $V_\lambda(G)$ in $\Gamma(\Delta_V[z], \mathcal{O}_{\Delta_V[z]}(d))^*$ is less than or equal to the multiplicity of $V_\lambda(K)$ in $\Gamma(\Delta_Z[z], \mathcal{O}_{\Delta_Z[z]}(d))^*$. If $\Delta_Z[z]$ is projectively normal, then the two multiplicities are equal for all λ and $d \geq 0$, and $\Delta_V[z]$ is also projectively normal.*

Proof. Since the stabilizer $G_{\hat{z}}$ contains U, and U is normalized by K, the stabilizer of every point in Z contains U; in other words, the action of U on Z is trivial. Thus Z can be considered a P-module. The embedding map $i : Z \to V$ is then P-equivariant. By restriction, we get a P-equivariant, closed embedding $i : \check{\Delta}_Z[z] \to \check{\Delta}_V[z]$, where $\check{\Delta}_V[z] \subseteq V$ and $\check{\Delta}_Z[z] \subseteq Z$ denote the affine cones of $\Delta_V[z]$ and $\Delta_Z[z]$. Hence, the corresponding surjection $i^* : R_V[z] \to R_Z[z]$ is P-equivariant. Since it is degree preserving, by restriction, we get a P-equivariant surjection $i^* : R_V[z]_d \to R_Z[z]_d$ for every d. Since the action of U on Z is trivial, we get the dual injection $i : (R_Z[z]_d)^* \to (R_V[z]_d^*)^U$.

Let M be any irreducible G-submodule of $R_V[z]_d$. Not all functions in M can vanish at z—otherwise, arguing as in the proof of Proposition 4.2, we can conclude that the functions in M vanish identically on the affine cone $\check{\Delta}_V[z]$, which is not possible. It follows that the restriction map i^* is nonzero on M. Thus $N = i^*(M)$ is a nonzero K-module, with trivial U-action. Dually, this means $i(N^*)$ is a nonzero K-submodule of $(M^*)^U$. If $M^* = V_\lambda(G)$, then $(M^*)^U = V_\lambda(K)$ (Lemma 8.3), and hence is irreducible. So $(M^*)^U \simeq i(N^*)$. Thus the injection $i : (R_Z[z]_d)^* \to (R_V[z]_d^*)^U$ is an isomorphism. Hence the multiplicity of $V_\lambda(G)$ in $R_V[z]_d^*$ is equal to the multiplicity of $V_\lambda(K)$ in $R_Z[z]_d^*$, and, moreover, $V_\alpha(K)$ can occur in $R_Z[z]_d^*$ only if α is also a dominant weight of G. This proves (a).

The proof of (b) is similar. The embedding map $i : Z \to V$ induces a P-equivariant

map $i^* : \Gamma(\Delta_V[z], \mathcal{O}_{\Delta_V[z]}(d))^* \to \Gamma(\Delta_Z[z], \mathcal{O}_{\Delta_Z[z]}(d))^*$, which need not be a surjection in general. Let M be any irreducible G-submodule of $\Gamma(\Delta_V[z], \mathcal{O}_{\Delta_V[z]}(d))$. One shows similarly that $N = i^*(M)$ is a nonzero K-submodule of $\Gamma(\Delta_Z[z], \mathcal{O}_{\Delta_Z[z]}(d))$, with trivial U-action, and $(M^*)^U \simeq i(N^*)$. This proves the first statement of (b).

If $\Delta_Z[z]$ is projectively normal, i.e., its homogeneous coordinate ring is integrally closed, then $\Gamma(\Delta_Z[z], \mathcal{O}_{\Delta_Z[z]}(d)) = R_Z[z]_d$, for $d \geq 0$ (cf. Hartshorne [9, p. 126]). Hence i^* is a surjection, for $d \geq 0$, since the restriction $i^* : R_V[z]_d \to R_Z[z]_d$ is surjective, and $R_V[z]_d \subseteq \Gamma(\Delta_V[z], \mathcal{O}_{\Delta_V[z]}(d))$. Now we prove equality of multiplicities as in (a). Since the multiplicity of every $V_\lambda(G)$ in $\Gamma(\Delta_V[z], \mathcal{O}_{\Delta_V[z]}(d))$ or $R_V[z]_d$ is now the same, both being equal to the multiplicity of $V_\lambda(K)$ in $\Gamma(\Delta_Z[z], \mathcal{O}_{\Delta_Z[z]}(d))^*$, it now follows that $\Gamma(\Delta_V[z], \mathcal{O}_{\Delta_V[z]}(d)) = R_V[z]_d$, for all $d \geq 0$. Hence $R_V[z]$ is integrally closed and $\Delta_V[z]$ is projectively normal (cf. Hartshorne [9, p. 126]). □

Now let W be any linear representation of a connected, reductive group K, and let $R \subseteq K$ be a reductive subgroup. Fix a point $y \in P(W)$. Let Y be the smallest R-submodule of W containing $\hat{y}$.

PROPOSITION 8.2. *An irreducible K-module $V_\beta(K)$ can occur within $R_W[y]_d$ only if an R-module $V_\alpha(R)$, with $\alpha \lhd_R^K \beta$, occurs within $R_Y[y]_d$. Conversely, if an R-module $V_\alpha(R)$ occurs in $R_Y[y]_d$, then there exists a K-module $V_\beta(K)$, with $\beta \rhd_R^K \alpha$, in $R_W[y]_d$.*

Proof. The embedding $r : Y \to W$ is R-equivariant. Hence, we have an R-equivariant, closed embedding $r : \check{\Delta}_Y[y] \to \check{\Delta}_W[y]$ of the affine cone of $\Delta_Y[y]$, and the corresponding R-equivariant surjection $r^* : R_W[y] \to R_Y[y]$. Since this surjection is degree preserving, by restriction, we get an R-equivariant surjection $r^* : R_W[y]_d \to R_Y[y]_d$ for each d.

Let $V_\beta(K)$ be any irreducible K-module in $R_W[y]_d$. Arguing as in the proof of Proposition 8.1, we can conclude that its image under r^* is nontrivial. The image can thus be identified with an R-submodule of $V_\beta(K)$. If an R-module $V_\alpha(R)$ occurs in this image, then by definition (cf. section 7), $\alpha \lhd \beta$. Conversely, for every R-module $V_\alpha(R)$ that appears in $R_Y[y]_d$, there is a K-module $V_\beta(K)$ in $R_W[y]_d$ whose image contains $V_\alpha(K)$, and hence we must have $\beta \rhd \alpha$. □

Proof of Theorem 8.2. The first and second statements follow from Proposition 8.1, letting $z = v$, $Z = W$. The third statement follows from Proposition 8.2. The fourth statement follows statement (a) of Theorem 1.1, since, by definition of partial stability, v is a stable point of Y with respect to the action of R. □

Proof of Theorem 8.1. Suppose $V_\lambda(G)$ occurs in $R_V[v]_d$; i.e., $V_{i_G\lambda}(G)$ occurs in $R_V[v]_d^*$. Then by the first statement of Theorem 8.2, $V_{i_G\lambda}(K)$ occurs in $R_W[v]_d^*$. That is, $V_{i_K(i_G\lambda)}(K)$ occurs in $R_W[v]$. It now follows from the third and fourth statements of Theorem 8.2 that λ lies over some $R_{\hat{v}}$-admissible weight μ of R.

Conversely, it follows from Theorem 8.2 similarly that, for every $R_{\hat{v}}$-admissible dominant weight μ of R, $R_V[v]$ contains $V_\lambda(G)$ for some dominant weight λ of G lying over μ at v. □

9. Application in complexity theory. We now specialize the Borel–Weil theorem for partially stable points (section 8) to the orbit closure problem that arises in complexity theory (section 2). We follow the notation of section 2. Now $V = \mathrm{Sym}^m(Y)$ is a linear representation of $G = SL(Y) = SL_l(\mathbb{C})$, and $W = \mathrm{Sym}^n(X)$ is a representation of $SL(X) = SL_k(\mathbb{C})$. Let $\hat{G} = GL_l(\mathbb{C})$. Let i^l denote the involution on the weights of $GL_l(\mathbb{C})$ so that $V_\lambda(GL_l(\mathbb{C}))^* = V_{i^l\lambda}(GL_l(\mathbb{C}))$, for a weight λ. Recall that every weight λ of $GL_l(\mathbb{C})$ or its dual $i^l(\lambda)$ corresponds to a Young diagram of height at most l. Every weight of $GL_l(\mathbb{C})$ that occurs in $\mathbb{C}[V]_d^* = \mathrm{Sym}^d(V) =$

$\mathrm{Sym}^d(\mathrm{Sym}^m(Y))$ corresponds to a Young diagram of size md—this will be used implicitly in what follows.

THEOREM 9.1. (a) *Suppose $g \in P(V)$ is stable with respect to the action of G. Then a Weyl module $V_\lambda(G)$ occurs in $\Delta_V[g]$ iff it is $G_{\hat{g}}$-admissible.*

(b) *Suppose $f \in P(V)$ is of the form $\phi(h)$, $h \in P(W)$. Then $V_\lambda(\hat{G})$ can occur in $R_V[f]_d$ only if* (1) *the weight $i^l(\lambda)$ corresponds to a Young diagram with md boxes and height at most $k+1$, and* (2) *$V_{\lambda'}(GL_{k+1}(\mathbb{C}))$, with $\lambda' = i^{k+1} \circ i^l(\lambda)$, contains some $SL_k(\mathbb{C})_{\hat{h}}$-admissible module $V_\mu(SL_k(\mathbb{C}))$, where we consider $SL_k(\mathbb{C})$ as a subgroup of $GL_{k+1}(\mathbb{C})$ in a natural way. This means μ and λ' are related by (a variant of) Pieri's branching rule.*

Conversely, for every $SL_k(\mathbb{C})_{\hat{h}}$-admissible module $V_\mu(SL_k(\mathbb{C}))$, there exist a d and λ satisfying (1) *and* (2) *above such that $V_\lambda(\hat{G})$ occurs in $R_V[g]_d$.*

Proof. (a) follows from Theorem 1.1(a).

(b) The point $f \in P(V)$ is partially stable with defect one with respect to the action of $\hat{G} = GL_l(\mathbb{C})$ on $P(V)$, specifically, (R,P)-stable, with R and P as specificed in section 2. Now we apply Theorem 8.2 for the action of $\hat{G}$ on $P(V)$. We will only clarify why the height of $i^l(\lambda)$ is at most $k+1$. The reductive Levi subgroup of P under consideration is $K \simeq GL_{k+1} \times GL_{l-k-1}$, and the subgroup $1 \times GL_{l-k-1}$, where 1 denotes the identity in GL_{k+1}, is contained in the stabilizer $K_{\hat{f}}$. Suppose $V_\lambda(G)$ occurs in $R_V[f]_d$. The irreducible K-submodule of V containing f is just $\bar{W} = \mathrm{Sym}^m(\bar{X})$ defined in section 2. Hence, by Theorem 8.2, $V_{i_K \circ i^l \lambda}(K)$ is a nonzero K-submodule of $R_{\bar{W}}[f]_d$, where i_K is the involution on the weights of K. By Proposition 5.1, $V_{i_K \circ i^l \lambda}(K)$ and, hence, $V_{i^l \lambda}(K)$ must be $K_{\hat{f}}$-admissible and, hence, $1 \times GL_{l-k-1}$-admissible. For any $V_\alpha(GL_l(\mathbb{C}))$, where α is a Young diagram of height $\leq l$, the K-module $V_\alpha(K)$, with the same weight, is equal to $V_{\alpha_1}(GL_{k+1}) \otimes V_{\alpha_2}(GL_{l-k-1})$, where α_1 consists of the first $k+1$ rows of α and α_2 consists of the remaining $l-k-1$ rows; here an empty row is treated as a row with zero length. Let $\alpha = i^l(\lambda)$. Then $V_{\alpha_2}(GL_{l-k-1})$ must be trivial since $V_\alpha(K)$ is $1 \times GL_{l-k-1}$-admissible; thus $\alpha_2 = 0$, and $\alpha_1 = \alpha$. It follows that the length of α is at most $k+1$. The number of boxes in $i^l(\lambda)$ must be md since every irreducible $\hat{G}$-representation occurring in $\mathbb{C}[V]_d^* = \mathrm{Sym}^d(\mathrm{Sym}^m(Y))$ has degree md.

The rest follows from Theorems 8.2 and 8.1; details are left to the reader. ∎

10. Representation theoretic data associated with a partially stable point. We extend the definition of the representation theoretic data (Definition 6.1) to the partially stable case and illustrate its significance with an application to G/P.

DEFINITION 10.1. *Suppose $v \in P(V)$ is (R,P) stable, $P = KU$. We say that a G-submodule $M \subseteq \mathbb{C}[V]_d$ is admissible, with respect to v and d, if $(M^*)^U$ is* (1) *$(K, Sym^d(W))$-admissible, where W is the smallest K-submodule of V containing $\hat{v}$, and* (2) *it is also $R_{\hat{v}}$-admissible. Let Σ_v be the set of all nonadmissible G-submodules of $\oplus_d \mathbb{C}[V]_d = \mathbb{C}[V]$.*

Let $\Sigma_v(d) \subseteq \mathbb{C}[V]_d$ be the union of nonadmissible G-submodules of $\mathbb{C}[V]_d$.

Basis elements of the G-submodules in Σ_v will be called *nonadmissible basis elements*. The following is a generalization of Proposition 6.2.

PROPOSITION 10.2. *Suppose v is (R,P)-stable. Then the G-modules in the representation-theoretic data Σ_v associated with v are contained in $I_V[v]$.*

Proof. Let $P = KU$. Fix any irreducible G-submodule $S \subseteq R_V[v]_d$. The result will follow if we show that every such S is admissible with respect to v and d (Definition 10.1). It follows from the first statement of Proposition 4.2 that S^* must contain

a $G_{\hat{v}}$-invariant. Since $U \subseteq G_{\hat{v}}$, this implies that $(S^*)^U$ contains an $R_{\hat{v}}$-invariant.

Let W be the smallest K-submodule of V containing $\hat{v}$. It remains to show that $(S^*)^U$ is $(K, \mathrm{Sym}^d(W))$-admissible. Since v, and hence $\hat{v}$, is stabilized by U, and U is normalized by K, W is also a P-module with trivial U-action. Let $\Phi = G \cdot W \subseteq V$. Consider the induced vector bundle $G \times_P W$ [27] with base space G/P and fiber W. Then Φ is the image of the natural G-equivariant map $\phi : G \times_P W \to V$ that maps (g, x), $g \in G$, $x \in W$, to $gx \in V$. We also have the associated map $\tilde{\phi} : G \times_P P(W) \to P(V)$. Since $\tilde{\phi}$ is proper, its image $\tilde{\Phi}$ is closed. The G-variety Φ is just the affine cone of $\tilde{\Phi}$, and is closed. $\Delta_V[v]$ is a closed G-subvariety of $\tilde{\Phi}$, and its affine cone $\check{\Delta}_V[v]$ is a closed G-subvariety of Φ. Hence, $R_V[v]$ is a G-summand of the homogeneous coordinate ring $R[\tilde{\Phi}]$ of $\tilde{\Phi}$. So every irreducible G-submodule of $R_V[v]$ can be thought of as an irreducible G-submodule of $R[\tilde{\Phi}]$. An element of $R[\tilde{\Phi}]_d$ is a regular function on Φ of degree d. Its pull-back via ϕ is a global section of the bundle $B = G \times_P (\mathrm{Sym}^d W)^*$. Hence, an irreducible G-submodule S of $R_V[v]_d$ corresponds to a nonzero irreducible G-submodule of $\Gamma(G/P, B)$. The second statement of Proposition 4.2 applied to B, in conjunction with Schur's lemma, implies that, given any such S, S^* must contain a P-submodule isomorphic to a P-submodule of $\mathrm{Sym}^d(W)$; i.e., $(S^*)^U$ must be $(K, \mathrm{Sym}^d(W))$-admissible. □

10.1. Example: G/P. Proposition 10.2 suggests we study to what extent the data Σ_v determines the ideal $I_V[v]$. In this section we shall show that for G/P the data Σ_v determines $I_v[v]$ completely. This observation was a starting point for Theorem 1.2 and Conjecture 1.9.

Let G be a simply connected, semisimple group G, and let $P \subseteq G$ be its parabolic subgroup, with Levi decomposition $P = KU$. Consider any embedding of G/P in $P(V)$, where $V = V_\lambda(G)$ is an irreducible G-representation and λ is a dominant weight lying in the interior of the face of the dominant Weyl chamber in correspondence [5] with P. Let $v \in P(V)$ correspond to its highest weight vector. Then G/P must actually be the orbit of v in $P(V)$ [5]; i.e., $\Delta_V[v] \simeq G/P$. Recall that v is (L, P)-stable, with defect zero, where L is the semisimple Levi subgroup of P (Example 1 in section 7).

Basis elements of $\Sigma_v(2)$ are equivalent to the Grassman–Plücker syzygies in the case of Grassmanian and, more generally, the quadratic straightening relations of the standard monomial theory [13] in the ideal of G/P.

PROPOSITION 10.3.

1. $\mathbb{C}[V]_d = V_{d\lambda}(G)^* \oplus \Sigma_v(d)$.
2. $R_V[v]_d = V_{d\lambda}(G)^*$.
3. *$I_V[v]$ is generated by the basis elements of $\Sigma_v(2)$, the nonadmissibility data of degree two.*

Remark. The second statement is one part of the Borel–Weil theorem (cf. section 8). Compare its proof here with the one based on Bruhat decomposition [11].

Proof. 1. Since $\mathbb{C}[V]_d^* \simeq \mathrm{Sym}^d(V_\lambda)$ contains a unique highest weight vector with weight $d\lambda$, its G-module decomposition is of the form

$$\mathbb{C}[V]_d^* = V_{d\lambda} + \sum_\mu V_\mu, \tag{6}$$

where each μ is some dominant weight smaller than $d\lambda$, in the usual ordering on the weights. Let $W = \mathbb{C}_\lambda$ be the one-dimensional representation (character) of P corresponding to the weight λ, so that $\mathrm{Sym}^d(W) = \mathbb{C}_{d\lambda}$. We want to show (cf. Definition 10.1) that each $V_\mu = V_\mu(G)$, $\mu \neq d\lambda$, is not admissible at v, i.e., V_μ^U is not

$(K, \mathbb{C}_{d\lambda})$-admissible, or, in other words, that V_μ, as a P-module, cannot contain $\mathbb{C}_{d\lambda}$ as a P-submodule (with trivial U-action): Otherwise let $w \in V_\mu$ be a basis vector of this one-dimensional module. Since w is invariant under the unipotent subgroup of P, it must be the highest weight vector of V_μ, and μ must belong to the interior of the face of the dominant Weyl chamber that corresponds to P [5]. Moreover, as a P-module, the line $\mathbb{C}w$ corresponding to w cannot be isomorphic to $\mathbb{C}_{d\lambda}$ unless $\mu = d\lambda$. Hence $V_\mu^* \subseteq \Sigma_v(d)$ (Definition 10.1). This proves Statement 1.

2. By Proposition 10.2, $\Sigma_v(d) \subseteq I_V[v]$, for all d. Hence, this follows from Statement 1 since $R_V[v]_d$ is clearly nonzero.

3. This is now a consequence of the second fundamental theorem for G/P in the standard monomial theory (cf. Theorem 7.5 in [13]), which states that the ideal $I_V[v]$ is generated by the functions in $\mathbb{C}[V]_2$ that vanish on $\Delta_V[v]$. By Statement 1, these are contained in $\Sigma_v(2) \subseteq I_V[v]$. □

11. SFT for the orbit of a partially stable, excellent point. Now we shall prove Theorem 1.2 for partially stable points with defect zero, by reducing it to the stable case that we have already proved. Let V be a linear representation of G. Let $P \subseteq G$ be a parabolic subgroup with Levi decomposition $P = KU = TLU$. We shall assume that the group R in the definition of (R, P)-stability satisfies the restriction in (5), as it does in our applications (cf. section 2).

A precise statement of Theorem 1.2 in the partially stable case is as follows.

THEOREM 11.1. *Let $V = V_\lambda(G)$. Let $v \in P(V)$ be an (R, P)-stable point with defect zero. Let W be the smallest K-submodule of V containing $\hat{v}$. Assume that* (1) *$L \subseteq R \subseteq K$, and* (2) *$R_{\hat{v}} \subseteq R$ is R-separable and characterizes v, considered as a point in $P(W)$. Then the orbit $Gv \subseteq P(V)$ is determined by the representation-theoretic data Σ_v (Definition* 10.1*) within some G-invariant neighborhood of the orbit. Specifically, there exists a G-invariant neighborhood $Z \subseteq P(V)$ such that Gv is a closed subvariety of Z and the zero set (scheme) in Z of the basis elements of the G-modules in Σ_v coincides with Gv.*

For example, suppose $W = \mathrm{Sym}^n(X)$ is embedded via ϕ in $V = \mathrm{Sym}^m(Y)$, as in section 2. Suppose that (1) f is a stable point in $P(W)$ with respect to the action of $R = SL(X) = SL_{n^2}(\mathbb{C})$, and (2) $R_{\hat{f}}$ characterizes f and is R-separable. Then $\phi(f)$ is a partially stable point of the type above.

Let $\Phi = G \cdot W \subseteq V$ as in the proof of Proposition 10.2. As we observed there, it is the image of the natural G-equivariant map $\phi : G \times_P W \to V$ that maps (g, x), $g \in G$, $x \in W$, to $gx \in V$, and we also have the associated map $\tilde{\phi} : G \times_P P(W) \to P(V)$. Since $\tilde{\phi}$ is proper, its image $\tilde{\Phi}$ is closed. The G-variety Φ is just the affine cone of $\tilde{\Phi}$. Let $R[\tilde{\Phi}]$ be the homogeneous coordinate ring of $\tilde{\Phi}$.

Our goal is to show that the orbit Gv of v is determined scheme-theoretically by the representation theoretic data within some G-invariant neighborhood of the orbit. Since Gv is contained in $\tilde{\Phi}$, our first goal is to understand the geometry of $\tilde{\Phi}$. Once this is done, we shall be able to reduce the present case to the stable case that has already been analyzed.

When v corresponds to the highest weight vector of $V = V_\lambda(G)$, $\tilde{\Phi} = \Delta_V[v] = G/P$. Hence we wish to generalize the results in section 10.1.

The geometry of $\tilde{\Phi}$. We say that $V_\alpha(G)$ is (K, U, W, d)-admissible if $(V_\alpha(G)^*)^U$ contains an irreducible K-submodule that also occurs in $\mathrm{Sym}^d(W)$, and that it is non–(K, U, W, d)-admissible otherwise.

PROPOSITION 11.1. *Every G-submodule in $R[\tilde{\Phi}]_d$ is (K,U,W,d)-admissible. Hence, every non–(K,U,W,d)-admissible G-submodule of $\mathbb{C}[V]_d$ belongs to the homogeneous ideal of $\tilde{\Phi}$.*

The proof is an easy modification of the proof of Proposition 10.2.

The following is a generalization of Proposition 10.3. Recall that W is a P-module with trivial U-action (cf. proof of Proposition 10.2).

PROPOSITION 11.2. (1) *As a G-module, $R[\tilde{\Phi}]_d$ is isomorphic to the space $\Gamma_d = \Gamma(G/P, Sym^d(W^*))$ of global sections of the vector bundle $G \times_P Sym^d(W^*)$.*

(2) *$\mathbb{C}[V]_d = \Gamma_d \oplus (\oplus_\beta V_\beta(G))$, where $V_\beta(G)^*$, for any β, cannot contain an irreducible K-submodule that also occurs in $Sym^d(W)$. In particular, each $V_\beta(G)$ is non–(K,U,W,d)-admissible and hence belongs to the ideal of $\tilde{\Phi}$.*

(3) *The ideal of $\tilde{\Phi}$ is generated (actually spanned) by non–(K,U,W,d)-admissible G-submodules of $\mathbb{C}[V]_d$.*

For the proof of this proposition we shall need a lemma. Let $P = KU = TLU$ be the Levi decomposition as above. We think of the root system of K as a subsystem of that of G. Let l be any linear functional l on the weight space of G with respect to which the usual ordering of the roots of G is defined; here it is assumed that l is irrational with respect to the weight lattice. Let

$$V_\lambda(G) = V_\lambda(K) \oplus \bigoplus_\mu V_\mu(K) \tag{7}$$

be a decomposition of $V_\lambda(G)$ as a K-module. Let v_β be the highest weight vector of $V_\beta(K)$ occurring in this decomposition with respect to l. Let $w_T(\beta) = w_T(v_\beta)$ denote its T-weight, i.e., the weight with respect to the central torus $T \subseteq K$.

The following is a complement to Lemma 8.3. Let ϕ be the projection of the dominant weights of G onto the largest face F of the dominant Weyl chamber that is orthogonal (in the Killing norm) to the simple roots of $\mathcal{K}$, the Lie algebra of K. Note that (1) $w_T(\alpha) = w_T(\phi(\alpha))$, for any dominant weight, since $w_T(\gamma) = 0$ for any simple root γ of K, and (2) $w_T(\phi(\alpha)) \neq w_T(\phi(\beta))$ if $\phi(\alpha) \neq \phi(\beta)$. Order the projected weights in F according to the restriction of l to F. This induces an order on the T-weights $w_T(\alpha)$'s.

LEMMA 11.2. *For every μ in (7), $w_T(\mu) = w_T(v_\mu)$ is less than $w_T(\lambda) = w_T(v_\lambda)$ for an appropriate l.*

Proof. Let W denote the Weyl group of G. For a simple root g, let W_g be the reflection in the hyperplane perpendicular to g.

The weights of $V_\lambda(G)$ are contained in the convex hull C of the conjugates of λ under the Weyl group elements [5]. Let A be the affine space, perpendicular to F, spanned by λ and $W_g(\lambda)$'s, where g ranges over the simple roots of $\mathcal{K}$. Its intersection with C is a face of C—call it L; it is the smallest face of C containing λ and $W_g(\lambda)$, for each simple root g of $\mathcal{K}$.

CLAIM 11.3. *The weight vectors of $V_\lambda(G)$, whose weights are contained in L, span the irreducible K-submodule $V_\lambda(K) \subseteq V_\lambda(G)$ with weight λ.*

Proof of the claim. Let $\mathcal{G}, \mathcal{K}$ denote the Lie algebras of G and K, and $U(\mathcal{G}), U(\mathcal{K})$ the corresponding universal enveloping algebras. We know that $V_\lambda(G)$ is spanned by αv_λ, where v_λ is the highest weight vector of $V_\lambda(G)$ and $\alpha \in U(\mathcal{G})$ ranges over all monomials in the negative roots of $\mathcal{G}$. If we order the roots appropriately, the Poincare–Birkoff–Witt theorem implies that α is of the form $\alpha_1\alpha_2$, where $\alpha_2 \in U(\mathcal{K})$ is a monomial in the negative roots of $\mathcal{K}$ and α_1 is a monomial in the remaining negative roots of $\mathcal{G}$. Then αv_λ is nonzero with weight in L iff $\alpha_1 = 1$. But $\alpha_2 v_\lambda$, as

α_2 ranges over all monomials in the negative roots of $\mathcal{K}$, clearly span $V_\lambda(K) \subseteq V_\lambda(G)$. This proves the claim.

It follows from the claim that no $\mu \neq \lambda$ in (7) can belong to L. We shall choose an irrational l such that the weights of $V_\lambda(G)$ within L have higher l-coordinates than the remaining weights of $V_\lambda(G)$; it clearly exists.

Consider the restriction of the linear function l to F. Then $l(\phi(\alpha))$ is higher than $l(\phi(\beta))$ for any weight β of $V_\lambda(G)$ not contained in L. Since no $\mu \neq \lambda$ in (7) can belong to L, the result follows. □

Proof of Proposition 11.2. Since w is P-stable, its stabilizer contains U. Since U is normalized by K, it follows that every point in W is also stabilized by U. By Lemma 8.3, $W = W_\lambda = V_\lambda(G)^U = V_\lambda(K)$.

The decomposition of $V = V_\lambda$ as a K-module is of the form

$$V = V_\lambda = W_\lambda \oplus \bigoplus_\mu W_\mu,$$

where, for each μ, $w_T(v_\mu) < w_T(v_\lambda)$ (Lemma 11.2). Let $W' = \bigoplus_\mu W_\mu$. By induction, and using the formula

$$\mathbb{C}[V]_d^* = \mathrm{Sym}^d(V) = \mathrm{Sym}^d(W_\lambda \oplus W') = \sum_{i+j=d} \mathrm{Sym}^i(W_\lambda) \otimes \mathrm{Sym}^j(W'),$$

it follows that $\mathbb{C}[V]_d^*$ has a K-module decomposition of the form

$$\mathbb{C}[V]_d^* = \mathrm{Sym}^d(V) = \mathrm{Sym}^d(W) \oplus W_d, \tag{8}$$

where the T-weight of the highest-weight-vector of each K-submodule of W_d is strictly smaller than the T-weight of the highest-weight-vector of each K-submodule in $\mathrm{Sym}^d(W)$. Hence no irreducible K-module can occur in both $\mathrm{Sym}^d(W)$ and W_d, considered as abstract K-modules; i.e., $\mathrm{Hom}(\mathrm{Sym}^d(W), W_d)^K = 0$.

Now consider a G-module decomposition

$$\mathbb{C}[V]_d^* \simeq \mathrm{Sym}^d(V_\lambda(G)) = \sum_\mu c_\mu^\lambda V_\mu(G), \tag{9}$$

where all $c_\mu^\lambda \geq 0$ and μ ranges over all dominant weights of G less than or equal to $d\lambda$. We do not know this decomposition explicitly; finding an explicit decomposition is a special case of the unsolved plethysm problem [5]. It follows from (9) that

$$\mathbb{C}[V]_d^{*U} \simeq \mathrm{Sym}^d(V_\lambda(G))^U = \sum_\mu c_\mu^\lambda V_\mu(G)^U = \sum_\mu c_\mu^\lambda V_\mu(K), \tag{10}$$

where the last step follows from Lemma 8.3. Since $W = V^U$, $\mathrm{Sym}^d(W)$ is a U-submodule of $\mathrm{Sym}^d(V)$. Hence it follows from (10) that each weight β of K that occurs in $\mathrm{Sym}^d(W)$ with nonzero multiplicity d^β also occurs as a weight of G in $\mathrm{Sym}^d(V)$ with multiplicity at least d^β. On the other hand, by the Borel–Weil theorem and Lemma 8.3 (cf. also Frobenius reciprocity [4]),

$$(\Gamma_d^*)^U = \mathrm{Sym}^d(W). \tag{11}$$

It follows that as a G-module,

$$\mathbb{C}[V]_d^* = \Gamma_d^* \oplus \bigoplus_\mu c_\mu V_\mu(G), \tag{12}$$

for suitable μ's. On the other hand, comparing this equation with (8), it follows that no $V_\mu(G)$ here can contain an irreducible K-submodule that also occurs in $\mathrm{Sym}^d(W)$. This proves the second statement of the proposition.

It remains to show that Γ_d is a G-submodule of $R[\Phi]_d$. By (11), $\Gamma_d^U = \mathrm{Sym}^d(W^*)$. Hence by the second statement, in conjunction with Lemma 8.3, this is equivalent to showing that $\mathrm{Sym}^d(W^*)$ is a K-submodule of $R[\Phi]_d^U$. This is clear, since we have the canonical U-equivariant embedding of W within Φ, the U-action on W being trivial. □

When $\tilde{\Phi} = G/P$, by the standard monomial theory, we know that nonadmissible basis elements of degree two generate the ideal of G/P (section 10.1). Analogously, in the context of Proposition 11.2, one can ask for a degree bound c such that the basis elements of non–(K, U, W, d)-admissible G-submodules of $\mathbb{C}[V]_d$, $d \le c$, generate the ideal of $\tilde{\Phi}$. This seeks an extension of the standard monomial theory to $\tilde{\Phi} = GW$.

Reduction to the stable case. Now we are ready to prove Theorem 11.1. Let $V = V_\lambda(G)$. Let v be an (R, P)-stable point with defect zero, as hypothesized, and let $W = V_\lambda(K)$ be the smallest K-submodule of V containing $\hat{v}$. Since $L \subseteq R \subseteq K$, K and R are both products of the form $LT(K)$ and $LT(R)$, respectively, where $T(K)$ and $T(R)$ are tori. Hence an irreducible K-module is also an irreducible R-module. In particular, W is an irreducible R-module with the action of the torus $T(R)$ being determined by a character; i.e., the action of $T(R)$ on $P(W)$ is trivial. Hence, any R-invariant subset of $P(W)$ is also K-invariant, and, in particular, $Rv = Kv \subseteq P(W)$.

The orbit $Gv \subseteq P(V)$ is contained in $\tilde{\Phi}$. By Proposition 11.2, the ideal of $\tilde{\Phi}$ is generated (actually spanned) by the non–(K, U, W, d)-admissible G-submodules of $\mathbb{C}[V]_d$. These submodules are contained in the nonadmissibility data Σ_v associated with v (cf. Definition 10.1). Let $\hat{\Sigma}_v$ be the set of remaining G-submodules of $\mathbb{C}[V]$ in Σ_v. A G-submodule $M \subseteq \mathbb{C}[V]_d$ belongs to $\hat{\Sigma}_v$ iff $(M^*)^U$ is not $R_{\tilde{v}}$-admissible. We shall show that there exists a G-invariant neighborhood Z of Gv in $\tilde{\Phi}$ such that Gv is a closed subvariety of Z and Gv is determined within Z by the data $\hat{\Sigma}_v$; i.e., the zero set of the (basis elements of) the G-modules in $\hat{\Sigma}_v$, restricted to Z, coincides with Gv scheme-theoretically.

Consider the G-equivariant map $\tilde{\phi} : G \times_P P(W) \to \tilde{\Phi}$.

Claim 11.4. *$\tilde{\phi}^{-1}(v)$ is a point.*

Proof of the claim. Suppose to the contrary. Then there exists $g \notin P$ and a $w \in P(W)$ such that $\phi(g, w) = v$, i.e., $gw = v$, and hence, $w = g^{-1}(v)$. Since v is (R, P)-stable, $U \subseteq G_v \subseteq P$ (Definition 7.1). Since $w \in P(W)$, and the U-action on W is trivial,

$$U \subseteq G_w = (G_v)^{g^{-1}} \subseteq P^{g^{-1}}.$$

Thus both P and $P^{g^{-1}}$ contain U. This implies that $P = P^{g^{-1}}$ (by Lemma 5.2.5(ii) in [30] and Corollary 11.17(iii) in [2]). Thus g^{-1} normalizes P. Since the normalizer of P is P itself (Theorem 11.16 in [2]), it follows that $g \in P$, a contradiction.

Let us denote the point $\phi^{-1}(v)$ by $\tilde{v}$. Since $\tilde{\phi}$ is surjective, to show that Gv is scheme-theoretically determined within a G-invariant neighborhood by the data $\hat{\Sigma}_v$, it suffices to show that $\tilde{\phi}^{-1}(Gv) = G \cdot \tilde{\phi}^{-1}(v) = G\tilde{v} \subseteq G \times_P P(W)$ is determined scheme-theoretically within some G-invariant neighborhood by the set $\tilde{\phi}^{-1}(\hat{\Sigma}_v)$ of the pull-backs of the G-modules in $\hat{\Sigma}_v$. But since $G_{\tilde{v}} = G_v \subseteq P$, the normal space to $G\tilde{v}$ can be identified with the normal space to its restriction to the slice $\tilde{\phi}^{-1}(P(W)) \simeq P(W)$, which in turn, corresponds to the normal space to the orbit $Rv = Kv \subseteq P(W)$. By the

Jacobian criterion (Proposition 6.4), it now suffices to show that the set $\tilde{\phi}^{-1}(\hat{\Sigma}_v)_{P(W)}$ of the restrictions of the modules in $\tilde{\phi}^{-1}(\hat{\Sigma}_v)$ to the fixed slice $\tilde{\phi}^{-1}(P(W)) \simeq P(W)$ of the bundle $G \times_P (W)$ determines the orbit of $Rv = Kv \subseteq P(W)$ within some K-invariant neighborhood of this orbit.

By Proposition 11.2, $R[\Phi]_d$ is isomorphic to the space $\Gamma_d = \Gamma(G/P, \mathrm{Sym}^d(W^*))$ of global sections of the bundle $G \times_P \mathrm{Sym}^d(W^*)$. By the Borel–Weil theorem and Lemma 8.3 (see also the Frobenius reciprocity in [4]), the set of restrictions of the modules in Γ_d to the slice $P(W)$ can be identified with Γ_d^U: If $M \in R[\Phi]$ and is isomorphic to $V_\lambda(G)$, then the restriction of $\tilde{\phi}^{-1}(M)$ to $P(W)$ corresponds to M^U, which is isomorphic to $V_\lambda(K)$ (Lemma 8.3). Hence, the restrictions of the modules in $\tilde{\phi}^{-1}(\hat{\Sigma}_v)$ to the slice $P(W)$ consists of precisely the K-modules in $\mathbb{C}[W]$ that do not contain any $R_{\hat{v}}$-invariant. Since K and R are of the form $LT(K)$, $LT(R)$, an irreducible K-module is also an irreducible R-module, and the subspace of $\mathbb{C}[W]$ spanned by non–$R_{\hat{v}}$-admisible K-submodules coincides with the subspace spanned by non–$R_{\hat{v}}$-admisible R-submodules. Thus, $\tilde{\phi}^{-1}(\hat{\Sigma}_v)_{P(W)}$ consists of precisely the non–$R_{\hat{v}}$-admisible R-modules in $\mathbb{C}[W]$. Since $v \in P(W)$ is stable with respect to the action of R on $P(W)$, we can now apply Theorem 6.1 for the stable case. It implies that $Rv \subseteq P(W)$ has an R-invariant, and hence, K-invariant, neighborhood Y such that Rv as a subvariety of Y is determined scheme-theoretically by $\tilde{\phi}^{-1}(\hat{\Sigma}_v)_{P(W)}$.

This shows that $\tilde{\phi}^{-1}(\hat{\Sigma}_v)_{P(W)}$ determines the orbit of $Rv = Kv \subseteq P(W)$ within a K-invariant neighborhood of the orbit.

This proves Theorem 11.1. □

12. *G*-separability. We now study the notion of G-separability (Definition 6.3), which is of interest in the context of Theorem 1.2.

PROPOSITION 12.1.

1. *A semisimple group H, embedded in $G = H \times H$ diagonally, is strongly G-separable.*
2. *$H = SL_k(\mathbb{C})$ is a strongly G-separable subgroup of $G = SL_n(\mathbb{C})$ if $k > (n+1)/2$.*
3. *$H = SL_k(\mathbb{C}) \times SL_l(\mathbb{C}) \subseteq G = SL_{k+l}(\mathbb{C})$, with natural embedding, is strongly G-separable.*

Remark. The last statement can be generalized to semisimple Levi subgroups of maximal parabolic subgroups of classical simple groups if one uses, instead of the decomposition formula in (13), Littelmann's restriction rule [14].

Proof. (1) By Schur's lemma, a G-module $V_\alpha(H) \otimes V_\beta(H)$, where $\otimes$ denotes the external tensor product here, is H-admissible iff $V_\beta(H) \simeq V_\alpha(H)^*$; i.e., $\beta = i_H(\alpha)$, where i_H is the involution on dominant H-weights (section 7). Any nontrivial representation $V_\lambda(H)$ occurs in the non–H-admissible G-module $V_\lambda(H) \otimes 1_H$, where 1_H denotes the trivial H-module. So H is clearly G-separable.

Strong G-separation follows from the following more general fact.

CLAIM 12.2. *$V_\lambda(H)$ occurs in the non–H-admissible G-module $V_\delta(H) \otimes V_\rho(H)$, $\delta = \lambda + \beta$ $\rho = i_H(\beta)$, for any dominant H-weight β.*

Proof of the claim. By Schur's lemma, this is equivalent to showing that

$$\begin{aligned} \mathrm{Hom}(V_\delta(H) \otimes V_\rho(H), V_\lambda(H)) &= V_{\lambda+\beta}(H)^* \otimes V_\rho(H)^* \otimes V_\lambda(H) \\ &= V_{\lambda+\beta}(H)^* \otimes V_\beta(H) \otimes V_\lambda(H) \end{aligned}$$

contains an H-invariant. By Schur's lemma again, this is equivalent to showing that $V_{\lambda+\beta}(H)$ occurs in $V_\beta(H) \otimes V_\lambda(H)$, which is clear.

(2) Consider a nontrivial $V_\lambda(SL_k(\mathbb{C}))$, where λ is a Young diagram of height h less than k. We shall exhibit a non–H-admissible $V_\mu(SL_n(\mathbb{C}))$ containing it. If h is greater than $n-k$, we let $\mu = \lambda$. Otherwise, let μ be a Young diagram obtained by adding $n-k-h+1$ boxes to the first column of λ. Its height is $n-k+1 < k$. By Pieri's branching rule, it is easy to see that $V_\mu(SL_n(\mathbb{C}))$ contains $V_\lambda(SL_k(C))$ but not the trivial representation of $SL_k(\mathbb{C})$. More generally, if μ' is a Young diagram obtained by appropriately extending, i.e., adding boxes to the first $n-k$ rows of μ, then $V_{\mu'}(SL_n(\mathbb{C}))$ contains $V_\lambda(SL_k(C))$ but not the trivial representation of $SL_k(\mathbb{C})$. There are infinitely many such μ's. So $SL_k(\mathbb{C})$ is strongly separable.

(3) Assume that $k \geq l$, the other case being similar. Consider a nontrivial H-module $L = V_\alpha(SL_k(\mathbb{C})) \otimes V_\beta(SL_l(\mathbb{C}))$, where α and β correspond to Young diagrams of height less than k and l, respectively. We shall exhibit a non–H-admissible G-module $V_\lambda(G)$ containing it. We identify α and β with the partitions: $\alpha = (\alpha_1, \alpha_2, \ldots)$, where α_i denotes the length of the ith row of the corresponding Young diagram, and $\beta = (\beta_1, \beta_2, \ldots)$. We proceed by cases.

Case 1. Either α does not correspond to a rectangular Young diagram of height l, or β is not trivial.

Let $\lambda = \alpha + \beta = (\alpha_1 + \beta_1, \ldots)$. Note that the height of λ is less than k. We have [5]

$$V_\lambda(GL_{l+k}(\mathbb{C})) = \sum_{\rho,\delta} N^\lambda_{\rho,\delta} V_\rho(GL_k(\mathbb{C})) \otimes V_\delta(GL_l(\mathbb{C})), \tag{13}$$

where $N^\lambda_{\rho,\delta}$ denotes the Littlewood–Richardson coefficient. From this it easily follows that $V_\lambda(SL_{k+l}(\mathbb{C}))$ contains the representation $V_\alpha(SL_k(\mathbb{C})) \otimes V_\beta(SL_l(\mathbb{C}))$ of $SL_k(\mathbb{C}) \times SL_l(\mathbb{C})$. But it cannot contain the trivial H-representation: If $\rho \neq 0$ and δ (possibly zero) correspond to rectangular Young diagrams with height k and l, respectively—so that $V_\rho(SL_k(\mathbb{C}))$ and $V_\delta(SL_l(\mathbb{C}))$ are trivial—then $N^\lambda_{\rho,\delta}$ is easily seen to be zero; otherwise the height of λ will be at least k. On the other hand, if $\rho = 0$, then $\lambda = \delta$. Since the height of β is less than l, the definition of λ then implies that $\alpha = \delta$ and $\beta = 0$; a contradiction.

More generally, let α' be any Young diagram obtained from α by adding columns of length k. Let $\lambda' = \alpha' + \beta$. Then $V_{\lambda'}(SL_{k+l}(\mathbb{C}))$ also contains $V_\alpha(SL_k(\mathbb{C})) \otimes V_\beta(SL_l(\mathbb{C}))$ but not the trivial representation of $SL_k(\mathbb{C}) \times SL_l(\mathbb{C})$. Moreover, there are infinitely many such λ's.

Case 2. α is rectangular of height l and width w, and $\beta = 0$.

We can assume that $k > l$; otherwise $V_\alpha(SL_k(\mathbb{C}))$ too will be trivial. For any integer $r \geq 0$, let λ be the Young diagram whose first r columns are of height k, the $(r+1)$st column is of length $l+1$, the columns numbered $r+2, \ldots, r+w$ are of height l, and the column numbered $r+w+1$ is of height $l-1$. Then it follows from (13) that $V_\lambda(GL_{l+k}(\mathbb{C}))$ contains $V_\rho(GL_k(\mathbb{C})) \otimes V_\delta(GL_l(\mathbb{C}))$, where ρ is obtained from α by adding to its left r columns of length k, and δ consists of a single column of height l. Clearly $V_\rho(GL_k(\mathbb{C})) \otimes V_\delta(GL_l(\mathbb{C}))$ is isomorphic to $V_\alpha(SL_k(\mathbb{C})) \otimes V_\beta(SL_l(\mathbb{C}))$ as an $SL_k(\mathbb{C}) \times SL_l(\mathbb{C})$-module. But it does not contain the trivial $SL_k(\mathbb{C}) \times SL_l(\mathbb{C})$-module; this too follows from (13). Moreover, there are infinitely many such λ.

This proves strong G-separability of H. □

For us, it is important to know if the stabilizers of the points that arise in the context of complexity theory are separable (cf. section 2). The connected component of the stabilizer of $\det(Y)$ in $SL_{n^2}(\mathbb{C})$, where Y is an $n \times n$ matrix, contains $SL_n(\mathbb{C}) \times$

$SL_n(\mathbb{C}) \subseteq SL(Y) = SL_{n^2}(\mathbb{C})$ (section 2.1). Regarding this subgroup we make the following conjecture.

CONJECTURE 12.3. *$SL_n(\mathbb{C}) \times SL_n(C)$ is a strongly separable subgroup of $SL_{n^2}(\mathbb{C})$.*

Here the embedding corresponds to the natural embedding $SL(V) \otimes SL(V) \subseteq SL(V \otimes V)$, $V = C^n$. Specifically, letting $V_\lambda(n)$ denote $V_\lambda(SL_n(\mathbb{C}))$ in what follows, the conjecture can be reformulated as follows.

CONJECTURE 12.4. *For every nontrivial Weyl module $V_\lambda(n) \otimes V_\mu(n)$ of $SL_n(\mathbb{C}) \times SL_n(\mathbb{C})$, such that $|\lambda| = |\mu| \pmod n$, there exist (infinitely many) Weyl modules $V_\rho(n^2)$ of $SL_{n^2}(\mathbb{C})$ whose decomposition as an $SL_n(\mathbb{C}) \times SL_n(\mathbb{C})$-module contains $V_\lambda(n) \otimes V_\mu(n)$ but not the trivial $SL_n(\mathbb{C}) \times SL_n(\mathbb{C})$-module.*

The restriction $|\lambda| = |\mu| \pmod n$ is required to ensure (cf. Definition 6.3) that $V_\lambda(n) \otimes V_\mu(n)$ occurs in some representation of $SL_{n^2}(\mathbb{C})$; cf. (14) below.

The conjecture can be reformulated in terms of the symmetric group as follows. Let $\hat{V}_\gamma(n^2)$ be a Weyl module of $GL_{n^2}(\mathbb{C})$. Embed $GL_n(\mathbb{C}) \times GL_n(\mathbb{C}) = GL(\mathbb{C}^n) \times GL(\mathbb{C}^n)$ in $GL(\mathbb{C}^n \otimes \mathbb{C}^n) = GL_{n^2}(\mathbb{C})$. The decomposition of $\hat{V}_\gamma(n^2)$ as a $GL_n(\mathbb{C}) \times GL_n(\mathbb{C})$-module is of the form

$$\hat{V}_\gamma(n^2) = \sum_{\alpha,\beta} c_{\alpha,\beta,\gamma} \hat{V}_\alpha(n) \otimes \hat{V}_\beta(n); \tag{14}$$

here $c_{\alpha,\beta,\gamma}$ can be nonzero only if $|\alpha| = |\beta| = |\gamma|$. To get the decomposition of $\hat{V}_\gamma$ as an $SL_n(\mathbb{C}) \times SL_n(\mathbb{C})$-module, we reduce the Young diagrams occurring on the right-hand side by removing columns of length n. This does not change their sizes modulo n; this explains the restriction $|\lambda| = |\mu| \pmod n$ in the conjecture. By Littlewood's symmetry conditions [5], the coefficients $c_{\alpha,\beta,\gamma}$ do not depend on the ordering of α, β, and γ.

Given a Young diagram δ, $|\delta| = m$, let W_δ denote the corresponding irreducible representation, the Specht module, of the symmetric group S_m. Then the coefficient $c_{\alpha,\beta,\gamma}$ occurring in the preceding decomposition is the same as the one occurring in the decomposition of the tensor product $W_\alpha \otimes W_\beta$ as an S_m-module,

$$W_\alpha \otimes W_\beta = \sum_\gamma c_{\alpha,\beta,\gamma} W_\gamma,$$

where $m = |\alpha| = |\beta|$; cf. [5].

For any λ of height less than n and $m = |\lambda| \pmod n$, let $\lambda(m)$ be the unique Young diagram of size m obtained by adding to α columns of length n. Then the preceding conjecture is equivalent to saying the following:

For every nontrivial pair of Young diagrams (λ, μ) of height less than n, and such that $|\lambda| = |\mu| \pmod n$, there exist an $m = |\lambda| = |\mu| \pmod n$, $m \geq n$, and a ρ of size m such that W_ρ occurs in the decomposition of $W_{\lambda(m)} \otimes W_{\mu(m)}$ as an S_m-module, but not in the decomposition of $W_\delta \otimes W_\delta$, where δ is the rectangular Young diagram of height n and size m.

If $|\lambda| = |\mu| \neq 0 \pmod n$, the last restriction is vacuous, because no such δ exists, and hence we have the following proposition.

PROPOSITION 12.5. *If $|\lambda| = |\mu| \neq 0 \pmod n$, Conjecture 12.4 holds.*

So, let us assume that $|\lambda| = |\mu| = 0 \pmod n$ in what follows.

PROPOSITION 12.6. *Conjecture 12.4 holds for $n = 2$.*

The main difficulty in extending the proof below to $n > 2$ is that an explicit decomposition of the tensor product of two arbitrary Specht modules is not yet known.

Proof. We need to show that for every nontrivial pair of (λ, μ) of row-shaped Young diagrams, with $|\lambda|$ and $|\mu|$ even, there exist an even m and a ρ of size m such that W_ρ occurs in the decomposition of $W_{\lambda(m)} \otimes W_{\mu(m)}$ as an S_m-module, but not in the decomposition of $W_\delta \otimes W_\delta$, where δ is the rectangular Young diagram of height 2 and width $m/2$. We shall show that there exists such a ρ for every large enough $m \geq 4(|\lambda| + |\mu|)$. Fix such an m.

Given a Young diagram γ, we shall let γ_i denote the number of boxes in its ith row from the top. We assume that the topmost row has the highest length in the diagram. We shall denote $\lambda(m)$ and $\mu(m)$ by $\bar{\lambda}$ and $\bar{\mu}$, respectively. Since λ and μ are row shaped, we shall let λ and μ denote the lengths of their row as well. Since $|\bar{\lambda}| = |\bar{\mu}| = m$, $\bar{\lambda}_2 - \bar{\lambda}_1 = \lambda$, and $\bar{\mu}_2 - \bar{\mu}_1 = \mu$, we have $\bar{\lambda}_2 = m/2 - \lambda/2$ and $\bar{\mu}_2 = m/2 - \mu/2$. Since $\bar{\lambda}$, $\bar{\mu}$, and δ have two rows, we can use the decomposition formula of Remmel and Whitehead [29].

First, we shall try to find a required ρ with two rows. Let (a, b), $a \geq b$, denote the two-row Young diagram with the top row of length a and the bottom row of length b. Suppose we are given Young diagrams $(k, h), (r, l), (d, c)$ of size m. Because of Littlewood's symmetry conditions we can assume that $l \leq h \leq c$. With this condition, the formula in [29, Theorem 3.3] says that

$$c_{(r,l),(k,h),(d,c)} = (1 + w - v)\chi(w \geq v), \tag{15}$$

where $w = \lfloor (l + h - c)/2 \rfloor$, $v = \max(0, \lceil (l + h + c - m)/2 \rceil)$, and the function χ is one if $w \geq v$ and zero otherwise.

By Littlewood's symmetry condition, $c_{\delta,\delta,\rho} = c_{\rho,\delta,\delta}$. Applying the preceding formula with $(r, l) = \rho$ and $(k, h) = (d, c) = \delta = (m/2, m/2)$, we conclude that this coefficient is nonzero iff $\lfloor \rho_2/2 \rfloor \geq \lceil \rho_2/2 \rceil$. That is, iff ρ_2 is even. So we need to find a ρ, with ρ_2 odd, such that $c_{\bar{\lambda},\bar{\mu},\rho}$ is nonzero. Because of symmetry, we can assume that $\bar{\lambda}_2 \leq \bar{\mu}_2$. We will try to find ρ such that

$$\rho_2 \leq \bar{\lambda}_2. \tag{16}$$

Then setting $(k, h) = \rho$, $(r, l) = \bar{\lambda}$, and $(d, c) = \bar{\mu}$ in (15), we conclude that $c_{\bar{\lambda},\bar{\mu},\rho} = c_{\rho,\bar{\lambda},\bar{\mu}}$ is nonzero iff

$$\lfloor (\rho_2 + \bar{\lambda}_2 - \bar{\mu}_2)/2 \rfloor \geq \max(0, \lceil (\rho_2 + \bar{\lambda}_2 + \bar{\mu}_2 - m)/2 \rceil), \tag{17}$$

i.e., iff

$$\lfloor (\rho_2 - \lambda/2 + \mu/2)/2 \rfloor \geq \max(0, \lceil (\rho_2 - \lambda/2 - \mu/2)/2 \rceil). \tag{18}$$

We now proceed by cases.

Case 1. $\mu \neq 0$.

In this case the condition in (18) can be satisfied if

$$\rho_2 \geq (\lambda + \mu)/2 \tag{19}$$

and $\mu \geq 2$, which holds since μ is nonzero and even. But there are many odd ρ_2's such that (16) and (19) are satisfied if, say, $m \geq 4(\lambda + \mu)$.

Case 2. $\mu = 0$, and $\lambda/2$ is odd.

In this case, (18) is satisfied if we let $\rho_2 = \lambda/2$, which is nonzero—otherwise (λ, μ) will be trivial—and odd, as required. Since m is large enough, (16) is also satisfied.

It remains to consider the following case.

Case 3. $\mu = 0$, and $\lambda/2$ is even.

In this case, the required two-row ρ does not exist. So we shall find an appropriate $\rho = (\rho_1, \rho_2, \rho_3, \rho_4)$ with four rows such that $\rho_3 = \rho_4$.

Given Young diagrams (k, h), (m, l), (d, c, a, a) (entries in nonincreasing order) with m boxes such that $a > 0$ and $\lceil (h+1)/2 \rceil \leq h - c$, the Remmel–Whitehead formula [29, Theorem 3.1] says that

$$c_{(k,h),(m,l),(d,c,a,a)} = \sum_{r=h-c}^{\min(l, \lfloor \frac{l-a+h-c}{2} \rfloor)} 1 - \sum_{r=\max(a, l+h+a-m-1)}^{\min(l, \lfloor \frac{h-1}{2} \rfloor, \lfloor \frac{l+h+a+c-m-1}{2} \rfloor)} 1. \tag{20}$$

We will set $(k, h) = (m, l) = \delta = (m/2, m/2)$ and $(d, c, a, a) = \rho = (\rho_1, \rho_2, \rho_3, \rho_4)$ in this formula. For the formula to be applicable, we need to ensure that

$$\lceil \lceil (h+1)/2 \rceil = (m/2+1)/2 \rceil \leq h - c = m/2 - \rho_2. \tag{21}$$

If, in addition,

$$\rho_2 + \rho_3 < m/2, \tag{22}$$

we get that

$$\begin{aligned} c_{\delta,\delta,\rho} &= \sum_{\frac{m}{2}-\rho_2}^{\lfloor \frac{(m-\rho_3-\rho_2)}{2} \rfloor} 1 - \sum_{r=\rho_3}^{\lfloor \frac{\rho_3+\rho_2-1}{2} \rfloor} 1 \\ &= \left\lfloor \frac{(m-\rho_3-\rho_2)}{2} \right\rfloor - \left(\frac{m}{2} - \rho_2\right) + \rho_3 - \left\lfloor \frac{\rho_3+\rho_2-1}{2} \right\rfloor, \\ &= \left\lfloor \frac{\rho_2-\rho_3}{2} \right\rfloor - \left\lfloor \frac{\rho_2-\rho_3-1}{2} \right\rfloor, \end{aligned}$$

which is 1 if $\rho_2 - \rho_3$ is even, and zero otherwise.

So we need to find a ρ with $\rho_2 - \rho_3$ odd, satisfying (21) and (22), such that $c_{\bar{\mu},\bar{\lambda},\rho}$ is nonzero. Since $\mu = 0$, we have $\bar{\mu}_1 = \bar{\mu}_2 = m/2$. Also, recall that $\bar{\lambda}_2 = m/2 - \lambda/2$. Set $(k, h) = \bar{\mu} = (m/2, m/2)$, $(m, l) = \bar{\lambda}$, and $(d, c, a, a) = \rho$ in (20). We shall choose a four-row ρ, with nonzero ρ_3, such that $\rho_2 - \rho_3$ is odd, ρ_2 and ρ_3 are sufficiently larger than λ, and also such that the difference between $m/2$ and $\rho_2 + \rho_3$ is sufficiently larger than λ. This is possible if m is large enough compared to λ. In this case, the upper index of the first sum in (20) becomes $\lfloor \frac{\bar{\lambda}_2 - \rho_3 + m/2 - \rho_2}{2} \rfloor$, and the lower index is $m/2 - \rho_2$. So the contribution of the first term is $\lfloor \frac{\rho_2 - \rho_3 - \lambda/2}{2} \rfloor$. Since λ is nonzero, the lower index of the second sum in (20) is equal to $\frac{\lambda}{2} - 1 + \rho_3$. The upper index, assuming that m is large enough and $m/2 - \rho_2 - \rho_3$ is sufficiently larger than λ, becomes $\lfloor \frac{-\lambda/2 + \rho_2 + \rho_3 - 1}{2} \rfloor$. Assuming that ρ_2 and ρ_3 are sufficiently larger than λ, it is larger than the lower index. Hence the second term becomes

$$\frac{\lambda}{2} - 1 + \rho_3 - \left\lfloor \frac{-\lambda/2 + \rho_2 + \rho_3 - 1}{2} \right\rfloor = \frac{\lambda}{2} - 1 + \left\lfloor \frac{\lambda/2 - \rho_2 + \rho_3 + 1}{2} \right\rfloor.$$

Thus

$$c_{\bar{\mu},\bar{\lambda},\rho} = \left\lfloor \frac{\rho_2 - \rho_3 - \lambda/2}{2} \right\rfloor + \frac{\lambda}{2} - 1 + \left\lfloor \frac{\lambda/2 - \rho_2 + \rho_3 + 1}{2} \right\rfloor = \frac{\lambda}{2} - 1.$$

This is nonzero, since $\lambda/2$, being nonzero and even, is at least two. So we can choose a ρ, with $\rho_2 - \rho_3$ odd, and subject to the preceding conditions, as required. ∎

Acknowledgments. We are grateful to Madhav Nori for his guidance and encouragement during the course of this work, and to C. S. Seshadri and Burt Totaro for helpful discussions.

REFERENCES

[1] D. Akhiezer, *Homogeneous Complex Manifolds*, Encyclopaedia Math. Sci. 10, Springer-Verlag, Berlin, 1986.
[2] A. Borel, *Linear Algebraic Groups*, Springer-Verlag, New York, 1991.
[3] A. Borel and H. Chandra, *Arithmetic subgroups of algebraic groups*, Ann. of Math. (2), 75 (1962), pp. 485–535.
[4] R. Bott, *Homogeneous vector bundles*, Ann. of Math. (2), 66 (1957), pp. 203–248.
[5] W. Fulton and J. Harris, *Representation Theory*, Springer-Verlag, New York, 1991.
[6] H. Grauert and K. Fritzsche, *Several Complex Variables*, Springer-Verlag, New York, Heidelberg, 1976.
[7] H. Grauert and R. Remmert, *Coherent Analytic Sheaves*, Springer-Verlag, Berlin, 1984.
[8] H. Grauert and R. Remmert, *Theory of Stein Spaces*, Springer-Verlag, Berlin, New York, 1979.
[9] R. Hartshorne, *Algebraic Geometry*, Springer-Verlag, New York, Heidelberg, 1977.
[10] G. Kempf, *Instability in invariant theory*, Ann. of Math. (2), 108 (1978), pp. 299–316.
[11] A. Knapp, *Lie Groups Beyond an Introduction*, Birkhäuser Boston, Boston, 1996.
[12] V. Lakshmibai and C. S. Seshadri, *Geometry of G/P-V*, J. Algebra, 100 (1986), pp. 462–557.
[13] V. Lakshmibai, P. Littlemann, and P. Magyar, *Standard monomial theory and applications*, in Representation Theories and Algebraic Geometry, A. Broer, ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998, pp. 319–364.
[14] P. Littelmann, *Paths and root operators in representation theory*, Ann. of Math. (2), 142 (1995), pp. 499–525.
[15] D. Littlewood, *Products and plethysms of characters with orthogonal, symplectic and symmetric groups*, Canad. J. Math., 10 (1958), pp. 17–32.
[16] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[17] D. Luna, *Slices Étales*, Bull. Soc. Math. France Mém., 33 (1973), pp. 81–105.
[18] D. Luna and Th. Vust, *Plongements d'espaces homogénes*, Comment. Math. Helv., 58 (1983), pp. 186-245.
[19] I. MacDonald, *Symmetric functions and Hall polynomials*, The Clarendon Press, Oxford University Press, New York, 1995.
[20] G. Margulis, *Explicit constructions of concentrators*, Problemy Inf. Transm., 9 (1973), pp. 325–332.
[21] H. Matsumara, *Commutative Ring Theory*, Cambridge University Press, Cambridge, UK, 1980.
[22] H. Minc, *Permanents*, Addison-Wesley, Reading, MA, 1978.
[23] M. Nagata, *Polynomial Rings and Affine spaces*, CBMS Reg. Conf. Ser. Math. 37, AMS, Providence, RI, 1978.
[24] Y. Matsushima, *Espaces homogènes de Stein des groupes de Lie complexes*, Nagoya Math. J., 16 (1960), pp. 205–218.
[25] K. Mulmuley and M. Sohoni, *Geometric complexity theory, P vs. NP and explicit obstructions*, in Advances in Algebra and Geometry (Hyderabad, 2001), C. Musili, ed., Hindustan Book Agency, New Delhi, India, 2003, pp. 239–261.
[26] K. D. Mulmuley and M. Sohoni, *Geometric complexity theory* I: *An approach to the P vs. NP and related problems*, SIAM J. Comput., 31 (2001), pp. 496–526.
[27] D. Mumford, J. Fogarty, and F. Kirwan, *Geometric Invariant Theory*, Springer-Verlag, Berlin, 1994.
[28] A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.
[29] J. Remmel and T. Whitehead, *On the Kronecker product of Schur functions of two row shapes*, Bull. Belg. Math. Soc. Simon Stevin, 1 (1994), pp. 649–683.
[30] T. Springer, *Linear algebraic groups*, in Algebraic Geometry IV: Linear Algebraic Groups, Invariant Theory, Encyclopaedia Math. Sci. 55, Springer-Verlag, Berlin, 1994, pp. 1–121.
[31] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1207–1282

FAULT-TOLERANT QUANTUM COMPUTATION WITH CONSTANT ERROR RATE*

DORIT AHARONOV† AND MICHAEL BEN-OR†

Abstract. This paper shows that quantum computation can be made fault-tolerant against errors and inaccuracies when η, the probability for an error in a qubit or a gate, is smaller than a constant threshold η_c. This result improves on Shor's result [*Proceedings of the 37th Symposium on the Foundations of Computer Science*, IEEE, Los Alamitos, CA, 1996, pp. 56–65], which shows how to perform fault-tolerant quantum computation when the error rate η decays polylogarithmically with the size of the computation, an assumption which is physically unreasonable. The cost of making the quantum circuit fault-tolerant in our construction is polylogarithmic in time and space. Our result holds for a very general local noise model, which includes probabilistic errors, decoherence, amplitude damping, depolarization, and systematic inaccuracies in the gates. Moreover, we allow exponentially decaying correlations between the errors both in space and in time. Fault-tolerant computation can be performed with any universal set of gates. The result also holds for quantum particles with $p > 2$ states, namely, p-qudits, and is also generalized to one-dimensional quantum computers with only nearest-neighbor interactions. No measurements, or classical operations, are required during the quantum computation. We estimate the threshold of our construction to be $\eta_c \simeq 10^{-6}$, in the best case. By this we show that local noise is in principle not an obstacle for scalable quantum computation. The main ingredient of our proof is the computation on states encoded by a quantum error correcting code (QECC). To this end we introduce a special class of Calderbank–Shor–Steane (CSS) codes, called polynomial codes (the quantum analogue of Reed–Solomon codes). Their nice algebraic structure allows all of the encoded gates to be transversal. We also provide another version of the proof which uses more general CSS codes, but its encoded gates are slightly less elegant. To achieve fault tolerance, we encode the quantum circuit by another circuit by using one of these QECCs. This step is repeated polyloglog many times, each step slightly improving the effective error rate, to achieve the desired reliability. The resulting circuit exhibits a hierarchical structure, and for the analysis of its robustness we borrow terminology from Khalfin and Tsirelson [*Found. Phys.*, 22 (1992), pp. 879–948] and Gács [*Advances in Computing Research: A Research Annual: Randomness and Computation*, JAI Press, Greenwich, CT, 1989]. The paper is to a large extent self-contained. In particular, we provide simpler proofs for many of the known results we use, such as the fact that it suffices to correct for bit-flips and phase-flips, the correctness of CSS codes, and the fact that two-qubit gates are universal, together with their extensions to higher-dimensional particles. We also provide full proofs of the universality of the sets of gates we use (the proof of universality was missing in Shor's paper). This paper thus provides a self-contained and complete proof of universal fault-tolerant quantum computation in the presence of local noise.

Key words. quantum computation, quantum fault tolerance, noise and decoherence, quantum error correction, concatenated codes, density matrices, polynomial codes, quantum Reed–Solomon codes, universal set of gates

AMS subject classifications. 81P68, 68Q01

DOI. 10.1137/S0097539799359385

*Received by the editors July 21, 1999; accepted for publication (in revised form) October 28, 2007; published electronically July 23, 2008. A preliminary version of this paper, under the name "Fault-Tolerant Quantum Computation with Constant Error," was published in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, El Paso, TX, 1997, pp. 176–188. An extended version quite similar to the current version is posted online at http://arxiv.org/abs/quant-ph/9906129. This research was supported by The Israel Science Foundation, grant 69/96, and the Minerva Leibniz Center at the Hebrew University in Jerusalem.

http://www.siam.org/journals/sicomp/38-4/35938.html

†School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel (doria@cs.huji.ac.il, benor@cs.huji.ac.il). The second author is the incumbent of the Jean and Helena Alfassa Chair in Computer Science.

1. Introduction. The area of quantum algorithms has witnessed remarkable discoveries over the past decade, the most remarkable of which is Shor's groundbreaking discovery of a polynomial quantum algorithm for factoring [82]. These results indicate a possibility that quantum computers exhibit exponential speedups over classical computers. It is yet unclear whether and how large scale quantum computers will be physically realizable (see [68] for possible implementation schemes). However, as in any physical system, quantum computers will *in principle* be subjected to noise, such as decoherence [103, 93, 94], and inaccuracies. Without error corrections, the effect of noise will accumulate and ruin the entire computation [97, 30, 71, 65, 66, 20]. Thus, the question of whether quantum computation can be protected against noise cannot be separated from the subject of quantum computational complexity.

One cannot hope to achieve fault tolerance of any computational model, be it quantum or classical, unless the noise model is restricted in some way. In this paper we adopt the assumption that the noise is *local*. Roughly, it means that the noise is independent between the different components of the system. This seems to be a fairly natural assumption to work with from a physical point of view and is often assumed in the classical case. We characterize the amount of noise by a parameter η which we call the *error rate*. The most basic variant of local noise with error rate η is called *independent probabilistic noise* and is defined as the following process: Each gate undergoes an arbitrary error with independent probability η, and in addition, for every time step, the qubits that do not participate in any gate undergo errors, too, each with independent probability η. Local noise can be defined much more generally, to include decoherence, systematic errors, and other kinds of physical processes. We will deal with these later in this paper, but for a first reading it is best to keep in mind the simplest local noise model of independent probabilistic noise.

This paper settles the question of whether quantum computation preserves its computational power in the presence of local noise. We give a positive answer to this question, for a very wide variety of cases under the title of local noise. To do this, we show how to simulate an unreliable quantum circuit by one that is robust to local noise of error rate η, as long as η is below a certain (constant) threshold. The overhead is only polylogarithmic in both space and time. By this we show that, in principle, local noise is not an obstacle against the physical realization of quantum computers. We do not attempt here to optimize the threshold value but rather to achieve a systematic and simple mathematical structure and as general a proof as possible. In the rest of the introduction, we state and explain the result in more detail, describe its context and limitations, and provide an outline of the proof.

1.1. Background on quantum error correction and fault tolerance. The analogue question of protecting classical computation from errors was already studied by von Neumann in 1956, when he showed that classical computation can be made robust to noise with constant error probability per gate [67]. This was done by using computation on redundant information, encoded by a repetition code.

The similar question for quantum computation, however, is far more complicated. Even the simpler question of transmitting quantum information reliably over a noisy channel poses a lot of conceptual obstacles in comparison with its classical analogue, and, in fact, scientists were skeptical about this possibility [97, 62]. There were several valid reasons for these concerns. First, the fact that quantum states cannot be cloned [99] implies that straightforward redundancy techniques cannot help. Second, Hilbert space is continuous, and this implies that it may be difficult to distinguish between bona fide quantum information and a state with a small error. Third, and

possibly the most difficult problem, is the fact that quantum noise can cause occasional *measurements* of the quantum state by the environment which lead to a collapse of the quantum state—a process which seems to cause irreversible damage to the quantum information. Finally, it seems that, in order to correct the error, one must measure the state to figure out what error had occurred. It seems impossible to perform such a measurement without collapsing the state and losing the quantum information.

These pessimistic beliefs were disputed and replaced with cautious optimism with the discoveries of the first examples of quantum error correcting codes (QECCs), encoding one qubit by nine [83] or seven [88] qubits and allowing for the correction of one faulty qubit. The main ingredients that allowed this discovery are as follows: (1) Despite the seemingly infinitely versatile set of possible quantum errors, all quantum errors on one qubit can actually be written as combinations of two errors (called *bit-flip* and *phase-flip*). It is thus sufficient to correct for these two errors. This discretization of the errors allowed turning to classical techniques of error correction. (2) In order to discover the error that had occurred, it suffices to perform a partial measurement on the state, which does not necessarily collapse the part of the state carrying the important information.

Immediately after this discovery, Calderbank and Shor [28], and independently Steane [89], presented a general construction of a large class of good quantum error correcting codes,[1] now called Calderbank–Shor–Steane (CSS) codes, which are based on known classical error correcting codes. These results were followed by the development of a whole theory for quantum error correction, including the important definition of stabilizer codes, and with many examples of QECCs (see, e.g., [68, 87] for more information and references). The theory of QECCs shows that, in principle, quantum information can be transmitted over a noisy channel in a reliable and efficient way.

The existence of QECCs is not in itself sufficient to ensure the possibility of quantum computation in the presence of noise. The reason is as follows. To compute in the presence of noise, one might try to encode the quantum state by using a QECC. In order to prevent the accumulation of errors, one must perform error corrections frequently, in between the computational steps. However, when dealing with noisy computation we are faced with several new problems that do not exist when dealing with the problem of transmitting information over noisy channels. First, in the information transmission setting, one assumes that the encoding and decoding processes are error-free. In the case of noisy computation, on the other hand, the error correction procedure itself is done in the presence of noise. It can thus introduce additional errors into the state and quite possibly cause more harm than help. A second problem is that the process of computation involves interactions between different qubits. If a gate is applied on two qubits, where one of them is faulty, it is likely that at the end of the operation the two qubits are faulty, or, in other words, the gate has caused the *propagation* of the error from a faulty qubit to an error-free qubit. One must therefore be able to compute on encoded states by using procedures and error corrections which do not allow the errors to propagate too much. Such procedures which limit the propagation of errors are called *fault-tolerant.*

In [84], Shor showed how to design fault-tolerant procedures for a universal set of quantum gates.[2] These procedures are applied to states encoded by a code from

[1] Good codes are codes which have a nonzero asymptotic transmission rate, in the presence of a constant error rate per bit.

[2] A set of gates is universal if an arbitrary unitary transformation can be constructed by using only gates from the set.

a certain class of CSS codes. In order to use these fault-tolerant constructions so as to improve the reliability of the quantum circuit, one would first encode the qubits in the original circuit by using the CSS quantum error correcting code and then apply fault-tolerant computation and fault-tolerant error corrections on these encoded states alternately. Shor showed that the resulting quantum circuit performs the desired computation reliably when the error rate, or the fault probability at each time step, per qubit or gate, decays polylogarithmically with the size of the quantum circuit. This result is a major improvement over the performances of quantum circuits without error corrections, in which the error probability is required to decay as one over the size of the circuit in order for the computation to succeed [21].

Nevertheless, the assumption that Shor used, namely, that the error probability decays polylogarithmically with the size of the computer, is a physically unrealistic assumption. We would like to assume that the resources required for applying one elementary operation are fixed and independent of the input size and, in particular, that the probability for a fault in each computer element, as well as the inaccuracy in each gate, is a constant number independent on the number of computer components.

1.2. Results. In this paper we improve on Shor's result and show how to perform fault-tolerant quantum computation in a more realistic model of computation, in which the error rate η is a fixed parameter, independent of the size of the computation.

THEOREM 1 (the threshold result, roughly). *There exists a threshold $\eta_0 > 0$ such that the following holds. Let $\epsilon > 0$. If Q is a quantum circuit operating on n input qubits for t time steps using s two- and one-qubit gates, there exists a quantum circuit Q' with depth, size, and width overheads which are polylogarithmic in n, s, t, and $1/\epsilon$ such that, in the presence of local noise of error rate $\eta < \eta_0$, Q' computes a function which is within ϵ total variation distance from the function computed by Q.*

The locality assumption is at the core of our threshold result. Our methods cannot handle errors which hit large sets of qubits chosen by an adversary. Within this realm of local noise, our fault tolerance result holds for a very general noise model. Our first threshold theorem (Theorem 12) is devoted to proving the threshold result for the independent probabilistic noise model. We use it as a basis to prove several other versions of the threshold theorem. Section 8 extends the result to work with what we call general local noise (Theorem 13). This version allows us to include in the noise model also decoherence, amplitude and phase damping, depolarization, systematic inaccuracies in the gates, leakage errors (of some sort—not disappearance of particles), and more. In fact, our result seems to apply to almost any conceivable quantum process as long as it is local. Later on, we generalize Theorem 12 in another direction (section 10) and show that our construction is reliable even when the probabilistic noise model allows correlations in space and time, as long as these correlations decay exponentially in some well defined sense.

We further generalize the threshold result by considering not only generalizations of the noise model but also different constraints to which the quantum system may be subjected. In particular, Theorem 14 shows that fault tolerance against general local noise can be achieved with any universal set of gates (not necessarily with the set of gates that we show how to encode fault tolerantly). This may be important in the likely scenario in which the set of gates implementable in the laboratory is different than the one we use in our constructions. Furthermore, in various implementation schemes only nearest-neighbor gates are available. Theorem 15 shows that fault tolerance can be achieved even when the particles of the quantum computer are

set on a one-, two-, or three-dimensional grid, and only nearest-neighbor interactions are allowed.

The various versions of the threshold theorem lead to different values of the threshold.

1.3. Assumptions on the architecture of the quantum system. It might be helpful to make explicit various issues related to the requirements from the physical realization of the quantum computer in order for our fault tolerance scheme to work.

- *Redundancy in the input.* We always assume that the input is given to the quantum circuit in many copies. This redundancy requirement is unavoidable if we want to achieve robustness to noise, because otherwise we would have negligible probability that we even start the computation correctly. This is also assumed in the classical scenario [67].
- *Redundancy in the output.* The output of the circuit is also given in a redundant way. In order to interpret the output string, one needs to calculate a certain majority function of the many output bits, which are all equal in the ideal case but not in the presence of errors.
- *Noise on wires vs. noise on gates.* In some quantum systems, the noise and decoherence on the wires are negligible. In other words, errors occur only while the qubits participate in a gate. In this paper we do not make such an assumption, and we assume that the wires are noisy, too. If one is dealing with a quantum system with noisy wires, then one must bear in mind that, in order to achieve fault tolerance, the quantum system must satisfy two important requirements.
  1. *Parallelism.* It is required that gates on different qubits can be applied in parallel, so that many gates can be applied at the same time step. It can be shown that, without parallelism, fault tolerance is impossible when wires are noisy, since error corrections cannot be applied fast enough to prevent accumulation of errors [5].
  2. *A supply of fresh qubits.* Another requirement is that fresh or clean qubits namely, qubits in the state $|0\rangle$, are available at any time and not just in the beginning. In other words, it must be possible to *restart* a qubit, namely, initialize it to the state $|0\rangle$, in the middle of the computation. Once again, without this assumption, reliable quantum computation is impossible because there is no way to discard entropy which accumulates rapidly in the circuit due to noise [7].
- *Intermediate measurements.* One might imagine cases in which measurements can be applied during the quantum computation, and so parts of the intermediate computation (e.g., computations inside the error correction procedure) can be performed by a classical computer. Moreover, in some cases it is also reasonable to assume that classical computations can be performed infinitely fast. This would be the case if the time it takes to perform a classical operation is negligible in comparison to the time it takes to perform a quantum gate. Of course, such cases are significantly simpler to handle than the general case. The reason is that, to a very large accuracy, classical computation can be assumed to be error-free, and so intermediate computations, such as parts of the error correction, can then be assumed to be error-free. In this paper we do not require this assumption. Measurements in the middle of the computation are not necessary in order to achieve fault tolerance. Though allowing intermediate measurements simplifies the situation considerably, we

make an effort to show that fault tolerance can be achieved without using classical operations or measurements. We do this because of two reasons. One is purely theoretical: One would like to know that measurements and classical operations are not essential and that the quantum model is complete in the sense that it can be made fault-tolerant within itself. Even more importantly, the assumption on the infinitely fast classical computation breaks down at some point and does not scale with the size of the input. One should certainly try to avoid such an assumption when attempting to prove a threshold theorem that holds asymptotically. The other reason we make an effort to avoid intermediate measurements is practical: In some suggestions for physical realizations of quantum computers (such as the NMR-based quantum computation—see [68]), it is very difficult to perform intermediate measurements during the quantum computation, and therefore everything should be done within the quantum computation model, without the help of classical computers.

- *Different sets of gates.* Our main fault-tolerant construction (Theorem 12) uses very specific universal sets of gates (see subsections 4.2 and 5.2 for the definitions). Nevertheless, one can actually convert the fault-tolerant circuit into one that uses any universal set of gates. This is proved and explained in section 9. Again, this is important for practical purposes, because the set of gates that is implementable in the laboratory might be entirely different than the ones that are easy to implement fault-tolerantly.
- *Nearest-neighbor interactions.* In some implementation schemes, interaction is limited to nearest-neighbor particles. We show that this is not a restriction in terms of fault tolerance: Resilience to noise can be achieved even under very restricted geometrical conditions, namely, that qubits are arranged in a one-dimensional lattice and gates are allowed only between nearest neighbors.

We summarize the division of our threshold theorems with respect to their assumptions on the architecture of the system. All of our theorems allow noise on the wires and thus assume parallelism and a constant supply of fresh qubits. Also, all of our theorems work without the assumption of intermediate measurements. Our first two versions of the threshold result, Theorems 12 and 13, and also section 10, are concerned with quantum systems subject to the following two assumptions on their architecture: First, a gate can be applied on any set of particles, regardless of their physical location. In other words, qubits that participate in a gate need not be nearest neighbors. Second, the final realization of the quantum circuit may consist of gates taken out of two particular sets, which we call $\mathcal{G}_1$ or $\mathcal{G}_2$. The other versions of our theorem release these assumptions. Theorem 15 shows that nearest-neighbor interactions suffice, and Theorem 14 allows the use of arbitrary universal sets of gates, including sets that contain only two-qubit gates.

1.4. Proof overview. We outline here the proof of our first threshold result, Theorem 12, which holds for the simplest noise model of independent probabilistic errors. The extensions to other cases build upon this case and do not require much more innovation.

1.4.1. Ingredients for the construction. There are several ingredients that are required in the proof.

1. A quantum error correcting code. To this end we define a new type of a quantum error correcting code (QECC) called *polynomial quantum error correcting code*. This is the quantum analogue of the Reed–Solomon code [64],

and it is a special case of CSS codes over large fields. We draw intuition here from Ben-Or, Goldwasser, and Wigderson [24], who used Reed–Solomon codes to achieve classical fault-tolerant distributed computation. An alternative version of the proof uses the general class of CSS code that Shor used in [84].

2. To compute on states encoded by QECCs, we need fault-tolerant maintenance procedures. First, we require a fault-tolerant way to perform error correction, which would allow us to correct errors in encoded states without introducing too many new errors. Second, we need a zero-state preparation procedure, namely, a procedure which prepares a state that encodes $|0\rangle$ (and uses fresh qubits for this purpose). These encoded zero states will be used as ancilla states for error correction as well as for computation. Both of these procedures can be performed in many different ways (e.g., [4]). Here we choose to work with the nice idea used by Steane for error correction [89]. Finally, we need a fault-tolerant decoding procedure which would allow us to read the logical value of the encoded state. This is not difficult to design.
3. To perform the actual computation, we need a universal set of gates which can be performed fault-tolerantly on encoded states. Each gate g needs to be replaced by an *encoded gate*, which takes the encoding of a state $|\alpha\rangle$ to the encoding of $g|\alpha\rangle$. Most importantly, the encoded gates need to be applied fault-tolerantly, i.e., without letting the errors propagate too much. Here is where the reason we chose to work with polynomial codes becomes apparent. Due to their algebraic structure, these codes exhibit the following very nice property: The entire universal set of quantum gates can be performed in a manner which is called *transversal*, or *bitwise*. This term refers to the application of the encoded gate g simply by applying g on each of the qubits in the code word individually (see Figure 1.1).
   This means that the set of encoded gates is extremely simple. Unfortunately, some complication cannot be avoided. The transversal operations in the case of polynomial codes indeed perform the desired computation, but in some cases they cause the degree of the polynomials involved in the code to increase (to see why this might happen, consider the case in which two polynomials are multiplied). To avoid accumulation of this effect, we perform a fairly simple procedure called *degree reduction*. This procedure, too, can be applied transversally, except it requires the help of ancilla zero states. Overall, we get a set of fault-tolerant procedures which possesses a nice algebraic structure. We also provide fault-tolerant procedures for the alternative codes we use, namely, the more general CSS codes that are used in Shor's scheme. Our constructions are based on those used by Shor [84], adapted to our re-

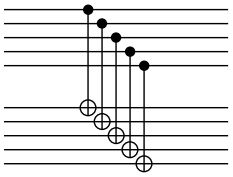


FIG. 1.1. *Encoded* CNOT *gate on two logical qubits encoded by five qubits each. Five* CNOT *gates are applied, where the ith gate is applied from the ith qubit in the first block to the ith qubit in the second block.*

quirement of no measurements during the computation. The fault-tolerant application of the Toffoli gate in this case does not possess as nice an algebraic structure as the procedures we design for polynomial codes.

4. We next need to show that the set of gates which we use is indeed universal, so that arbitrary quantum computations can be performed using our scheme. A known result due to Solovay [85] and Kitaev [50, 52] implies that it suffices for this purpose to show that the set of gates in question generates a dense subset in the group of unitary matrices. We use algebraic and field theoretical tools, and employ a few known group theoretical results, to prove that both sets of gates we use indeed satisfy this density requirement. In the case of polynomial codes, the proof is fairly involved. We note that a proof of the universality of the set of gates used by Shor was missing in [84].

1.4.2. The fault-tolerant circuit. We now want to combine all of the above ingredients to construct a fault-tolerant circuit from a given circuit M_0 which is not protected against errors. We want to make the circuit reliable against a constant error rate η, independent of the size of the computer n.

The first step of the idea is already present in Shor's original fault-tolerant result (except for using different codes and fault-tolerant procedures): Compute on states encoded by a QECC. We use a QECC that encodes one qubit into, say, m qubits. The circuit M_1 which computes on encoded states is defined as a *simulation* of M_0. A qubit in M_0 transforms to a block of m qubits in M_1, and each gate in M_0 transforms to a fault-tolerant procedure in M_1 applied on the corresponding blocks. Note that procedures might take more than one time step to perform. Thus, a time step in M_0 is mapped to a time interval in M_1, which is called a *working period*. In order to prevent accumulation of errors, at the beginning of each working period an error correction is applied on each block. The working period now consists of two stages: a correction stage and a computation stage. To be precise, the initialization and output stages in M_1 require some special attention, in order to encode the redundant input, and also decode the output states into a redundant classical output. We do not elaborate on this part here.

The idea is therefore to apply alternately computation stages and correction stages, hoping that during the computation stage the damage that accumulated is still small enough so that the corrections are able to handle it. We will soon argue that if the error rate is below a certain threshold, the reliability of M_1 is better than that of M_0. Let us assume this for the moment. In any case, we need to reduce the effective error rate of the final circuit to something which is inverse polynomially small, which would guarantee that with high probability there is effectively no error in the entire computation. Some thought would lead the reader to the conclusion that, regardless of how we choose the QECC in the above scheme, it seems impossible to achieve an improvement from a constant error rate to an effective error rate that is inverse polynomial.

Our solution is fairly simple to state. The crucial point here is that the new circuit M_1 might not be reliable enough for our purposes, but it suffices that it is even *slightly* more reliable than the original circuit. If this is the case, we can continue to improve the reliability by repeating the same process as we did for the original circuit, but starting from the new, more reliable circuit M_1. We do this for several levels; let us denote the number of levels by r, so the final circuit is M_r. How many levels are needed? It turns out that the improvement in the error rate is doubly exponential in the number of levels, and so it suffices that r is taken to be polyloglog in the

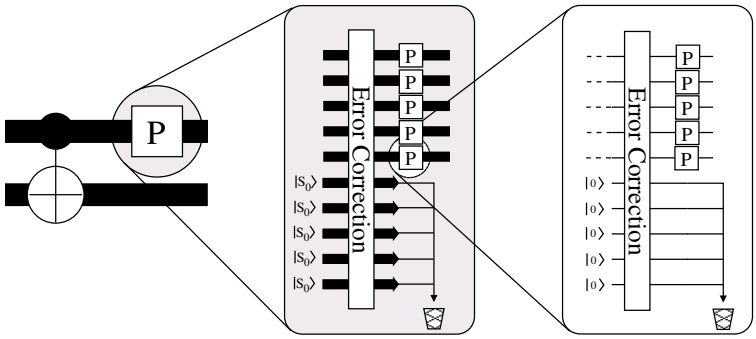


FIG. 1.2. *This figure shows the two-level simulation of a two-gate circuit. When looking at it at the coarsest resolution, we see the circuit on the left. The gates we see are those gates which* M_0 *consists of, namely, a* CNOT *followed by a one-qubit gate. By increasing the resolution a bit, we see the next level circuit* M_1*. The figure shows only part of it, namely, the second gate in* M_0 *transformed in* M_1 *to its encoded version. This includes an error correction of that block (with the help of an ancilla), followed by a transversal application of the one-qubit gate. By increasing the resolution yet one last time, we get to see the actual physical circuit, namely,* M_2*. Here we show only the encoding of one one-qubit gate in* M_1*.*

parameters of M_0.

The resulting circuit M_r exhibits a hierarchical structure resembling the one in Figure 1.2.

Each qubit in the original circuit transforms to a block of qubits in the next level, and they in their turn transform to a block of blocks in the second simulation, and so on. A gate in the original circuit transforms to a procedure in the next level, which transforms to a larger procedure containing smaller procedures in the next level, and so on. The final circuit computes in all of the levels: The largest procedures, computing on the largest (highest-level) blocks, correspond to operations on qubits in the original circuit. The smaller procedures, operating on smaller blocks, correspond to computation in lower levels. Note that each level simulates the error corrections in the previous level and adds error corrections in the current level. The final circuit, thus, includes error corrections of all of the levels, where during the computation of error corrections of larger blocks, smaller blocks of lower levels are being corrected. The lower the level, the more often error corrections of this level are applied, which is in correspondence with the fact that smaller blocks are more likely to be quickly damaged. This is the source of the advantage of using recursive simulations, rather than one step of simulation, as is done in Shor's scheme [84].

One other way to understand this is that the hierarchical structure takes advantage of a very important fact, namely, that the errors are located *randomly*. It is easy to see that the resulting QECC, namely, a concatenation of the QECC for r levels, can correct very few (less than a fraction of n) errors if their locations are chosen by an adversary, but as long as their locations are random, all is fine.

We note that the final circuit in one of our constructions (the one that uses polynomial codes) is made of p-qudits rather than qubits. This circuit can be easily converted to one that works with qubits by replacing each qudit by the appropriate number of qubits and each gate by a sequence of gates on these qubits. The analysis of robustness of this final circuit follows exactly the lines of section 9.

1.4.3. New reliability and the threshold value. We now want to argue that the reliability of the simulating circuit M_1 is larger than that of the original circuit

M_0, as long as the error rate is below a certain threshold. In other words, we want to explain why, below a certain threshold error rate, we should expect some improvement in reliability from one level of simulation to the next.

In the original circuit, the occurrence of one fault may cause the failure of the computation. In contrast, suppose that one fault occurs in each procedure of the simulating circuit. If the fault-tolerant procedures allow this fault to propagate to, say, at most one error in each block, then these faults would have no effect on the correctness of the simulation since they would be corrected during the error correcting stages. In other words, if in each encoded gate and the preceding error correction procedure the number of faults is such that they can cause only less than the correctable number of errors, then the error corrections will prevent the errors from accumulating. (In fact, this argument needs to be made much more carefully (see Lemma 9), but for the sake of the introduction here it suffices.) The *effective* error rate of M_1 is thus, roughly, the probability for more than the correctable number of faults to occur in one procedure.

Just for the sake of illustration, let us consider an example in which our QECC corrects one error but not two and a fault-tolerant procedure allows one fault to propagate to at most one error in each of the final blocks. We now estimate the effective error rate in a slightly incorrect way, but which demonstrates the idea. In this case, the effective error rate is the probability for more than one fault to occur in one procedure. In other words, we need to consider all ways in which two or more faults can occur in one procedure. This is a combinatorial calculation, which behaves approximately like

$$\eta_{\text{eff}} \approx \left(\begin{array}{c} A \\ 2 \end{array} \right) \eta^2,$$

where A is the number of locations an error can possibly occur during the application of one procedure. Now observe that, for η_{eff} to be strictly smaller than η, we get the requirement $\eta < \frac{1}{\binom{A}{2}}$. Thus, if η is below a certain threshold that depends on A, we gain an improvement in the reliability of the circuit. This is the origin for the *threshold* in the threshold theorem. See section 12 for estimations of the values in our construction.

1.4.4. The proof. The analysis of the scheme turns out to be quite complicated. To do this, we borrow terminology used by Khalfin and Tsirelson [49] and Gács [42] to analyze self-correcting classical cellular automata. First, we distinguish between two notions which we have been using so far as if they are the same: the *error* in the state of the qubits, i.e., the set of qubits which have errors, and the actual *faults*, namely, the occasions where noise operators were applied. We use the term *fault path* to denote the list of locations, namely, points in time and space, where faults had occurred (in a specific run of the computation).

We then define the notion of sparse errors and sparse fault paths. Naturally, due to the hierarchical structure of this scheme, these notions are defined recursively. A block in the lowest level is said to be k-close to its correct state if it does not have more than k errors, and a higher level block is "k-close" to its correct state if it does not contain more than k blocks of the previous level which are far from their correct state. If all of the blocks of the highest level are k-close to being correct, we say that the set of errors, or the deviation, is *k-sparse*.

Which set of faults does not cause the state to be too far from correct in the above metric? The answer is recursive, too: A computation of the lowest-level procedure

is said to be undamaged if not too many faults occurred in it. Computations of higher-level procedures are not damaged if they do not contain too many lower-level procedures which are damaged. A fault path will be called *sparse* if the computations of all of the highest-level procedures are not damaged.

The proof of the threshold result is done by showing that the probability for "bad" faults, i.e., the probability for the set of faults not to be sparse, decays as a double exponential with the number of levels r. Thus, bad faults are negligible, if the error rate is below the threshold. This is the easy part. The more complicated task is to show that the "good" faults are indeed good; i.e., if the set of faults is sparse enough, then the set of errors is also kept sparse until the end of the computation. This part is done by using a double induction on the number of levels r.

1.5. Related work and the state of the art in quantum fault tolerance. A preliminary version of the threshold result (for independent probabilistic errors) was published in the proceedings version of this paper [3]. Different variants of the threshold result for probabilistic noise were independently discovered at about the same time (1996) by Knill, Laflamme, and Zurek [56, 57, 58], who used Steane's seven-qubit QECC, and by Kitaev [50], who used toric codes. All of these works are based on the same idea of applying one scheme recursively to get a hierarchical structure and achieve approximately the same estimated threshold value of 10^{-6}.

An important unique contribution of our work is that our proof does not require intermediate measurements and classical operations during the quantum computation: Our fault-tolerant circuits use only quantum gates and, thus, work entirely within the framework of the quantum model. All works on fault tolerance we are aware of (other than the current paper) work under the assumption that intermediate measurements are allowed, and, moreover, classical computation can be performed infinitely fast. This makes the problem significantly easier, because under these assumptions large parts of the computation can be assumed to be error-free.

Our proof, as well as the proof of Kitaev [50], requires using a QECC that corrects two errors. It turns out that our construction works for a QECC that corrects one error, but a different analysis is required. An idea of how to perform this analysis was suggested in [56, 57, 58], by using the notion of overlapping rectangles. This idea was recently made into a proof by Aliferis, Gottesman, and Preskill [14]. A different proof was independently given by Reichardt [78].

Since the discovery of the threshold result mentioned above, intensive scientific effort was put into generalizing, simplifying, and improving the fault-tolerant networks. This effort started with the important work by Gottesman [45, 44] showing that fault tolerance can be achieved with any stabilizer code and with the works of Preskill [73, 74] and of Steane [90, 91]; it continued with a long line of works which we will not attempt to survey here for lack of space. The main goal of this line of work is practical: that of improving the threshold and making it realistic.

We mention a few main lines of developments. The first is that of developing the usage of ancilla state distillation, also known as *ancilla factories*. In the basic fault-tolerant schemes, one initiates the computation with states encoding 0, and the preparation of these states can be thought of as a separate process. Steane [90], Gottesman and Chuang [48], and finally Bravyi and Kitaev [27] have each shown how to use more and more cleverly designed ancillary states to significantly simplify the main computational process and improve the threshold.

The creation of ancilla states of high fidelity is the bottleneck in the threshold calculations in the above-mentioned schemes. In a breakthrough work, Knill [54] has

suggested to use concatenation of quantum error *detection* (rather than correction) codes, for the distillation of ancilla states. If an error is detected, the ancilla state can be discarded, without affecting the ongoing computation. The overhead in the number of qubits is large, but numerical estimations [54] of the threshold of this scheme revealed a value of up to several percent. Reichardt [76] and independently Aliferis, Gottesman, and Preskill [15] provide a rigorous proof of a much worse threshold than the estimated one, but still the best rigorously proven today: 10^{-3}. It is left open to close the gap between the numerics and the theoretical work.

Completely different methods to achieve fault tolerance were also explored. The first idea in this context is the beautiful idea of Kitaev, of achieving fault tolerance in the different model of topological quantum computation with anyons [51]. In this model of computation, the information is protected by topological means, and no concatenation is required.

Another interesting development along nonstandard lines is that of using subsystem codes [59]. These are a generalization of quantum error correcting codes. Aliferis and Cross [13] recently showed how to use subsystem codes due to Bacon [16], to get a threshold of $1.9 \cdot 10^{-4}$.

So far, we have discussed only methods which did not take into consideration geometrical restrictions on the system. Another issue which is explored in this paper is that of fault-tolerant quantum computation with nearest-neighbor interactions between qubits in one-, two, or three-dimensional systems. We prove a threshold result under such assumptions in section 11. This was proved independently also by Gottesman in [47]. In contrast to [47], our proof of generalization to low dimensions is very simple (see section 11), since one only has to keep track of the error propagation in one level, and the general threshold theorem (section 7) takes care of the rest. Both results provide very small values of the threshold. The lower bound on the threshold value in the case of $1D$ was improved recently to 10^{-6} in [92]. In two dimensions, better results are known. Svore, Divincenzo, and Terhal [95] have proved a threshold in $2D$ which is of the order of 10^{-5}, and Raussendorf and Harrington [75] have provided a scheme in which topological codes are used, in which the simulated threshold is of the order of 10^{-3}.

All of the above works consider the case of probabilistic noise only. Yet another different line of work in the area of fault tolerance is the study of the limitations and applicability of the threshold result in the presence of more general noise models. This direction is explored quite thoroughly in the current paper, where we prove the threshold result for a general local noise model (section 8) and, moreover, (slightly) relax the locality requirement to allow exponentially decaying correlations in the noise process (section 10). Following the preliminary version of this work [4], and based on similar methods, Terhal and Burkard [96], and recently Aliferis, Gottesman, and Preskill [14], proved threshold results in the presence of non-Markovian noise. The most recent development in this context [9] shows that fault tolerance can be achieved even in the presence of correlations which decay *polynomially* in space.

We also mention here an alternative approach to fault tolerance which has attracted interest, namely, "decoherence-free subspaces" (DFSs) [41, 102, 63, 101, 55]. A DFS is a subspace of the Hilbert space, which, under certain assumptions on the noise, is completely unaffected by the noise process. The computation is then performed entirely within this subspace [17, 18]. The DFS method might indeed be very useful in the presence of special types of noise, such as collective decoherence (see references above), and can thus deal with various correlated noise processes. However, we note that the assumptions on the noise made in the DFS model are quite

restrictive and, in particular, do not allow local noise. If one wants to correct for local noise, these methods must be combined with the standard methods of fault tolerance presented in this paper.

Finally, since the publication of the preliminary version of this paper [3], polynomial codes have found several applications in other areas of quantum computation such as secure multiparty computation [36] and quantum secret sharing [33]. Nonbinary codes were defined independently also by Chuang, Leung, and Yamamoto [31] and Knill [53]. Fault tolerance with higher-dimensional particles was discussed also in [46].

1.6. Conclusions and open problems. This paper implies that quantum computation can in theory be carried out in the presence of noise, as long as the noise satisfies a fairly reasonable assumption, namely, locality, and as long as the error rate in the system can be decreased below a certain constant value. Within this locality assumption, our results holds for a very general noise model. Moreover, our result holds for quantum systems with fairly limited constraints, such as no classical computation and subject to geometrical restrictions. Thus, this paper provides a thorough and general solution to the problem of quantum computation in the presence of local noise.

The value of the threshold is of utmost importance to the practicality of the theorem. We did not attempt to optimize the threshold value in our fault tolerance scheme, and our estimated threshold of $\approx 10^{-6}$ (see section 12) is far from being practical, according to the state of the art in experimental systems. As mentioned in subsection 1.5, since the preliminary publication of this work there have been many improvements. The best current rigorous results give a threshold value of approximately 10^{-3}, whereas computer simulations of the best constructions say that the threshold value is of the order of one to several percent. Providing rigorous proofs for the latter is a challenge. Any improvement in the threshold value is significant, as it brings quantum computation closer to being practical. We note that one should be careful in interpreting these numbers, since the threshold depends on many parameters; see section 12 and subsection 1.3 for further discussion of this matter.

Another parameter which is crucial for practical purposes is the space efficiency of the fault-tolerant construction, namely, the overhead in the number of qubits. Once again, we have not attempted to optimize this parameter. This parameter has achieved significantly less attention in the literature than the threshold value. See [77, 54] for some discussion of this matter.

It is of course desirable to save time as well. Our scheme requires a polylogarithmic blowup in the depth of the circuit. In the original version of this paper [3, 4] we conjectured that it might be possible to use a quantum analogue of multilinear codes [86], to reduce the multiplicative factor of $O(\log(n))$ to a factor of $O(\log(\log(n))$. Recently, Ahn [10] indeed achieved fault tolerance with this improved depth overhead, by using clever combinatorial considerations for toric codes. An open question is whether it is possible to reduce the time cost to a constant, as in the classical case. We conjecture that the answer is negative.

Another important open problem is to further relax the assumptions on the noise model and to make them as realistic as possible. This is a natural continuation of the work presented in sections 8 and 10 of the current paper and in several papers of the past few years [96, 14, 9].

The results achieved in this work shed light on a very interesting question, regarding the transition from quantum to classical physics [103]. Traditionally, this transition is viewed as a gradual transition (but see [49]). The threshold result suggests that in some cases this transition can actually be viewed as a *phase transition*,

occurring at a critical error rate, above which the system abruptly loses its ability to entangle large sets of particles. This is supported by the result of this paper, showing that at a very small error rate quantum systems can maintain their quantum nature, together with another result of ours [5], showing that at very high error rates quantum computers can be simulated efficiently by a classical Turing machine. In [2] one of us formalized this phase transition idea and provided a physical explanation for this computational difference between high and low error rates, by showing that in fault-tolerant circuits a phase transition in a certain property of entanglement called *entanglement length* occurs at a critical error rate. This draws a natural connection between the transition from quantum to classical and the notion of entanglement. The connection between the quantum-classical transition, entanglement, and quantum phase transitions has since become an active field of research (see, e.g., [69, 70]). We believe that the physical realization of fault-tolerant quantum computers might enable far better understanding of this fundamental question.

1.7. Organization of paper and instructions as to how to read it. Section 2 starts with a rigorous definition of the model of quantum circuits which we use, namely, quantum circuits with mixed states. We define the basic noise model, of independent probabilistic errors, and its various generalizations. Section 3 provides the definitions of quantum error correcting codes in general and the particular QECC codes we use in this paper. Section 4 describes how to apply fault-tolerant procedures for polynomial codes. Section 5 describes how to apply fault-tolerant gates on states encoded by CSS QECC. Section 6 proves the universality of the sets of gates defined in the previous two sections. Section 7 proves the threshold result for independent probabilistic noise. This is the main result of the paper. Section 8 generalizes the threshold result to general local noise. In section 9 we generalize our result to circuits which use any universal set of gates and not necessarily the set of gates for which we have constructed fault-tolerant procedures. In section 10 we relax the assumption of independence and explain how to deal with exponentially decaying correlations in the noise. Section 11 proves the threshold result for quantum circuits with restricted geometry, i.e., one-dimensional quantum circuits. In section 12 we discuss the estimation of the threshold values in various cases.

Many readers would be satisfied with understanding the main ingredients of the proof of the threshold result, for the simplest noise model of independent probabilistic noise, and for one type of QECC, either general CSS codes or polynomial codes. Such readers might also be less interested in the details of the proof that the set of gates we use is universal. The relevant chapters for those readers are sections 2.8, 2.9, 3, 4 (or 5), and 7. In general, the paper is written in a very modular way, so the readers can skip to the interesting parts.

Each section starts with a short overview. A reader who is interested in a more detailed overview than the one provided in the introduction, but still not in all of the details, can read only these overviews.

2. The model of noisy quantum circuits.

2.1. Overview. This section provides the necessary definitions for this paper. For more details on the quantum computation model consult [68]; for more on quantum mechanics consult [34, 79, 72].

The standard model [38, 39, 100] of quantum circuits with unitary gates allows only unitary operations on qubits. However, noisy quantum systems are not isolated from other quantum systems, usually referred to as the environment. Their inter-

actions with the environment are unitary, but, when restricting the system to the quantum computer alone, the operation on the system is no longer unitary, and the state of the quantum circuit is no longer a vector in the Hilbert space. It is indeed possible to stay within the unitary model with pure states, by keeping track of the state of the environment. However, this environment is not part of the computer, and it is assumed that we have no control or information on its state. We find it more elegant to follow the framework of the physicists and to work within the model of quantum circuits with mixed states defined by Aharonov, Kitaev, and Nisan [6]. In this model, the state of the set of qubits is always defined: It is a probability distribution over pure states, i.e., a *mixed state*. It is described by a *density matrix* (see below). The quantum gates in this model are not necessarily unitary: Any physically allowed operator on qubits is a quantum gate. In particular, this model allows an operation which adds a blank qubit to the system, discards a qubit, or applies a measurement in some basis. The model of quantum circuits with mixed states is equivalent in computational power to the standard model of quantum circuits [6] but seems more adequate for the study of noisy quantum computation. Since noise is a dynamical process which depends on time, the circuits we consider are leveled; i.e., gates are applied at discrete time steps.

To include noise in our model, we add noise operators which act in between the time steps of the circuit. The simplest model for noise is the independent probabilistic noise. After every time step, we consider the sets of qubits that interacted in the previous time step. Each such set undergoes a fault (i.e., an arbitrary quantum operation) with independent probability η, and η is referred to as the *error rate*. A qubit that did not participate in a gate undergoes the same process on its own.

The probabilistic noise process can be generalized to a more realistic model of noise, which is the following process. After each time step, each set of qubits participating in the same gate goes through a physical operator which is at most η-far from the identity (in some operator metric discussed below). This model includes, apart from probabilistic errors, also decoherence, amplitude and phase damping, systematic inaccuracies in the gates, and more (see [68] for an explanation of these terms). Two important assumptions were made in this definition: independence between different faults in space, i.e., locality, and independence in time, which is called the Markovian assumption. We remain within this framework until the end of the paper, and only in section 10 do we relax it to allow also exponentially decaying correlations in both time and space.

We note that all definitions here use qubits, namely, two-state particles, but can trivially be extended to any finite-dimensional particles, namely, to *qudits*.

2.2. Pure states. A quantum physical system in a *pure state* is described by a unit vector in a Hilbert space, i.e., a vector space with an inner product. In the *Dirac* notation a pure state is denoted by $|\alpha\rangle$. The physical system which corresponds to a quantum circuit consists of n quantum two-state particles, called qubits. The Hilbert space of a qubit is a two-dimensional complex space and is denoted by $\mathcal{H}_2$. We choose an orthonormal basis for this space and denote its vectors by $|0\rangle$ and $|1\rangle$. The Hilbert space of n qubits is the n-fold tensor product of $\mathcal{H}_2$, namely, $\mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_2$, its dimension is 2^n, and it is isomorphic to $\mathcal{C}^{2^n}$. As a basis for this space we choose all possible tensor products of the basis vectors of the individual qubits $|i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_n\rangle$, where i_j is either 0 or 1. We often use the notation $|i_1, i_2, \ldots, i_n\rangle$ or $|i\rangle$ for brevity. A pure state $|\alpha\rangle \in \mathcal{H}_2^n$ is a *superposition* of the basis states: $|\alpha\rangle = \sum_{i=1}^{2^n} c_i|i\rangle$, with $\sum_{i=1}^{2^n} |c_i|^2 = 1$. The transposed-complex conjugate of $|\alpha\rangle$ is denoted by $\langle\alpha|$. The

inner product between $|\alpha\rangle$ and $|\beta\rangle$ is denoted by $\langle\alpha|\beta\rangle$.

2.3. Mixed states and density matrices. In general, a quantum system is not in a pure state but in a *mixed state*, namely, a probability distribution, or a *mixture* of pure states, denoted by $\{\alpha\} = \{p_k, |\alpha_k\rangle\}$. This means that the system is with probability p_k in the pure state $|\alpha_k\rangle$. This description is not unique, as different mixtures might represent the same physical system. An equivalent unique description uses *density matrices*. A density matrix ρ on $\mathcal{H}_2^n$ is an Hermitian (i.e., $\rho = \rho^\dagger$) semipositive-definite matrix, of dimension $2^n \times 2^n$, with trace $\mathrm{Tr}(\rho) = 1$. A pure state $|\alpha\rangle = \sum_i c_i|i\rangle$ is represented by the density matrix $\rho_{|\alpha\rangle} = |\alpha\rangle\langle\alpha|$ (by definition, $\rho(i,j) = \langle i|\rho|j\rangle$). A mixture $\{\alpha\} = \{p_s, |\alpha_s\rangle\}$ is associated with the density matrix $\rho_{\{\alpha\}} = \sum_s p_s|\alpha_s\rangle\langle\alpha_s|$. This association is not one-to-one, but it is *onto* the density matrices, because any density matrix describes the mixture of its eigenvectors, with the probabilities being the corresponding eigenvalues.

Given a density matrix of n qubits, one can assign a density matrix to a subsystem A of $m < n$ qubits. We say that the rest of the system, represented by the Hilbert space $\mathcal{G} = \mathbf{C}^{2^{n-m}}$, is *traced out* and denote the new matrix by $\rho|_A$. We have $\rho|_A(i,j) = \sum_{k=1}^{2^{n-m}} \rho(ik, jk)$, and k runs over a basis for $\mathcal{G}$. In words, this means averaging over $\mathcal{G}$. The new density matrix $\rho|_A$ is called the *reduced* density matrix to A. If the state of the qubits which are traced out is in tensor product with the state of the other qubits, then discarding the qubits means simply erasing their state. However, if the state of the discarded qubits is not in tensor product with the rest, the reduced density matrix is always a mixed state.

2.4. Measurements. A quantum system can be *measured*, or observed. The measurement is a probabilistic process which, given a density matrix, yields a pair: a classical output and the associated density matrix. In this paper we will use only a measurement of one qubit in the computational basis. Let P_0 (P_1) be a projection on the subspace spanned by all states in which the measured qubit is 0 (1). For a given density matrix ρ, the classical output is $m \in \{0,1\}$ with probability $\Pr(m) = \mathrm{Tr}(P_m\rho)$. Given the outcome m, the resulting state of the quantum system after the measurement is equal to $\Pr(m)^{-1}P_m\rho P_m$ (this has the same meaning as a conditional probability distribution). Any measurement thus defines a probability distribution over the possible outputs.

2.5. Quantum operators and quantum gates. To define the possible transformations on density matrices, we consider linear operators on operators (sometimes called *superoperators*). A superoperator is called *positive* if it sends positive-semidefinite Hermitian matrices to positive-semidefinite Hermitian matrices. Superoperators can be extended to operate on larger spaces by taking a tensor product with the identity operator. A superoperator is a *completely positive map* if all of its extensions are positive. For a superoperator to be physically permissible, it must take density matrices to density matrices. Thus, it must be a completely positive map which is trace-preserving. It turns out that any superoperator which satisfies these conditions is indeed physically permissible, as it takes density matrices to density matrices [60, 61]. Thus quantum superoperators are defined as follows.

DEFINITION 1. *A* permissible superoperator T *from k to ℓ qubits is a trace-preserving, completely positive, linear map from density matrices on k qubits to density matrices on ℓ qubits. Its action on a density matrix ρ is denoted as follows:* $\rho \longmapsto T \circ \rho$.

Linear operations on mixed states preserve the probabilistic interpretation of the

mixture: $T \circ \rho = T \circ (\sum_s p_s |\alpha_s\rangle\langle\alpha_s|) = \sum_s p_s T \circ (|\alpha_s\rangle\langle\alpha_s|)$.

A very important example of a quantum superoperator is the superoperator corresponding to the standard unitary transformation $|\alpha\rangle \mapsto U|\alpha\rangle$, which sends a quantum state $\rho = |\alpha\rangle\langle\alpha|$ to the state $U\rho U^\dagger$.

Another important superoperator is to discard a set of qubits. This operation corresponds to tracing out the discarded qubits. In this paper, a discarding qubit gate will always be applied to qubits which ideally (if no error occurred) are in tensor product with the rest of the system.

One more useful quantum superoperator is the one which adds a blank qubit to the system $V_0 : |\xi\rangle \mapsto |\xi\rangle \otimes |0\rangle : \mathbf{C}^{2^n} \to \mathbf{C}^{2^{n+1}}$. This is described by the superoperator $T : \rho \mapsto \rho \otimes |0\rangle\langle 0|$.

A lemma by Choi [29] and Hellwig and Kraus [60, 61] asserts that any permissible superoperator from k to l qubits can be described as a combination of the above three operations: Add $k + l$ blank qubits, apply a unitary transformation on the $2k + l$ qubits, and then trace out $2k$ qubits. One can think of this as the definition of a permissible superoperator. We define a quantum gate to be the most general quantum operator.

DEFINITION 2. *A quantum gate from k to ℓ qubits is a permissible superoperator from k to ℓ qubits.*

In this paper we will use only three types of quantum gates: discard or add qubits and, of course, unitary gates. However, quantum noise will be allowed to be an arbitrary permissible superoperator (with the required restrictions as explained below).

We mention here that it is possible to describe a measurement as a superoperator. For this, we replace the classical result of a measurement $m \in \{0, 1\}$ by the density matrix $|m\rangle\langle m|$ in an appropriate Hilbert space $\mathcal{M}$. We then present the measurement as a superoperator T:

$$T\rho \;=\; \sum_m (P_m \rho P_m) \otimes \Big(|m\rangle\langle m|\Big). \tag{2.1}$$

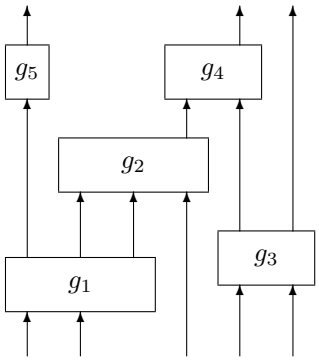
2.6. The trace metric on density operators. A useful metric on density matrices is the *trace* metric. It is induced from the *trace norm* for general Hermitian operators. The trace norm on Hermitian matrices is half the sum of the absolute values of the eigenvalues. $\|\rho\| = \frac{1}{2}\sum_i |\lambda_i|$. We thus have that $0 \le \|\rho_1 - \rho_2\| \le 1$. It was shown in [6] that the trace distance between two matrices equals the measurable distance between them in the following sense. Consider the two probability distributions D_1, D_2 which one gets when applying the same measurement to the two density matrices. Each such measurement thus induces some statistical difference between the outcomes, quantified by the total variation distance between the two distributions.[3] The trace distance is exactly the maximum over all possible measurements of half the total variation distance.

2.7. Quantum circuits with mixed states. We now define a quantum circuit.

DEFINITION 3. *Let $\mathcal{G}$ be a family of quantum gates.* A quantum circuit that uses gates from $\mathcal{G}$ *is a directed acyclic graph. Each node v in the graph is labeled by a gate $g_v \in \mathcal{G}$ from k_v to ℓ_v qubits. The in-degree and out-degree of v are equal k_v and ℓ_v, respectively.*

[3] The total variation distance between two probability distributions D_1, D_2 is defined to be $\sum_j |D_1(j) - D_2(j)|$.

Here is a schematic example of such a circuit.



The circuit operates on a density matrix as follows.

DEFINITION 4 (final density matrix). *Let Q be a quantum circuit. Choose a topological sort for Q: $g_t, \ldots, g_1$, where g_j are the gates used in the circuit. The final density matrix for an initial density matrix ρ is $Q \circ \rho = g_t \circ \cdots \circ g_2 \circ g_1 \circ \rho$.*

It is easy to see that $Q \circ \rho$ is well defined and does not depend on the topological sort of the circuit (see [6] for a detailed proof). At the end of the computation, the r output qubits are measured, and the classical outcome, which is a string of r bits, is the output of the circuit. For an input string i, the probability for an output string j is $\langle j|(Q \circ |i\rangle\langle i|)|j\rangle$.

2.8. Quantum circuits with independent probabilistic noise. To introduce noise to the quantum circuits, we consider leveled quantum circuits, where each level corresponds to one time step. Let us denote by T the number of time steps in the quantum circuit. The gates are then applied at integer times from 1 to T. We recall that, due to the input gate, qubits can be input to the circuit at different time steps. Faults occur in between time steps. We define a *location* (namely, a possible place where a fault can occur) as follows.

DEFINITION 5. *A set $(b_1, b_2, \ldots, b_\ell, t)$ is a* location *in the quantum circuit Q if the qubits $b_1, \ldots, b_\ell$ participate in the same gate in Q at time step t, and no other qubit participates in that gate. For this matter, the absence of a gate is viewed as the identity gate on one qubit, so if a qubit b does not participate in any gate at time t, then (b, t) is a location in Q as well.*

We will need the following terminology for our analysis of fault tolerance.

DEFINITION 6 (fault path). *The list of locations where faults had occurred (in a specific run of the computation) is called a fault path.*

DEFINITION 7 (error rate). *In the independent probabilistic noise model, each location is faulty with independent probability η, and this probability is called the error rate of the circuit.*

Each fault path $\mathcal{F}$ is thus assigned a probability $\Pr(\mathcal{F})$, which is a function of the number of locations in $\mathcal{F}$, the total number of locations in the circuit, and η.

The fault path does not determine the exact faults that occur in the circuit. In our model, once the fault path is chosen, an adversary chooses the exact noise superoperators applied on the faulty locations. That is, given a fault path, the adversary chooses T permissible superoperators; each one is applied on the faulty locations of time t in the fault path. We denote this choice of superoperators by $\mathcal{E}(\mathcal{F}) = (\mathcal{E}_T, \ldots, \mathcal{E}_2, \mathcal{E}_1)$, where $\mathcal{E}_t$ is a permissible superoperator applied by the noise process on the faulty locations of time step t.

We can now define the model precisely.

DEFINITION 8 (the independent probabilistic noise model). *Let Q be a leveled quantum circuit of T time steps. Let $\mathcal{E}(\mathcal{F})$ be a function from fault paths on Q to T-tuples of superoperators (as above). The noisy quantum circuit is denoted by $Q^{\mathcal{E}}$ and is defined as follows. We first define the final density matrix for a given choice of a fault path to be the result of applying the noise superoperators in between the time steps: $Q^{\mathcal{E}}(\mathcal{F}) \circ \rho = \mathcal{E}_t \circ g_t \circ \cdots \circ \mathcal{E}_2 \circ g_2 \circ \mathcal{E}_1 \circ g_1 \circ \rho$. The final density matrix of the circuit $Q^{\mathcal{E}} \circ \rho$ is defined as a weighted average over the final density matrices for each fault path:*

$$Q^{\mathcal{E}} \circ \rho = \sum_{\mathcal{F}} Pr(\mathcal{F}) Q^{\mathcal{E}}(\mathcal{F}) \circ \rho. \tag{2.2}$$

We say that a noisy quantum computer computes a function f with error ϵ in the presence of error rate η if $Q^{\mathcal{E}}$ defined above computes the function f with an error probability at most ϵ for any adversarial choice of faults $\mathcal{E}$.

This model includes, for example, probabilistic measurements of individual qubits, as well as the depolarization model in which each qubit is randomly replaced by a completely random qubit, i.e., a qubit in the identity density matrix.

2.9. Quantum circuits with general local noise. Most of the paper uses only the independent probabilistic noise model defined above. However, in section 8 we extend the threshold result to hold for a much more general noise model. In the general noise model, we replace the probabilistic faults which occur with probability η in each location with a different noise operator which is applied at each and every location in the quantum circuit, after every time step. The only restriction on the noise operator applied at a specific location is that it is within η distance to the identity. Thus, in between time steps, a *noise operator* of the following form operates on *all* of the qubits (here we give an example for the noise operator after time step t):

$$\mathcal{E}(t) = \mathcal{E}_{A_{1,t}}(t) \otimes \mathcal{E}_{A_{2,t}}(t) \otimes \cdots \otimes \mathcal{E}_{A_{l_t,t}}(t). \tag{2.3}$$

$A_{i,t}$ runs over all possible locations at time t, and for each one of them,

$$\|\mathcal{E}_{A_{i,t}}(t) - I\| \leq \eta. \tag{2.4}$$

The norm we use here is the *diamond norm* on superoperators [50, 6]. The exact definition of the diamond norm is not needed in this paper. We merely use the following properties (see [6]):

1. $\|T\rho\| \leq \|T\|\|\rho\|$.
2. $\|TR\| \leq \|T\|\|R\|$.
3. $\|T \otimes R\| = \|T\|\|R\|$.
4. The norm of any permissible superoperator T is equal to 1.

This defines the general local noise model with error rate η.

An interesting example of such a noise process is a restricted version of the independent probabilistic noise model. Suppose for each one of the superoperators in the product (2.3) we write

$$\mathcal{E}_{A_{i,t}}(t) = \left(1 - \frac{\eta}{2}\right) I_{A_{i,t}} + \frac{\eta}{2} \mathcal{E}'_{A_{i,t}}(t), \tag{2.5}$$

where $\mathcal{E}'_{A_{i,t}}(t)$ is some permissible superoperator. This operator is within η distance from the identity, and so this satisfies the conditions of local noise (2.3), (2.4) with error rate η. It is easy to see that this gives a restricted version of the independent noise model with error rate $\eta/2$, because the operator $\mathcal{E}_{A_{i,t}}$ can be interpreted as if the operator $\mathcal{E}'_{A_{i,t}}(t)$ is applied with probability $\eta/2$, and otherwise nothing is done to the qubit. Note that we do not get the independent probabilistic model in its full generality because (2.3) implies that the noise operator on the faulty locations of each time step is a tensor product of operators, one on each location.

This gives a very general error model, under the locality restriction. The model includes many physical sources for noise and errors, such as decoherence, systematic inaccuracies in the gates, amplitude and phase damping, and more (for explanations of these terms, see [43, 68]). As for leakage errors, this is a term that can mean one of two things. The first is that the state of the particle, due to noise, leaves the subspace spanned by computational states. This can be easily taken care of by defining the computational space to include all possible states of the particle, and so this is included implicitly in our discussion (allowing, as we do, higher-dimensional particles). The other meaning of leakage errors is the disappearance of a particle. We do not deal with this source of errors in this paper.

The general local noise model can be further relaxed to allow exponentially decaying correlations in time and space, but we delay the exact definition to section 10 where we deal with such correlations.

3. Quantum error correcting codes.

3.1. Overview of QECC, CSS codes, and polynomial codes. We will use here only QECCs which encode one qubit into, say, m qubits, and we call the set of m qubits a *block*. By linearity, if $|0\rangle$ is encoded by $|S_0\rangle$ and $|1\rangle$ by $|S_1\rangle$, then $a|0\rangle + b|1\rangle$ is encoded by $a|S_0\rangle + b|S_0\rangle$. It follows that such a quantum code is a two-dimensional subspace in the Hilbert space of m qubits. We note that we slightly abuse language here, confusing between the code space and the map encoding the states into this subspace. The exact meaning should be clear from the context. The state of the encoded qubit is sometimes called the *logical* state. The code is said to correct q errors if the logical state is recoverable given that not more than q errors occurred in the block. Roughly speaking, we say that no more than q errors occurred if the density matrix of the remaining $m - q$ qubits did not change.

There is a big difference between classical ECCs and QECCs, in that, for QECCs, not only basis states but also quantum superpositions should be recoverable after an error occurred. The codes we use here are CSS codes [28], and their construction is based on two ideas that enable the protection of superpositions.

1. Despite the fact that quantum operations are extremely versatile and can be continuous, the most general fault or quantum operation on a qubit can be described as (loosely speaking) a linear combination of four simple operations: the identity, i.e., no error at all, a bit-flip ($|0\rangle \leftrightarrow |1\rangle$), a phase-flip ($|0\rangle \longmapsto |0\rangle, |1\rangle \longmapsto -|1\rangle$), or both a bit-flip and a phase-flip. Thus correcting for these three errors suffice. This important fact was first proved by Bennett et al. [22], by using a beautiful group symmetry argument and a notion called *twirl*. We give here a simple proof based on an idea by Steane [89].
2. In order to correct bit-flips, one can use the analogue of classical error correcting codes. To correct phase-flips, one observes that phase-flips are actually bit-flips in the Fourier basis. One can therefore correct bit-flips in the Fourier transform basis, and this will translate to correcting phase-flips in the correct

basis. To correct general errors, one would first correct bit-flips and then correct bit-flips in the Fourier transformed basis, which translates to correcting phase-flips in the original basis.

CSS codes [28, 89] were first presented for qubits, that is, over the field F_2. We give here a proof for the fact that CSS codes are quantum codes, which is simpler than the original proof [28], and generalize it to the field F_p for any prime[4] $p > 2$.

Next, we define a new class of quantum codes, which are called *polynomial codes*. The idea underlying their construction is based on a theorem by Schumacher and Nielsen [80] which asserts that a quantum state can be corrected only if no information about the state has leaked to the environment through the noise process. More precisely, if the reduced density matrix on any t qubits does not depend on the logical qubit, then there exists a unitary operation which recovers the original state even if the environment interacted with t qubits, i.e., t errors occurred. This is reminiscent of the situation in classical secret-sharing schemes [81] and hints at the following possibility: We should divide the "secret," i.e., the logical qudit, among many parties (i.e., physical qudits) such that no t parties share any information about the secret. We adopt Shamir's secret-sharing scheme [81] which suggested to use random polynomials, evaluated at different points in a field of p elements, as a way to divide a secret among m parties. A random polynomial of degree d is chosen, and then each party gets the evaluation of the polynomial at a different point in the field F_p. The secret is the value of the polynomial at 0. To adopt this scheme to the quantum setting, we simply replace the random polynomial by a superposition of all polynomials of the appropriate degree to get a QECC. The result is the quantum analogue of Reed–Solomon codes [64]. It turns out that polynomial codes are a special case of CSS codes over a large field F_p.

The remainder of the section assumes basic knowledge in classical error correcting codes, which can be found in [64].

3.2. Quantum codes. A quantum code is a subspace of some dimension, say, k, in the Hilbert space of $m > k$ qubits. A word in the code is any vector in the subspace or, more generally, any density matrix ρ supported on the subspace. We say that a density matrix ρ' on m qubits is a word with q errors in the qubits $b_1, \ldots, b_q$, if ρ' can be written as $\mathcal{E}(b_1, \ldots, b_q) \circ \rho$, where ρ a word in the code, and $\mathcal{E}(b_1, \ldots, b_q)$ is some permissible superoperator on the q qubits. A quantum circuit $\mathcal{R}$ is said to correct an error $\mathcal{E}(b_1, \ldots, b_q)$ on ρ if

$$\mathcal{R} \circ \mathcal{E}(b_1, \ldots, b_q) \circ \rho = \rho. \tag{3.1}$$

We can now define a quantum error correcting code which corrects q errors.

DEFINITION 9 (QECC). *An $[[m, k, q]]$ quantum error correcting code is a subspace of dimension 2^k in the Hilbert space of m qubits such that the following holds. Let $\{|\alpha_i\rangle\}_{i=1}^k$ be a basis for C. There exists a quantum circuit $\mathcal{R}$, called an error correction procedure, such that for any i, j and $\mathcal{E}(b_1, \ldots, b_q)$*

$$\mathcal{R} \circ \mathcal{E}(b_1, \ldots b_q) \circ |\alpha_i\rangle\langle\alpha_j| = |\alpha_i\rangle\langle\alpha_j|.$$

Note that, from linearity, the last requirement implies the correction of any word in the code and not just the basis states. Note that $\mathcal{R}$ may (and, in fact, has to) use extra qubits during its application, but at the end all ancillary qubits are discarded, and the original state should be recovered.

[4] In fact, the results in this paper can be extended to work over any field, but we do not provide the details here.

3.3. From general errors to bit-flips and phase-flips. We now prove that it suffices to correct bit-flips and phase-flips in order to correct for general errors. To do this, we use the linearity of quantum operators. We recall the definition of the Pauli operators (slightly modified for simplicity here):

$$\mathcal{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ \sigma_y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},\ \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{3.2}$$

σ_x is called a bit-flip, since it takes $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. σ_z is called a phase-flip, as it multiplies $|1\rangle$ by a minus sign, while leaving $|0\rangle$ as is. σ_y is a combination of the two, since $\sigma_y = \sigma_z\sigma_x$. Let us define a *Pauli error* on a qubit as the application of one of the nontrivial Pauli matrices on that qubit. We note that the Pauli operators form a basis for the matrices on one qubit.

We proceed to many qubits. Consider all possible strings e of length m in the alphabet $\mathcal{I}, X, Y, Z$. The set of 4^m matrices:

$$\sigma_e = \sigma_{e_m} \otimes \cdots \sigma_{e_1}, \quad e \in \{\mathcal{I}, X, Y, Z\}^m, \tag{3.3}$$

form a basis for the linear operators on the Hilbert space of m qubits. We call it the Pauli basis. It is easy to check that this is an orthonormal basis with respect to the inner product $(V, U) = \frac{1}{2^m}\mathrm{tr}(VU^\dagger)$ for $2^m \times 2^m$ matrices.

DEFINITION 10. *We say that σ_e, for $e \in \{\mathcal{I}, X, Y, Z\}^m$, is a Pauli error of length q, if the string e has at most q coordinates that are not the identity.*

Before we show that it suffices to correct Pauli errors, we need two definitions.

DEFINITION 11. *We say that $\mathcal{R}$ corrects Pauli errors of length q in C if for any two basis states in the code, $|\alpha\rangle$ and $|\alpha'\rangle$, and any Pauli error of length q, σ_e, we have*

$$\mathcal{R} \circ \sigma_e|\alpha\rangle\langle\alpha'|\sigma_e^\dagger = |\alpha\rangle\langle\alpha'|. \tag{3.4}$$

DEFINITION 12. *We say that $\mathcal{R}$ detects Pauli errors of length q in C if for any two basis states of the code, $|\alpha\rangle$ and $|\alpha'\rangle$, and any two different Pauli errors of length q, $\sigma_e \neq \sigma_{e'}$,*

$$\mathcal{R} \circ \sigma_e|\alpha\rangle\langle\alpha'|\sigma_{e'}^\dagger = 0. \tag{3.5}$$

The reason for the fact that such $\mathcal{R}$ is said to detect errors is the following. Suppose $\mathcal{R}$ first writes down e on an ancilla $\mathcal{R} \circ \sigma_e|\alpha\rangle = |\alpha\rangle \otimes |e\rangle$. Then for two different errors the two ancilla states are orthogonal, and thus when discarding the ancilla qubits we get zero. If $\mathcal{R}$ does not write e on an ancillary register, but instead e can be discovered from the extra information written on the ancilla qubits, then discarding the ancilla register would result similarly in 0 for two different errors. In this paper we use error correcting procedures which not only correct but also detect the errors. We can now prove the following.

THEOREM 2. *Let C be a quantum code, and let $\mathcal{R}$ correct and detect any Pauli error of length q in C. Then R corrects any general error on any q qubits in C.*

Proof. We recall (section 2) that any permissible noise operator $\mathcal{E}$ on q qubits can be written as a combination of three operators: adding $2q$ blank qubits, applying a unitary transformation on the $3q$ qubits, and then discarding $2q$ qubits. We write the error operator as $\mathcal{E} = \mathcal{T} \circ \mathcal{L}$, where $\mathcal{T}$ is the discarding qubits operator. In the most general case, $\mathcal{L}$ is the following operation on q qubits. For all $0 \leq i \leq 2^q - 1$,

$$\mathcal{L} : |i\rangle \longmapsto |0^{2q}\rangle|i\rangle \longmapsto \sum_{j=0}^{2^q-1} |A_{i,j}\rangle \otimes |j\rangle, \tag{3.6}$$

where all of the coefficients are put into the vectors $|A_i^j\rangle$ which are states of $2q$ qubits (these vectors are not necessarily unit vectors). Observe that $\mathcal{L}$ can be presented as a multiplication of the state of the right register of q qubits by

$$\sum_{i,j} |A_{i,j}\rangle \otimes |j\rangle\langle i|. \tag{3.7}$$

The above object can be viewed as a $2^q \times 2^q$ matrix, whose entries are not real numbers but operators. In particular, each entry is a tensor product with a 2^{2q}-dimensional vector $|A_{i,j}\rangle$.

We would like to present this object in terms of the Pauli basis. This is easy to do by recalling that the Pauli basis is orthonormal. Define A to be the matrix whose entries are the vectors $|A_{i,j}\rangle$. We get that the object in (3.7) is equal to

$$\sum_{e \in \{\mathcal{I},X,Y,Z\}^q} |A_e\rangle \otimes \sigma_e,$$

where $|A_e\rangle = (A, \sigma_e) = \frac{1}{2^q}\mathrm{tr}(A\sigma_e^\dagger)$ ($A\sigma_e^\dagger$ is a matrix of vectors, the trace of which is a vector).

We thus have that

$$\mathcal{L} = \sum_{e,e'} |A_e\rangle\langle A_{e'}| \otimes \sigma_e \cdot \sigma_{e'}^\dagger. \tag{3.8}$$

We can now see that $\mathcal{R}$ indeed corrects for the error $\mathcal{E} = \mathcal{T} \circ \mathcal{L}$, by the following:

$$\begin{aligned} \mathcal{R} \circ \mathcal{E} \circ |\alpha\rangle\langle\alpha'| &= \mathcal{R} \circ \mathcal{T} \circ \sum_{e,e'} |A_e\rangle\langle A_{e'}| \otimes \sigma_e |\alpha\rangle\langle\alpha'| \sigma_{e'}^\dagger \\ &= \left(\mathcal{T} \circ \sum_e |A_e\rangle\langle A_e| \right) \otimes |\alpha\rangle\langle\alpha'| = |\alpha\rangle\langle\alpha'|, \end{aligned} \tag{3.9}$$

where in the second equality we have used the fact that $\mathcal{T}$ commutes with $\mathcal{R}$ because they operate on different qubits and that $\mathcal{R}$ corrects and detects errors. In the last equality we used the fact that $\mathcal{T} \circ \sum_e |A_e\rangle\langle A_e| = c$ does not depend on $|\alpha\rangle, |\alpha'\rangle$. By choosing $|\alpha\rangle = |\alpha'\rangle$, we have that $\mathrm{Tr}(\mathcal{R} \circ \mathcal{E} \circ |\alpha\rangle\langle\alpha|) = \mathrm{Tr}(c|\alpha\rangle\langle\alpha|) = c$, but on the other hand $\mathcal{R} \circ \mathcal{E}$ is trace-preserving, so $c = 1$. □

3.4. Calderbank–Shor–Steane codes. We give here the definition of CSS codes, which is a slight modification (and simplification) of the definition from [28] but gives the same codes. A linear code of length m and dimension k is a subspace of dimension k in F_2^m, where F_2^m is the m-dimensional vector space over the field F_2 of two elements. Let C_1, C_2 be two linear codes of length m such that $\{0\} \subset C_2 \subset C_1 \subset F_2^m$, and let us define a quantum code by taking the superpositions of all words in a coset of C_2 in C_1 to be one basis word in the code. We have

$$\forall a \in C_1/C_2 : |S_a\rangle = \frac{1}{\sqrt{2^{\dim(C_2)}}} \sum_{w \in C_2} |w + a\rangle. \tag{3.10}$$

Note that $|S_a\rangle$ is well defined and does not depend on the representative of the coset since if $(a_1 - a_2) \in C_2$, then $|S_{a_1}\rangle = |S_{a_2}\rangle$. Also, for different cosets the vectors are orthogonal. Thus, this defines a basis for a subspace of dimension $2^{\dim(C_1)-\dim(C_2)}$.

This is our quantum code. Note that the support of $|S_a\rangle$ are words in the code C_1. We see that bit-flips can be corrected by using classical error correction techniques for the code C_1 (that is, the classical error correction is performed unitarily, adding ancillary qubits).

Before we discuss how to correct phase-flips, let us define a very important quantum gate on one qubit, called the *Hadamard* gate, or the Fourier transform over F_2:

$$H = H^{-1} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}. \tag{3.11}$$

Observe that

$$H\sigma_z H^{-1} = \sigma_x. \tag{3.12}$$

This means that a phase-flip transforms to a bit-flip in the Fourier transform basis. By applying the Hadamard gate on each qubit in $|S_a\rangle$, we get the state:

$$\begin{aligned} |C_a\rangle &= H \otimes H \otimes \cdots \otimes H |S_a\rangle = \frac{1}{\sqrt{2^{m+\dim(C_2)}}} \sum_{b=0}^{2^m-1} \sum_{w \in C_2} (-1)^{(w+a)\cdot b} |b\rangle \\ &= \frac{1}{\sqrt{2^{m-\dim(C_2)}}} \sum_{b \in C_2^\perp} (-1)^{a\cdot b} |b\rangle, \end{aligned} \tag{3.13}$$

which is a superposition of words in the perpendicular subspace $C_2^\perp$. Thus, to correct phase-flips, one transforms to the Fourier basis and corrects bit-flips in the code $C_2^\perp$. These observations led to the following theorem due to Calderbank and Shor [28]. We give here a simple proof of this theorem, based on Theorem 2.

THEOREM 3 (CSS codes). *Let C_1 and $C_2^\perp$ be linear codes over F_2, of length m, such that $0 \subset C_2 \subset C_1 \subset F_2^m$, and such that $C_2^\perp, C_1$ correct q errors. Then the subspace spanned by $|S_a\rangle$ for all $a \in C_1/C_2$ is a $[[m, 2^{\dim(c_1)-\dim(C_2)}, q]]$ QECC. The error correction procedure $\mathcal{R}$ is constructed by correcting bit-flips with respect to C_1 in the S-basis, rotating to the C-basis by applying Hadamard on each coordinate, correcting with respect to $C_2^\perp$, and rotating back to the S-basis.*

Proof. We define the procedure $\mathcal{R}_{C_1}$ to be a unitary embedding of m qubits into the space of $2m$ qubits, by

$$\mathcal{R}_{C_1} |i\rangle = |w(i)\rangle \otimes |e(i)\rangle \tag{3.14}$$

for each $i \in F_2^m$, where $w(i) \in C_1$ is a string of minimal distance to i, and $e(i) \in \{0,1\}^m$ satisfies $w(i) + e(i) = i$. Since this is a one-to-one transformation, it is a unitary embedding, and therefore it is a permissible superoperator. Let $e_b \in \{0,1\}^m$ have at most q 1′s in it. Let $\mathcal{E}_b$ be an error operator which is the corresponding tensor product of bit-flips (σ_x) and identities. In the coordinates where e_b is 0, $\mathcal{E}_b$ has the identity, and in the coordinates where it is 1, $\mathcal{E}_b$ has a bit-flip. Then for any $|\alpha\rangle, |\alpha'\rangle$ supported on C_1, we have

$$R_{C_1} \circ \mathcal{E}_b \circ |\alpha\rangle\langle\alpha'| = |\alpha\rangle\langle\alpha'| \otimes |e_b\rangle\langle e_b|. \tag{3.15}$$

$\mathcal{R}_{C_2^\perp}$ is defined similarly:

$$\mathcal{R}_{C_2^\perp} |j\rangle = |w(j)\rangle \otimes |e(j)\rangle, \tag{3.16}$$

where $w(j) \in C_2^\perp$ is a string of minimal distance to j, and $e(j) \in \{0,1\}^m$ satisfies $w(j) + e(j) = j$. Let $e_f \in \{0,1\}^m$ have at most q 1′s. Let $\mathcal{E}_f$ be the error operator which is the corresponding tensor product of phase-flips (σ_z) and identities. Then for any $|\beta\rangle, |\beta'\rangle$ supported on $C_2^\perp$, we have

$$R_{C_2^\perp} \circ \mathcal{E}_f \circ |\beta\rangle\langle\beta'| = |\beta\rangle\langle\beta'| \otimes |e_f\rangle\langle e_f|. \tag{3.17}$$

Denote by $\mathcal{H}$ the operator that applies the Hadamard gate H on every qubit: $\mathcal{H} = H \otimes H \otimes \cdots \otimes H$. We claim that the operator

$$\mathcal{R} = \mathcal{T}_f \circ \mathcal{T}_b \circ \mathcal{H} \circ \mathcal{R}_{C_2^\perp} \circ \mathcal{H} \circ \mathcal{R}_{C_1} \tag{3.18}$$

is the desired error correcting procedures. $\mathcal{T}_f, \mathcal{T}_b$ are the operators discarding the qubits added for $\mathcal{R}_{C_2^\perp}, \mathcal{R}_{C_1}$, respectively.

By Theorem 2, it is enough to show that this procedure corrects and detects q Pauli errors. To show that it corrects q Pauli errors, write the error vector e in two parts e_b and e_f as follows: e_b is 1 in the coordinates where σ_x or σ_y occurred and 0 elsewhere. e_f is 1 in the coordinates where σ_z or σ_y occurred and 0 elsewhere. We can therefore write the error operator $\mathcal{E}$ as a product of two operators $\mathcal{E} = \mathcal{E}_f \circ \mathcal{E}_b$, by using the fact that $\sigma_y = \sigma_z \sigma_x$. We now have

$$\mathcal{R} \circ \mathcal{E} \circ |\alpha\rangle\langle\alpha'| = \mathcal{T}_f \circ \mathcal{T}_b \circ \mathcal{H} \circ \mathcal{R}_{C_2^\perp} \circ \mathcal{H} \circ \mathcal{R}_{C_1} \circ \mathcal{E}_f \circ \mathcal{E}_b \circ |\alpha\rangle\langle\alpha'|. \tag{3.19}$$

We would like to start developing the expression above by applying $\mathcal{R}_{C_1}$, but the phase errors stand in the way. For this, note that for any string $|i\rangle$ we have

$$\mathcal{R}_{C_1} \mathcal{E}_f |i\rangle = \mathcal{E}_f R_{C_1} |w(i)\rangle \otimes \mathcal{E}_f |e(i)\rangle.$$

This means that

$$R_{C_1} \mathcal{E}_f = (\mathcal{E}_f \otimes \mathcal{E}_f) R_{C_1}. \tag{3.20}$$

We apply (3.20) as well as (3.15), to the right-hand side of (3.19). We get

$$\mathcal{R} \circ \mathcal{E} \circ |\alpha\rangle\langle\alpha'| = \mathcal{T}_f \circ \mathcal{H} \circ \mathcal{R}_{C_2^\perp} \circ \mathcal{H} \circ \mathcal{E}_f \circ |\alpha\rangle\langle\alpha'| \otimes \mathcal{T}_b \circ \mathcal{E}_f \circ |e_b\rangle\langle e_b|, \tag{3.21}$$

where we have used the fact that $\mathcal{T}_b$ commutes with all operators except R_{C_1}. Since $\mathcal{E}_f$ is trace-preserving, we can now apply $\mathcal{T}_b$ and get a scalar which is exactly 1. We proceed with

$$\begin{aligned} &\mathcal{T}_f \circ \mathcal{H} \circ \mathcal{R}_{C_2^\perp} \circ \mathcal{H} \circ \mathcal{E}_f \circ |\alpha\rangle\langle\alpha'| \\ &= \mathcal{T}_f \circ \mathcal{H} \circ \mathcal{R}_{C_2^\perp} \circ (\mathcal{H} \circ \mathcal{E}_f \circ \mathcal{H}) \circ (\mathcal{H} \circ |\alpha\rangle\langle\alpha'|) \\ &= \mathcal{T}_f \circ \mathcal{H} \circ (\mathcal{H} \circ |\alpha\rangle\langle\alpha'|) \otimes |e_f\rangle\langle e_f| = |\alpha\rangle\langle\alpha'|. \end{aligned} \tag{3.22}$$

We thus see that $\mathcal{R}$ indeed corrects q Pauli errors. It is left to show that $\mathcal{R}$ also detects q Pauli errors. To show that

$$\mathcal{R} \circ \sigma_e |\alpha\rangle\langle\alpha'| \sigma_{e'}^\dagger = 0,$$

we observe that if $e \neq e'$, then either $e_b \neq e_b'$ or $e_f \neq e_f'$. We can repeat the previous argument. If $e_b \neq e_b'$, we get zero due to $\mathcal{T}_b \circ |e_b\rangle\langle e_b'| = 0$. Otherwise, we have $e_f \neq e_f'$, and we get zero because $\mathcal{T}_f \circ |e_f\rangle\langle e_f'| = 0$. □

3.5. CSS codes over F_p. The theory of quantum error corrections can be generalized to quantum computers which are composed of quantum particles of $p > 2$ states, called qudits. If we want to stress the dimensionality of the particles, we call them p-qudits. To generalize the notion of bit-flips and phase-flips to p-qudits, define the following two matrices:

- B : $B|a\rangle = |(a+1) \text{mod p}\rangle$,
- P : $P|a\rangle = w^a|a\rangle$,

where $w = e^{\frac{2\pi i}{p}}$. The analogue of Pauli matrices are combinations of powers of these matrices, i.e., the p^2 matrices:

$$B^c P^{c'} \quad \forall c, c' \in F_p. \tag{3.23}$$

Just as we did in the case of Pauli matrices, we can consider the p^{2m} m-fold tensor products of such matrices and show that they form a basis for matrices on m p-qudits. This basis is orthonormal, with respect to the inner product $(U, V) = \frac{1}{p^m}\text{tr}(UB^\dagger)$.

Like in the case of qubits, errors of type B transform to errors of type P and vice versa, via the analogue of the Hadamard, namely, the appropriate Fourier transform. In fact, there are several possible analogues of the Hadamard matrix for the case of F_p. Let $w = e^{2\pi i/p}$ and $w_l = w^l$. Then we define the lth Fourier transform over F_p to be

$$W(w_l) : |a\rangle \longmapsto \frac{1}{\sqrt{p}} \sum_{b \in F} w^{lab}|b\rangle. \tag{3.24}$$

It can be easily checked that

$$\forall c \in F_p, \quad W(w_l)P^c W(w_l)^{-1} = B^{c\ell^{-1}}, \quad W(w_l)B^c W(w_l)^{-1} = P^{c\ell}. \tag{3.25}$$

We can now define CSS codes over F_p in a very similar way as it is done for F_2. We first fix a choice of inner product over F_p^m with which we work: We choose the coefficients c_i in the bilinear form $\vec{a} \cdot \vec{b} = \sum_{i=1}^m c_i a_i b_i$. This fixes what we mean by the orthogonal code $C_2^\perp$. The statements and proofs of Theorems 2 and 3 are generalized to F_p by using the above definition of the generalized bit-flip B and the generalized phase-flip P, where F_2^m is replaced by F_p^m and the Hadamard gate on the ith qubit is replaced by the Fourier transform $W(w_{c_i})$ over the ith qudit.

3.6. Polynomial quantum codes. We define polynomial codes over the field F_p, for p a prime. We set d to be an upper bound on the degree of the polynomials used in the code and m to be the length of the code. Let $m < p$ be the number of elements in the field F_p with which we will be working. Set $\alpha_1, \ldots, \alpha_m$ to be m distinct nonzero elements of the field F_p. We consider the set of polynomials over F_p of degree at most d:

$$V_d = \{f(\cdot) \in F_p[x], \deg(f) \leq d\}, \tag{3.26}$$

where $F_p[x]$ is the field of polynomials with coefficients in F_p. Define the following classical codes:

$$\begin{aligned} C_1 &= \{(f(\alpha_1), \ldots, f(\alpha_m))|f(\cdot) \in V_d\} \subset F_p^m, \\ C_2 &= \{(f(\alpha_1), \ldots, f(\alpha_m))|f(\cdot) \in V_d, f(0) = 0\} \subset C_1. \end{aligned} \tag{3.27}$$

We can now define the quantum code:

$$\forall a \in F_p, \quad |S_a\rangle = \frac{1}{\sqrt{p^d}} \sum_{f(\cdot)\in V_d, f(0)=a} |f(\alpha_1), \ldots, f(\alpha_m)\rangle = \frac{1}{\sqrt{p^d}} \sum_{w\in C_2} |w + \vec{a}\rangle, \tag{3.28}$$

where $\vec{a}$ is the vector of length m with a at each coordinate. Since C_2 has p different cosets in C_1, the dimension of the code is p, and the code encodes exactly one p-qudit. We prove the following theorem.

THEOREM 4. *A polynomial code of degree d with length m over F_p is a* $[[m, p, \min\{\lfloor \frac{m-d-1}{2} \rfloor, \lfloor \frac{d}{2} \rfloor\}]]$ QECC.

Proof. Two different words in C_1 agree on at most d coordinates, and thus C_1 is a linear code of distance $m-d$. It can thus correct and detect $\lfloor (m-d-1)/2 \rfloor$ errors. $C_2^\perp$ (under any choice of the inner product in which the coefficients are nonzero) is a linear code of minimal distance $\geq d+1$. This is true since the projection on any d coordinates of the code C_2 contains all possible strings of length d, and therefore the only vector of length d orthogonal to all of the vectors is the 0 vector. Thus, $C_2^\perp$ corrects and detects $\lfloor d/2 \rfloor$ errors. Theorem 4 follows from Theorem 3 for F_p. □

We call the polynomial code which uses polynomials of degree $m-d-1$ (the codegree of d) the *dual code* of the code which uses polynomials of degree d. By the above lemma, the two codes correct the same number of errors.

4. Fault-tolerant gates for polynomial codes.

4.1. Overview. In this section we define $\mathcal{G}_1$ (subsection 4.2), the universal set of gates with which we work when computing with polynomial codes. We note that our set of gates involves only one- and two-qudit gates. We show how to apply the gates in $\mathcal{G}_1$ on encoded states, in a fault-tolerant manner. Our fault-tolerant procedures will all have what we call *spread* 1. This notion will be defined shortly (subsubsection 4.3.4), but roughly it means that one fault in a procedure can cause at most one error in the final state of the procedure. We augment the fault-tolerant gates by fault-tolerant procedures for error correction, zero-state preparation and decoding (a zero-state preparation procedure prepares a state $|S_0\rangle$, and a decoding procedure takes $|S_a\rangle$ to m copies of a). These procedures also have spread 1. This section thus proves the following theorem.

THEOREM 5. *For any gate in $\mathcal{G}_1$, there exists a fault-tolerant procedure with spread* 1 *that computes the gate on states encoded by a quantum polynomial code. There exist also fault-tolerant error correction, decoding, and zero-state preparation procedures for this code, which all have spread* 1. *Moreover, all of these procedures use only gates from $\mathcal{G}_1$ and in particular do not use measurements.*

The requirement that the procedures use only gates from $\mathcal{G}_1$ is imposed so that the scheme can be applied recursively, as we will do in section 7.

In an ideal situation, all gates can be applied transversally, as in Figure 1.1, in which case it is clear that one fault can propagate to at most one error in each block. Unfortunately, we do not know of any universal set of gates and any code in which all gates can be applied transversally in the simple way depicted by Figure 1.1. It is here that the advantage of polynomial codes comes into play. Due to the algebraic properties of polynomials, all of the gates in $\mathcal{G}_1$ can be applied essentially transversally: The gates are applied either transversally or by first preparing some states of the form $|S_0\rangle$ and then applying some operations transversally on the computational blocks and the ancilla blocks together.

To achieve this nice property, we observe that in the case of polynomial codes the transversal application of the gates always achieves the correct result, except for one problem: Instead of getting the final state as a superposition of polynomials with degree d, for some gates we end up with the correct logical dit, except that it is encoded with polynomials of a different degree. This can be illustrated if one considers gates which involve multiplication of polynomials, such as the generalized Toffoli gate, where the degree d becomes $2d$. Likewise, the Fourier transform gate takes d to the codegree $m-d-1$. To get back into the code that uses polynomials of degree d, we make use of a procedure called *degree reduction*. In the classical case, degree reduction was used by Ben-Or, Goldwasser, and Wigderson [24] to achieve fault-tolerant classical distributed computation. These techniques can be adapted to the quantum case, as was done in the original version of this paper [3, 4]. Here, however, we present a much simpler construction that reduces the degree by teleportation between states of different degrees. This technique, which was used in [8, 36], is based on ideas in [48].

In order for the above ideas to work, it must be that, even after the degree has changed, the state is still inside a QECC so that errors can be corrected. For this reason we work with codes of length $m = 3d + 1$ such that the quantum polynomial code of twice the degree (which is equal to the codegree) corrects the same number of errors (see Theorem 4).

It is therefore the case that, for polynomial codes, all fault-tolerant procedures can be applied in a remarkably simple manner, namely, transversally with the help of ancilla zero states.

We start with the definition of the set of gates $\mathcal{G}_1$ and continue to general definitions related to encoded gates and to fault-tolerant procedures, such as transversal and semitransversal operations and the spread of errors inside an encoded gate. We then proceed to the description of fault-tolerant encoded gates and finally to fault-tolerant error correction, zero-state preparation, and decoding procedures. We note that much work is put into making these final three procedures measurement-free. At the end of this section we remark about how to simplify procedures when measurements and classical computations can be used.

4.2. The set of gates for polynomial codes $\mathcal{G}_1$. We fix $m = 3d + 1$ in the polynomial codes we use. We work with the following set of gates, which we denote by $\mathcal{G}_1$:

1. $\mathsf{NOT}_p(c)$: $\forall\ c \in F_p$, $|a\rangle \longmapsto |a + c\rangle$ (also denoted B^c).
2. CNOT_p: $|a, b\rangle \longmapsto |a, a + b\rangle$.
3. CNOT_p^{-1}: $|a, b\rangle \longmapsto |a, a - b\rangle$.
4. SWAP: $|a\rangle|b\rangle \longmapsto |b\rangle|a\rangle$.
5. Multiplication by a constant: $0 \neq c \in F_p$: $|a\rangle \longmapsto |ac\rangle$.
6. P^c (generalized phase): $\forall c \in F_p\ |a\rangle \longmapsto w^{ca}|a\rangle$, for $w = e^{2\pi i/p}$.
7. Generalized controlled phase of order c: $\forall c \in F_p\ |a\rangle|b\rangle \longmapsto (w)^{abc}|a\rangle|b\rangle$.
8. $W(c)$ (generalized Fourier transform (of order c)): $|a\rangle \longmapsto \frac{1}{\sqrt{p}} \sum_{b \in F_p} w^{abc}|b\rangle \forall 0 < l < p$.
9. Generalized Toffoli: $|a\rangle|b\rangle|c\rangle \longmapsto |a\rangle|b\rangle|c + ab\rangle$.
10. Adding a qudit in the state $|0\rangle$.
11. Discarding a qudit.

All of the additions and multiplications are in F_p (i.e., modulo p). We will often denote $\mathsf{NOT}_p(1)$ simply by NOT_p. We note that, if the characteristic of the field we work with is not 2, we can replace the generalized Toffoli gate in the above construction

with the following gate:

- Squaring gate: $|a\rangle|b\rangle \longmapsto |a\rangle|b + a^2/2\rangle$.

We get a set of gates which consists of two-qudit gates without any three-qudit gate involved. In either case, the set is proved to be universal in section 6. This set is by no means a minimal set for universality, but the fault-tolerant procedures become simpler and shorter if we have a larger repertoire of fault-tolerant gates that we can use.

4.3. Fault-tolerant procedures—general definitions.

4.3.1. Encoded gates. Say we have a unitary gate g which is applied on the state $|\alpha\rangle$ in the original circuit. We now want to apply the corresponding gate to the state encoding $|\alpha\rangle$. We denote the encoding of $|\alpha\rangle$ by $\phi(|\alpha\rangle)$.

DEFINITION 13. *A sequence of gates Q is said to encode a gate g for the code C, and is denoted by $\Phi(g)$, if, for any superposition $|\alpha\rangle$,*

$$\Phi(g)\phi(|\alpha\rangle) = \phi(g|\alpha\rangle).$$

4.3.2. Transversal and semitransversal gates. The way to apply a gate which is the simplest and which allows the least propagation of errors is the transversal application.

DEFINITION 14. *Consider a gate g on, say, k qudits (k is between 1 and 3 in this paper's case). Consider k blocks of m qudits each. Let us label the qudits in each block from 1 to m (from left to right). We say that the gate g is applied* transversally *on $1, 2$, or 3 encoded blocks if, in order to apply $\Phi(g)$ on k encoded blocks, it suffices to apply the gate g m times, each time on all qudits labeled by the same label.*

We will sometimes need to modify the above transversal construction by just a little bit. Instead of applying the same gate on the set of ith qudits, independent of i, we allow ourselves to apply a gate which depends on the index i. The structure of the circuit remains the same, as in Figure 1.1. We call this way of encoding a gate *semitransversal*. In both cases, it is clear that one fault of a gate during the procedure can affect at most one qubit in each block at the end of the procedure. We now make these notions slightly more precise.

4.3.3. Errors versus faults. To analyze the propagation of errors in our fault-tolerant procedures, we need to make an important distinction between *errors*, which are the actual deviations of the quantum state from being correct, and *faults*, which are the events that occur that cause the qubits to have errors. Let us start by defining what we mean by errors. This requires some definition since the state of one qudit is not well defined in the quantum model, and so we cannot consider one qudit and say that it is "correct" or not.

DEFINITION 15 (deviation). *Consider a density matrix ρ' of a set of qudits B. We say that ρ' is deviated from the correct matrix ρ on the set of qudits $A \subseteq B$ if $\rho'|_{B-A} = \rho|_{B-A}$.*

This definition coincides with the more operative notion of errors which we used in the discussion about error correction in section 3.

CLAIM 1. *Let ρ be a (correct) density matrix of a pure state on a set of qudits B. The matrix ρ' is deviated from ρ on a set of qudits A if and only if there exists a permissible quantum operator on the set A which takes ρ to ρ'.*

Proof. For one direction, suppose that the deviation in ρ' is confined to the set of qudits A. Consider a *purification* of ρ', namely, a state $|\psi'\rangle$ of the set of qudits B plus extra qudits C, such that $|\psi'\rangle\langle\psi'||_B = \rho'$. Such a state exists by, e.g., [68, page 110].

Note that reducing the matrices $|\psi\rangle\langle\psi| \otimes |0\rangle_C\langle 0|_C$, $|\psi'\rangle\langle\psi'|$ to the set of qudits $B - A$ results in the same matrix. By standard results (see, e.g., [68, page 111, Exercise 2.81]), there is a unitary matrix U acting on $C \cup A$ which takes $|\psi\rangle \otimes |0\rangle_C$ to $|\psi'\rangle$. Thus, to get from ρ to ρ', we add the qudits in C in the state $|0\rangle_C$, apply U on $A \cup C$, and discard C. We have designed a permissible operator on A that takes ρ to ρ'. The other direction of the claim is trivial. □

The above claim relates the notions of deviation and error operators. Thus, if ρ is the density matrix of the correct state (which is always a pure state in our construction), we can loosely say that the set of qudits A are the faulty qudits or that the *errors* occurred on the qudits in A. We note that the set A is not uniquely defined. In the rest of the paper, however, in every place where we assume something about the deviation set A, it suffices to choose an arbitrary set of qudits that satisfies the relevant assumption and continue from there.

We will use the above correspondence between error operators and deviations many times in the paper. The way we do this is that we prove, say, that the deviation from the correct state at some stage in our construction is confined to some set of qudits. This implies, by Claim 1, that one can view the density matrix as if an error operator was applied on the deviated qudits, and so all of the results about quantum error correction apply. From now on, therefore, we can talk only about deviation.

The notion of a fault is completely different from the notion of error or deviation. To understand this difference, recall subsection 2.8, where the notions of "locations" and "fault paths" were defined. A *fault* in the circuit is thus the noise operator which is applied at a certain location that appears in the fault path.

We will analyze the effect of faults occurring at certain locations, on the resulting errors in the final states.

4.3.4. Spread. In order to analyze how faults in the circuit affect the errors in the final state, we define the notion of a "spread" of a circuit or a procedure. In most cases, a very simple consideration is required: It is clear that a fault in a location $(q_1, \ldots, q_l, t)$ can affect a qubit q' at time $t' > t$ only if there is a path in the circuit from $(q_1, \ldots, q_l, t)$ to (q', t'). In other words, if we know the correct propagation of some density matrix in the circuit, an additional fault at a certain location can cause a deviation from that correct propagation only in locations affected by the location of the fault via such a path in the circuit. For transversal procedures, it is thus easy to see that one fault can affect at most one qudit in each block. We say that the spread of the procedure is 1.

Unfortunately, for the error correction, zero-state preparation, and decoding procedures, a more careful analysis is required, because the circuit itself is far from being transversal. In such a case one needs to actually take into account the computation performed by the circuit, in order to bound the propagation of errors. It is sufficient for our purposes that the error propagation is limited only when the total number of errors is small. For example, we cannot hope to control the number of errors in the output if the number of errors in the input to an error correction procedure is large.

Definition 16. *We consider procedures that compute on states encoded by a QECC which can correct q errors. We say that the procedure has* spread 1 *if the following holds. Consider a fault path with k faulty locations in this procedure. Suppose that the input state to the procedure is deviated on a set of qudits which has at most f qudits in each of the blocks on which the procedure works. We require that, as long as $f + k \leq q - 1$, adding one additional faulty location to the fault path increases the final*

deviation in each one of the relevant blocks by at most 1 *(while leaving the number of errors in the other blocks unchanged).*

Except for sections 9 and 11, all of our procedures have spread equal to 1. However, there are cases in which more complicated situations arise, where the propagation of errors is confined to a small number of qudits but larger than 1. For example, such is the case when we introduce geometrical constraints to the system, which cause more propagation of errors. The above definition can be generalized to spread equal to ℓ.

DEFINITION 17. *We consider procedures that compute on states encoded by a QECC which can correct q errors. We say that the procedure has* spread ℓ *if the following holds. Consider a fault path with k faulty locations in this procedure. Suppose that the input state to the procedure is deviated on a set of qudits which has at most f qudits in each of the blocks on which the procedure works. We require that, as long as $(f+k)\ell \leq q-\ell$, adding one additional faulty location to the fault path increases the final deviation in each one of the relevant blocks by at most ℓ (while leaving the number of errors in the other blocks unchanged).*

The proofs of section 7, showing that our general hierarchical scheme (without exact specification of the code being used) is fault-tolerant, are done for the more general case of spread ℓ. This does not impose any additional difficulty in the proof. For a first reading, it is perhaps simpler to keep the definition of spread 1 in mind.

4.3.5. Issues related to ancillas. In some of the procedures, we will use ancilla qudits as extra working space. At the end of the procedure these qudits will be discarded, in order to get exactly the state we need. As was explained in subsection 2.5, we will always discard qudits which are (in an ideal noiseless situation) in tensor product with the rest of the system. In this case the operation of discarding qudits means simply erasing their state, and the resulting state is a pure state. This is necessary if we want to apply unitary operations on the encoded states. We will describe a procedure by specifying what it does to basic states of the code. It is easy to see that if for any input basis state the ancilla qubits at the end of the procedure are in a tensor product with the rest of the qubits, and their state does not depend on the input basis state, i.e.,

$$|S_a\rangle \longmapsto |S_{g(a)}\rangle \otimes |A\rangle, \tag{4.1}$$

where A is independent of a, then for any input superposition for the procedure the ancilla qubits are in tensor product with the rest of the qudits, and thus they can be discarded simply by erasing them.

4.4. Transversal and semitransversal gates for polynomial codes. We begin with the simplest cases.

LEMMA 1. *The first seven gates:* $\mathsf{NOT}_p(c), \mathsf{CNOT}_p, \mathsf{CNOT}_p^{-1}, \mathsf{SWAP}$*, multiplication by a constant, generalized phase, and generalized controlled phase, can all be applied in a transversal or semitransversal manner.*

Proof. It is easy to check that the first five gates, namely, $\mathsf{NOT}_p(c)$, CNOT_p, CNOT_p^{-1}, SWAP, and multiplication by a constant different than zero, can be applied transversally by applying the gate on corresponding qudits from the different blocks. We give here just one example, for the $\mathsf{NOT}_p(c)$ gate:

$$|S_a\rangle = \sum_{w\in C_2} |a+w_1\rangle\otimes\cdots\otimes|a+w_m\rangle \longmapsto \sum_{w\in C_2} |a+c+w_1\rangle\otimes\cdots\otimes|a+c+w_m\rangle = |S_{a+c}\rangle. \tag{4.2}$$

The other four gates are similar. The generalized phase is just slightly more complicated. It can be applied semitransversally. Define c_l as the interpolation coefficients such that

$$\forall\ f \in F[x],\ \deg(f) \le m-1, f(0) = \sum_{i=1}^{m} c_i f(\alpha_i). \tag{4.3}$$

We apply on the lth qudit the gate $|a\rangle \longmapsto w^{c_l a}|a\rangle$. This achieves the desired operation because

$$\begin{aligned} |S_a\rangle &= \frac{1}{\sqrt{p^d}} \sum_{f\in V, f(0)=a} |f(\alpha_1), \ldots, f(\alpha_m)\rangle \\ &\longmapsto \frac{1}{\sqrt{p^d}} \sum_{f\in V, f(0)=a} \Pi_{i=1}^{m} w^{c_l f(\alpha_l)} |f(\alpha_1), \ldots, f(\alpha_m)\rangle \\ &= \frac{1}{\sqrt{p^d}} \sum_{f\in V, f(0)=a} w^a |f(\alpha_1), \ldots, f(\alpha_m)\rangle = w^a |S_a\rangle. \end{aligned} \tag{4.4}$$

Finally, the generalized controlled phase of order c can also be applied semitransversally, by using the same idea. On the lth coordinate we apply the generalized controlled phase of order $c \cdot c_l$. We get

(4.5)

$$\begin{aligned} |S_a\rangle|S_b\rangle &= \frac{1}{p^d} \sum_{f\in V, f(0)=a} |f(\alpha_1), \ldots, f(\alpha_m)\rangle \sum_{g\in V, g(0)=b} |g(\alpha_1), \ldots, g(\alpha_m)\rangle \\ &\longmapsto \frac{1}{p^d} \sum_{f,g\in V, f(0)=a, g(0)=b} \Pi_{i=1}^{m} w^{cc_l f(\alpha_l) g(\alpha_l)} |f(\alpha_1), \ldots, f(\alpha_m)\rangle |g(\alpha_1), \ldots, g(\alpha_m)\rangle. \end{aligned}$$

Since f, g are both of degree at most d, the product of the two polynomials is of degree at most $2d$, and the interpolation applies. If we denote $f \cdot g = h$, we get $\Pi_{i=1}^{m} w^{cc_l f(\alpha_l) g(\alpha_l)} = w^{c\sum_l c_l h(\alpha_l)} = w^{ch(0)} = w^{cf(0)g(0)} = w^{abc}$, as we wanted. ∎

4.5. Rotation to the dual code. The remaining two gates, namely, the generalized Fourier transform and the generalized Toffoli (or the squaring gate), are not as simple, since the transversal application changes the degree of the polynomials involved. To this end we add a superscript d or $2d$ denoting the degree of the polynomials used in the code, as in $|S_a^d\rangle$ or $|S_b^{2d}\rangle$. We start by showing the effect of a semitransversal Fourier transform on a state encoded by using degree d' polynomials. Denote by $w_l = w^{c_l}, l = 1, \ldots, m$, for c_l the interpolation coefficients, as in the proof of Lemma 1. Recall that in our notation

$$W(c_l) : |a\rangle \longmapsto \frac{1}{\sqrt{p}} \sum_{b\in F_p} w_l^{ab} |b\rangle. \tag{4.6}$$

We apply $W(c_l)$ to the lth qudit for all $1 \le l \le m$.

$$|S_a^{d'}\rangle \longmapsto W(c_1) \otimes W(c_2) \otimes \cdots \otimes W(c_m) |S_a^{d'}\rangle. \tag{4.7}$$

Claim 2 (rotation to the dual code). *The transformation of* (4.7) *performs the generalized Fourier transform, except it moves the word to the dual code, namely, the*

polynomial code with the codegree $m-d'-1$*:*

$$|S_a^{d'}\rangle \longmapsto \frac{1}{\sqrt{p}} \sum_{b\in F_p} w^{ab} |S_b^{m-d'-1}\rangle.$$

Proof. Let us denote the final state of the transformation by $|\alpha\rangle$:

$$|S_a^{d'}\rangle = \frac{1}{\sqrt{p^d}} \sum_{f\in V, f(0)=a} |f(\alpha_1),\dots,f(\alpha_m)\rangle \tag{4.8}$$

$$\longmapsto |\alpha\rangle = \frac{1}{\sqrt{p^{d+m}}} \sum_{b_1,b_2,\dots,b_m\in F} \sum_{f\in V, f(0)=a} w^{\sum_{l=1}^m c_l f(\alpha_l) b_l} |b_1,\dots,b_m\rangle.$$

For each string $b_1,\dots,b_m \in F_p$, associate the unique polynomial $b(x)$ which satisfies $b(\alpha_l)=b_l$ and has degree $\deg(b)\le m-1$. The exponent of w in (4.8) can be written in a much simpler form when $b(x)$ is of degree $\deg(b)\le m-d'-1$. For such $b(x)$, the polynomial $h(x)=b(x)f(x)$ is of degree $\deg(h)\le m-1$ so

$$\sum_{l=1}^m c_l f(\alpha_l) b(\alpha_l) = \sum_{l=1}^m c_l h(\alpha_l) = h(0) = f(0)b(0). \tag{4.9}$$

Hence, the sum over all b with $\deg(b)\le m-d-1$ in (4.8) gives

$$\frac{1}{\sqrt{p^{d+m}}} \sum_{b_1,b_2,\dots,b_m\in F,\deg\ b(x)\le m-d-1} \sum_{f\in V, f(0)=a} w^{b(0)f(0)} |b_1,\dots,b_m\rangle \tag{4.10}$$

$$= \frac{1}{\sqrt{p^{m-d}}} \sum_{b_1,b_2,\dots,b_m\in F,\deg\ b(x)\le m-d-1} w^{b(0)a} |b_1,\dots,b_m\rangle$$

$$= \frac{1}{\sqrt{p}} \sum_{b\in F_p} w^{ab} \frac{1}{\sqrt{p^{m-d-1}}} \sum_{b_1,b_2,\dots,b_m\in F,\deg\ b(x)\le m-d-1, b(0)=b} |b_1,\dots,b_m\rangle$$

$$= \frac{1}{\sqrt{p}} \sum_{b\in F_p} w^{ab} |S_b^{m-d'-1}\rangle.$$

Now we claim that the sum over the rest of the b's must vanish. The reason is that the norm of the above vector is 1. Now $|\alpha\rangle$ can be written as a sum of two vectors: the contribution from b's with $\deg(b)\le m-d-1$ and that from the rest of the b's. The two are orthogonal, since different $|b\rangle$'s are orthogonal. Hence, the squared norm of $|\alpha\rangle$, which is 1 (because the operation is unitary and we started with a norm one vector), is the sum of the squared norms of the contribution of $\deg(b)\le m-d-1$, which is also 1, and the norm of the orthogonal vector. Thus, the norm of the sum over b's with $\deg(b) > m-d'-1$ must vanish. ∎

4.6. Degree reduction and degree increase.

DEFINITION 18. *A* degree reduction *is a procedure which takes a state* $|S_a^{d'}\rangle$ *and returns the state* $|S_a^d\rangle$*, for* $d' > d$*.* Degree increase *is defined similarly, except we require that* $d' < d$*.*

To construct these procedures, we need to have at our disposal a *zero-state preparation* procedure, namely, a fault-tolerant procedure which generates the state $|S_0^d\rangle$. In fact, we will need a zero-state preparation procedure also for the polynomial code of degree $2d = m-d-1$, so that we have the states $|S_0^{m-d-1}\rangle$ available, too. We

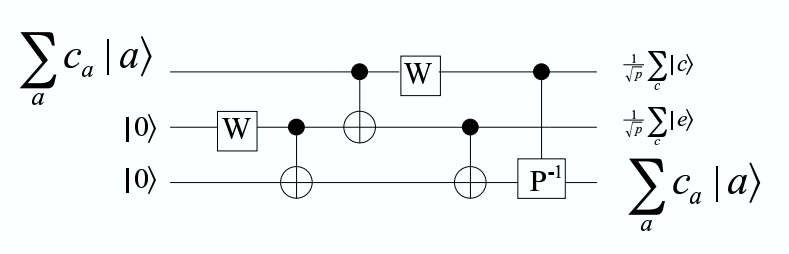


FIG. 4.1. *Teleportation of a general state of the topmost p-qudit to the bottommost one.*

will see how to perform these procedures in subsection 4.13, and for now we merely assume that we have them at our disposal.

The degree increase is derived by using teleportation. The idea is to apply teleportation from a state encoded with polynomials of degree d to a state encoded with polynomials of degree $2d$. We first recall how teleportation works for p-qudits (see Figure 4.1) and then consider its encoded version.

The first qudit is in a general state $\sum_{a\in F_p} c_a|a\rangle$, and the other two qudits are in the state $|0\rangle$. The first step is transforming the last two qudits to the analogue of an EPR pair for F_p: We apply the generalized Fourier transform W on the second qudit and then a CNOT_p^{-1} from the second qudit to the third one. This gives the state

$$\frac{1}{\sqrt{p}} \sum_{a,b\in F_p} c_a|a\rangle|b\rangle|-b\rangle.$$

We then apply CNOT_p from the first qudit to the second, which gives

$$\frac{1}{\sqrt{p}} \sum_{a,b\in F_p} c_a|a\rangle|b+a\rangle|-b\rangle.$$

Next, we apply a generalized Fourier transform on the first qudit to get

$$\frac{1}{p} \sum_{a,b,c\in F_p} c_a w^{ac}|c\rangle|b+a\rangle|-b\rangle.$$

We then apply CNOT_p from the second qudit to the third:

$$\frac{1}{p} \sum_{a,b,c\in F_p} c_a w^{ac}|c\rangle|b+a\rangle|a\rangle.$$

Finally, we apply a generalized controlled phase of order -1 from the first qudit to the third, to get

$$\frac{1}{p} \sum_{a,b,c\in F_p} c_a|c\rangle|b+a\rangle|a\rangle = \frac{1}{p} \sum_{a,e,c\in F_p} c_a|c\rangle|e\rangle|a\rangle = \frac{1}{\sqrt{p}} \sum_{c\in F_p} \otimes \frac{1}{\sqrt{p}} \sum_{e\in F_p} \otimes|c\rangle|e\rangle \otimes \sum_a c_a|a\rangle,$$

which is the desired teleportation.

The degree increase is basically an encoding of the above circuit, by using the correct choice of degrees for each block. We work on three blocks of qudits. The first

block is a general state of one qudit encoded by using polynomial codes of degree d. The second block is initially in the state $|S_0^d\rangle$, and the last block is in the state $|S_0^{2d}\rangle$. All of the gates in the teleportation are now applied transversally, except for the generalized Fourier transform which is applied semitransversally, as in Claim 2. We thus see that, after applying the first two gates in the teleportation circuit transversally, we get that the last two blocks are in the state

$$\frac{1}{\sqrt{p}} \sum_b |S_b^{2d}\rangle |S_{-b}^{2d}\rangle.$$

The next gate, which is the encoded CNOT_p, is now applied from a word in the code of degree d to that of degree $2d$. It is easy to check that this does not matter to the correctness of this gate. After the next step, namely, the semitransversal generalized Fourier transform, Claim 2 implies that all blocks are encoded by using degree $2d$ polynomials, and so the correctness follows from the correctness of the teleportation circuit.

The construction we have shown for degree increase does not work for degree reduction, since, for example, the first CNOT_p does not work correctly if the target state is of a smaller degree than that of the control state. To bypass this problem, we can instead perform a degree increase in the dual code. This is done as follows. We start with a state of degree $2d$. We first apply the semitransversal generalized Fourier transform (4.7). This takes us to the dual code (of degree d), by Claim 2. We now apply a degree increase, to get the same state encoded by using polynomials of degree $2d$. Finally, we apply the reverse of the transformation in (4.7), which achieves the desired result, again, by Claim 2.

4.7. Fault-tolerant Fourier transform. To achieve the generalized Fourier transform of order 1, we simply apply the transformation of (4.7). By Claim 2, this yields the desired state but in the wrong degree. Applying a degree reduction solves the problem. To achieve a generalized Fourier transform $W(c)$ of order $c \neq 1$, we use w^c instead of w everywhere in subsection 4.5.

4.8. Fault-tolerant generalized Toffoli and squaring. To apply the generalized Toffoli gate on $|S_a\rangle|S_b\rangle|S_c\rangle$, we first increase the degree of the third register to $2d$. We now apply the general Toffoli gate transversally on the m coordinates, which gives $|S_a\rangle|S_b\rangle|S_{ab+c}^{2d}\rangle$, as is easy to check. We now apply a degree reduction on the third register, and this achieves the desired result. The squaring gate is applied in exactly the same manner.

4.9. Remaining gates. The fault-tolerant version of the gate that adds a qubit in the state $|0\rangle$ is simply the zero-state preparation procedure. We will see how to perform this procedure in subsection 4.13. The fault-tolerant gate that discards a qudit can obviously be done transversally by discarding all of the qudits in the block. This completes our description of fault-tolerant procedures for the set $\mathcal{G}_1$.

4.10. Fault-tolerant error correction. Our construction of the error correction procedure is based on the simplest known quantum error correcting technique, namely, Steane's error correction [90]. Once again, we assume here that we have at our disposal a fault-tolerant zero-state preparation procedure, which we will show later.

The error correction procedure is composed of two stages: The first stage detects and corrects dit-flips (faults of type B), by using classical error correction techniques for the code C_1. The second stage applies on each coordinate the generalized Fourier

transform (4.7). We then correct dit-flips by using classical error correction techniques for the code C_2. Finally, we rotate back by applying the inverse of the generalized Fourier transform on each coordinate. By subsections 3.5 and 3.6 this achieves the desired error correction. It is therefore sufficient to describe how to correct dit-flips fault-tolerantly.

Let us first describe one part of the error correction procedure. Generate an ancilla block in the state $|S_0^{2d}\rangle$, by using the fault-tolerant zero-state preparation procedure. Apply a generalized Fourier transform semitransversally, as in (4.7), on the ancilla block. Its state is now $\frac{1}{\sqrt{p}}\sum_{a\in F_p}|S_a^d\rangle$. Then apply CNOT_p transversally from the block we would like to correct to the ancilla blocks. Now observe that there is a classical circuit that, given any string from the ancilla block, can output the value of e_i, the ith dit of the error vector on the original block, as long as the number of faults is at most q, the number of errors that C_1 corrects. To see this, note that, for any $w \in C_2, w' \in C_1$, and e a word in F_p^m, the transversal CNOT_p takes

$$|w+e\rangle|w'\rangle \mapsto |w+e\rangle|w'+e\rangle,$$

and we can find e by finding the closest word in C_1 to the string $w'+e$.

To correct dit-flips, we repeat the above procedure m times: Generate m ancilla states, rotate them to the dual code, apply CNOT_p transversally from the computational block to each one of the ancilla blocks, and then apply on the ith ancilla block a circuit that outputs $p-r$, where B^r is the dit-flip that occurred on the ith coordinate. Finally, apply a CNOT_p from this output dit into the ith coordinate of the computation block.

By ignoring the zero-state preparation procedures in the above construction for now, it is easy to see that one fault in the above procedure can affect only one qudit in the final block.

4.11. Error correction which projects any word into the code. We would like to insert here one important modification, which is not necessary for the fault-tolerant error correction but will become crucial when applying the error corrections in the recursive scheme. This is the requirement that the error correction takes any word to some word in the code, regardless of the number of faults in the original state. This requirement applies, of course, only if during the error correction there are no faults.

Roughly, this is done by checking whether too many errors occurred and, if so, replacing the entire block by another block which is initialized in the state $|S_0\rangle$. However, we should be careful to keep the procedure fault-tolerant. We do this in the following way: Before starting the correction procedure, we generate another ancilla state $|S_0\rangle$, by using the state preparation procedure. When computing the value of e_i in the error correction procedure, we also compute whether the total number of faults is at most q and write the answer on another qudit. Let us call this an *indication bit*. The CNOT which checks if the ith dit is wrong and if so applies NOT on the i*th* qubit is replaced by a generalized Toffoli gate which takes as an input also the ith indication bit and also checks if the number of faults is at most q. We then swap the ith qudit with the ith qudit of the state $|S_0\rangle$, conditioned that the ith indication bit indicates that the number of faults is larger than q. Such a conditional swap can be achieved by a small circuit which uses only the classical gates from $\mathcal{G}_1$.

To see that this procedure indeed takes any word to some word in the code, observe that this is true if no fault occurs during the procedure itself. Since the procedure is performed fault-tolerantly, the final state will differ from a word in the code only in the qubits affected by an error.

4.12. More issues regarding ancillas. In subsection 4.3.5 we required that, in all of our procedures, the state of the ancilla qudits that are discarded at the end of a procedure is in tensor product with and independent of the state of the computer (in a noiseless situation). This requirement was imposed so that the ancilla states are in tensor product with the state of the computer even if this state is a superposition of the basis states and not only in a basis state like in our analysis. This requirement can be released in two cases: the zero-state preparation and the decoding procedures.

The zero-state preparation procedure is supposed to get as an output one basis state. Thus, we can release the above requirement about ancilla qudits and demand only that the ancilla state is in tensor product with the computational qubits at the end of the procedure. This requirement can also be released in the decoding procedure, since once we decode a state in our scheme, we do not use it any more. Hence, we should check only that it gives the correct answer when measured.

4.13. Fault-tolerant zero-state preparation. We need to show how to generate $|S_0\rangle$ with spread 1. By Definition 16, we need to check the propagation of errors only under the assumption that the total number of errors in the input string, plus the number of faults during the procedure, is at most q, the number of errors that the codes can correct. In this case, we can assume that the number of faults is at most q, since the zero-state preparation has no input. The construction is based on one basic design, which is essentially concatenated with itself, up to some modifications.

4.13.1. Zero-state preparation resilient to one fault. Assume for a moment that $q = 1$, so that we only have to make sure that the spread of the state preparation procedure is 1 if there is one fault in the procedure. In this situation, our state preparation procedure is the following construction. Let $\mathcal{Q}$ be some quantum circuit that generates $|S_0\rangle$ from $|0^m\rangle$, without any fault-tolerant requirement. We start with $5m$ blank qubits and apply $\mathcal{Q}$ on the first, second, third, fourth, and fifth m-tuples of qubits. We now apply a circuit that is very similar to the error correction circuit: It attempts to detect errors with respect to the code containing one word, namely, $|S_0\rangle$. To detect dit-flips in the first block, we apply CNOT_p transversally from the first block to the second one and then perform some computation on the qudits of the second block to assess the number of dit-flips. This time, we do not attempt to infer the exact error vector, or to correct the state, but just to decide whether we accept the state or not. To do this, we copy each dit in the block to m different dits. The circuit we apply on the second block gets as an input the ith copy of these dits, namely, a string in F_p^m, and outputs whether its distance from the code C_2 is more than 1 or not. The output bit is called the *dit-flips indication bit*. We perform m independent calculations to get m such indication bits.

We use the third block to detect phase-flips. We would like to follow a similar construction as for the detection of dit-flips. The difference is that we first rotate both the first block and the third block to the dual code, as in (4.7). After we apply CNOT_p from the first to the third block, we detect errors by using the third block, except that the error detection is done with respect to the code C_1 for polynomials with the codegree. Unfortunately, this scheme requires some modification. The problem is that if one fault occurred in the generation of the third block, such that after its rotation this block contains many phase-flips, these phase-flips will propagate through the CNOT_p gates to the first block but will not be detected. To prevent this, before we apply the above construction, we use the fourth block to detect for dit-flips of the third block. This is done just as the above dit-flip detection with the first and second blocks. This initial check creates m additional indication bits. We condition each of

the CNOT_p gates from the first to the third block on the relevant additional indication bit—hence instead of applying a transversal CNOT_p, we in fact apply generalized Toffoli gates transversally. After the application of the Toffoli gates, we rotate the first block back to the dual of the dual code, namely, to the original code. We now compute from the third block m phase-flip indication bits, independently, exactly as we have computed the dit-flip indication bits before.

In fact, we need to insert one last modification in this construction. We would like to be able to bound the total number of dit-flips and phase-flips, and not just each of them separately, so that we can bound the deviation. Hence, instead of the two separate indication bits, we actually calculate one indication bit, as follows. The circuit gets as an input two strings in F_p^m. It checks whether there exists one coordinate such that it suffices to change this coordinate in both strings, in some way (including leaving it as is), to put the first string in C_2 and the second string in C_1 (with the codegree). If there is no such coordinate, this bit is turned to 1.

Finally, we apply transversally the following three-qudit transformation, on the ith dit of the first block, the ith indication bit, and the ith dit of the fifth block. The transformation swaps the ith dits of the first and fifth blocks, conditioned that the indication bit is 1 (namely, more than one error was detected). This three-qudit transformation can be constructed from classical gates in $\mathcal{G}_1$.

CLAIM 3. *If exactly one fault in the above construction occurred, then the final state has at most one error.*

Proof. By subsections 3.5 and 3.3 we can treat the error as if it is a linear combination of dit-flips and phase-flips. Let us first consider the case of a fault occurring in one of the non-fault-tolerant circuits preparing the states $|S_0\rangle$. If the fault occurs in the fifth circuit, nothing happens, since no swap occurs. If the fault occurs in the first circuit, it could be that this fault propagated to more than one qudit in $|S_0\rangle$. If there is more than one coordinate in which either a dit-flip or a phase-flip occurred, the remainder of the circuit, which is fault-free, will detect it. Hence, all indication bits would indicate that the first and fifth blocks should be swapped. If the fault propagated to at most one qudit, this would not cause a swap, but this is OK since there will only be one error in the final state.

A similar argument applies if the fault occurs in the second block. If there is more than one faulty qudit in the $|S_0\rangle$ state of this block, this will result in swapping the first block with the correct state (the fifth block). If there is one error, this will not cause a swap but can only cause one error in the final state.

What about the third block? If the $|S_0\rangle$ state of the third block has more than one dit-flip, these will be detected in the dit-flip preliminary detection of the third block, and the conditioning on the additional indication dits will prevent them from propagating. Hence, we can assume that there is at most one dit-flip in this state. One such dit-flip can propagate from the third block to the first block through the CNOT_p gates, after the rotation, but can cause at most one error. Now let us add phase-flips. After the rotation of the third block, these transform to dit-flips. If there are at least 2 of them, a swap will occur and no error will be caused in the final block. If there is at most one such dit-flip, it has no affect at all since it does not propagate through the CNOT_p gates. Overall, we have seen that one fault of any kind in the third block can cause at most one error in the final block.

Similar arguments apply to show that one fault in the fourth block can cause at most one error. If the fourth $|S_0\rangle$ state has more than one dit-flip, nothing will happen due to the additional indication bits. But even if it has at most one dit-flip, this does not propagate through the CNOT_p gates from the third to the fourth block,

and so the dit-flips have no effect. As for the phase-flips, these can propagate to the third block, and the argument proceeds as in the case of the third block.

The next case is a fault that occurs during the CNOT_p and rotation gates. Since these are applied transversally, one fault can affect only exactly one qudit in each block. Let us see how these errors propagate through the remaining fault-free circuit. As for the propagation of a dit-flip, since one dit-flip cannot cause the indication bits to turn into 1, we have that all indications bits will be zero. As for a phase-flip, this does not propagate at all through the remaining gates. Hence, as no swap will occur, the final state will have at most one error.

A more subtle case is the case in which the fault occurred during the copying of the bits of the second (or third, or fourth) block. Suppose a fault occurred while copying the ith bit. It could be that the ith indication dit is flipped and/or had a phase flip, and, possibly, the ith dit in the second block is flipped, too, and/or suffers from a phase-flip. Let us first consider just the dit-flips and concentrate on a dit-flip in the copied dit itself. The problem is that in this case the remaining copied dits will all be wrong! Fortunately, this does not pose a problem. This affects exactly one dit in the input string of each circuit that computes an indication bit. Because of the fact that the indication bits turn to 1 only if they see more than one error, all indication dits (except, possibly, for the one in which the fault actually occurred) will still be 0. So in this case no swap occurs, except for possibly in one qudit, and the state, which was error-free to begin with, will have at most one error. An extra phase-flip in the same location of the dit-flip, if one occurred, does not change this analysis. A phase-flip in the copied qudit itself does not propagate via the CNOT_p gates to the target dit. Hence, by the end of the copying stage, the error will still be confined to the qudits where the fault had occurred, namely, one qudit in the second or third or fourth block, and one indication bit. So once again, at most one qudit will be swapped. Note that if just a phase-flip occurred in the indication bit, this does not affect the final state at all because it does not propagate through the SWAP gates.

Finally, a fault in the circuit calculating one of the indication bits, or a fault in one of the three-qudit circuits, can cause only one error since the remainder of the circuit is transversal. □

4.13.2. Zero-state preparation resilient to q faults. Let us now consider the case of $q > 1$. The above construction does not work any more, even for $q = 2$, since two faults can ruin completely both the first and the last block, which are constructed in a non-fault-tolerant way. Instead, we apply the above construction in a way that is sort of concatenated. For $q = 2$, we concatenate it once with itself, in the following way: We first apply the above construction five times, to get five $|S_0\rangle$ states—we call these states the first-level states. We then apply the above construction once more, except for two changes. The first change is that, instead of applying the original non-fault-tolerant circuit to construct the five $|S_0\rangle$ states, we use the five first-level states that we have generated. The second change is that we set the threshold of the indication bits in the second-level construction to be 2 instead of 1; namely, we only flip the indication bit to 1 if more than *two* errors were detected.

To increase q to any value, we simply apply the above concatenation q times, each time by using as input states the final states from the previous level of concatenation and increasing the threshold for the indication bits by 1. Overall, the scheme that allows for q faults, denoted by Q_q, uses 5^q $|S_0\rangle$ states.

CLAIM 4. *If $k \leq q$ faults occur during the application of Q_q, the final output block will have at most k errors.*

Proof. We prove this by induction. The case of $q = 1$ was essentially given in Claim 3, where it was shown that if $k = 1$, then the final block will have at most one error. It is obvious that if $k = 0$, the state will have no error. Now assume for q, and prove for $q + 1$. We divide this proof into two cases.

In the first case, in each one of the five q-level blocks that are used to generate the final block, there are at most q faults. In this case we can apply the induction. Let $x_1, \ldots, x_5$ be the number of faults occurring in the five circuits generating the q-level blocks, respectively. By induction, the number of errors in the final states is at most $x_1, \ldots, x_5$, respectively. If x is the number of faults occurring in the final part of the circuit, we have that $x_1 + \cdots + x_5 + x = k$. The next step in the circuit involves CNOT_p gates, which do not increase the number of errors in each block beyond k. The next step is the copying of each dit m times and computation of the indication bits. Recall that, at this step, the threshold for the indication bit to flip is for the number of errors it sees to be larger than $q + 1$. However, the calculation of each indication bit gets as an input two strings, such that the union of their errors is confined to at most $k \leq q + 1$ qudits, and so, unless the calculation itself involves a fault, the indication bit will remain 0. Hence, in this stage, too, one fault can propagate only to one error. The remainder of the circuit is transversal. This means that the total number of faults in the final $q + 1$-level block is at most k.

In the other case, $q + 1$ faults occurred during the generation of one of the five q-level blocks which are input to the final $(q + 1)$ level. In this case, the remainder of the circuit is fault-free. We consider five subcases: The faulty block is the first, second, third, fourth, or fifth $|S_0\rangle$s. Suppose first that it is the first block. Then either the final state of the faulty block has more than $q+1$ errors, and then the entire block will be swapped with the fifth block and the final state will be error-free, or the final state of the first block has $q + 1$ errors or less, in which case it will not be swapped, but still the number of errors is at most $k = q + 1$. A similar argument works if the faulty block is the second block. Suppose that there are x dit-flips, and suppose that y phase-flips have propagated to the first block. If $x + y > q + 1$, there will be a swap. Otherwise, this will cause at most $k = q + 1$ errors. Suppose that the faulty block is either the third or the fourth block. Observe that phase-flips before the rotation become dit-flips, and these do not propagate from the third block to the first block. Either there are more than $q + 1$ of these phase-flips, and they cause a SWAP, or they do not affect the final state. Hence, we need consider only the effect of the at most $q + 1$ dit-flips, which after the rotation become phase-flips, and can propagate to at most $q + 1$ errors in the final state. Finally, if the faulty block is the fifth block, the final state will not be swapped, so there will not be errors at all. ∎

This implies that the zero-state preparation procedure has spread 1.

4.14. A fault-tolerant decoding procedure. A decoding procedure applies

$$|S_a\rangle \longmapsto |A_a\rangle|\vec{a}\rangle, \tag{4.11}$$

where the state $|A_a\rangle$ is an ancillary state which depends on a. (We can discard this state at the end of the procedure; we will see that, whenever the decoding procedure is applied, the state encodes a well-defined logical bit a.) To apply this transformation, we compute a independently m times from the state $|S_a\rangle$. To do this, we add m^2 blank qubits and copy each qudit from $|S_a\rangle$ m times to m different blank qudits, by using m CNOT_p gates. We get m "copies" of $|S_a\rangle$. These are, of course, not real copies of $|S_a\rangle$, since they are entangled. However, each word in the classical code is copied m times. On each copy of the word we apply the quantum analogue of the

classical computation that computes what is the logical bit that the word encodes. The answer, which is a if no error occurred, is written on another blank qubit. For this computation we use the classical gates in $\mathcal{G}_1$. We might need some extra blank qudits as working space. The computation of a is done in the shortest way possible, regardless of whether it is fault-tolerant. An error in this computation can affect only the one copy of a which it computes. A fault in the first stage of copying the qubits m times can affect only one qubit in each of the copies, and, if the number of errors in S_a plus the number of faults in the first stage is smaller than the number of errors correctable by the code, these faults have no effect. One fault in the second stage of the procedure, during the computation of one of the a's, can affect only the correctness of that a.

Remark 1. We remark regarding the significantly simpler version of the error correction, decoding, and zero-state preparation procedures, in case measurements and classical computation are allowed.

In the error correction procedure, no repetition is required, and one ancilla state is needed for either dit-flips or phase-flips. This saved a factor of m in the construction.

The simplification is most notable in the case of the zero-state preparation procedure. In this case, we can omit the entire concatenated construction. We use the circuit as in the case of $q = 1$ and simplify it further as follows. Observe that, after the CNOT_p gates from the computational block to the ancilla blocks it suffices to measure every qudit in the ancilla blocks and perform the calculation of the error vector classically, without copying the dits first. Hence, the construction is transversal except for the classical computation which does not introduce errors, and the spread is 1.

Finally, in the decoding procedure we can omit the copying of the dits.

5. Fault-tolerant gates for CSS codes over F_2. In this section we give an alternative construction to the one using polynomial codes. Here we use a restricted class of CSS codes, which were used by Shor in [84]. Shor showed how to apply a universal set of gates (which we denote by $\mathcal{G}_2$) on states encoded by such codes. It turns out that almost all of the gates in $\mathcal{G}_2$ can be applied transversally. The only complicated procedure is the Toffoli gate. We repeat the constructions of Shor for the simple gates for completeness. As for the Toffoli gate, we essentially use Shor's construction, except that we adopt it to our framework in which no measurements are allowed, which requires some extra work.

5.1. Some restrictions on the CSS codes. In the following, we will put some restrictions on the CSS codes which we will use. This is done in order to be able to apply several gates transversally, as will be seen shortly. We start with C_1, a punctured doubly even self-dual code, and set $C_2 = C_1^{\perp}$. A punctured self-dual code is a code which is obtained from a self-dual code C' (namely, a code for which $C' = C'^{\perp}$) by deleting one coordinate. We also require that C' is a doubly even code; i.e., the weight of each word in the code is divisible by 4. To see that in this case $C_2 = C_1^{\perp} \subset C_1$, as in the definitions of CSS codes, observe that if $v \perp C_1$, then $v0 \perp C'$, so $v0 \in C'^{\perp} = C'$, so $v \in C_1$. We will denote that $C_1 = C$ and $C_2 = C^{\perp}$.

We now claim that there are only two cosets of $C^{\perp}$ in C. If the length of C is m, then $\dim(C'^{\perp}) = \dim(C') = (m+1)/2$. Hence $\dim(C) = (m+1)/2$ as well, since $|C| = |C'|$, because no two words in C' are mapped to the same word in C by the punctuation. Hence, $\dim(C^{\perp}) = m - (m+1)/2 = (m-1)/2$, and so $\dim(C) - \dim(C^{\perp}) = 1$. Observe that C includes the all-one vector: $\vec{1} \in C$. This is true since $1^{m+1} \in C'^{\perp}$, because C' is even, and since C' is self-dual, $1^{m+1} \in C'$. Hence $1^m \in C$. Observe also that the length m must be odd due to the above considerations.

This implies that $\vec{1} \notin C^{\perp}$. The two code words in our quantum code can thus be written as

$$|S_{\vec{0}}\rangle = \sum_{w \in C^{\perp}} |w\rangle, \tag{5.1}$$
$$|S_{\vec{1}}\rangle = \sum_{w \in C^{\perp}} |w + \vec{1}\rangle.$$

$|S_{\vec{0}}\rangle$ and $|S_{\vec{1}}\rangle$ can be thought of as encoding $|0\rangle$ and $|1\rangle$, respectively. We will make use of the fact that $\vec{a} \cdot \vec{b} \bmod 2 = ab$, for $a, b \in 0, 1$, and that $\vec{a} + \vec{b} = \overrightarrow{a+b}$. This fact allows us to shift easily between operations on vectors and operations on the bits they represent. We will therefore usually omit the vectors in the notations of $|S_{\vec{0}}\rangle$ and $|S_{\vec{1}}\rangle$, unless there is ambiguity.

5.2. The set of gates for CSS codes $\mathcal{G}_2$. We work with the following set of gates, which we denote by $\mathcal{G}_2$:

1. NOT: $|a\rangle \longmapsto |1 - a\rangle$.
2. CNOT: $|a, b\rangle \longmapsto |a, a + b\rangle$.
3. Phase: $|a\rangle \longmapsto i^a |a\rangle$.
4. SWAP: $|a\rangle|b\rangle \longmapsto |b\rangle|a\rangle$.
5. Controlled phase: $|a\rangle|b\rangle \longmapsto (-1)^{ab}|a\rangle|b\rangle$.
6. Hadamard: $|a\rangle \longmapsto \frac{1}{\sqrt{2}} \sum_b (-1)^{ab} |b\rangle$.
7. Toffoli gate: $|a, b, c\rangle \longmapsto |a, b, c + ab\rangle$.
8. Adding a qubit in the state $|0\rangle$.
9. Discarding a qubit.

All of the additions and multiplications above are in F_2 (i.e., modulo 2). Section 6 shows that this set of gates is universal. We remark here once again that, like the set $\mathcal{G}_1$, the set $\mathcal{G}_2$ is by no means a minimal universal set of gates, but the fault-tolerant procedures become simpler and shorter if we have a larger repertoire of fault-tolerant gates that we can use. The following theorem shows how to perform gates from $\mathcal{G}_2$ on encoded states fault-tolerantly.

THEOREM 6. *Fix a CSS code that satisfies the restrictions of subsection* 5.1. *For any gate in $\mathcal{G}_2$, there exists a fault-tolerant procedure with spread* 1 *that computes the gate on states encoded by the code. There exist also fault-tolerant error correction, decoding, and zero-state preparation procedures for this code, which all have spread* 1. *Moreover, all of these procedures use only gates from $\mathcal{G}_2$ and in particular do not use measurements.*

The constructions of the error correcting, decoding, and zero-state preparation procedures follows exactly the construction of section 4. It remains to show the constructions of the computational gates.

5.3. Transversal gates. In the following we omit overall normalization factors, since all vectors are known to be unit vectors. We also set $a, b \in C/C^{\perp}$. We start with the NOT gate:

$$\begin{aligned} |S_a\rangle &= \sum_{w \in C^{\perp}} |a_1 + w_1\rangle \otimes \cdots \otimes |a_m + w_m\rangle \\ &\longmapsto \sum_{w \in C^{\perp}} |a_1 + 1 + w_1\rangle \otimes \cdots \otimes |a_m + 1 + w_m\rangle = |S_{a+\vec{1}}\rangle. \end{aligned} \tag{5.2}$$

For CNOT,

(5.3)
$$\begin{aligned}|S_a\rangle|S_b\rangle &= \sum_{w\in C^\perp} |a_1+w_1\rangle\otimes\cdots\otimes|a_m+w_m\rangle \sum_{w'\in C^\perp} |b_1+w'_1\rangle\otimes\cdots\otimes|b_m+w'_m\rangle\\ &\longmapsto \sum_{w\in C^\perp} |a_1+w_1\rangle\otimes\cdots\otimes|a_m+w_m\rangle \sum_{w'\in C^\perp} |a_1+b_1+w_1+w'_1\rangle\otimes\cdots\\ &\quad\otimes|a_m+b_m+w_m+w'_m\rangle\\ &= |S_a\rangle|S_{a+b}\rangle,\end{aligned}$$

where the last equality follows from the fact that $C^\perp$ is a linear subspace, and therefore summing over $w+w'$ for a fixed w in the code is equivalent to summing over w'. For the phase gate, apply the gate $|a\rangle \longmapsto i^a|a\rangle$ three times on each coordinate. This gives

(5.4)
$$|S_a\rangle = \sum_{w\in C^\perp} |a_1+w_1\rangle\otimes\cdots\otimes|a_m+w_m\rangle \longmapsto \sum_{w\in C^\perp} i^{3(\sum_k a_k+w_k)}|a_1+w_1\rangle\otimes\cdots\otimes|a_m+w_m\rangle.$$

This is the desired result, because of the following fact. Since C is obtained from a doubly even self-dual code C' by deleting one coordinate, it is easy to see that $C^\perp \subset C$ are exactly those words in which the deleted coordinate was 0. Thus, all words in $C^\perp$ have weight which is divisible by 4, and all words in C but not in $C^\perp$ have weight which is 3 mod 4.

For the encoded controlled phase gate,

$$|S_{\vec a}\rangle|S_{\vec b}\rangle = \sum_{w,w'\in C^\perp} |\vec a+w\rangle|\vec b+w'\rangle \longmapsto \sum_{w,w'\in C^\perp} (-1)^{(\vec a+w)\cdot(\vec b+w')}|\vec a+w\rangle|\vec b+w'\rangle. \tag{5.5}$$

Now $(\vec a+w)\cdot(\vec b+w') = \vec a\cdot\vec b \bmod 2$. This is true since $\vec a\in C$ and $w'\in C^\perp$ so $\vec a\cdot w' = 0 \bmod 2$, and likewise $w\cdot\vec b = w\cdot w' = 0 \bmod 2$. Moreover, $\vec a\cdot\vec b \bmod 2$ is equal to ab, and so the final state is indeed the desired state. Finally, for the Hadamard gate,

$$\begin{aligned}|S_a\rangle &= \sum_{w\in C^\perp} |a_1+w_1\rangle\otimes\cdots\otimes|a_m+w_m\rangle \longmapsto \sum_{x\in F_2^m}\sum_{w\in C^\perp} (-1)^{(a+w)\cdot x}|x\rangle\\ &= \sum_{x\in C}(-1)^{a\cdot x}|x\rangle = \sum_{b\in C/C^\perp}\sum_{w\in C^\perp}(-1)^{a\cdot(b+w)}|b+w\rangle = \sum_b (-1)^{a\cdot b}|S_b\rangle.\end{aligned} \tag{5.6}$$

It remains to show how to apply the Toffoli gate on encoded states.

5.4. The fault-tolerant Toffoli gate. To apply the Toffoli gate, we roughly follow Shor's scheme, where we construct an ancillary state denoted by $|A_0\rangle$ and use it to obtain the Toffoli gate. The main difference from Shor's scheme is in the construction of the ancillary state, which is not completely straightforward if one wants to avoid using measurements.

Our constructions of encoded gates for polynomial codes, which we discussed in section 4, is much simpler than the construction we get for the Toffoli gate. First, for the Toffoli gate we need not only zero-state preparations but also the preparation of a three-block ancilla state. Second, the subsequent operations given the ancilla state are not transversal as in the case of the polynomial codes. The work of [48]

shows how to encode a universal set of gates by procedures which consist of ancilla state preparation followed by transversal operations (namely, teleportation). This, however, is done by assuming measurements and classical computation.

5.4.1. Construction of the ancilla state $|A_0\rangle$. Define the ancilla state

$$|A_0\rangle = \frac{1}{2}\sum_{a,b}|S_aS_bS_{ab}\rangle. \tag{5.7}$$

We also define

$$|A_1\rangle = \frac{1}{2}\sum_{a,b}|S_aS_bS_{1-ab}\rangle, \tag{5.8}$$

which is easily convertible to $|A_0\rangle$ by applying NOT on the third block. Note that the state

$$\frac{1}{\sqrt{2}}(|A_0\rangle + |A_1\rangle) = \frac{1}{2\sqrt{2}}(|S_0\rangle + |S_1\rangle)(|S_0\rangle + |S_1\rangle)(|S_0\rangle + |S_1\rangle) \tag{5.9}$$

is easy to construct by preparing three zero states $|S_0\rangle$ and then applying an encoded Hadamard on each block. In order to convert this state to $|A_0\rangle$, we use states of the form

$$|S_{cat}\rangle = \frac{1}{\sqrt{2}}(|S_0\rangle^m + |S_1\rangle^m), \tag{5.10}$$

which we call encoded cat states. An encoded cat state can be achieved by applying m zero-state preparation procedures to get $|S_0\rangle^m$, followed by an encoded Hadamard on the first block, and then CNOT gates from the first block to all other blocks.

We will also make use of the transformation

$$|S_a\rangle^m|A_b\rangle \longmapsto (-1)^{ab}|S_a\rangle^m|A_b\rangle \tag{5.11}$$

for bits a, b. The transformation (5.11) is performed by applying

$$|S_a\rangle|b\rangle|c\rangle|d\rangle \longmapsto (-1)^{a(bc+d)}|S_a\rangle|b\rangle|c\rangle|d\rangle \tag{5.12}$$

on the ith block in the encoded cat state and the ith bit in each of the three blocks of $|A_0\rangle + |A_1\rangle$. By applying this transformation for $1 \le i \le m$ we get

$$|S_a\rangle^m|S_b\rangle|S_c\rangle|S_d\rangle \longmapsto (-1)^{a(bc+d)}|S_a\rangle^m|S_b\rangle|S_c\rangle|S_d\rangle, \tag{5.13}$$

which is exactly the desired transformation of (5.11). Note that transformation (5.12) need not be fault-tolerant. We do not care if one fault ruins the entire block $|S_a\rangle$.

Let us start with the state

$$\frac{1}{\sqrt{2^{r+1}}}(|S_0\rangle^m + |S_1\rangle^m)^r(|A_0\rangle + |A_1\rangle), \tag{5.14}$$

where r will be chosen soon. Now apply transformation (5.11) between each one of the r encoded cat states and the last register, and between every two subsequent applications of transformation (5.11), perform an error correction on the last register.

The reason for the error corrections will become clear shortly. The resulting state would then be

$$\frac{1}{\sqrt{2^{r+1}}}(|S_0\rangle^m + |S_1\rangle^m)^r|A_0\rangle + \frac{1}{\sqrt{2^{r+1}}}(|S_0\rangle^m - |S_1\rangle^m)^r|A_1\rangle. \tag{5.15}$$

We would now like to "measure" (without any measurement) the r cat states, in order to decide (essentially, by taking majority over the r "measurement" results) whether the cat states are in their plus or minus states. If the answer is "plus," we would know that the state of the final three registers is $|A_0\rangle$, the desired state, and if "minus," we would know it is $|A_1\rangle$, in which case we could apply NOT on the third block to get $|A_0\rangle$.

We can now explain why the error corrections were added in between the different applications of transformation (5.11) that led to the state of (5.15): Transformation (5.11) is derived by m applications of transformation (5.13). Hence, a single error in the last register can cause a flip of sign in the encoded cat state, from "plus" to "minus" or vice versa. If this single error is not corrected, the signs will be wrong in all of the remaining cat states, too.

To apply the measurement of the sign of one encoded cat state in a fault-tolerant manner, we observe that applying encoded Hadamard gates on all m blocks in one encoded cat state results in

$$\begin{aligned}
&\frac{1}{\sqrt{2}}(|S_0\rangle^m + |S_1\rangle^m) \longmapsto \frac{1}{\sqrt{2^{m-1}}} \sum_{i=0, i\cdot\vec{1}=0}^{2^m} |S_{i_1}\rangle|S_{i_2}\rangle \dots |S_{i_m}\rangle, \\
&\frac{1}{\sqrt{2}}(|S_0\rangle^m - |S_1\rangle^m) \longmapsto \frac{1}{\sqrt{2^{m-1}}} \sum_{i=0, i\cdot\vec{1}=1}^{2^m} |S_{i_1}\rangle|S_{i_2}\rangle \dots |S_{i_m}\rangle.
\end{aligned} \tag{5.16}$$

This means that the answer plus or minus is determined by the parity of the strings in the sum. To compute the parity fault-tolerantly, we apply a fault-tolerant decoding procedure on each block, to get a string of m (ideally equal) bits. We then compute the parity transversally: We consider all of the first bits in the blocks (there are m of them) and compute their parity, by using a quantum circuit made of Toffoli, CNOT, and NOT gates. Likewise, we consider all of the second bits and compute their parity, and so on. For each encoded cat state, we therefore get m parity bits. We compare the parity bits of the different encoded cat states transversally by applying m majority votes of the form:

$$|a_1, a_2, \dots, a_r, b\rangle \longmapsto |a_1, a_2, \dots, a_r, b + maj(a_i)\rangle. \tag{5.17}$$

Each such majority vote can be constructed once again from Toffoli, CNOT, and NOT gates, since they are universal for classical computations. Finally, we apply CNOT transversally from the result of the majority vote to the qubits in the third block of $|A_0\rangle$.

It is left to see that the spread of this procedure is 1. We will consider errors at different stages of the procedure. The construction of an encoded cat state allows one fault to propagate to one error since it is composed of the state preparation procedure and CNOTs applied transversally. This is true also for the construction of the state $\frac{1}{\sqrt{2}}(|A_0\rangle + |A_1\rangle)$ because we used only the fault-tolerant state preparation procedure and the transversal Hadamard gate. A more subtle consideration is required for

transformation (5.11). An error during this transformation can cause a whole block in one of the cat states in the state (5.13) to be affected, together with one qubit in each one of the last three blocks. However, one such block can ruin the parity bits of only one encoded cat state. As long as the number of faulty parity bits is less than $r/2$, the majority vote will still give the correct parity. An error in the parity computations or in the majority vote cannot affect more than one qubit in $|A_0\rangle$, since they are done transversally.

We choose r to be $2k_0+1$, where $k_0 = \lfloor q/2 \rfloor$. The reason is that, in our proof, we will distinguish between cases with at most k_0 faults in a rectangle, which we treat as the good case, and other cases which will be shown to be rare. Hence, we would like to be able to tolerate $\lfloor q/2 \rfloor$ faults in a rectangle. Choosing $r = 2k_0 + 1$ will do the trick.

5.4.2. Construction of Toffoli gate given $|A_0\rangle$. The construction of the Toffoli gate given $|A\rangle$ follows Shor's scheme almost exactly, with minor changes due to the fact that measurements are replaced by CNOT gates to additional blank qubits. Also, classical conditioning on the results of the measurements is replaced by unitary gates on the computation qubits and extra blank qubits carrying the results of the measurements. Here is how this is done. We will first generate "half" a Toffoli gate:

$$|S_a\rangle|S_b\rangle|S_0\rangle \longmapsto |S_a\rangle|S_b\rangle|S_{ab}\rangle. \tag{5.18}$$

A Toffoli gate can be generated from transformation (5.18) in the following way. We start with the three blocks on which we want to apply Toffoli: $|S_a\rangle|S_b\rangle|S_c\rangle$. Then we generate $|S_0\rangle$ on an extra block, by using the zero-state preparation procedure. We then apply transformation (5.18) on the first two blocks and the newly generated $|S_0\rangle$. Then we apply an encoded CNOT from our original third block $|S_c\rangle$ to the new block. We finally apply an encoded Hadamard on the original third block. This gives the overall transformation:

$$|S_a\rangle|S_b\rangle|S_c\rangle|S_0\rangle \longmapsto \frac{1}{\sqrt{2}}|S_a S_b S_{c+ab}\rangle(|S_0\rangle + (-1)^c|S_1\rangle). \tag{5.19}$$

Note that if the fourth block was not there, we would be done, because the operation on the first three blocks is exactly the Toffoli gate. We next decode the fourth block. Hence, we would like to apply the operation

$$|S_a\rangle|S_b\rangle|S_c\rangle \longmapsto (-1)^{ab+c}|S_a\rangle|S_b\rangle|S_c\rangle \tag{5.20}$$

on the first three blocks, conditioned that the decoded qubits are 1.

This can be applied transversally, in the following way. First, apply a controlled phase on $|S_c\rangle$ and the decoded qubits, in a transversal manner. This will give the factor $(-1)^c$. To apply $|S_a\rangle|S_b\rangle \longmapsto (-1)^{ab}|S_a\rangle|S_b\rangle$ conditioned on the decoded qubits, apply transversally the operation: $|a\rangle|b\rangle|d\rangle \longmapsto (-1)^{abd}|a\rangle|b\rangle|d\rangle$, where $|d\rangle$ is a decoded qubit. This achieves the correct transformation since we know that the controlled phase can be applied transversally. This operation can be applied by adding a blank qubit $|a\rangle|b\rangle|0\rangle|d\rangle$ and applying a Toffoli gate on the first three qubits, followed by a controlled phase on the last two qubits, and then by a Toffoli gate again on the first three qubits.

It is left to show how to construct transformation (5.18) on two blocks B_1 and B_2. This is done by generating the ancilla state $|A_0\rangle$ as before. Now apply an encoded

CNOT from the first block of $|A_0\rangle$ to the first block B_1 and from the second block of $|A\rangle$ to the second block B_2. This achieves the transformation

$$|S_c\rangle|S_d\rangle|A_0\rangle \longmapsto \sum_{a,b} |S_{a+c}\rangle|S_{b+d}\rangle|S_a\rangle|S_b\rangle|S_{ab}\rangle. \tag{5.21}$$

We note that the last three blocks have a strong connection to the Toffoli gate. More precisely, we note that projecting the above sum on the subspace where the first two blocks are $|S_0\rangle|S_0\rangle$ implies that $a = c, b = d$, and the gate which is achieved on the last three blocks is exactly the desired Toffoli gate. If the first two blocks are the state $|S_0\rangle|S_1\rangle$, this implies that $a = c, b = \neg d$, and so the gate which is achieved is the encoded version of $|c, d\rangle \mapsto |c, \neg d, c\neg d\rangle$. Similarly for the two other possibilities we get the encoded versions of $|c, d\rangle \mapsto |\neg c, d, \neg cd\rangle$ and $|c, d\rangle \mapsto |\neg c, \neg d, \neg c\neg d\rangle$, respectively.

To adjust for these deviations from the Toffoli gate, we apply decoding procedures to the first two blocks and then apply the following corrections transversally on the five blocks:

$$\begin{aligned}
&|0,0\rangle|a,b,c\rangle \longmapsto |0,0\rangle|a,b,c\rangle,\\
&|0,1\rangle|a,b,c\rangle \longmapsto |0,1\rangle\mathsf{NOT}(2)\mathsf{CNOT}(1,3)|a,b,c\rangle,\\
&|1,0\rangle|a,b,c\rangle \longmapsto |1,0\rangle\mathsf{NOT}(1)\mathsf{CNOT}(2,3)|a,b,c\rangle,\\
&|1,1\rangle|a,b,c\rangle \longmapsto |1,1\rangle\mathsf{NOT}(1)\mathsf{NOT}(2)\mathsf{NOT}(3)\mathsf{CNOT}(1,3)\mathsf{CNOT}(2,3)|a,b,c\rangle.
\end{aligned} \tag{5.22}$$

It is easy to check that this achieves the desired corrections. The transformation in (5.22) is a reversible transformation on five qubits and can therefore be constructed by a constant number of classical gates from the set $\mathcal{G}_2$.

The spread of this procedure is 1, since we use the decoding procedure which has spread 1 and everything else is done transversally.

6. Universality of the sets of gates $\mathcal{G}_1$ and $\mathcal{G}_2$. In this section we prove the universality of the sets of gates we use. Roughly, a set of gates is said to be universal if the subgroup generated by the set of gates is dense in the group of unitary operations on n qubits $U(2^n)$ (for p-qudits the group is $U(p^n)$). A beautiful theorem by Kitaev [50, 52] and Solovay and Yao [85] implies that if a set $\mathcal{G}$ is universal, then any quantum circuit that uses arbitrary gates with constant fan-in can be replaced by one that uses only gates from $\mathcal{G}$, such that the overhead in time and space of the new circuit is only polylogarithmic. This theorem gives meaning to the notion of universality, since it implies that restricting the computation to a universal gate set implies only polylogarithmic overhead, and thus one can regard such a set as sufficient for quantum computation.

Our proof of universality of the set $\mathcal{G}_2$, used for CSS codes, is a simple reduction to a set of gates shown to be universal by Kitaev [50]. A similar result was achieved independently by Boykin et al. [26]. The proof that the set of gates $\mathcal{G}_1$ used for polynomial codes is universal is much more complicated. It is based on geometrical arguments on the special unitary group $SU(n)$, together with some basic facts from the theory of finite fields.

We start with a detailed discussion of the notion of universality. We then proceed to prove some geometrical lemmas that we will use in our universality proofs, and, finally, we prove the universality of the gate sets $\mathcal{G}_1$ and $\mathcal{G}_2$.

6.1. Universal sets of gates—basic results.

DEFINITION 19. *Let $p \geq 2$. A set of gates $\mathcal{G}$ on $k \geq 2$ p-qudits is said to be universal if it is closed under inversion and $\mathcal{G} \cup \{e^{i2\pi\theta} I\}_{real\ \theta}$ generates a dense subset in $U(p^k)$.*

The inclusion of the scalars of the form $e^{2\pi i\theta}$ (which are of absolute value 1 and are sometimes called *phases*) does not have any physical effect, since multiplying a gate by such a phase does not change the resulting density matrix.

The fact that this definition indeed captures the notion of universality is summarized by the following theorem.

THEOREM 7. *Consider $\epsilon > 0$, a quantum circuit Q that uses arbitrary gates (of constant fan-in), and a universal set of gates $\mathcal{G}$. The circuit Q can be translated to a circuit Q_ϵ that uses only gates from $\mathcal{G}$ such that the following three conditions hold:*

1. *Q_ϵ computes a function that approximates the function computed by Q to within total variation distance ϵ.*
2. *Q_ϵ is only polylogarithmically larger and deeper than Q.*
3. *The description of Q_ϵ can be efficiently computed given the description of Q.*

The proof of this theorem is well known for the case of qubits, and we extend it here for the case of p-qudits. The proof is based on two results. The first result, known as the Solovay–Kitaev theorem, states that "density implies efficiency": If a set operating on some Hilbert space is universal, then it can be used to approximate any unitary matrix on the same space exponentially rapidly. Moreover, the sequence of gates from $\mathcal{G}$ that achieves the approximation can be found efficiently.

THEOREM 8 (see Kitaev [50, 52] and Solovay [85]). *Let $\mathcal{G}$ be a universal set of gates over $U(p^k)$ for some integers $p, k \geq 2$. Then there exists a Turing machine A that, given a matrix $M \in U(p^k)$ and $\epsilon > 0$, outputs a sequence of gates $g_1, \ldots, g_\ell \in \mathcal{G}$ such that $\|M - g_\ell \cdot g_{\ell-1} \ldots g_1\| < \epsilon$, and both ℓ and the running time of A are polynomial in $\log(1/\epsilon)$.*

The proof of this beautiful and fundamental theorem uses Lie groups and Lie algebras and is beyond the scope of this paper.

To complete the proof of Theorem 7 we need another fact: An operation on any number of qubits (qudits) can be achieved by using gates that operate on only two qubits (qudits). This was proved for the case of qubits by DiVincenzo [40], a proof which was simplified by Barenco et al. [19]. We give here a proof for the general case of qudits, by using similar ideas to those used by Deutsch [39] and Barenco et al. [19].

THEOREM 9. *Let $\mathcal{G}$ be a universal set of gates on $k \geq 2$ p-qudits. Consider the Hilbert space of $m > k$ p-qudits. Then the set of gates achieved by extending the gates in $\mathcal{G}$ to operate on m p-qudits generates a dense subset of $U(p^m)$.*

Proof. We define a generalized Toffoli gate on m qudits $T_m(Q)$ to be a gate which applies Q on the mth qudit conditioned that the first $m-1$ qudits are in the state $p-1$ (see Figure 6.1).

The conditioned Q can be applied on the rth qudit, instead of the mth one, in which case we denote the gate by $T_{m,r}(Q)$.

CLAIM 5. *The set of gates $T_{m,r}(Q)$ can be approximated to within an arbitrary accuracy by using the extensions of gates from G to m qudits.*

Proof. We first show in Figure 6.2 an explicit sequence of generalized Toffoli gates on $m-1$ qudits, $T_{m-1}(Q)$'s, which achieves $T_m(Q)$.

We denote that $V = Q^{\frac{1}{p}}$, and $\oplus$ denotes here the $\mathsf{NOT}_p = \mathsf{NOT}_p(1)$ operation. It is easy to check that the above circuit indeed gives the desired controlled Q, by considering what happens to the basis states, in two cases: All first $m-2$ qudits

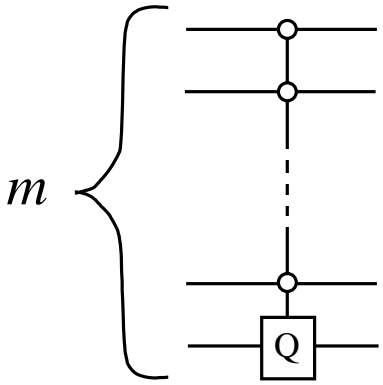


FIG. 6.1. *The controlled operation in F_2 is replaced here by conditioning the application of Q on the fact that the first $m-1$ qudits are in the state $p-1$. Note the difference from the usual conditioning, which for the case of $m=2$ would give CNOT_p. To distinguish it from the standard kind of conditioning, we denote this controlling operation by an empty circle.*

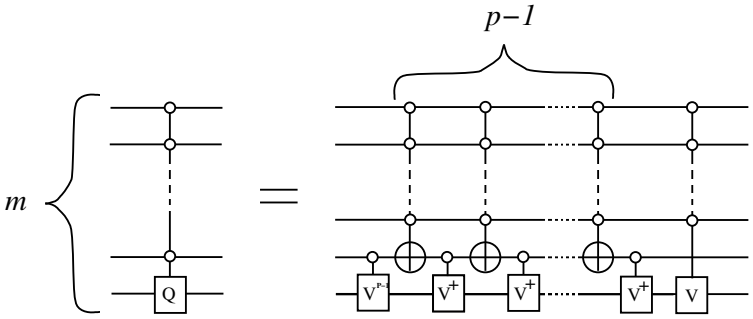


FIG. 6.2. *The gate $T_m(Q)$ can be written as a sequence of gates of the form $T_{m-1}(Q)$. After the first controlled V^{p-1}, we apply $T_{m-1}(\mathsf{NOT}_p)$, followed by a controlled $V^\dagger$, and repeat this pair of gates $p-1$ times. At the end we apply V controlled on the first $p-2$ qudits.*

are equal to $p-1$, or not. By using the above scheme recursively we can construct a circuit which uses k-qudit gates of the form of $T_k(Q)$ and achieves $T_m(Q)$ for any $m > k$ and any one-qudit Q. Since the set $\mathcal{G}$ is universal, the gates of the form $T_k(Q)$ can be approximated. Note that the recursion starts with two-qudit gates, which is the reason why we require $k \geq 2$. The construction is similar for $T_{m,r}(Q)$. □

The gate $T_m(Q)$ can be seen as applying Q on the subspace spanned by the last p basis vectors while applying identity on the rest. The next step is to construct a generalization of the above gate, i.e., a gate which applies $Q \in U(p)$ on the subspace spanned by any p basis vectors $|i_1\rangle, \ldots, |i_p\rangle$ while applying identity on the rest of the basis vectors. Denote this matrix by $T_m(Q, i_1, \ldots, i_p)$. We note that since $\mathcal{G}$ is universal, it can be used to approximate any one-qudit gate, so we are also allowed to use in our construction NOT_p gates.

CLAIM 6. *The gate $T_m(Q, i_1, \ldots, i_p)$ can be constructed by using gates of the form $T_{m,r}(Q)$ together with NOT_p gates.*

Proof. Consider the set of gates $\{T_{m,r}(Q)\}$, where Q runs over all permutation matrices on the Hilbert space of one qudit and r runs over all m qudits. We first prove that this set, augmented with NOT_p gates, generates all permutation matrices

on m qudits. To prove this, it suffices to construct all matrices of the form $\tau_{i,j}$ for two strings $i, j \in \{0, p-1\}^m$ that differ in only one coordinate. The matrix $\tau_{i,j}$ is defined to be the transformation that switches the two basis vectors $|i\rangle, |j\rangle$ and $\tau_{i,j}|k\rangle = |k\rangle$ for any $k \neq i, j$. Let the coordinate on which i, j disagree be the rth coordinate, and let this coordinate be equal to a in the string i and to b in the string j. To construct $\tau_{i,j}$, apply NOT_p gates on all of the coordinates except the rth coordinate, so that all of these coordinates in both strings become equal to $p-1$. Now apply $T_{m,r}(Q)$, with Q on the rth coordinate being the matrix which permutes $|a\rangle$ and $|b\rangle$, leaving the rest of the basis vectors untouched. Then reverse all of the NOT_p gates on all of the coordinates but the rth one. This gives $\tau_{i,j}$ and therefore all permutations on basis vectors. The general $T_m(Q, i_1, \ldots, i_p)$ for any Q can now be achieved by first permuting the basis vectors $i_1, \ldots, i_p$ to the last p vectors, applying $T_m(Q)$, and then permuting back. □

The last step is to use $T_m(Q, i_1, \ldots, i_p)$ to construct a general $p^m \times p^m$ unitary matrix U. Let us denote the p^m eigenvectors of U by $|\psi_j\rangle$ with corresponding eigenvalues $e^{i\theta_j}$. U is specified by $U|\psi_j\rangle = e^{i\theta_j}|\psi_j\rangle$. Define

$$U_k|\psi_j\rangle = \begin{cases} |\psi_j\rangle & \text{if } k \neq j, \\ e^{i\theta_k}|\psi_k\rangle & \text{if } k = j. \end{cases} \tag{6.1}$$

Then $U = \Pi_{k=1}^{p^m} U_k$. It is left to show how to construct U_k. For this we will show how to construct a transformation R with the following properties: R takes $|\psi_k\rangle$ to $\lambda|(p-1)^m\rangle$ for some complex number λ of absolute value 1. We don't care what R does to the rest of the vectors. Given such an R, we can use it to construct U_k as follows. We first apply R. We then apply the generalized Toffoli gate which takes $|(p-1)^m\rangle$ to $e^{i\theta_k}|(p-1)^m\rangle$ and does nothing to the rest of the basis states. Then we apply R^{-1} which takes $\lambda|(p-1)^m\rangle$ to $|\psi_k\rangle$. This indeed achieves U_k, as can be easily checked.

To construct R, and similarly R^{-1}, we do the following. We start with $|\psi_k\rangle$, and we first make the coefficient in front of $|0^m\rangle$ zero, by a rotation in the plane spanned by $|0^m\rangle$ and $|(p-1)^m\rangle$. This is a special case of the $T_m(Q, i_1, \ldots, i_p)$ which we have constructed before. Thus, the weight of $|0^m\rangle$ has been shifted to $|(p-1)^m\rangle$. In the same way, the weights in front of all basis vectors, one by one, can be shifted to $|(p-1)^m\rangle$, and this achieves R. □

6.2. Useful lemmas for proving universality. We proceed to state two geometrical lemmas. These will be used in the proofs of universality of the sets of gates $\mathcal{G}_1$ and $\mathcal{G}_2$. These lemmas are due to Kitaev, who used them for proving universality of the set of gates in [50].

LEMMA 2. *Let $n \geq 3$. Let $|\alpha\rangle \in \mathcal{C}^n$. Let H be the subgroup in $SU(n)$ which fixes $|\alpha\rangle$, and let $V \in U(n)$ be a matrix which does not leave the subspace spanned by $|\alpha\rangle$ invariant. Then the subgroup generated by the subgroups H and $V^{-1}HV$ is dense in $SU(n)$.*

The proof of this lemma can be found in the solution of Problem 8.11 in [52].

LEMMA 3. *Let U_1 and U_2 be two noncommuting matrices in $SU(2)$ such that their eigenvalues are not integer roots of unity. The subgroup generated by U_1, U_2 is dense in $SU(2)$.*

Proof. If x is an element of $SU(2)$ not of finite order, then the closed subgroup generated by x is connected and of dimension 1. The closed group generated by both U_1 and U_2 is thus connected and is noncommutative. Any connected noncommutative subgroup of $SU(2)$ is *all* of $SU(2)$. □

6.3. Universality of the set of gates $\mathcal{G}_1$ for polynomial codes. We would now like to show that the set of gates for polynomial codes $\mathcal{G}_1$ is universal.

THEOREM 10. *Consider p-qudits for a prime $p > 3$. The set of gates $\mathcal{G}_1$ together with all phase factors is universal for $U(p^5)$.*

Let us first show that we have at our disposal all classical gates on two qudits. Let F be a finite field of characteristic p for some prime p and size $q = p^r$ for some r. Consider the basic classical reversible gates on F-valued registers, including addition and multiplication by a nonzero constant, reversible addition, $[a, b] \to [a, a + b]$, and Toffoli $[a, b, c] \to [a, b, c + ab]$. For a characteristic different than 2, we also consider the second basic set, which is the above set of gates but with the Toffoli gate replaced by the squaring gate: $[a, b] \to [a, b + a^2/2]$. We claim the following.

LEMMA 4 (folklore). *Both sets of basic reversible gates, applied on $k \geq 3$ qudits, generate the full permutation group S_{q^k}. In both cases an ancillary register in the state 0 is added to the system.*

Proof. We start with the first set of basic gates. Denote by G_k the group generated by the basic gates on k registers. For a function $g : F^{k-1} \to F$ we define the permutation $\pi_g : F^k \to F^k$ by

$$\pi_g([a_1, \ldots, a_k]) = [a_1 + g(a_2, \ldots, a_k), a_2, \ldots, a_k].$$

The coordinate to which g is added is called the target coordinate. The main result of Ben-Or and Cleve [23] is that, for $k \geq 3$ and any such function g, $\pi_g \in G_k$.

We now use functions of the form π_g to generate all of S_{q^k}. First, recall that any permutation on S_{q^k} can be written as a product of transpositions of the form $\tau_{i,j}$ for $i, j \in F^k$ such that the string i differs from the string j in only one coordinate ($\tau_{i,j}$ leaves all other coordinates unchanged). It thus suffices to generate $\tau_{i,j}$ from functions of the form π_g. To do this, we add one ancillary dit in the state 0. Let us assume that the coordinate in which the strings i, j differ is the rth one. Define $g : F^k \to F$ to be the function that takes a string s to the value of the rth coordinate in $\tau_{i,j}(s)$. We have that $\pi_g([a_0, a_1, \ldots, a_k]) = [a_0 + g(a_1, \ldots, a_k), a_1, \ldots, a_k]$, which operates on an ancillary coordinate (a_0) plus the k original coordinates, is at our disposal.

To achieve $\tau_{i,j}$, we apply π_g. This is followed by π'_g, which we define to be $\pi'_g([a_0, a_1, \ldots, a_k]) = [a_0, a_1, a_{r-1}, a_r - g(a_1, \ldots, a_{r-1}, a_0, a_{r+1}, \ldots, a_k), a_{r+1}, \ldots, a_k]$. We note that

$$[a_1, \ldots, a_k] \to [a_1 - g(a_2, \ldots, a_k), a_2, \ldots, a_k]$$

is in G_k, by applying π_g $q - 1$ times. Hence, by renaming coordinates, π'_g is also at our disposal. By the definition of g we have that $\pi'_g \pi_g [0, a_1, \ldots, a_k] = [g[a_1, \ldots, a_k], a_1, \ldots, a_{r-1}, 0, a_{r+1}, \ldots, a_k]$ which is the desired transposition up to swapping the coordinates.

The proof for the second set of basic reversible gates is similar by using the fact that in this case $\pi_g \in G_k$ for any $k \geq 2$, which was also proved in [23]. ∎

Remark 2. In fact, the same result can be proved without the ancillary qudit, by using results from Coppersmith and Grossman [35]. We do not give details here.

Since $\mathcal{G}_1$ contains all basic gates, this implies that we have at our disposal all classical gates on three (and therefore also on two) qudits. Next, we prove an analogue for F_p of the well-known fact that one-qubit gates and classical two-qubit gates are universal for qubits [19].

LEMMA 5. *The group G generated by the set of gates consisting of all one-qudit gates $SU(C^p)$ and all classical two-qudit gates is the special unitary group on two qudits $SU(C^{p^2})$.*

Proof. We will use the fact that the lemma we are trying to prove is already known for the case of $p = 2$. Let S be the two-dimensional subspace in C^p spanned by the first two basis vectors $|0\rangle$ and $|1\rangle$. Clearly, $SU(S)$ is in $U(C^p)$. Let S_ℓ be the two-dimensional subspace in the Hilbert space of the ℓth qudit, for $\ell \in \{0, 1\}$, and let $A = S_1 \otimes S_2$. Since we have at our disposal all classical gates, we also have all permutations of basis vectors of A. We can use the fact that A is isomorphic to the Hilbert space of two qubits, and Lemma 5 for the case of qubits, and so this implies that $SU(A)$ can be generated. We now augment A by one computational basis state at a time to get the entire space.

Consider $|i_1\rangle, \ldots, |i_{p^2-2}\rangle$ to be a sequence of computational basis vectors which completes the basis of A, $|00\rangle, |10\rangle, |01\rangle, |11\rangle$, to a basis for C^{p^2}. Define $A_j = A \oplus |i_1\rangle \oplus \cdots \oplus |i_j\rangle$. We prove by induction on j that we can generate $SU(A_j)$. Suppose that we can generate $SU(A_{j-1})$. The classical gate which permutes $|i_j\rangle$ and $|i_{j-1}\rangle$, leaving all other states unchanged, is at our disposal. This gate is in $SU(A_j)$ and does not leave the subspace spanned by $|i_j\rangle$ invariant. Hence, we can apply Lemma 2 to get all of $SU(A_j)$. ∎

Lemma 5 implies that it would suffice to show that all one-qudit gates are at our disposal. Denote by Q the one-qudit matrix of the form

$$Q = \begin{pmatrix} w & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}, \tag{6.2}$$

which applies a phase factor of w on the state $|0\rangle$.

LEMMA 6. *Q and Q^{-1} are in the subgroup generated by $\mathcal{G}_1$ on three qudits.*

Proof. We will generate $Q \otimes I \otimes I$, which applies the following transformation:

$$\begin{aligned} |0\rangle|a\rangle|b\rangle &\longmapsto w|0\rangle|a\rangle|b\rangle, \\ |j\rangle|a\rangle|b\rangle &\longmapsto |j\rangle|a\rangle|b\rangle, \quad j \neq 0. \end{aligned} \tag{6.3}$$

To achieve this transformation, we view this gate as applying multiplication by w of the second qudit, conditioned that the first qudit is 0. Recall that in our notation P is a one-qudit gate that applies a phase shift $P|a\rangle = w^a|a\rangle$, and B is a one-qudit gate that applies a bit shift $B|a\rangle = |a + 1 \bmod \mathrm{p}\rangle$. Both B and P are in our repertoire. Denote by $\Lambda(B)$ the controlled B, which applies B on the second qudit, conditioned that the first qudit is 0, and applies the identity on the second qudit if the state of the first qudit is anything but 0. $\Lambda(B)$ is also at our disposal by Lemma 4. Now consider the commutator

$$P^{-1} \cdot \Lambda(B)^{-1} \cdot P \cdot \Lambda(B), \tag{6.4}$$

where the P's are applied to the second qudit. This is exactly the gate we want, since, if the first qudit is 0, the matrix which is applied on the second qudit is $P^{-1} \cdot B^{-1} \cdot P \cdot B = wI$. If the first qudit is in a basic state which is not $|0\rangle$, the matrix which is applied on the second qudit is the identity. ∎

We now consider the two commutator matrices in $\langle \mathcal{G}_1 \rangle$, the group generated by $\mathcal{G}_1$:

$$X = HQH^{-1}Q^{-1}, \quad Y = HQ^{-1}H^{-1}Q, \tag{6.5}$$

where H is the generalized Fourier transform. We also define the two-dimensional subspace S:

$$S = span\left\{ |0\rangle, \sum_{b\in F_p, b\neq 0} |b\rangle \right\}. \tag{6.6}$$

The claim is that X and Y operate as the identity on the orthogonal subspace to S.

LEMMA 7. *X and Y operate as the identity on $S_i^{\perp}$.*

Proof. It is easy to write down explicitly the matrix elements of $X = HQH^{-1}Q^{-1}$ and $Y = HQ^{-1}H^{-1}Q$:

$$X_{ab} = \begin{cases} \delta_{ab} + \frac{1}{p}(w-1) & \text{if } b \neq 0, \\ (\delta_{ab} + \frac{1}{p}(w-1))w^{-1} & \text{if } b = 0, \end{cases} \tag{6.7}$$

$$Y_{ab} = \begin{cases} \delta_{ab} + \frac{1}{p}(w^{-1}-1) & \text{if } b \neq 0, \\ (\delta_{ab} + \frac{1}{p}(w^{-1}-1))w & \text{if } b = 0. \end{cases}$$

To see that X and Y operate as the identity on the subspace orthogonal to S, consider the matrices $X - I$ and $Y - I$, which satisfy (6.7) if we subtract δ_{ab} from each term. We show that the orthogonal vectors to S are all in the kernel of $X - I$ and $Y - I$. A vector v orthogonal to S satisfies

$$v_0 = 0, \quad \sum_{b,b\neq 0} v_b = 0, \tag{6.8}$$

and thus

$$\sum_{b\in F_p} (X-I)_{ab} v_b = \frac{1}{p}(w-1)\sum_{b\neq 0} v_b = 0, \tag{6.9}$$

$$\sum_{b\in F_p} (Y-I)_{ab} v_b = \frac{1}{p}(w^{-1}-1)\sum_{b\neq 0} v_b = 0. \qquad \square$$

We denote by X' and Y' the two matrices confined to S. We claim that these matrices generate a dense subgroup in the group of 2×2 unitary matrices $U(2)$ operating on S. We will want to use Lemma 3, and the main effort is to prove that the eigenvalues of X' and Y' are not integer roots of unity. The proof of this fact is based on some basic results regarding cyclotomic fields and Galois fields, which can be found in [98].

LEMMA 8. *For $p > 3$, the eigenvalues of X' and Y' are not integer roots of unity.*

Proof. The subspace S is spanned by the orthonormal basis vectors $|0\rangle$ and $|\alpha\rangle = \frac{1}{\sqrt{p-1}} \sum_{b\in F_p, b\neq 0} |b\rangle$. By (6.7) and a little algebra we get

$$X'|0\rangle = \left(1 + \frac{1}{p}(w-1)\right) w^{-1}|0\rangle + \frac{\sqrt{p-1}}{p}(w-1)|\alpha\rangle, \tag{6.10}$$

$$X'|\alpha\rangle = \frac{\sqrt{p-1}}{p}(w-1)w^{-1}|0\rangle + \left(1 + \frac{p-1}{p}(w-1)\right)|\alpha\rangle,$$

and for Y we get the transformation

$$Y'|0\rangle = \left(1 + \frac{1}{p}(w^{-1}-1)\right) w|0\rangle + \frac{\sqrt{p-1}}{p}(w^{-1}-1)|\alpha\rangle, \tag{6.11}$$

$$Y'|\alpha\rangle = \frac{\sqrt{p-1}}{p}(w^{-1}-1)w|0\rangle + \left(1 + \frac{p-1}{p}(w^{-1}-1)\right)|\alpha\rangle.$$

It is easy to verify that X', and similarly Y', have determinant 1. The characteristic polynomial for X' is $\lambda^2 - \mathrm{Tr}(X')\lambda + \mathrm{Det}(X')$, which amounts to

$$f(\lambda) = \lambda^2 - \frac{2 + (p-1)(w + w^{-1})}{p}\lambda + 1. \tag{6.12}$$

Y' has exactly the same characteristic polynomial. We want to show that the roots of this polynomial are not integer roots of unity. Let us assume that one of the roots of the above polynomial is a primitive nth root of unity, denoted by ζ_n, for some integer n. The other solution is the complex conjugate of ζ_n, and we have

$$\frac{2 + (p-1)(w + w^{-1})}{p} = \zeta_n + \zeta_n^{-1}. \tag{6.13}$$

We will first prove that $n = p$ or $2p$. Denote by $Q(w)$ and $Q(\zeta_n)$ the Galois extensions of the field of rationals obtained by adjoining w and ζ_n, respectively, to the field of rationals Q. Also, denote by $Q(w)^+$ the maximal real subfield of $Q(w)$ obtained by extending Q by $w + w^{-1}$, and similarly denote the maximal real subfield of $Q(\zeta_n)$ by $Q(\zeta_n)^+$. By (6.13), $Q(\zeta_n)^+ = Q(w)^+$.

The degree of the extension $\deg(Q(w)/Q(w)^+)$ is exactly 2, since w is a root of the minimal two degree polynomial $x^2 - (w + w^{-1})x + 1$ over the field $Q(w)^+$. Similarly, $\deg(Q(\zeta_n)/Q(\zeta_n)^+) = 2$. On the other hand, $\deg(Q(w)/Q) = p - 1$ and $\deg(Q(\zeta_n)/Q) = \phi(n)$, by Theorem 2.5 in [98]. Now, for three fields, F_1, F_2, and F_3 such that F_3 extends F_2 which extends F_1, we have $\deg(F_3/F_1) = \deg(F_3/F_2)\deg(F_2/F_1)$. It follows that

$$\deg(Q(w)^+/Q) = \frac{p-1}{2}, \quad \deg(Q(\zeta_n)^+/Q) = \frac{\phi(n)}{2}. \tag{6.14}$$

But $Q(w)^+ = Q(\zeta_n)^+$, which implies that the degrees of extensions are equal, so $\phi(n) = p - 1$.

Now if $p > 3$, then $w + w^{-1} \notin Q$, since $\deg(Q(w + w^{-1})/Q) = (p-1)/2 > 1$. Since $w + w^{-1} \in Q(w) \cap Q(\zeta_n)$, we have $Q \neq Q(w) \cap Q(\zeta_n)$. If p and n were relatively prime, we would have $Q = Q(w) \cap Q(\zeta_n)$ (by Proposition 2.4 in [98]), and so p must divide n, say, $n = p^r m$, with m coprime to p. This implies that

$$\phi(n) = p^{r-1}(p-1)\phi(m). \tag{6.15}$$

Since, as we have seen before, $\phi(n) = p - 1$, we have $p^{r-1}\phi(m) = 1$. We deduce that $r = 1$ and $\phi(m) = 1$. This can be satisfied only if $m = 1$ or $m = 2$, namely, $n = p$ or $n = 2p$.

In the first case of $p = n$ we get

$$\frac{1 + (p-1)\cos(2\pi/p)}{p} = \cos(2k\pi/p), \tag{6.16}$$

where we have set $\xi_n = e^{2\pi i k/n}$. If $p > 3$, (6.16) is not satisfied by any integer k, since the left-hand side is a convex combination of $\cos(2\pi/p)$ and 1, and no real part of a pth root of unity lies between these two points.

In the second case of $p = 2n$, we get

$$\frac{1 + (p-1)\cos(2\pi/p)}{p} = \cos(k\pi/p). \tag{6.17}$$

Ruling this out is similar to ruling out (6.16), except that we have to show that the case of $k = 1$ does not hold, namely, $\frac{1+(p-1)\cos(2\pi/p)}{p} \neq \cos(\pi/p)$. This is true because the cosine function is a concave function. The lemma follows. □

We can now show that we have at our disposal all one-qudit gates.

CLAIM 7. *For $p > 3$, the set $\mathcal{G}_1$ operating on three qudits generates all one-qudit gates.*

Proof. Since we are allowed to operate on three qudits, we can generate Q by Lemma 6. We can thus generate X and Y. It is easy to see that, if $w \neq \pm 1$ or, equivalently, $p \neq 2$, the off-diagonal terms of $X'Y' - Y'X'$ are not zero, and so X' and Y' do not commute. We can therefore apply Lemma 3, by using the fact that the eigenvalues are not roots of unity, by Lemma 8. We thus have a dense subset of the special unitary group $SU(2)$ on all subspaces S. We now want to augment the subspace S by one basis vector at a time, to get the entire $SU(p)$ group. To do this, we define $S_0 = S$ and, for $p > j \geq 1$, $S_j = S \oplus |1\rangle \oplus \cdots \oplus |j\rangle$. We have seen that we can generate $SU(S_0)$. Assume by induction that we can generate $SU(S_{j-1})$, and let us see that we can generate $SU(S_j)$. Let $|\gamma\rangle$ be the state orthogonal to S_{j-1} inside S_j. If we set $|\delta\rangle = \ sum_{k=j+1}^{p-1} |k\rangle$, then it is easy to check that $|\gamma\rangle$ is a nontrivial combination of $|\delta\rangle$ and $|j\rangle$. We have at our disposal a classical gate which exchanges $|0\rangle$ and $|j\rangle$. Moreover, this gate is in $SU(S_j)$. This gate does not leave the subspace spanned by $|\gamma\rangle$ invariant. Hence we can apply Lemma 2 and get all of $SU(S_j)$. Since $SU(S_{p-1}) = SU(p)$, the claim is proved. ∎

The proof of Theorem 10 now follows easily.

Proof of Theorem 10. We have shown that we have at our disposal all one-qudit gates. Lemma 4 shows that, by working on five qudits, we also have in our repertoire all classical gates on three qudits and, in particular, all classical gates on two qudits which act trivially on the third qudit. Lemma 5 implies that we can generate all two-qudit gates. The theorem follows from Theorem 9, which shows that these matrices can be used to construct all matrices on five qudits $U(p^5)$. ∎

6.4. Universality of the set of gates $\mathcal{G}_2$ used for CSS codes. The set of gates used for CSS codes is shown here to be universal. The theorem is based on an argument by Kitaev [50] which is given here for completeness.

THEOREM 11. *The set of gates $\mathcal{G}_2$ together with all phase factors is universal for $U(2^5)$.*

Proof. We denote by P the gate that takes $|1\rangle$ to $i|1\rangle$ and does nothing to $|0\rangle$. $\Lambda(P)$ is the controlled P, namely, the gate which applies P on a second qubit only if the first qubit is $|1\rangle$ and does nothing otherwise. The proof is based on a result by Kitaev [50], which asserts that the set of gates $\{\Lambda(P), H\}$ is universal. Since the Hadamard gate H is already in our set $\mathcal{G}_\in$, we need only to show how to construct $\Lambda(P)$ from our set of gates. We will denote by $T_{a_1,\ldots,a_k}$ a generalized Toffoli on the k qubits $a_1, \ldots, a_k$. This gate applies NOT on the kth qubit conditioned that the first $k-1$ qubits are in the state $|1\rangle$. We first construct $T_{1,2,4,5}$ out of five qubits. This can be done by using three-qubit Toffoli gates as follows:

$$
\begin{aligned}
a, b, c, d, e \quad &\overset{T_{1,2,3}}{\longmapsto} \\
a, b, c + ab, d, e \quad &\overset{T_{3,4,5}}{\longmapsto} \\
a, b, c + ab, d, e + cd + abd \quad &\overset{T_{1,2,3}}{\longmapsto} \\
a, b, c, d, e + cd + abd \quad &\overset{T_{3,4,5}}{\longmapsto} \\
a, b, c, d, e + abd,
\end{aligned}
\tag{6.18}
$$

which is exactly $T_{1,2,4,5}$. Define $X = P_4^3 T_{1,2,3,4} P_4 T_{1,2,3,4}$, where P_4 applies the gate P to the fourth qubit. X takes $|1110\rangle \longmapsto i|1110\rangle$, $|1111\rangle \longmapsto -i|1111\rangle$ and does nothing

to the other basis states. X^2 is the three-qubit gate which gives $|111\rangle \longmapsto -|111\rangle$ and identity on the rest of the basis vectors, tensored with identity on the fourth qubit. Now, to construct $\Lambda(P)$, we apply $P_3^3 T_{1,2,3} P_3 T_{1,2,3}$ followed by X^2. The first sequence of gates gives $|110\rangle \longmapsto i|110\rangle, |111\rangle \longmapsto -i|111\rangle$ and identity on the rest. By applying X^2 we get $\Lambda(P)$ tensored with identity on the third qubit. The theorem now follows from Kitaev [50].

We now provide Kitaev's argument which uses Lemmas 2 and 3 to show that $\Lambda(P)$ and the Hadamard gate are universal. Denote that

$$X_1 = H_1 \Lambda(P)_{1,2} H_1, \qquad X_2 = H_2 \Lambda(P)_{2,1}^{-1} H_2. \tag{6.19}$$

Define $Y_1 = X_1 X_2^{-1}$ and $Y_2 = X_2 X_1^{-1}$. Note that Y_1 and Y_2 both operate as the identity on the two states $|00\rangle$ and $|\eta\rangle = |01\rangle + |10\rangle + |11\rangle$. Denote by L the subspace orthogonal to $|00\rangle$ and $|\eta\rangle$. Then $Y_1\ Y_2 \in SU(L)$. Y_1, and Y_2 do not commute, and their eigenvalues are $\frac{1}{4}(1 \pm \sqrt{15})$. Hence, by Lemma 3 they generate a dense subgroup in $SU(L)$. Now add to Y_1, Y_2 also the gate $\Lambda(P)$ itself. This gate fixes $|00\rangle$ but does not stabilize the space $|\eta\rangle$. We can use Lemma 2 to show that the set $\{Y_1, Y_2, \Lambda(P)\}$ generates a dense subgroup in $SU(L \oplus |\eta\rangle)$. Finally, add H_1 to the set $Y_1, Y_2, \Lambda(P)$. We have seen that $Y_1, Y_2, \Lambda(P)$ generates a dense set in the subgroup that fixes $|00\rangle$, while H_1 is not in this subgroup. Hence, $H_1, Y_1, Y_2, \Lambda(P)$ (which are all gates generated by H and $\Lambda(P)$) generate a dense subgroup of $SU(L \oplus |\eta\rangle \oplus |00\rangle) = SU(4)$. Together with all phase factors, we get the unitary group on two qubits. □

7. Fault tolerance for independent probabilistic noise. In this section we prove our main result: the threshold theorem for fault tolerance in the presence of probabilistic noise with a constant error rate. To do this we use the ingredients we have developed in the previous sections: quantum error correcting codes and fault-tolerant procedures on states encoded by these codes. The construction is based on a simulation of an unreliable circuit M_0 by another circuit M_1, which computes the same function. The new circuit M_1 is deeper and requires more space than M_0 but only by a constant factor. The advantage is that, under certain conditions on the error rate, M_1 is more reliable than the original circuit M_0. In such a case, the simulation can be applied recursively, each time achieving further improvement in the effective error rate. The final quantum circuit M_r, which is only polylogarithmically larger and deeper than M_0, can be shown to be fault-tolerant against a constant error rate. To analyze the propagation of errors in M_r, we define the notion of sparse errors and sparse fault paths, by using hierarchical definitions that fit the hierarchical structure of the construction. The threshold theorem is proved in two parts. First, we show that sparse fault paths are good, meaning that they cause sparse errors which do not affect the final result. Second, we show that nonsparse fault paths are rare, as long as the error rate is below a certain threshold.

For this section, we use the terminology of qubits. Everything works in exactly the same way if the particles are qudits instead, as is required in the scheme which uses the polynomial codes. In this case, the final circuit will of course consist of p-qudits, where p is some finite dimension.[5]

[5]One can of course apply our construction, which uses polynomial codes, such that the final circuit is implemented with qubits. For this, we simply replace each p-qudit in the final circuit with $\lceil \log(p) \rceil$ qubits. The proof that the new circuit is fault-tolerant follows similar ideas as in section 9.

7.1. Quantum computation codes. In order to compute on encoded states, we need a quantum code accompanied with a universal set of gates which can be applied fault-tolerantly (namely, with small spread) on states encoded by C. The code should also be accompanied with fault-tolerant decoding, zero-state preparation, and error correction procedures. We define the following.

DEFINITION 20. *A quantum code is called a quantum computation code with spread* 1 *if it is accompanied with a universal set of gates G with fault-tolerant procedures and with fault-tolerant zero-state preparation, decoding, and correction procedures, all with spread* 1. *Moreover, we require that*

(1) *all procedures use only gates from G, and*

(2) *the correction procedure takes any density matrix to some word in the code.*

The first restriction in the definition of a quantum computation code allows us to use this code in a recursive construction. The second restriction is required for a more subtle reason, as will become clear in the proof of Lemma 9.

Theorem 5 implies that the polynomial QECC accompanied by the set of gates $\mathcal{G}_1$ is a quantum computation code with spread 1. Theorem 6 implies the same for CSS codes over F_2 restricted as in section 5.1, with the set of gates $\mathcal{G}_2$.

In our proof of fault tolerance, we will be interested not in the spread of one procedure but in the spread of a *sequence* of two procedures, namely, an encoded gate preceded by an error correction (this will be referred to as a rectangle). For the case of procedures of spread 1, this does not matter, since by augmenting two such procedures one after the other, the resulting circuit consisting of the pair of procedures also has spread 1.

Unfortunately, this does not hold for the case in which the spread of the procedures we augment is larger than 1. In fact, if the spread of the procedures is ℓ', the spread of a circuit consisting of two of them one after the other might be ℓ'^2. We take this into account in our definition of a quantum computation code which uses procedures of spread larger than 1. A quantum computation code with spread ℓ is defined as in Definition 20, except that we require that the spread of the decoding and zero-state preparation procedure, as well as the spread of a sequence of an encoded gate preceded by an error correction procedure, be ℓ.

Obviously, we must require that the number of errors that the code can correct q be larger than the spread of the code, so that we can tolerate at least one fault in a rectangle. In fact, we require more than that for our proof to work: We need $2\ell \leq q$, as will be seen in Lemma 9.

7.2. Recursive simulations. From now on, fix a quantum computation code C. It encodes one qubit on m qubits, it corrects q errors, and it is accompanied with a universal set of gates $\mathcal{G}$ which can be performed fault-tolerantly. We will fix m to be a constant which does not grow with n. Let M_0 be a quantum circuit using gates from $\mathcal{G}$. We simulate M_0 by a more reliable circuit M_1, as follows. Each qubit is replaced by a block of qubits. Each time step in M_0 transforms in M_1 to a working period, which consists of two stages. In the first stage, an error correction procedure is applied on each block. At the second stage, each gate which operated in the simulated time step in M_0 is replaced in M_1 by its procedure, operating on the corresponding blocks.

The input of M_1 is the input to M_0, where each input bit is duplicated m times. Before any computation is done on this input, we apply in M_1 a zero-state preparation procedure, then apply CNOT_p transversally from the given input string $|a^m\rangle$ to the encoded state $|S_0\rangle$, and discard the m initial dits. At the end of the computation we will again use redundancy: For each block, we apply a decoding procedure which

decodes the state to m copies of the logical bit, and the output of M_1 is defined as the majority of the bits in each block.

The above mapping, denoted by $M_1 = \phi(M_0)$, constitutes one level of the simulation. The mapping ϕ is then applied once again, this time on M_1, to give M_2. We repeat this r levels to get $M_r = \phi^r(M_0)$, an r-simulating circuit of M_0. The number of levels r is $O(\text{polyloglog}(V(M_0)))$, where $V(M_0)$ is the volume of M_0, i.e., the number of locations in M_0 (see Definition 5 of the term *location*). The output of M_r is defined by taking recursive majority on the outputs. This means that first we take the majority in each block of size m, then we take the majority, of m such majority bits, and so on for r levels, to give one output bit.

7.3. Blocks and rectangles. The recursive simulations induce a definition of s-blocks: Every qubit transforms to a block of m qubits in the next level, and this block transforms to m blocks of m qubits, and so on. One qubit in M_{r-s} thus transforms to m^s qubits in M_r. This set of qubits in M_r is called an s-block. This induces a division of the qubits in M_r to s-blocks, and this division is a refinement of the division to $s+1$-blocks. A 0-block in M_r is simply a qubit. In the same way, one can define s-working periods. Each time step in M_0 transforms to w time steps in M_1, and an s-working period is the time interval in M_r which corresponds to one time step in M_{r-s}.

The recursive simulation induces a partition of the set of locations in M_r to generalized rectangles. An r-rectangle in M_r is the set of locations which originated from one location in M_0. This is best explained by an example: Consider a CNOT gate which is applied in M_0 at time t on qubits q_1, q_2. The location $((q_1, q_2), t)$ in M_0 transforms in M_1 to error correction procedures on both blocks, followed by the procedure of the CNOT gate. The set of locations in these three procedures is the 1-rectangle in M_1 which originated from the location $((q_1, q_2), t)$ in M_0. More generally, an s-rectangle in M_r is the set of points in M_r which originated from one location in M_{r-s}. Note that the partition to s-rectangles is a refinement of the partition to $(s+1)$-rectangles. A 0-rectangle in M_r is just one location.

7.4. Sparse errors and sparse faults. In a noiseless scenario, the state of M_r at the end of each r-working period encodes the state of M_0 at the end of the corresponding time step. However, we assume that errors occur in M_r, and we want to analyze those. In order to analyze the propagation of errors in M_r, we need to distinguish between the actual faults that occur during the computation and the errors that are caused in the state. First, we focus on the errors in the states and define a distance between encoded states. The hierarchy of blocks requires a recursive definition.

DEFINITION 21. *Let B be the set of qubits in n r-blocks. An (r,k)-sparse set of qubits A in B is a set of qubits in which, for every r-block in B, there are at most k $(r-1)$-blocks such that the set A in these blocks is not $(r-1,k)$ sparse. A $(0,k)$-sparse set of qubits A is an empty set of qubits.*

Two density matrices ρ_1 and ρ_2 of the set of qubits B are said to be (r,k)-deviated if there exists an (r,k)-sparse set of qubits $A \subseteq B$, with $\rho_1|_{B-A} = \rho_2|_{B-A}$. The deviation satisfies the triangle inequality since the union of two sets which are (r,l_1)- and (r,l_2)-sparse, respectively, is (r,l_1+l_2)-sparse, by induction on r.

We will see that a computation is successful if the error at the end of each r-working period is sparse enough. The question is which fault paths keep the errors sparse. We will show in Lemma 9 that this is guaranteed if the fault path is sparse.

DEFINITION 22. *A set of locations in an r-rectangle is said to be (r,k)-sparse if there are no more than k $(r-1)$-rectangles in which the set is not $(r-1,k)$-sparse. A $(0,k)$-sparse set in a 0-rectangle is an empty set. A fault path in M_r is (r,k)-sparse if, in each r-rectangle, the set is (r,k)-sparse.*

7.5. The good part: Sparse fault paths keep the error sparse. We claim that if the fault path is sparse enough, then the error corrections keep the deviation small. The number of effective errors at any given level is thus constantly maintained below the dangerous zone. This is true at all levels, and indeed we prove it by induction on the level index r.

LEMMA 9. *Let C be a computation code that corrects q errors, with spread ℓ. Let M_r be the r-simulation of M_0 by C. Consider a computation subjected to an (r,k)-sparse fault path with $2k\ell \le q$. At the end of each r-working period the error is $(r,q/2)$-sparse.*

A condition of the form $k\ell \le q$ is natural, since k faults can indeed propagate to $k\ell$ errors if the spread is ℓ, and so we have to require that $k\ell \le q$ for the error correction to work. Our condition is in fact more restrictive, having the additional factor of 2, for technical reasons that will become apparent in the proof.

Proof. It is instructive to first prove this lemma for $r = 1$. This is done by induction on the time step t. For $t = 0$ the deviation is zero. Suppose that the density matrix at the end of the tth working period is $(1,q/2)$-deviated from the correct density matrix. If no errors occur during the tth working period, the error corrections would have corrected the state to $\phi(\rho(t))$, and the procedures would have taken it to the correct state $\phi(\rho(t+1))$. However, k faults did occur in each rectangle. We will use the fact that the rectangle has spread ℓ. Let us add the k faults to the fault path in the rectangle one by one. While adding the ith fault, we have $(q/2 + (i-1))\ell \le q - \ell$, because $i \le k$, and $2k\ell \le q$. Hence, this satisfies the requirement in Definition 17, and so we can deduce that each of the faults we add increases the deviation in the state at the end of the rectangle by at most ℓ qubits in each block. Thus, we have in each block at most $k\ell$ qubits which are affected by the faults, and the final deviation is at most $k\ell \le q/2$. This proves the theorem for $r = 1$.

For general r, we prove two assertions together, by using induction on r. The first assertion implies the desired result.

1. Consider n r-blocks, in a density matrix ρ_r which is $(r,q/2)$-deviated from $\phi^r(\rho_0)$, where ρ_0 is a density matrix of n qubits. At the end of an r-working period which r-simulates the operation of g_0 on ρ_0, with an (r,k)-sparse set of faults, the density matrix is $(r,q/2)$-deviated from $\phi^r(g_0 \circ \rho_0)$.
2. Consider n r-blocks in the state ρ_r such that x of the blocks are in an arbitrary density matrix and y of them are in a density matrix which is $(r,q/2)$-deviated from some word in the code $\phi^r(\rho_0)$. Consider an r-working period which r-simulates the operation g_0 with an (r,k)-sparse set of faults, applied on ρ_r. We claim that at the end of the r-working period the state is the same as if we started with the y blocks as they are and the x blocks corrected to some word in the code. More precisely, at the end of the r-working period the density matrix is $(r,q/2)$-deviated from $\phi^r(g_0 \circ \rho_0')$, where ρ_0' is a density matrix of n qubits which when reduced to the qubits corresponding to the y good blocks is equal to ρ_0.

For $r = 1$ the proof of the first assertion is as before. The second assertion is true because of a similar argument, using the second requirement in the definition of a quantum computation code (Definition 20), namely, that the error correction

procedure takes any density matrix to a word in the code. Let us now assume both claims for r and prove each of the claims for $r+1$.

Proof of Assertion 1. We consider an $(r+1)$-working period operating on ρ_{r+1}, $(r+1)$-simulating the procedure encoding g_0 on ρ_0. The $(r+1)$-simulation working period can be seen as an r-simulation of one working period in M_1 which 1-simulates some computation in M_1, consisting of an error correction procedure followed by an encoded gate.

Consider for a moment the 1-simulation circuit. We know that this circuit has spread ℓ. By using the fact that $k\ell + q/2 \leq q$, we know that if at most k faults occurred in this 1-simulation, and if the input state is $(1, q/2)$-deviated from being correct, then the output state is at most $(1, q/2)$-deviated from the correct state, namely, from $\phi(g_0(\rho_0))$.

We now turn back to the r-simulation of this circuit, namely, to our $(r+1)$-working period. We now assume a wrong assumption: The state at the end of each r-working period (in our $(r+1)$-working period) is $(r, q/2)$-deviated from some word in the code. This means that the $q/2$ problematic input blocks are $(r, q/2)$-deviated from some word in the code and, moreover, that at the end of each of the problematic r-rectangles the state of these blocks is $(r, q/2)$-deviated from some word in the code. In this case the proof of the first assertion follows easily from our first induction assumption, as follows.

Let w be the number of r-working periods in the $(r+1)$-working period. By the induction assumption on the first assertion, we have that at the end of each one of the r-working periods the matrix is $(r, q/2)$-deviated from what it is supposed to be by the computation we are r-simulating, namely, by the 1-simulation with spread ℓ. After w applications of this induction assumption, we get that the matrix at the end is $(r, q/2)$-deviated from a matrix $\phi^r(\rho_1)$, where ρ_1 is $(1, q/2)$-deviated from $g_0(\rho_0)$. This implies that the final density matrix is $(r+1, q/2)$-deviated from the correct matrix $\phi^{r+1}(g_0(\rho_0))$.

To release the wrong assumption, we use the induction hypothesis on the second assertion. We claim that nothing changes in the above argument if we apply, before each r-working period, an error correction procedure on each r-block, which has an (r, k)-sparse set of faults. By the induction assumption on the first assertion, for blocks which are $(r, q/2)$-deviated from some word in the code, this will remain the case after this error correction. Consider a block which is not $(r, q/2)$-deviated from some word in the code. Then the error correction will indeed change it to a block which is $(r, q/2)$-deviated from some word in the code. If the next r-rectangle that operates on that block is bad (namely, the fault path in this r-rectangle is not (r, k)-sparse), then the fact that we have added the correction does not change anything in the argument, since the faults can be chosen adversarially to ruin the entire block. If the next r-rectangle is good (namely, the fault path in it is (r, k)-sparse), then by the induction assumption on the second assertion this correction will happen in any case, and thus does not matter. Hence, the above argument goes through.

Proof of Assertion 2. The proof follows almost exactly the same argument as for the first assertion. We consider again an r-simulation of a 1-simulation. The first stage of the 1-simulation we consider here is a 1-error correction which takes any word to some word in the code, and so it is supposed to take the 1-blocks corresponding to the y-blocks to their state $\phi(\rho_0)$ and the x-blocks to some state. Hence, the 1-simulation is supposed to take the state to $\phi(\rho_0')$ as in the assertion. The remainder of the procedure is supposed to apply $\phi(g_0)$ to this state. Once again impose the wrong assumption as in the proof of the first assertion, and use the fact that the

spread of the 1-circuit is ℓ. By the induction assumption on the first assertion, we have that, at the end of each one of the r-working periods in the r-simulation of the above 1-simulation, the matrix is $(r, q/2)$-deviated from what it is supposed to be. We get that, under this assumption, the final matrix would be $(r, q/2)$-deviated from a matrix $\phi^r(\rho_1)$, where ρ_1 is $(1, q/2)$-deviated from $\phi(\rho')$. This implies that the final matrix is $(r+1, q/2)$-deviated from $\phi^r(\rho'_0)$. The relaxation of the wrong assumption is done exactly as before. □

This lemma implies that, if the faults are sufficiently sparse, the level of deviation can be kept below a certain value for the entire computation, and so at the end of the computation the set of errors is sparse. We need to show that in this case the recursive majority of the output bits indeed gives the correct answer.

LEMMA 10. *Let $2q+1 \leq m$. Let the final density matrix of M_r be $(r, q/2)$-deviated from the correct one. Consider the distribution on n bit strings which is obtained by measuring the output and taking recursive majority on each r-block. This distribution is equal to the output distribution of M_0.*

Proof. Let ρ_r be the correct final density matrix of M_r. Let ρ'_r be the final density matrix which is $(r, q/2)$-deviated from ρ_r. This remains true if we apply a measurement of all of the qubits and also if we discard certain r-blocks which do not correspond to the output of the computation. We can thus assume that ρ_r and ρ'_r are the density matrices of the output qubits, that they are mixtures of basis states, and that ρ'_r is $(r, q/2)$-deviated from ρ_r.

Note that, due to the reading procedure, the correct density matrix ρ_r is in the subspace which is spanned by basis states in which all of the coordinates in one r-block are equal, i.e., are of the form $|i_1^{m^r}, i_2^{m^r}, \ldots, i_n^{m^r}\rangle$. We have to show that, when we measure the qubits as a density matrix which is $(r, q/2)$-deviated from such a matrix, and we take the recursive majority, we get the correct distribution.

The matrix ρ_r can be written as $\{p_i, |\alpha_i\rangle\}$, where $|\alpha_i\rangle = |i_1^{m^r}, i_2^{m^r}, \ldots, i_n^{m^r}\rangle$. All qubits in an r-block are equal, since the matrix is correct. The probability that the recursive majority string is i is exactly p_i. ρ'_r can be written as $\{q_j, |\beta_j\rangle\}$, where $|\beta_j\rangle$ are basis vectors. Let A be a subset of qubits such that $\rho_r|_A = \rho'_r|_A$ and A contains all of the qubits on which ρ_r operates, except an $(r, q/2)$-sparse set. We know that the reduced density matrix of ρ'_r to A is the same as that of ρ_r, and so we have that ρ'_r is supported only by $|\beta_j\rangle$ such that their restrictions to qubits in A are strings which agree on all qubits in each r-block. Moreover, we have

$$\sum_{j, \beta_j|_A = \alpha_i|_A} q_j = p_i. \tag{7.1}$$

It is easy to see that the recursive majority string for $|\beta_j\rangle$ in the above sum is the same as that of $|\alpha_i\rangle$ and therefore is equal to i. Hence, we have that the probability for the recursive majority to be i in ρ'_r is p_i, and this completes the proof. □

7.6. The bad part: Nonsparse fault paths are rare below the threshold. Let us first analyze the improvement in the effective error rate when going from M_0 to M_1. By Lemma 9 we know that, as long as the number of faults k in each rectangle in M_1 satisfies $k \leq q/2\ell$ (where q is the number of errors the code corrects and ℓ is the spread of the code), then the computation at the end is correct. Therefore, the probability for more faults than $q/2\ell$ in one rectangle is a good definition of the effective error rate: If no such event happened, than there is no error. We can easily upper bound this probability from above. We set $k_0 = \lfloor q/2\ell \rfloor$ and denote by A the

number of locations in the largest rectangle in any simulation using C. Then

$$\eta_{\text{eff}} = \binom{A}{k_0+1} \eta^{k_0+1} \tag{7.2}$$

is an upper bound on the probability for more than k_0 faults in one rectangle. We refer to this bound as the effective error rate of the new circuit M_1 (we have obviously overestimated the effective error rate in this definition, but we do not attempt to optimize our analysis here).

DEFINITION 23 (the threshold). *We define the threshold η_c to be the error rate for which $\eta_{\text{eff}} = \eta$:*

$$\eta_c = \frac{1}{\binom{A}{k_0+1}^{1/k_0}}. \tag{7.3}$$

The following claim justifies the term *threshold*.

CLAIM 8. *Suppose that $k_0 \geq 1$. For any η below the threshold $\eta < \eta_c$, we have that the effective error rate is strictly smaller than the actual error rate: $\eta_{\text{eff}} < \eta$.*

Proof. Note that $x = \eta/\eta_c < 1$.

$$\eta_{\text{eff}} = \binom{A}{k_0+1} \eta^{k_0+1} = \binom{A}{k_0+1} \eta_c^{k_0+1} x^{k_0+1} = \eta_c x^{k_0+1} = \eta x^{k_0} < \eta. \qquad \square$$

We now show that, below the threshold, bad (namely, nonsparse) fault paths are rare.

LEMMA 11. *If $\eta < \eta_c$, $\exists \delta > 0$ such that the probability $P(r)$ for the fault path restricted to an r-rectangle to be (r, k_0)-sparse is larger than $1 - \eta^{(1+\delta)^r}$.*

Proof. Let δ be such that

$$\binom{A}{k_0+1} \eta^{k_0+1} < \eta^{1+\delta}. \tag{7.4}$$

Such a δ exists for η below the threshold, by Claim 8.

The proof of the lemma follows by induction on r. The probability for a 0-rectangle, i.e., one location, to have faults which are $(0, k_0)$-sparse, namely, the probability that in this location a fault did not occur, is $1 - \eta$.

Assume for r, and let us prove for $r+1$. For the faults in an $(r+1)$-rectangle not to be $(r+1, k_0)$-sparse, there must be at least $k_0 + 1$ r-rectangles in which the fault path is not (r, k_0)-sparse. So

$$P(r+1) \geq 1 - \binom{A}{k_0+1} (1 - P(r))^{k_0+1} > 1 - \eta^{(1+\delta)^{r+1}},$$

by using the induction assumption and the fact that $\eta^{(1+\delta)^r} < \eta < \eta_c$, and hence (7.4) can be applied with $\eta^{(1+\delta)^r}$ instead of η. $\square$

We can now prove the threshold theorem.

THEOREM 12 (the threshold theorem for probabilistic noise). *Let $\epsilon > 0$. Let C be a computation code using a set of gates $\mathcal{G}$. There exist a constant threshold $\eta_c > 0$ and constants c_1, c_2, c_3 such that the following holds. Let Q be a quantum circuit, with n input qubits (qudits), which operates t time steps, uses s gates from $\mathcal{G}$, and has v locations. There exists a quantum circuit Q' which operates on $n \cdot O(\log^{c_1}(\frac{v}{\epsilon}))$ qubits*

(qudits), for time $t \cdot O(\log^{c_2}(\frac{v}{\epsilon}))$*, and uses* $v \cdot O(\log^{c_3}(\frac{v}{\epsilon}))$ *gates from* $\mathcal{G}$ *such that, in the presence of probabilistic noise with error rate* $\eta < \eta_c$*,* Q' *computes a function which is within* ϵ *total variation distance to that computed by* Q*.*

Proof. We set M_0 to be equal to Q. Let $k_0 = \lfloor q/2\ell \rfloor$ as before. We set δ to be as in (7.4). We choose r such that $v\eta^{(1+\delta)^r} < \epsilon$. We then define Q' to be M_r generated according to the above scheme. By Lemma 11, we have that the probability for a fault path to be bad, i.e., not (r, k_0)-sparse, is smaller than ϵ. By Lemmas 9 and 10, the (r, k_0)-sparse fault paths give correct outputs. The resulting function (or distribution) is thus within ϵ total variation distance from correct.

As for the overhead, the number of qubits in Q' is nm^r, the number of time steps is tw^r, where w is the largest number of time steps in one rectangle, and the number of gates is vA^r. The proof follows from our choice of r, which turns out to be some constant times $\log\log(\epsilon/v)$. ∎

8. The threshold result for general noise. So far, we have dealt only with probabilistic faults. Actually, the circuit generated by the above recursive scheme is robust also against general local noise. This makes the applicability of the result much wider.

We prove this by writing the noise operator on each qubit as the identity plus a small error term. By expanding the error terms in powers of η, we get a sum of terms, each corresponding to a different fault path. The threshold result for general noise is again proved by dividing the faults into good and bad parts. First, we show that the bad part is negligible, i.e., that the norm of the sum of all terms which correspond to nonsparse fault paths is small. The proof that the good faults are indeed good, i.e., that the error in the terms corresponding to sparse fault paths is sparse, is based on the proof for probabilistic errors, together with some linearity considerations.

The threshold one gets in this case is slightly worse.

8.1. Fault paths in the case of general noise. The notion of fault paths is less clear in the case of general noise. To define fault paths, write the final density matrix of the noisy circuit as follows:

$$\rho(t) = \mathcal{E}(t) \cdot \mathcal{L}(t) \cdot \mathcal{E}(t-1) \cdot \mathcal{L}(t-1) \ldots \mathcal{E}(0) \cdot \mathcal{L}(0)\rho(0). \tag{8.1}$$

In the above equation, $\mathcal{E}(t)$ is the noise operator operating at time t, and $\mathcal{L}(t)$ is the computation operator at time t. According to our noise model (2.3), $\mathcal{E}(t)$ can be written as a tensor product of operators, operating on the possible locations of faults at time t, $A_{i,t}$. Each such operator can be written as a sum of two operators, by using (2.4):

$$\mathcal{E}_{A_{i,t}}(t) = (1-\eta)I + \mathcal{E}'_{A_{i,t}}(t), \quad \|\mathcal{E}'_{A_{i,t}}(t)\| \le 2\eta. \tag{8.2}$$

We can replace all of the error operators in (8.1) by the products of operators of the form (8.2). We get:

$$\begin{aligned}\rho(t) = {} & \left(\otimes_{A_{i,t}}((1-\eta)I + \mathcal{E}'_{A_{i,t}})\right) \cdot \mathcal{L}(t) \cdot \left(\otimes_{A_{i,t-1}}((1-\eta)I + \mathcal{E}'_{A_{i,t-1}})\right) \cdot \mathcal{L}(t-1) \\ & \ldots \left(\otimes_{A_{i,0}}((1-\eta)I + \mathcal{E}'_{A_{i,t-1}})\right) \mathcal{L}(0)\rho(0).\end{aligned} \tag{8.3}$$

We can open up the brackets in the above expression. We get a sum of terms, where in each term, for each set of qubits $A_{i,t}$ at time t, we operate either $(1-\eta)I$ or $\mathcal{E}'_{A_{i,t}}(t)$. Thus, each term in the sum corresponds to a certain fault path. More

precisely, a fault path is a subset of the locations $\{A_{i,t}\}_{i,t}$, and the term in the sum which corresponds to this fault path is exactly the term in which we apply $\mathcal{E}'_{A_{i,t}}(t)$ on all locations in the fault path and apply $(1-\eta)I$ on the rest. As was done in the probabilistic case, we can now divide the above sum (8.3) into two parts: the sum over the *good* fault paths and the sum over the *bad* fault paths. We define the good fault paths to be those which are (r, k_0)-sparse, and the bad ones are all of the rest. We write

$$\rho(t) = \mathcal{L}_g \cdot \rho(0) + \mathcal{L}_b \cdot \rho(0). \tag{8.4}$$

We will treat each part separately.

8.2. The bad part: Nonsparse fault paths are negligible below the threshold. We show that the trace norm of the *bad* part is negligible, when η is below the threshold for general noise.

DEFINITION 24. *The threshold for general noise $\eta'_c > 0$ is defined to be the error rate η such that*

$$e \begin{pmatrix} A \\ k_0+1 \end{pmatrix} (2\eta)^{k_0+1} = 2\eta, \tag{8.5}$$

where, as before, $k_0 = \lfloor q/2\ell \rfloor$, q is the number of errors which the code corrects, ℓ is the spread of the code, and A is the maximal number of locations in a rectangle. The threshold for general noise for the code C is thus

$$\eta'_c = \frac{1}{2} e^{-k_0} \begin{pmatrix} A \\ k_0+1 \end{pmatrix}^{-k_0}. \tag{8.6}$$

Note that there is a slight difference from the threshold in the case of probabilistic noise. A factor of 2 is added to η, and the factor of e is added to the whole definition. These differences are due to the fact that the norm of the good operators in the general noise case is not smaller than 1 but can also be slightly larger than 1. η'_c is thus taken to be smaller than the threshold for probabilistic noise η_c.

We now prove that the contribution of the bad fault paths is indeed small.

LEMMA 12. *Let $\eta \langle \eta'_c, \epsilon \rangle 0$. Let M_0 use v locations. Let $r = \text{polyloglog}(\frac{v}{\epsilon})$. Then in M_r*

$$\|\mathcal{L}_b\| \leq \epsilon, \quad \|\mathcal{L}_g\| \leq 1 + \epsilon.$$

Proof. We shall rewrite the sum over all fault paths, by collecting together all of the operations according to in which r-rectangles they were done. In other words, we order the r-rectangles in some topological order, and then apply all of the operators that belong to one rectangle, before we start with a different rectangle. We now open up the operators corresponding to one r-rectangle, by using (8.2). We denote by $\mathcal{L}_b(i)$ the sum over all operators on the ith r-rectangle that correspond to errors on a bad fault path in this rectangle. Similarly, $\mathcal{L}_g(i)$ is the sum over all operators on the ith rectangle that correspond to good fault paths. If there are v procedures, we can write

$$\rho(t) = (\mathcal{L}_g(v) + \mathcal{L}_b(v)) \cdot (\mathcal{L}_g(v-1) + \mathcal{L}_b(v-1)) \ldots (\mathcal{L}_g(1) + \mathcal{L}_b(1))\rho(0) \tag{8.7}$$

for $\rho(0)$ being the initial density matrix. We first prove that

$$\forall 1 \leq i \leq v, \quad \|\mathcal{L}_b(i)\| \leq (2\eta)^{(1+\delta)^r}, \quad \|\mathcal{L}_g(i)\| \leq 1 + (2\eta)^{(1+\delta)^r}, \tag{8.8}$$

where δ is defined by

$$e\begin{pmatrix} A \\ k_0+1 \end{pmatrix}(2\eta)^{k_0+1} < (2\eta)^{1+\delta}. \tag{8.9}$$

Such a δ exists because η is below the threshold defined in (8.6). The proof of inequality (8.8) for $\eta < \eta_c'$ follows the lines of Lemma 11.

We use induction on r. It is useful to add the superscript r and denote by $\mathcal{L}_b^r(i)$ the sum over all bad fault paths in an r-rectangle. For $r = 0$, a 0-rectangle is simply one location. Hence $\mathcal{L}_b^0(i)$, which is the sum over all *bad* fault paths in this location, consists of only one term: the gate applied in this location (which could be the identity) followed by one noise operator. By (8.2), and the properties of the norm on superoperators in section 2.9, $\|\mathcal{L}_b^0(i)\| \le 2\eta$ and $\|\mathcal{L}_g^0(i)\| \le 1 + 2\eta$.

We now assume for r and prove for $r+1$. For the faults in an $(r+1)$-rectangle *not* to be $(r+1, k_0)$-sparse, there must be at least k_0+1 r-rectangles in which the fault is not (r, k_0)-sparse. So by the induction assumption on both $\mathcal{L}_b(i)$ and $\mathcal{L}_g(i)$

$$\begin{aligned} \|\mathcal{L}_b^{r+1}(i)\| &\le \begin{pmatrix} A \\ k+1 \end{pmatrix}((2\eta)^{(1+\delta)^r})^{k+1}(1+(2\eta)^{(1+\delta)^r})^{A-k-1} \\ &\le e\begin{pmatrix} A \\ k+1 \end{pmatrix}((2\eta)^{(1+\delta)^r})^{k+1}, \end{aligned} \tag{8.10}$$

where we have used the fact that $(1+(2\eta)^{(1+\delta)^r})^{A-k-1} < e$, since $(2\eta)^{(1+\delta)^r} < 2\eta$ and $2\eta A \le 1$. The right-hand side is $\le (2\eta)^{(1+\delta)^{r+1}}$ by using the fact that $(2\eta)^{(1+\delta)^r} < 2\eta$ and the fact that the threshold condition of (8.6) is satisfied for $\eta < \eta_c'$. This proves the induction step for $\mathcal{L}_b(i)$. By using $\|\mathcal{L}_g(i) + \mathcal{L}_b(i)\| = 1$ we can prove the induction step also for $\mathcal{L}_g(i)$. To prove the statement, we consider bad fault paths, i.e., at least one r-rectangle is bad. If there are v rectangles, we have

$$\|\mathcal{L}_b\| \le v \cdot (1 + (2\eta)^{(1+\delta)^r})^{v-1}(2\eta)^{(1+\delta)^r}. \tag{8.11}$$

Setting $r = \text{polyloglog}(\frac{v}{\epsilon})$ gives the desired result. ∎

8.3. The good part: Sparse fault paths give almost correct outputs.

LEMMA 13. *Let $\eta < \eta_c', \epsilon > 0$. Let M_0 have v locations. Let P_i be the probability to measure the string i in M_0 (operating without noise). Let $r = \text{polyloglog}(\frac{v}{\epsilon})$, as in Lemma* 12*, and let M_r be defined as before. We consider the operator $\mathcal{L}_g$ as in* (8.4)*, and $\rho(0)$ as the input density matrix of M_r. Then*

$$\left|\sum_{j \mapsto i}[\mathcal{L}_g \cdot \rho(0)]_{j,j} - P_i\right| \le \epsilon,$$

where $j \mapsto i$ means that taking recursive majority on the string j results in the string i.

Proof. The measurement of the output qubits of any density matrix of nm^r qubits induces some probability distribution over n-bit strings by taking the recursive majority. For any density matrix ρ which is $(r, q/2)$-deviated from the correct final density matrix of M_r, this probability distribution is the same one as the output distribution of M_0. In other words, the probability for each n-bit string is the same as that of M_0.

Since $\mathcal{L}_g$ corresponds to sparse fault paths, we would now like to apply Lemma 9 to show that $\mathcal{L}_g\rho(0)$ is $(r, q/2)$-deviated from the correct final matrix. However,

since we are dealing with general noise, the matrix $\mathcal{L}_g\rho(0)$ is not necessarily a density matrix or even a positive-semidefinite matrix. This is because the operators $\mathcal{E}'_{A_{i,t}}$ that it involves are not necessarily completely positive.

To bypass this technical issue, we recall that

$$\mathcal{E}'_{A_{i,t}} = \mathcal{E}_{A_{i,t}} - (1-\eta)I, \tag{8.12}$$

which is a weighted sum of physically admissible operators ($\mathcal{E}_{A_{i,t}}$ and I). Hence, we can write $\mathcal{L}_g$ as a weighted sum of (r, k_0)-sparse fault paths that *do* correspond to physically admissible operators:

$$\mathcal{L}_g \cdot \rho(0) = \sum_f \lambda_f \mathcal{L}_f \cdot \rho(0), \tag{8.13}$$

where each term in the above sum $\mathcal{L}_f \cdot \rho(0)$ corresponds to an evolution of the initial density matrix subject to physically admissible faults operating in a set of locations which is (r, k_0)-sparse. Lemma 9 thus applies to each term in the sum. We get that each density matrix $\mathcal{L}_f \cdot \rho(0)$ is $(r, q/2)$-deviated from correct. This means that

$$\sum_{j\mapsto i}[\mathcal{L}_g \cdot \rho(0)]_{j,j} = \sum_f \lambda_f \sum_{j\mapsto i}[\mathcal{L}_f \cdot \rho(0)]_{j,j} = \left(\sum_f \lambda_f\right) P_i. \tag{8.14}$$

However, by using (8.13), we have

$$\left|\sum_f \lambda_f - 1\right| = |\operatorname{Tr}(\mathcal{L}_g\cdot\rho(0)) - 1| = |\operatorname{Tr}((\mathcal{L}_g+\mathcal{L}_b)\cdot\rho(0)) - 1 - \operatorname{Tr}(\mathcal{L}_b\cdot\rho(0))| = |\operatorname{Tr}(\mathcal{L}_b\cdot\rho(0))|. \tag{8.15}$$

By using Property 1 of the superoperator norm (section 2) and Lemma 12, we have

$$|\operatorname{Tr}(\mathcal{L}_b \cdot \rho(0))| \le \|\mathcal{L}_b\|\|\rho(0)\| \le \epsilon, \tag{8.16}$$

which completes the proof. □

8.4. The threshold theorem for general noise. We can now prove the threshold result for general noise.

THEOREM 13 (the threshold theorem for general noise). *Let $\epsilon > 0$. Let C be a computation code with gates $\mathcal{G}$. There exist a threshold $\eta'_c > 0$ and constants c_1, c_2, c_3 such that the following holds. Let Q be a quantum circuit, operating on n qubits for t time steps, which uses s gates from $\mathcal{G}$, and has v locations. There exists a quantum circuit Q' which operates on $n\log^{c_1}(\frac{v}{\epsilon})$ qubits, for time $t\log^{c_2}(\frac{v}{\epsilon})$, and uses $v\log^{c_3}(\frac{v}{\epsilon})$ gates from $\mathcal{G}$ such that, in the presence of general noise with error rate $\eta < \eta'_c$, Q' computes a function which is within ϵ total variation distance from that computed by Q.*

Proof. We construct Q' to be the r-simulation of Q, where r is chosen such that the requirements of Lemma 12 are satisfied with $\epsilon' = \epsilon/2$. We now estimate the total variation distance between the output distribution of Q' (after taking recursive majority) and that of Q. As before, we denote the probability to measure i at the

output of Q by P_i. The total variation distance is then

(8.17)
$$\frac{1}{2}\sum_i \left| \sum_{j\mapsto i} \rho(t)_{j,j} - P_i \right| = \frac{1}{2}\sum_i \left| \sum_{j\mapsto i} ([\mathcal{L}_b \cdot \rho(0)]_{j,j} + [\mathcal{L}_g \cdot \rho(0)]_{j,j}) - P_i \right|$$
$$\leq \frac{1}{2}\left(\sum_i \left| \sum_{j\mapsto i} [\mathcal{L}_b \cdot \rho(0)]_{j,j} \right| + \sum_i \left| \sum_{j\mapsto i} [\mathcal{L}_g \cdot \rho(0)]_{j,j} - P_i \right| \right).$$

Lemma 13 implies that the second term is at most ϵ'. The first term is bounded from above by the trace norm of $\mathcal{L}_b \cdot \rho(0)$ and hence, by Lemma 7.6, by ϵ'. The sum of the two gives $2\epsilon' = \epsilon$. □

9. Fault tolerance with any universal set of gates. So far, the reliable circuits which we have constructed can use only universal set of gates associated with a quantum computation code, such as the sets $\mathcal{G}_1$ and $\mathcal{G}_2$. This is an undesirable situation, both theoretically and practically. Theoretically, we would like to be able to show that the fault-tolerance result is robust, meaning that fault-tolerant quantum computation can be performed regardless of the universal set of gates which we use. Practically, it is likely that the sets of gates $\mathcal{G}_1$ or $\mathcal{G}_2$ are difficult to implement in the laboratory. We would like to be able to implement fault-tolerant quantum computation by using the gates that are most readily available to us. Indeed, in this section we provide the desired generalization and show that the threshold result holds for any universal set of gates $\mathcal{G}$. In other words, starting from a quantum circuit which uses an arbitrary universal set of gates $\mathcal{K}$, we can implement it fault-tolerantly with any universal set of gates $\mathcal{G}$ of our choice. We require only that $\mathcal{G}$ contains a gate which discards a qubit (qudit) and a gate which adds a blank qubit (qudit) to the circuit.

The idea of the proof is that we design the final circuit in three stages: We start from the original circuit which uses gates from $\mathcal{K}$ and simulate it by a circuit which uses one of the sets of gates for which a quantum computation code exists, e.g., $\mathcal{G}_1$. We then apply our fault-tolerant construction and get a circuit M_r which uses once again gates from the same set associated with the computation code, say, $\mathcal{G}_1$. Finally, we replace every gate g in M_r by a sequence of gates from $\mathcal{G}$ which approximates the gate g to within a constant μ. The final circuit is denoted M'_r. We now prove that this construction works, if μ is chosen correctly.

In the following, we assume that we use the fault-tolerant construction with the set of gates $\mathcal{G}_1$. The discussion can be easily changed to use the gate set $\mathcal{G}_2$.

DEFINITION 25. *For any gate g in $\mathcal{G}_1$, consider the smallest circuit (in terms of number of locations) that uses gates from $\mathcal{G}$ and approximates g to within accuracy μ. Denote this circuit by $Q(g,\mu)$. Let $S(\mu)$ be the maximum number of locations in $Q(g,\mu)$ taken over all g in $\mathcal{G}_1$.*

CLAIM 9. *Fix a gate $g \in \mathcal{G}_1$. Consider the superoperator $Q'(g,\mu)$ which corresponds to applying the circuit $Q(g,\mu)$ in the presence of general noise of error rate η. We claim that, if $\eta < 1/2S(\mu)$, $\|g - Q'(g,\mu)\| \leq 3S(\mu)\eta + \mu$.*

Proof. By the definition of $Q(g,\mu)$, it suffices to prove that $\|Q(g,\mu) - Q'(g,\mu)\| \leq 3S(\mu)\eta$. We write down the superoperator corresponding to $Q'(g,\mu)$, as is done in

(8.3). By using the triangle inequality, and (2.4), we have

$$\begin{aligned}\|Q(g,\mu)-Q'(g,\mu)\| &\le S(\mu)(2\eta)(1-\eta)^{S(\mu)-1} \\ &\quad + \binom{S(\mu)}{2}(2\eta)^2(1-\eta)^{S(\mu)-2}+\cdots+(2\eta)^{S(\mu)} \\ &= (1+\eta)^{S(\mu)}-(1-\eta)^{S(\mu)}.\end{aligned} \tag{9.1}$$

Simple combinatorics now implies the result, taking into account that $S(\mu)\eta \le 0.5$. ∎

We view the circuit M'_r as composed of generalized locations, where each generalized location corresponds to the set of locations arising from the approximation of one gate in M_r. Let us compare M'_r in the presence of noise, viewed in this resolution of generalized locations, to the circuit M_r operating without noise. Claim 9 implies that at each generalized location the error in M'_r, compared to the correct gate in M_r, is at most $3S(\mu)\eta+\mu$. We would like to apply our result from section 8, and so we need to require that $3S(\mu)\eta+\mu<\eta'_c$, the threshold for general noise. By optimizing for μ we get the following definition for the threshold:

$$\eta''_c = \min\left\{\max_{\mu<\eta'_c}\frac{\eta'_c-\mu}{3S(\mu)}, \frac{1}{2S(\mu)}\right\}. \tag{9.2}$$

If η is smaller than this threshold, then Claim 9 applies, and we get that the effective error rate $\eta_{\text{eff}} = 3S(\mu)\eta+\mu$ is indeed below the threshold for general noise. This allows us to prove the threshold theorem by using any universal set of gates in the presence of general noise.

THEOREM 14 (the threshold result in full generality). *Let $\epsilon>0$. Let $\mathcal{K}$ and $\mathcal{G}$ be two universal sets of quantum gates. There exist a constant $\eta''_c>0$ and constants c_1,c_2,c_3 such that the following holds. Given any quantum circuit Q with n input qubits, which operates for t time steps, uses s gates from $\mathcal{K}$, and has v locations, there exists a corresponding quantum circuit Q' which operates on $nO(\log^{c_1}(\frac{v}{\epsilon}))$ qubits, for time $tO(\log^{c_2}(\frac{v}{\epsilon}))$, and uses $vO(\log^{c_3}(\frac{v}{\epsilon}))$ gates from $\mathcal{G}$ such that, in the presence of general noise with error rate $\eta<\eta''_c$, Q' computes a function which is within ϵ total variation distance from the function computed by Q.*

Proof. We first approximate Q by a circuit M_0 which uses only gates from the set of gates $\mathcal{G}_1$ of a computation code C. Our new circuit M_0 computes the same function as Q up to total variation distance $\epsilon/2$. We do this by approximating every one of the s gates in Q by a sequence of gates from $\mathcal{G}$ that approximates the gate to within ϵ/s. Due to the Kitaev–Solovay theorem (see section 6), the number of gates required to replace each gate in Q is $\text{polylog}(s/\epsilon)$.

We now construct M_r, the r-simulation of M_0, which again uses gates from $\mathcal{G}_1$. This is done as in section 7, except that the choice of r is taken to suit the error rate $\eta_{\text{eff}} = 3S(\mu_0)\eta+\mu_0$.

To construct Q', we replace each location in M_r by a circuit of $S(\mu_0)$ locations, by using gates from $\mathcal{G}$ that approximate the gate performed in the location to within μ_0.

We would now like to prove that the final circuit is fault-tolerant against general noise of error rate $\eta<\eta''_c$ by using essentially the same lines as in section 8. To be able to apply this proof, we do the following. We group the locations in the final circuit M'_r to groups of $S(\mu_0)$ locations, where each group corresponds to the location from which it originated in M_r. Consider the superoperator $\mathcal{L}$ associated with the application of the gates and noise operators in one generalized location corresponding

to the gate g. By Claim 9, $\|\mathcal{L}-g\| \leq \eta_{\text{eff}}$. Hence, $\|g^{-1}\mathcal{L}-I\| \leq \eta_{\text{eff}}$. We can therefore consider an equivalent circuit to M'_r, where for each generalized location, described by a superoperator $\mathcal{L}$ which consists of a sequence of gates and noise operators, we instead put the gate g followed by a noise operator $g^{-1}\mathcal{L}$. The proof now follows from the proof of Theorem 13, by using $\ell = S(\mu_0)$ for the spread of the procedures. □

The threshold value we get for arbitrary sets of gates (9.2) is of course worse than the threshold values for the fault-tolerant constructions that use gates of computation codes. The value of the threshold depends on the exact set of gates $\mathcal{G}$, because it depends on the number of gates required to approximate the gates of the computation code.

10. Robustness against exponentially decaying correlations. We would now like to show how the above results hold also in the case of exponentially decaying correlations between the noise processes, in both space and time.

10.1. Adding correlations to the noise model. Two very important assumptions were made when introducing our general model of noise.

- Locality: No correlations between environments of different qubits, except through the gates.
- The Markovian assumption: The environment is renewed at each time step, and hence no correlations between the environments at different time steps.

Both of these assumptions can be slightly released, to allow exponentially decaying correlations in both space and time, while the results of this paper still hold.

To add exponentially decaying correlations to the probabilistic noise model, we generalize it in the following way. Instead of considering independent probabilities for error in each location, we require that the probability for a fault path which contains k locations is bounded by some constant times the probability for the same fault path in the independent errors model:

$$\Pr(\text{fault path with } k \text{ errors}) \leq c\eta^k(1-\eta)^{v-k}, \tag{10.1}$$

where v is the number of locations in the circuit. Then an adversary chooses a noise operator which operates on all of the qubits in the fault path, without any restrictions. The most general case is as follows. The adversary adds some blank qubits, which are called the environment, at the beginning of the computation. At each time step, the adversary can operate a general operator on the environment and the set of qubits in the fault path at that time step. This model allows correlations in space, since the noise operator need not be of the form of a tensor product of operators on different locations. Correlations in time appear because the environment that the adversary added in is not renewed at each time step, so noise operators of different time steps are correlated. Note that the independent probabilistic noise process is a special case of this process.

10.2. Proof of the threshold theorem with exponentially decaying correlations. We observe that all of the lemmas that we have used in order to prove the threshold theorem for probabilistic noise hold in this case, except for one step which fails. It is the step that shows that bad fault paths are rare, in the case of probabilistic noise (Lemma 11). The proof of this lemma relies on the independence of faults. We observe that the proof of Lemma 11 is actually a union bound. In this section we show that the same threshold as is used for probabilistic noise (Definition 23) guarantees that the bad fault paths are rare also in the presence of probabilistic noise with exponentially decaying correlations.

We use a union bound argument, as follows. Consider fault paths in v r-rectangles. If a fault path is bad, there must be at least one r-rectangle in which it is bad, namely, not (r, k_0)-sparse. Let us concentrate on this r-rectangle. We first count the number of bad fault paths in this r-rectangle that have a minimal number of faulty locations. To do this, denote by F_j the number of minimal fault paths in a j-rectangle. We have

$$F_1 \leq \begin{pmatrix} A \\ k_0+1 \end{pmatrix} \tag{10.2}$$

and

$$F_{j+1} \leq \begin{pmatrix} A \\ k_0+1 \end{pmatrix} F_j^{k_0+1}. \tag{10.3}$$

We can solve the recursion to get

$$F_r \leq \begin{pmatrix} A \\ k_0+1 \end{pmatrix}^{\frac{(k_0+1)^r-1}{k_0}}. \tag{10.4}$$

A minimal bad fault path contains exactly $(k_0+1)^r$ locations. Now, v r-rectangles contain vA^r locations. We can bound the number of bad fault paths in v r-rectangles, that consist of $(k_0+1)^r + i$ locations, as follows. We first choose one r-rectangle (this gives a factor of v). In this rectangle we pick one of the possible minimal bad fault paths (this gives a factor of F_r). We can then choose the rest of the locations arbitrarily. This gives that the number of bad fault paths in v r-rectangles consisting of $(k_0+1)^r + i$ locations is at most

$$v \begin{pmatrix} A \\ k_0+1 \end{pmatrix}^{\frac{(k_0+1)^r-1}{k_0}} \begin{pmatrix} vA^r - (k_0+1)^r \\ i \end{pmatrix}. \tag{10.5}$$

By using (10.1), we have that the overall probability of the bad fault paths is at most

(10.6)

$$\begin{aligned} &\sum_{i=0}^{vA^r-(k_0+1)^r} v \binom{A}{k_0+1}^{\frac{(k_0+1)^r-1}{k_0}} \binom{vA^r-(k_0+1)^r}{i} c\eta^{(k_0+1)^r+i}(1-\eta)^{vA^r-(k_0+1)^r-i} \\ &\leq cv \left(\binom{A}{k_0+1}^{\frac{1}{k_0}} \eta \right)^{(k_0+1)^r} \sum_{i=0}^{vA^r-(k_0+1)^r} \binom{vA^r-(k_0+1)^r}{i} \eta^i (1-\eta)^{vA^r-(k_0+1)^r-i} \\ &= cv \left(\binom{A}{k_0+1}^{\frac{1}{k_0}} \eta \right)^{(k_0+1)^r}. \end{aligned}$$

The expression above decays exponentially fast to zero with r, if η is strictly below the threshold for probabilistic noise (Definition 23). This completes the proof. □

11. Fault tolerance in a *d*-dimensional quantum computer. We now proceed to our final generalization of the threshold theorem. So far, we allowed a gate to operate on any set of qubits, regardless of the actual location of these qubits in space. Here we consider quantum systems with geometrical constraints: The qubits

are embedded in space, in a one-, two-, or three-dimensional grid, and gates can be applied only to nearest-neighbor qubits. In the case of dimensionality higher than 1, and gates of more than two qubits, we also require that all qubits in a gate lie on the same line. We show that the threshold result holds in full generality for d-dimensional quantum computers, for any $d \geq 1$.

In physical systems with geometrical constraints, discarding and adding qubits is a questionable process, unless we allow empty spots. To avoid this complication, we replace these two operations by one gate called RESTART, which is constructed by discarding a qubit and then adding a blank qubit instead of the discarded qubit. This allows us to remove entropy out of the system while maintaining the geometrical structure. Since the gates are restricted to operate on nearest neighbors, we also need to require that the set of gates we use contains the SWAP gate, so that, when we want to apply a gate on qubits that are not nearest neighbors, we can bring them closer together.

THEOREM 15 (threshold theorem for d-dimensional circuits). *Let $\epsilon > 0$. Let $d \geq 1$. Let $\mathcal{G}'$ and $\mathcal{G}''$ be two universal sets of quantum gates. There exist a threshold $\eta_c''' > 0$ and constants c_1, c_2, c_3 such that the following holds. Let Q' be a d-dimensional quantum circuit, with n input qubits, which operates t time steps, uses s gates from $\mathcal{G}'$, and has v locations. There exists a d-dimensional quantum circuit Q'' which operates on $nO(\log^{c_1}(\frac{v}{\epsilon}))$ qubits, for time $tO(\log^{c_2}(\frac{v}{\epsilon}))$, and uses $vO(\log^{c_3}(\frac{v}{\epsilon}))$ gates from $\mathcal{G}'' \cup \{\mathsf{SWAP}, \mathsf{RESTART}\}$ such that, in the presence of general noise with error rate $\eta < \eta_c'''$, Q'' computes a function which is ϵ-close to that computed by Q'.*

Proof. To prove the theorem, we need to modify the fault-tolerant procedures so that they apply gates on nearest neighbors only and also all of the qubits remain in the circuit throughout the computation.

Here is how one level of the simulation is done. We first pick a preferred direction, and each qubit will be extended to an array of qubits lying in that direction. The simulation will "stretch" the simulated circuit only in the preferred direction, by a constant factor. Let a be the maximal number of ancilla qubits used in any of the procedures of the computation code $\mathcal{G}$. Let m be the size of the block. A qubit in Q' will be replaced by $m + a$ qubits, placed in a line along the preferred direction. The ancilla qubits will serve as a working space, but we will also SWAP computational qubits with ancilla qubits, in order to bring computation qubits closer and operate gates on them.

The fault-tolerant procedures are thus modified as follows. First, instead of adding ancilla qubits during the procedure, we use only the ancilla qubits that are already there and apply a RESTART gate on an ancilla qubit one step before we use it in the procedure. Also, any gate g in the original procedure that operates on qubits that are far apart is replaced by a sequence of SWAP gates which bring the qubits that g operates on to nearest neighbor sites, followed by g, followed by another sequence of SWAP gates which bring the qubits back to their original sites. Since the simulated circuit Q' applies gates only on nearest neighbors, say, on $s \in \{1, 2, 3\}$ qubits in a row, the number of SWAP gates used in the above modification per gate g is at most $2(s-1)(m+a)$, i.e., a constant. This means that the modified procedure is larger than the original one by a constant factor (both in time and in space).

The claim is that the procedure is still fault-tolerant. This might seem strange since the SWAP gates operate on many qubits and seem to help in the propagation of errors. However, note that a SWAP gate which operates on a faulty qubit and an unaffected qubit does not propagate the error to the two qubits but keeps it confined to the original qubit, which is now in a new site. Hence, a SWAP gate which is not

faulty does not cause a propagation of error. If a fault does occur in a SWAP gate, then the two qubits participating in it are contaminated. If the fault occurred before the application of the gate g, then one of the qubits the gate g operates on is faulty, and hence this also causes the contamination of all of the qubits on which g operates (in the worst case). So an error in a SWAP gate is equivalent to an error in all of the original sites of the qubits participating in the gate and also the final site of the other qubit participating in the SWAP gate. This adds a factor of 2 at most to the original spread of the procedure. All other aspects of the theorem remain the same. □

12. Threshold estimations. Finally, it is left to estimate the exact threshold value for fault-tolerant quantum computation. However, there is not one such value: This value depends on many things such as which variants of the threshold theorem we use (probabilistic noise, general local noise, d-dimensional circuits) as well as the choice of the computation code and, mainly, the exact assumptions we make on the quantum system, most importantly, whether we allow classical computation in the middle of the quantum process or not. Since our results are mainly proofs of existence, where we have not attempted to optimize the threshold, we do not attempt to provide exact values of the threshold in all of these cases. Nevertheless, we give here a rough estimation of the threshold value in one of the simpler cases: the case of probabilistic independent noise, with no geometrical constraints on the system, and under the assumption that infinitely fast classical operations are allowed during the computation.

To estimate this threshold, we examine the formula for the threshold value in the case of independent probabilistic noise with no geometrical constraints, which is given by Definition 23. The threshold value depends on two parameters: A, the number of locations in the biggest rectangle in the simulation, and $k_0 = \lfloor q/2\ell \rfloor$, where we recall that q is the number of errors that the code can correct, and ℓ is its spread, which is 1 in both our constructions. We use polynomial codes of degree $d = 4$, i.e., length $m = 13$, so that $q = 2$, and so $k_0 = 1$. The threshold in our case is thus $\binom{A}{2}^{-1}$.

The parameter A, the size of the largest rectangle, is estimated by counting the number of locations where quantum (rather than classical) gates and qubits are involved. The bottleneck in our construction using polynomial codes, namely, the largest rectangle, is the one corresponding to applying transversal operation on two blocks, preceded by error correction on both of these blocks. The reason for this is as follows. First, we need to consider only two-qubit encoded gates because we use the squaring gate instead of the Toffoli gate in the set $\mathcal{G}_1$ (see subsection 4.2). Second, one might think that the largest rectangle involves the ancilla state preparation for the degree reduction as well. However, to avoid such large rectangles, we consider each step in the ancilla preparation as a rectangle of its own.

It remains to estimate the size of the above rectangle. For each error correction we need two zero-state preparations, and so altogether we need four zero-state preparations plus transversal operations. Since we use $q = 2$, every zero-state preparation requires 25 preliminary zero-state preparations (not done fault-tolerantly) plus transversal operations. We estimate the size of the rectangle in this case to be of the order of 10^3 locations, which implies a threshold of the order of $\simeq 10^{-6}$.

The threshold value in other cases, and under different sets of assumptions on the noise and on the constraints of the quantum system, can be estimated by using (8.6), (9.2), and the definition of η_c''' as is defined in section 11, where the estimation of the parameters involved should depend on the assumptions being used.

Acknowledgments. We thank Peter Shor for discussions about his result, Thomas Beth for helping to construct the fault-tolerant Toffoli gate without measurements for the CSS codes, Richard Cleve for asking the question of nearest neighbors and suggesting the solution, Noam Nisan for a fruitful discussion regarding the correct model to use, and Prasad Gospel, Denis Gaitsgori, Erez Lapid, and Bob Solovay for helpful discussions regarding the proofs of universality. We are grateful to Hoi-Kwong Lo for correcting an error in the definition of polynomial codes. Finally, we thank Alesha Kitaev for a very helpful remark regarding the generalization to the general noise model and Daniel Gottesman for very helpful remarks and corrections regarding the final draft of this paper.

REFERENCES

[1] D. Aharonov, *Quantum Computation*, Annual Reviews of Computational Physics 6, D. Stauffer, ed., World Scientific, Singapore, 1998.

[2] D. Aharonov, *A quantum to classical phase transition in noisy quantum computers*, Phys. Rev. A, 62 (2000), 062311.

[3] D. Aharonov and M. Ben-Or, *Fault-tolerant quantum computation with constant error*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), El Paso, TX, 1997, pp. 176–188.

[4] D. Aharonov and M. Ben-Or, *Fault-Tolerant Quantum Computation with Constant Error Rate*, preliminary extended version of [3], available online at http://arxiv.org/abs/quant-ph/9906129 (1999).

[5] D. Aharonov and M. Ben-Or, *Polynomial simulations of decohered quantum computers*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 1996, pp. 46–55.

[6] D. Aharonov, A. Y. Kitaev, and N. Nisan, *Quantum circuits with mixed states*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), Dallas, TX, 1998, pp. 20–30.

[7] D. Aharonov, M. Ben-Or, R. Impagliazzo, and N. Nisan, *Limitations of Noisy Reversible Computation*, available online at http://arxiv.org/abs/quant-ph/9611028 (1996).

[8] D. Aharonov and D. Gottesman, *Accuracy Thresholds: Can We Beat* 10^{-4}*?* Presentation at ITP Conference of Quantum Information, 2001, available online at http://online.itp.ucsb.edu/online/qinfo_c01/aharonov/.

[9] D. Aharonov, A. Y. Kitaev, and J. Preskill, *Fault tolerant quantum computation with long-range correlated noise*, Phys. Rev. Lett., 96 (2006), 050504.

[10] C. S. Ahn, *Extending Quantum Error Correction: New Continuous Measurement Protocols and Improved Fault-Tolerant Overhead*, Ph.D. thesis, Caltech, Pasadena, CA, 2004.

[11] R. Alicki, M. Horodecki, P. Horodecki, and R. Horodecki, *Dynamical description of quantum computing: Generic nonlocality of quantum noise*, Phys. Rev. A, 65 (2002), 062101.

[12] R. Alicki, D. A. Lidar, and P. Zanardi, *Internal consistency of fault-tolerant quantum error correction in light of rigorous derivations of the quantum Markovian limit*, Phys. Rev. A, 73 (2006), 052311.

[13] P. Aliferis and A. Cross, *Subsystem fault tolerance with the Bacon–Shor code*, Phys. Rev. Lett., 98 (2007), 220502.

[14] P. Aliferis, D. Gottesman, and J. Preskill, *Quantum accuracy threshold for concatenated distance-3 codes*, Quant. Inf. Comput., 6 (2006), pp. 97–165.

[15] P. Aliferis, D. Gottesman, and J. Preskill, *Accuracy threshold for postselected quantum computation*, Quant. Inf. Comput., 8 (2008), pp. 181–244.

[16] D. Bacon, *Operator quantum error correcting subsystems for self-correcting quantum memories*, Phys. Rev. A, 73 (2006), 012340.

[17] D. Bacon, J. Kempe, D. Lidar, and B. K. Whaley, *Universal fault-tolerant computation on decoherence-free subspaces*, Phys. Rev. Lett., 85 (2000), pp. 1758–1761.

[18] J. Kempe, D. Bacon, D. Lidar, and B. K. Whaley, *Theory of decoherence-free fault-tolerant universal quantum computation*, Phys. Rev. A, 63 (2001), 042307.

[19] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A, 52 (1995), pp. 3457–3467.

[20] A. Barenco, A. Ekert, K. A. Suominen, and P. Torma, *Approximate quantum Fourier transform and decoherence*, Phys. Rev. A, 54 (1996), pp. 139–146.
[21] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523.
[22] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, *Mixed state entanglement and quantum error correction*, Phys. Rev. A, 54 (1996), pp. 3824–3851.
[23] M. Ben-Or and R. Cleve, *Computing algebraic formulas using a constant number of registers*, SIAM J. Comput., 21 (1992), pp. 54–58.
[24] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness theorems for fault-tolerant distributed computing*, in Proceedings for the 20th Annual ACM Symposium on Theory of Computing (STOC), Chicago, 1988, pp. 1–10.
[25] A. Berthiaume and G. Brassard, *Oracle quantum computing*, in Proceedings of the Workshop on Physics of Computation: PhysComp '92, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 60–62.
[26] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, *A new universal and fault-tolerant quantum basis*, Inform. Process. Lett., 75 (2000), pp. 101–107.
[27] S. Bravyi and A. Y. Kitaev, *Universal quantum computation with ideal Clifford gates and noisy ancillas*, Phys. Rev. A, 71 (2005), 022316.
[28] A. R. Calderbank and P. W. Shor, *Good quantum error-correcting codes exist*, Phys. Rev. A, 54 (1996), pp. 1098–1105.
[29] M. D. Choi, *Completely positive linear maps on complex matrices*, Linear Algebra Appl., 10 (1975), pp. 285–290.
[30] I. L. Chuang, R. Laflamme, P. W. Shor, and W. H. Zurek, *Quantum computers, factoring, and decoherence*, Science, 270 (1995), pp. 1633–1635.
[31] I. L. Chuang, W. C. D. Leung, and Y. Yamamoto, *Bosonic quantum codes for amplitude damping*, Phys. Rev. A, 56 (1997), pp. 1114–1125.
[32] B. Cirel'son (Tsirelson), *Reliable Storage of Information in a System of Unreliable Components with Local Interactions*, Lecture Notes in Math. 653, Springer-Verlag, New York, 1978, pp. 15–30.
[33] R. Cleve, D. Gottesman, and H.-K. Lo, *How to share a quantum secret*, Phys. Rev. Lett., 83 (1999), pp. 648–651.
[34] C. Cohen-Tanoudji, *Quantum Mechanics*, Wiley, New York, 1977.
[35] D. Coppersmith and E. Grossman, *Generators for certain alternating groups with applications to cryptography*, SIAM J. Appl. Math., 29 (1975), pp. 624–627.
[36] C. Crépeau, D. Gottesman, and A. Smith, *Secure multi-party quantum computation*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 2002, pp. 643–652.
[37] E. Dennis, A. Y. Kitaev, A. Landahl, and J. Preskill, *Topological quantum memory*, J. Math. Phys., 43 (2002), pp. 4452–4505.
[38] D. Deutsch, *Quantum theory, the Church-Turing principle, and the universal quantum computer*, Proc. R. Soc. Lond. Ser. A, 400 (1985), pp. 97–117.
[39] D. Deutsch, *Quantum computational networks*, Proc. R. Soc. Lond. Ser. A, 425 (1989), pp. 73–90.
[40] D. P. DiVincenzo, *Two-bit gates are universal for quantum computation*, Phys. Rev. A, 51 (1995), pp. 1015–1022.
[41] L.-M. Duan and G.-C. Guo, *Reducing decoherence in quantum computer memory with all quantum bits coupling to the same environment*, Phys. Rev. A, 57 (1998), pp. 737–741.
[42] P. Gács, *Self correcting two dimensional arrays*, in Advances in Computing Research: A Research Annual: Randomness and Computation, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 223–326 (see, in particular, pp. 240–241 and 246–248).
[43] C. W. Gardiner, *Quantum Noise*, Springer, Berlin, 1991.
[44] D. Gottesman, *A theory of fault-tolerant quantum computation*, Phys. Rev. A, 57 (1998), pp. 127–137.
[45] D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, Ph.D. thesis, Caltech, Pasadena, CA; available online at http://arxiv.org/abs/quant-ph/9705052 (1997).
[46] D. Gottesman, *Fault-tolerant quantum computation with higher-dimensional systems*, Chaos Solitons Fractals, 10 (1999), pp. 1749–1758.
[47] D. Gottesman, *Fault-tolerant quantum computation with local gates*, J. Modern Opt., 47 (2000), pp. 333–345.
[48] D. Gottesman and I. Chuang, *Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations,* Nature, 402 (1999), pp. 390–393.
[49] L. A. Khalfin and B. S. Tsirelson, *Quantum/classical correspondence in the light of Bell's inequalities*, Found. Phys. 22 (1992), pp. 879–948.

[50] A. Y. Kitaev, *Quantum computations: Algorithms and error corrections*, Russian Math. Surveys, 52 (1997), pp. 1191–1249.
[51] A. Y. Kitaev, *Fault-tolerant quantum computation by anyons*, Ann. Physics, 303 (2003), pp. 2–30.
[52] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classical and Quantum Computation*, Grad. Stud. Math. 47, American Mathematical Society, Providence, RI, 2002.
[53] E. Knill, *Non-Binary Unitary Error Bases and Quantum Codes*, available online at http://arxiv.org/abs/quant-ph/9608048 (1996).
[54] E. Knill, *Quantum computing with very noisy devices*, Nature, 434 (2005), pp. 30–44.
[55] E. Knill and L. Viola, *Theory of quantum error correction for general noise*, Phys. Rev. Lett., 84 (2000), pp. 2525–2528.
[56] E. Knill, R. Laflamme, and W. Zurek, *Resilient quantum computation*, Science, 279 (1998), pp. 342–345.
[57] E. Knill, R. Laflamme, and W. H. Zurek, *Resilient Quantum Computation: Error Models and Thresholds*, available online at http://arxiv.org/abs/quant-ph/9702058 (1997).
[58] E. Knill and R. Laflamme, *Concatenated Quantum Codes*, available online at http://arxiv.org/abs/quant-ph/9608012 (1996).
[59] D. W. Kribs, R. Laflamme, D. Poulin, and M. Lesosky, *Operator quantum error correction*, Quantum. Inf. Comput., 6 (2006), pp. 383–399.
[60] K. Hellwig and K. Kraus, *Operations and measurements*. II, Comm. Math. Phys. 16 (1970), pp. 142–147.
[61] K. Kraus, *States, Effects and Operations. Fundamental Notions of Quantum Theory*, Springer-Verlag, Berlin, 1983.
[62] R. Landauer, *Is quantum mechanics useful?*, Philos. Trans. Roy. Soc. London Ser. A, 353 (1995), pp. 367–376.
[63] D. A. Lidar, L. C. Isaac, and B. K. Whaley, *Decoherence free subspaces for quantum computation*, Phys. Rev. Lett., 81 (1998), pp. 2594–2597.
[64] J. H. van Lint, *An Introduction to Coding Theory*, 2nd ed., Springer-Verlag, New York, 1992.
[65] C. Miquel, J. P. Paz, and R. Perazzo, *Factoring in a dissipative quantum computer*, Phys. Rev. A, 54 (1996), pp. 2605–2613.
[66] C. Miquel, J. P. Paz, and W. H. Zurek, *Quantum computation with phase drift errors*, Phys. Rev. Lett., 78 (1997), pp. 3971–3974.
[67] J. von Neumann, *Probabilistic logic and the synthesis of reliable organisms from unreliable components*, in Automata Studies, C. E. Shanon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, 1956, pp. 43–98.
[68] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[69] A. Osterloh, L. Amico, G. Falci, and R. Fazio, *Scaling of entanglement close to a quantum phase transition*, Nature, 416 (2002), pp. 608–610.
[70] T. Osborne and M. Nielsen, *Entanglement in a simple quantum phase transition*, Phys. Rev. A, 66 (2002), 032110.
[71] G. M. Palma , K.-A. Suominen, and A. K. Ekert, *Quantum computers and dissipation*, Proc. R. Soc. Lond. Ser. A, 452 (1996), pp. 567–584.
[72] A. Peres, *Quantum Theory: Concepts and Methods*, Kluwer Academic Press, Dordrecht, The Netherlands, 1993.
[73] J. Preskill, *Fault tolerant quantum computation*, in Introduction to Quantum Computation and Information, H.-K. Lo, S. Popescu, and T. P. Spiller, eds., World Scientific, River Edge, NJ, 1998, pp. 213–269.
[74] J. Preskill, *Reliable quantum computation*, Proc. Roy. Soc. London Ser. A, 454 (1998), pp. 385–410.
[75] R. Raussendorf and J. Harrington, *Fault tolerant quantum computation with high threshold in two dimensions*, Phys. Rev. Lett., 98 (2007), 190504.
[76] B. W. Reichardt, *Error-Detection-Based Quantum Fault Tolerance Against Discrete Pauli Noise*, Ph.D. thesis, University of California, Berkeley, CA, 2006. Available online at http://arxiv.org/abs/quant-ph/0612004.
[77] B. W. Reichardt, *Improved Ancilla Preparation Scheme Increases Fault-Tolerant Threshold*; available online at http://arxiv.org/abs/quant-ph/0406025 (2004).
[78] B. W. Reichardt, *Threshold for a Distance-Three Quantum Code*; available online at http://arxiv.org/abs/quant-ph/0509203 (2005).
[79] J. J. Saqurai, *Modern Quantum Mechanics*, revised ed., Addison–Wesley, Reading, MA, 1994.
[80] B. W. Schumacher and M. A. Nielsen, *Quantum data processing and error correction*, Phys. Rev. A, 54 (1996), pp. 2629–2635.

[81] A. Shamir, *How to share a secret*, Comm. ACM, 22 (1979), pp. 612–613.
[82] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[83] P. W. Shor, *Scheme for reducing decoherence in quantum computer memory*, Phys. Rev. A, 52 (1995), pp. 2493–2496.
[84] P. W. Shor, *Fault tolerant quantum computation*, in Proceedings of the 37th Symposium on the Foundations of Computer Science, IEEE Press, Los Alamitos, CA, 1996, pp. 56–65.
[85] R. Solovay and A. C.-C. Yao, manuscript, 1996.
[86] D. A. Spielman, *Highly fault-tolerant parallel computation*, in Proceedings of the 37th Annual IEEE Conference on Foundations of Computer Science, 1996, pp. 154–163.
[87] A. Steane, *Quantum computing and error correction*, in Decoherence and Its Implications in Quantum Computation and Information Transfer, A. Gonis and P. E. A. Turchi, eds., IOS Press, Amsterdam, 2001, pp. 284–298.
[88] A. Steane, *Error correcting codes in quantum theory*, Phys. Rev. Lett., 77 (1996), pp. 793–797.
[89] A. Steane, *Multiple particle interference and quantum error correction*, Proc. R. Soc. Lond. Ser. A, 452 (1996), pp. 2551–2577.
[90] A. Steane, *Active stabilisation, quantum computation and quantum state synthesis*, Phys. Rev. Lett., 78 (1997), pp. 2252–2255.
[91] A. M. Steane, *Space, time, parallelism and noise requirements for reliable quantum computing*, Fortschr. Phys., 46 (1998), pp. 443–458.
[92] A. M. Stephens, A. G. Fowler, and L. C. L. Hollenberg, *Universal fault tolerant quantum computation on bilinear nearest neighbor arrays*, Quant. Inf. Comput., 8 (2008), p. 330–344.
[93] A. Stern, Y. Aharonov, and Y. Imry, *Phase uncertainty and loss of interference: A general picture*, Phys. Rev. A, 41 (1990), pp. 3436–3448.
[94] A. Stern, Y. Aharonov, and Y. Imry, *Dephasing of interference by a back reacting environment*, in Quantum Coherence, J. Anandan, ed., World Scientific, Singapore, 1991, p. 201.
[95] K. M. Svore, D. P. Divincenzo, and B. M. Terhal, *Noise threshold for a fault-tolerant two-dimensional lattice architecture*, Quantum Inf. Comput., 7 (2007), pp. 297–318.
[96] B. Terhal and G. Burkard, *Fault-tolerant quantum computation for local non-Markovian noise*, Phys. Rev. A, 71 (2005), 012336.
[97] W. G. Unruh, *Maintaining coherence in quantum computers*, Phys. Rev. A, 51 (1995), pp. 992–997.
[98] L. C. Washington, *Introduction to Cyclotomic Fields*, Grad. Texts in Math. 83, Springer-Verlag, New York, 1982.
[99] W. K. Wootters and W. H. Zurek, *A single quantum cannot be cloned*, Nature, 299 (1982), pp. 802–803.
[100] A. C.-C. Yao, *Quantum circuit complexity*, Proceedings of the 33th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Los Alamitos, CA, 1993, pp. 352–361.
[101] P. Zanardi, *Dissipation and decoherence in a quantum register*, Phys. Rev. A, 57 (1998), pp. 3276–3284.
[102] P. Zanardi and M. Rasseti, *Noiseless quantum codes*, Phys. Rev. Lett., 79 (1997), pp. 3306–3309.
[103] W. H. Zurek, *Decoherence and the transition from quantum to classical*, Physics Today, 44 (1991), pp. 36–44.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1283–1299

MITOTIC CLASSES IN INDUCTIVE INFERENCE*

SANJAY JAIN† AND FRANK STEPHAN‡

Abstract. For the natural notion of splitting classes into two disjoint subclasses via a recursive classifier working on texts, the question of how these splittings can look in the case of learnable classes is addressed. Here the strength of the classes is compared using the strong and weak reducibility from intrinsic complexity. It is shown that, for explanatorily learnable classes, the complete classes are also mitotic with respect to weak and strong reducibility, respectively. But there is a weakly complete class that cannot be split into two classes which are of the same complexity with respect to strong reducibility. It is shown that, for complete classes for behaviorally correct learning, one-half of each splitting is complete for this learning notion as well. Furthermore, it is shown that explanatorily learnable and recursively enumerable classes always have a splitting into two incomparable classes; this gives an inductive inference counterpart of the Sacks splitting theorem from recursion theory.

Key words. recursion theory, inductive inference, mitotic classes, reducibilities

AMS subject classifications. 68Q32, 03D25, 03D30

DOI. 10.1137/070700577

1. Introduction.

A well-known observation is that infinite sets can be split into two parts of the same cardinality as the original set, while finite sets cannot be split in such a way; for example, the integers can be split into the sets of the even and odd numbers, while splitting a set of five elements would result in subsets of unequal sizes. In this sense, infinite sets are better than finite ones. The corresponding question in complexity and recursion theory is the following: Which sets can be split into two sets of the same complexity as the original set [1, 9, 10, 15]?

Ambos-Spies [1] defined one of the variants of mitoticity using many-one reducibilities. Here a set A is many-one reducible to a set B iff there is a recursive function f such that $A(x) = B(f(x))$. That is, one translates every input x for A into an input $f(x)$ for B, takes the solution provided by B (in the set or in its complement), and then copies this to obtain the solution for A. Similarly one also considers complexity-theoretic counterparts of many-one reductions; for example, one can translate an instance (G_1, G_2) of the graph-isomorphism problem into an instance ϕ of the satisfiability problem in polynomial time, where G_1 is isomorphic to G_2 iff ϕ is satisfiable. Here, one can choose the reduction such that one not only tests membership, but can also translate a solution of ϕ into an isomorphism between G_1 and G_2 whenever such a solution exists for ϕ. Indeed, NP-complete problems are characterized as those into which every NP problem can be translated. This general method of reducing problems and translating solutions (although here the translation of the solution is just the identity) occurs quite frequently in other fields of mathematics. In inductive inference, intrinsic complexity is based on the notion of reducing one learning problem $\mathcal{L}$ to another problem $\mathcal{H}$: First an operator translates a text T for a set L in $\mathcal{L}$ into a text $\Theta(T)$ for a set H in $\mathcal{H}$, and then another operator translates a solution E,

*Received by the editors August 20, 2007; accepted for publication (in revised form) April 3, 2008; published electronically July 23, 2008. This work was supported in part by NUS grants R252-000-212-112 and R252-000-308-112.

http://www.siam.org/journals/sicomp/38-4/70057.html

†School of Computing, National University of Singapore, Singapore 117590, Singapore (sanjay@comp.nus.edu.sg).

‡Department of Mathematics and School of Computing, National University of Singapore, Singapore 117543, Singapore (fstephan@comp.nus.edu.sg).

which is a sequence converging to an index e of H, into a solution for L given as a sequence converging to an index e' of L. Before explaining this in more detail, some terminology is necessary to make it precise.

- A partial recursive function is a partial function computed by a Turing machine, where the machine does not halt on inputs on which the function is undefined. A recursively enumerable set is the domain (or, equivalently, the range) of a partial recursive function. There is an acceptable numbering $W_0, W_1, W_2, \ldots$ of all recursively enumerable sets [16, section II.5]; this numbering will be kept fixed from now on.
- A general recursive operator Θ is a mapping from total functions to total functions such that there is a recursively enumerable set E of triples which satisfies the following: For every total function f and every x, y, $\Theta(f)(x) = y$ iff there is an n such that $(f(0)f(1)\ldots f(n), x, y) \in E$.
- A language is a recursively enumerable subset of the natural numbers.
- A class $\mathcal{L}$ is a set of languages. A family $L_0, L_1, L_2, \ldots$ is an indexing for $\mathcal{L}$ iff $\{(e, x) : x \in L_e\}$ is recursively enumerable and $\mathcal{L} = \{L_0, L_1, L_2, \ldots\}$.
- A text T (see [11]) is a mapping from the set $\mathbb{N}$ of natural numbers to $\mathbb{N} \cup \{\#\}$. Content of a text T, content(T), is the set $\{T(n) \mid n \in \mathbb{N} \wedge T(n) \in \mathbb{N}\}$. T is a text for L iff content$(T) = L$. $T[n]$ denotes the first n elements of the sequence T, that is, $T[n] = T(0)\,T(1)\ldots T(n-1)$. Furthermore, content$(T[n])$ denotes the set of natural numbers occurring in $T(0)\,T(1)\ldots T(n-1)$. As an example, $T = \#\#3\#\#8\#8\#\#7\#9\#^\infty$ is a text for the set $\{3, 7, 8, 9\}$, $T(0) = \#$, $T(2) = 3$, and $T[6] = \#\#3\#\#8$.
- A learner is a general recursive operator (see [22]) which translates T into another sequence E. The learner converges on T iff there is a single e such that $E(n) = e$ for almost all n—in this case one says that the learner converges on T to e. The learner explanatorily learns (see [5, 11]) T iff it converges on T to some index e such that W_e = content(T). A learner explanatorily learns L iff it explanatorily learns every text for L. A learner explanatorily learns $\mathcal{L}$ iff it explanatorily learns every $L \in \mathcal{L}$. Note that, in some cases, learning algorithms can also be described such that they use the indices from a given indexing for $\mathcal{L}$; such indices can always be translated into indices of the acceptable numbering $W_0, W_1, \ldots$ of all recursively enumerable sets.
- A classifier [21] is a general recursive operator which translates texts to sequences over $\{0, 1\}$. A classifier C converges on a text T to a iff $C(T[n]) = a$ for almost all n.
- For the learning criteria considered in this paper, one can assume without loss of generality [17] that the learner M computes $E(n)$ from input $T[n]$. $M(T[n])$ denotes this hypothesis. A similar convention holds for classifiers.

As an example, the class $\mathcal{D}$ of all finite sets is explanatorily learnable. The following learner explanatorily learns $\mathcal{D}$: On every finite sequence σ it outputs a canonical index for content(σ), so $M(\#\#3\#\#8\#8)$ outputs an index for $\{3, 8\}$. Then it is easy to see that M converges on every text for a finite set to the canonical index for this set. An example for a nonexplanatorily learnable class was obtained by Gold [11] who showed that $\mathcal{D} \cup \{L\}$ is not explanatorily learnable for any infinite set L. Let $\mathcal{D}_{even}$ and $\mathcal{D}_{odd}$ be the classes of finite sets with an even and odd number of elements, respectively. Then one can easily construct a classifier which separates $\mathcal{D}_{odd}$ from $\mathcal{D}_{even}$ by defining that $M(\sigma) = 1$ if content(σ) has an even number of elements and $M(\sigma) = 0$ otherwise.

Freivalds, Kinber, and Smith [6] consider reductions between learnability prob-

lems for function classes. Jain and Sharma [13] carried this idea over to the field of learning languages from positive data and formalized the following two reducibilities for learnability problems. The main difference between these two reducibilities is that Θ can be one-to-many in the case of the weak reducibility, as different texts for the same language can be mapped to texts for different languages, while for the strong reducibility this is not allowed, at least for languages in the given class.

- A class $\mathcal{L}$ is weakly reducible to $\mathcal{H}$ iff there are general recursive operators Θ and Ψ such that
  - whenever T is a text for a language in $\mathcal{L}$, $\Theta(T)$ is a text for a language in $\mathcal{H}$;
  - whenever E is a sequence which converges to an index e with $W_e = \text{content}(\Theta(T))$ for some text T of a language in $\mathcal{L}$, $\Psi(E)$ is a sequence converging to an e' with $W_{e'} = \text{content}(T)$.

  One writes $\mathcal{L} \leq_{weak} \mathcal{H}$ in this case.
- A class $\mathcal{L}$ is strongly reducible to $\mathcal{H}$ iff there are general recursive operators Θ, Ψ as above with the following additional constraint. Whenever T, T' are texts for the same language in $\mathcal{L}$, $\Theta(T), \Theta(T')$ are texts for the same language in $\mathcal{H}$. One writes $\mathcal{L} \leq_{strong} \mathcal{H}$ in this case. Furthermore, $\Theta(L)$ denotes the language content$(\Theta(T))$, where T is a text for L.

For example, the class $\{\{x\} : x \in \mathbb{N}\}$ of all singleton sets is strongly reducible to $\mathcal{D}$, but $\mathcal{D}$ is not weakly reducible to the class of all singleton sets [13]. Furthermore, $\mathcal{D}_{even}$ and $\mathcal{D}_{odd}$ are strongly reducible to each other by the same mapping (Θ, Ψ). The operator Θ translates every text for a set D to a text for $\{0\} \cup \{d+1 : d \in D\}$; the operator Ψ translates an index e, in a sequence of hypotheses, to an index $g(e)$ of the set $W_{g(e)} = \{x : x+1 \in W_e\}$.

Jain, Kinber, Sharma, and Wiehagen investigated these concepts in several papers [12, 13, 14]. They found that there are complete classes with respect to $\leq_{weak}$ and $\leq_{strong}$. Here a class $\mathcal{H}$ is complete with respect to $\leq_{weak}$ ($\leq_{strong}$) iff $\mathcal{H}$ can be explanatorily learned and for every explanatorily learnable class $\mathcal{L}$ it holds that $\mathcal{L} \leq_{weak} \mathcal{H}$ ($\mathcal{L} \leq_{strong} \mathcal{H}$). If $\sqsubset$ is a recursive dense linear ordering on $\mathbb{N}$ without least and greatest element (which makes $\mathbb{N}$ an order-isomorphic copy of the rationals), then

$$\mathcal{Q} = \{\, \{y \in \mathbb{N} \mid y \sqsubseteq x\} \mid x \in \mathbb{N}\}$$

is a class which is complete for both $\leq_{weak}$ and $\leq_{strong}$ (see [12]). The following classes are complete for $\leq_{weak}$ but not for $\leq_{strong}$ (see [13]):

$$\mathcal{I} = \{\, \{0, 1, \ldots, x\} \mid x \in \mathbb{N}\},$$
$$\mathcal{CS} = \{\, \mathbb{N} - \{x\} \mid x \in \mathbb{N}\}.$$

If one looks at $\mathcal{CS}$, one can easily see that it is the disjoint union of two classes of equivalent intrinsic complexity, namely, the class $\{\, \mathbb{N}-\{x\} \mid x \text{ is even}\}$ and $\{\, \mathbb{N}-\{x\} \mid x$ is odd$\}$. All three classes can be translated into each other, and a classifier can witness the splitting: If T is a text for a member of $\mathcal{CS}$, then the classifier converges in the limit to the remainder of x divided by 2 for the unique $x \notin \text{content}(T)$. This type of splitting can be formalized to the notion of a mitotic class.

DEFINITION 1. *Two infinite classes $\mathcal{L}_0$ and $\mathcal{L}_1$ are called a* splitting *of a class $\mathcal{L}$ iff $\mathcal{L}_0 \cup \mathcal{L}_1 = \mathcal{L}$, $\mathcal{L}_0 \cap \mathcal{L}_1 = \emptyset$, and there exists a classifier C such that, for all $a \in \{0,1\}$ and for all texts T with content$(T) \in \mathcal{L}_a$, C converges on T to a.*

A class $\mathcal{L}$ is strongly mitotic (weakly mitotic) *iff there is a splitting $\mathcal{L}_0, \mathcal{L}_1$ of $\mathcal{L}$ such that $\mathcal{L} \equiv_{strong} \mathcal{L}_0 \equiv_{strong} \mathcal{L}_1$ ($\mathcal{L} \equiv_{weak} \mathcal{L}_0 \equiv_{weak} \mathcal{L}_1$).*

The study of such notions is motivated by recursion theory [16, 22] where a recursively enumerable set is called mitotic iff it is the disjoint union of two other recursively enumerable sets which have the same Turing degree. The importance of this notion is reflected by Ladner's result that a recursively enumerable set is mitotic iff it is autoreducible, that is, iff there is an oracle Turing machine M such that $A(x) = M^{A\cup\{x\}}(x)$ for all x [15]. Furthermore the notion had been carried over to complexity theory where it is still an important research topic [1, 9, 10]. Thus, we feel it is interesting to explore this notion in the context of inductive inference too. The results show some interesting properties related to mitoticity. In particular, classes complete for explanatory learning with respect to $\leq_{strong}$ ($\leq_{weak}$) are strongly (weakly) mitotic. Thus, nonmitotic classes cannot be complete.

Although intrinsic complexity is not the exact counterpart of Turing degrees in recursion theory, it is the only type of complexity which is defined via reducibilities and not via measures such as counting mind changes or the size of long-term memory in inductive inference. Therefore, from the viewpoint of inductive inference, the above defined version of mitotic classes is reasonable. Indeed, there are some obvious parallels: In recursion theory, any recursively enumerable cylinder is mitotic, where a cylinder A is a set of the form $\{(x,y) \mid x \in B, y \in \mathbb{N}\}$ for some set $B \subseteq \mathbb{N}$. A corresponding cylindrification of a class $\mathcal{L}$ would be the class

$$\{\,\{(x,y) \mid y \in L\} \mid x \in \mathbb{N}, L \in \mathcal{L}\}.$$

It can easily be seen that this class is strongly mitotic and thus also weakly mitotic. Indeed, two constraints are placed in Definition 1 in order to be as near to the original definition of mitoticity in recursion theory as possible:

- In the recursion theoretic setting, if A is split into two recursively enumerable sets A_0, A_1 with $A_0 \equiv_T A_1$, then $A \equiv_T A_0 \equiv_T A_1$. Thus, in the definition of mitoticity for learning theory, it is required that all three classes, class $\mathcal{L}$ and its two halves $\mathcal{L}_1$ and $\mathcal{L}_2$, have the same intrinsic complexity degree.
- If A is recursively enumerable and mitotic and split into A_0, A_1, then there is a partial recursive function with domain A mapping the elements of A_a to a for all $a \in \{0,1\}$. For mitotic classes of languages, the corresponding function is a classifier which works correctly on all texts for the languages in the class. It is not required that the classifier converges on every text, as then many naturally strongly mitotic classes, such as $\mathcal{CS}$, would no longer be mitotic. This has a parallel in recursion theory: if one splits a maximal set (as defined in Remark 8 below) into two recursively enumerable sets A_0 and A_1, which are both not recursive, then the sets A_0 and A_1 are recursively inseparable.

Besides the reducibilities $\leq_{weak}$ and $\leq_{strong}$ considered here, other reducibilities have also been considered [12, 13]. This paper deals only with $\leq_{weak}$ and $\leq_{strong}$, as these two are the most natural and representative.

One emphasis of the current work is on the search for natural classes which split or do not split. Therefore it is always required that the class under consideration is learnable (under the criterion in consideration). Furthermore, one tries to show properties for complete classes, recursively enumerable classes, and indexed families. Angluin [2] defined that $\{L_0, L_1, L_2, \ldots\}$ is an indexed family iff the function $e, x \mapsto L_e(x)$ is recursive. For indexed families $\{L_0, L_1, L_2, \ldots\}$ one can assume, without loss of generality, that $L_n \neq L_m$ whenever $n \neq m$. A learner for this family is called exact iff it converges on every text for L_n to n.

In this paper, it is shown that classes complete for explanatory learning with respect to $\leq_{strong}$ ($\leq_{weak}$) are strongly (weakly) mitotic. Furthermore, there are classes

complete for explanatory learning with respect to $\leq_{weak}$ which are not strongly mitotic. Similar phenomena are studied for behaviorally correct learning. A counterpart to the Sacks splitting theorem is that every infinite recursively enumerable explanatory learnable class has a splitting into two incomparable subclasses. The relations between autoreducibility and mitoticity are also investigated. However, in contrast to the situation in complexity theory, these two notions turn out to be different.

The following remark is important for several proofs.

Remark 2. One says that a learner M or a classifier C converges on T to a value a iff $M(T[n]) = a$ or $C(T[n]) = a$ for almost all n, respectively. But it does not matter—in the framework of inductive inference—how fast this convergence is; the machine can be slowed down by starting with an arbitrary guess and later repeating hypotheses, if needed. Similarly, if one translates one text of a language L into a text of a language H, it is not important how fast the symbols of H show up in the translated text; it is only important that they show up eventually. Therefore the translator can put into the translated text a pause symbol, #, until more data are available or certain simulated computations have terminated.

Therefore, learners, operators translating texts, and classifiers can be made primitive recursive by the just mentioned delaying techniques. Thus one can have recursive enumerations $\Theta_0, \Theta_1, \Theta_2, \ldots$ of translators from texts to texts, $M_0, M_1, M_2, \ldots$ of learners, and $C_0, C_1, C_2, \ldots$ of classifiers such that, for every given translator, learner or classifier, this list contains an equivalent one. These lists can be used in proofs where diagonalizations are needed.

Given a text T and a number n, one denotes by $\Theta(T[n])$ the initial part $\Theta(T)[m]$ for the largest $m \leq n$ such that $\Theta(T)[m]$ is produced without accessing any datum in T beyond the nth position. Note that, for every m, there is an n such that $\Theta(T[n])$ extends $\Theta(T)[m]$ and $\Theta(T[n])$ can be computed from $T[n]$.

2. Complete classes. The two main results are that classes which are complete for $\leq_{strong}$ are strongly mitotic and classes which are complete for $\leq_{weak}$ are weakly mitotic. This stands in contrast to the situation in recursion theory where some Turing-complete recursively enumerable sets are not mitotic [15]. Note that certain classes which are complete only for $\leq_{weak}$ fail to be strongly mitotic; thus the main results cannot be improved.

THEOREM 3. *Every class which is complete for $\leq_{strong}$ is also strongly mitotic.*

Proof. Let $\mathcal{L}$ and $\mathcal{H}$ be any classes which are complete for $\leq_{strong}$. Then the class $\mathcal{K}$ consisting of the sets $I = \{1, 3, 5, 7, \ldots\}$, $J = \{0\} \cup I$, $\{2x + 3 : x \in H\}$, and $J \cup \{2x+2 : x \in H\}$ for every $H \in \mathcal{H}$ is also complete for $\leq_{strong}$. Since $\mathcal{L}$ is complete for $\leq_{strong}$, there is a translation Θ which maps languages in $\mathcal{K}$ to languages in $\mathcal{L}$ such that proper inclusion is preserved [14]. Thus there is some $e \in \Theta(J) - \Theta(I)$. As $\mathcal{H}$ is complete for $\leq_{strong}$, the subclasses $\{\Theta(\{2x + 3 : x \in H\}) : H \in \mathcal{H}\}$ and $\{\Theta(J \cup \{2x + 2 : x \in H\}) : H \in \mathcal{H}\}$ of $\mathcal{L}$ are also complete for $\leq_{strong}$. All members of the first class do not contain e, while all members of the second class contain e as an element. It follows that the subclasses $\mathcal{L}_0 = \{L \in \mathcal{L} : e \notin L\}$ and $\mathcal{L}_1 = \{L \in \mathcal{L} : e \in L\}$ are disjoint and complete for $\leq_{strong}$. Thus $\mathcal{L}$ can be classified by a classifier C, which conjectures 1 if e has shown up in the text so far and 0 otherwise. Therefore, $\mathcal{L}$ is strongly mitotic. ∎

The following notion is used to formulate Proposition 6 which is a central ingredient of Theorem 7. Furthermore, learners with certain properties are needed.

DEFINITION 4. *For any sequence T of symbols, let all(T) be the length of the*

shortest prefix of T containing all symbols which show up in T; that is, let

$$all(T) = \sup\{n+1 : content(T[n]) \subset content(T)\}.$$

Note that $all(T) < \infty$ iff $content(T)$ is a finite set.

The following remark combines some ideas of Blum and Blum [4] and Fulk [8].

Remark 5. Let $\mathcal{L}$ be an explanatorily learnable class. Then there is an explanatory learner M for $\mathcal{L}$ with the following properties:

- M is prudent [17]; that is, whenever M outputs an index e on some input data, then M explanatorily learns W_e.
- M is order independent [4]; that is, for every set L, either M diverges on all texts for L or M converges on all texts for L to the same index.
- For every text T and index e, if $M(T[n]) = e$ for infinitely many n, then $M(T[n]) = e$ for almost all n.

PROPOSITION 6. *Suppose $\mathcal{I} \leq_{weak} \mathcal{L}$ and M is a explanatory learner for $\mathcal{L}$ which satisfies the conditions in Remark* 5. *Then there is a reduction (Θ, Ψ) from $\mathcal{I}$ to $\mathcal{L}$ such that, for all texts T for a language in $\mathcal{I}$, M converges on $\Theta(T)$ to an index $e > all(T)$.*

Proof. Assume that (Θ'', Ψ'') witnesses that $\mathcal{I} \leq_{weak} \mathcal{L}$. Now a reduction (Θ', Ψ') from $\mathcal{I}$ to $\mathcal{I}$ is constructed such that Θ can be taken to be the composition of Θ' and Θ''.

The key idea for this is the following: One constructs (Θ', Ψ') from $\mathcal{I}$ to $\mathcal{I}$ such that, for every text T of a set in $\mathcal{I}$, M converges on $\Theta''(\Theta'(T))$ to an index $e > \text{all}(T)$. Note that $\text{all}(T)$ is finite for all texts for members of $\mathcal{I}$. By Remark 2, assume without loss of generality that Θ'' is primitive recursive. The idea is that Θ' translates a text T for $I_n = \{0, 1, \ldots, n\}$ to a text for $I_{2^n(1+2m)}$ for some m; Ψ' translates any sequence converging to an index of the set $I_{2^n(1+2m)}$ into a sequence converging to an index of I_n.

Given a sequence E of indices, $\Psi'(E)(s)$ is computed as follows. Let k be the least number such that $W_{E(s),s} \subseteq I_k$. Choose m, n such that $2^n(1+2m) = k$ and output the canonical index for I_n. It is easy to see that this translation works whenever E converges to an index of some set in $\mathcal{I}$.

CONSTRUCTION OF Θ'. The construction of Θ' is more involved. For the construction, the special properties of M from Remark 5 are important. The most adequate way to describe $\Theta'(T)$ is to build longer and longer finite prefixes $\tau_0, \tau_1, \tau_2, \ldots$ of this target $\Theta'(T)$. The construction starts with $\tau_0 = 0\#$, and, in stage s, the extension τ_{s+1} of τ_s is defined according to the first case which applies:

Case 1. $M(\Theta''(\tau_s)) \leq \text{all}(T[s])$. Then let τ_{s+1} be the first extension of τ_s found such that $M(\Theta''(\tau_{s+1})) \neq M(\Theta''(\tau_s))$.

Case 2. Case 1 does not hold, but $\text{content}(\tau_s) \neq I_{2^n(1+2m)}$ for all m, where n is the least number with $\text{content}(T[s]) \subseteq I_n$. Then let $\tau_{s+1} = \tau_s a$ for the least nonelement a of $\text{content}(\tau_s)$.

Case 3. Cases 1 and 2 do not hold. Then $\tau_{s+1} = \tau_s\#$.

Here $\Theta''(\tau_s)$ and $\Theta''(\tau_{s+1})$ are defined as in Remark 2 and can be computed from τ_s and τ_{s+1}, respectively.

VERIFICATION. For the verification, assume that a set $I_n = \{0, 1, 2, \ldots, n\} \in \mathcal{I}$ and a text T for I_n are given.

First note that, in Case 1 of the construction, the extension τ_{s+1} of τ_s can always be found. To see this, note that there are two texts T_1, T_2 extending τ_s for different sets in $\mathcal{I}$. It follows that $\Theta''(T_1)$ and $\Theta''(T_2)$ are texts of different sets and thus M

converges on them to different indices. Thus one can take a sufficiently long prefix of one of T_1, T_2 in order to get the desired τ_{s+1}.

Second, it can be shown by induction that $|\tau_s| > s$ at all stages s; this guarantees that Θ' is indeed a general recursive operator.

Third, one shows that M does not converge on $\Theta''(\Theta'(T))$ to any index less than or equal to all(T). By Case 1 in the construction, M cannot converge on $\Theta''(\Theta'(T))$ to an index $e \leq \text{all}(T)$. Thus, by Remark 5, there is a stage s_0 such that Case 1 of the construction is never taken after stage s_0 and content$(T[s_0]) = I_n$.

Fourth, one shows that $\Theta'(T)$ is a text for some language in $\mathcal{I}$. There is a least m such that content$(\tau_{s_0}) \subseteq I_{2^n(1+2m)}$. For all stages $s > s_0$, if content$(\tau_s) \subset I_{2^n(1+2m)}$, then τ_{s+1} is chosen by Case 2; else τ_{s+1} is chosen by Case 3. One can easily see that the resulting text $\Theta'(T) = \lim_{s\to\infty} \tau_s$ is a text for $I_{2^n(1+2m)}$. Indeed, $\Theta'(T) = \tau_{s_1}\#^\infty$ for $s_1 = s_0 + 2^n(1+2m) + 2$.

Thus it follows that Θ' maps every text of a set I_n to a text for some set $I_{2^n(1+2m)}$, as desired. So, for all texts T of sets in $\mathcal{I}$, M converges on $\Theta''(\Theta'(T))$ to some index $e > \text{all}(T)$. □

THEOREM 7. *Let $\mathcal{L}$ be an explanatorily learnable class which is complete for $\leq_{weak}$. Then $\mathcal{L}$ is weakly mitotic.*

Proof. Let $I_n = \{0, 1, \ldots, n\}$. By Proposition 6 there is a reduction (Θ, Ψ) from $\mathcal{I}$ to $\mathcal{L}$ and a learner M such that M satisfies the conditions outlined in Remark 5 and, for every text T of a member of $\mathcal{I}$, M converges on $\Theta(T)$ to an index $e > \text{all}(T)$. For this reason, using oracle K for the halting problem, one can check, for every index e, whether there is a text T for a language in $\mathcal{I}$ such that M on $\Theta(T)$ converges to e. This can be seen as follows: One can assume, without loss of generality, that, besides $\#$, no data item in a text is repeated. Also, among the texts for sets in $\mathcal{I}$, only the texts of the sets $\{0\}$, $\{0,1\}$, $\{0,1,2\}$, $\ldots$, $\{0,1,2,\ldots,e\}$ can satisfy all$(T) \leq e$. Thus, one has just to check the behavior of the given explanatory learner M for the class $\mathcal{L}$ on the texts $\Theta(T')$, for T' in the class

$$\mathcal{T}_e = \{T' \mid T' \in \{0, 1, \ldots, e, \#\}^e \cdot \#^\infty \wedge \text{content}(T) \in \mathcal{I}\}.$$

Now, define a classifier C such that on a text T, the nth guess of C is 1 iff there is an odd number $m \leq M(T[n])$ and a text $T'' \in \mathcal{T}_{M(T[n])}$ for I_m such that $M(\Theta(T'')[n]) = M(T[n])$.

For the verification that C is a classifier, assume that M converges on T to some index e. Then C converges on T to 1 iff there is an odd number m and a text T' for I_m in $\mathcal{T}_e$ such that M converges on the texts T and $\Theta(T')$ to the same number. Otherwise C converges on T to 0. If M does not converge on T, then T is not a text for a set in $\mathcal{L}$, and the behavior of C on T is irrelevant. Thus C is a classifier which splits $\mathcal{L}$ into two classes $\mathcal{L}_0$ and $\mathcal{L}_1$. These classes $\mathcal{L}_0$ and $\mathcal{L}_1$ contain the images of repetition-free texts of sets in the classes $\{I_0, I_2, I_4, \ldots\}$ and $\{I_1, I_3, I_5, \ldots\}$, respectively. Thus both classes are complete for $\leq_{weak}$, and the splitting of $\mathcal{L}$ into $\mathcal{L}_0$ and $\mathcal{L}_1$ witnesses that $\mathcal{L}$ is weakly mitotic. □

As several proofs use known properties of the maximal sets, the following remark summarizes some of these properties.

Remark 8. A set A is *maximal* (see [16]) iff (a) A is recursively enumerable, (b) A has an infinite complement, and (c) every recursively enumerable set B satisfies that either $B - A$ is finite or the complement of $A \cup B$ is finite.

A maximal set is dense simple; that is, if $a_0, a_1, a_2, \ldots$ gives the complement in the ascending order and f is a recursive function, then $f(a_n) < a_{n+1}$ for almost all n.

For any partial recursive function ψ and any maximal set A, the following statements hold.

- $\psi(x)$ is defined either for almost all $x \in \overline{A}$ or for only finitely many $x \in \overline{A}$.
- The set $\{x \notin A \mid \psi(x) \in A\}$ either is finite or contains almost all elements of $\overline{A}$.
- If, for every x, there is some $y > x$ such that $y \notin A$, $\psi(y)$ is defined, $\psi(y) > x$, and $\psi(y) \notin A$, then $\psi(z)$ is defined and $\psi(z) = z$ for almost all $z \in \overline{A}$.

These basic facts about the maximal sets will be used in several proofs. Odifreddi [16, pp. 288–294] provides more information on the maximal sets including the proof of their existence by Friedberg [7].

THEOREM 9. *There exists an indexed family $\{L_0, L_1, L_2, \ldots\}$ which is weakly mitotic and complete for $\leq_{weak}$, but not strongly mitotic.*

Proof. Let A be a maximal set with complement $\{a_0, a_1, \ldots\}$, where $a_n < a_{n+1}$ for all n. Now let $\mathcal{L}$ consist of the sets

- $\{x, x+1, x+2, \ldots, x+y\}$ for all $x \in A$ and $y \in \mathbb{N}$,
- $\{x, x+1, x+2, \ldots\}$ for all $x \notin A$.

As A is recursively enumerable, $\mathcal{L}$ can be represented as an indexed family. Explanatory learnability is also clear as the learner, on input σ, first determines $x = \min(\text{content}(\sigma))$ and then conjectures content(σ) if $x \in A_{|\sigma|}$, and conjectures $\{x, x+1, x+2, \ldots\}$ otherwise. Without loss of generality, it can be assumed that $a_0 > 0$ and thus $\mathcal{L}$ is a superclass of $\mathcal{I}$ and therefore complete for $\leq_{weak}$. By Theorem 7, $\mathcal{L}$ is weakly mitotic.

Let $\mathcal{L}_0$ and $\mathcal{L}_1$ be two disjoint classes with union $\mathcal{L}$. Without loss of generality, $\{a_0, a_0+1, a_0+2, \ldots\} \in \mathcal{L}_1$. Assume now by way of contradiction that $\mathcal{L} \leq_{strong} \mathcal{L}_0$ as witnessed by (Θ, Ψ). As Θ has to preserve the proper subset relation on the content of the texts while translating, every text of a set of the form $\{a_n, a_n+1, a_n+2, \ldots\}$ has to be translated into a text for a set of the form $\{a_m, a_m+1, a_m+2, \ldots\}$ (to preserve the property that translation of $\{a_n, a_n+1, a_n+2, \ldots\}$ has infinitely many subsets in the class).

Now consider the function f which outputs, on input x, the first element found to be in the range of the image $\Theta(\sigma)$ for some σ with $x = \min(\text{content}(\sigma))$. The function f is recursive, but by Remark 8 and A being a maximal set, the relation $f(a_n) < a_{n+1}$ holds for almost all n. It follows that, if n is sufficiently large, then some text of $\{a_n, a_n+1, a_n+2, \ldots\}$ is translated to a text of one of the sets $\{a_k, a_k+1, a_k+2, \ldots\}$ with $k \leq n$. Now fix a text T for $\{a_n, a_n+1, a_n+2, \ldots\}$. One can then inductively define a sequence of strings $\sigma_n, \sigma_{n-1}, \ldots, \sigma_0$ such that each sequence $\sigma_n\sigma_{n-1}\ldots\sigma_m T$ is a text for $\{a_m, a_m+1, a_m+2, \ldots\}$ and

$$\text{content}(\Theta(\sigma_n\sigma_{n-1}\ldots\sigma_m\sigma_{m-1})) \not\subseteq \text{content}(\Theta(\sigma_n\sigma_{n-1}\ldots\sigma_m T))$$

for each $m \leq n$. As Θ maps texts of infinite sets in $\mathcal{L}$ to texts of infinite sets in $\mathcal{L}$, one can conclude that

$$\text{content}(\Theta(\sigma_n\sigma_{n-1}\ldots\sigma_m T)) = \{a_m, a_m+1, a_m+2, \ldots\}.$$

Thus, for every m, some text of the set $\{a_m, a_m+1, a_m+2, \ldots\}$ is mapped to a text for the same set, contradicting the assumption that Θ does not have $\{a_0, a_0+1, a_0+2, \ldots\}$ in its range. Therefore $\mathcal{L}$ is not strongly mitotic. □

3. Incomplete learnable classes. Finite classes are not mitotic, and thus every nonempty class has a subclass which is not mitotic. For infinite classes, one can get

that the corresponding subclass is also infinite. The proof is a standard application of Ramsey's theorem: Given classifiers $C_0, C_1, C_2, \ldots$, one selects a subclass $\{H_0, H_1, H_2, \ldots\}$ of $\{L_0, L_1, L_2, \ldots\}$ such that each classifier C_n classifies $H_n, H_{n+1}, H_{n+2}, \ldots$ in the same way. The class $\{H_0, H_1, H_2, \ldots\}$ may not be an indexed family but may be a very thin class in the sense that the indices of H_n with respect to $L_0, L_1, L_2, \ldots$ are growing very fast. Alternatively, one can also take the $H_0, H_1, H_2, \ldots$ such that, for a given enumeration of primitive recursive operators, the text $\Theta_n(T_m)$ of the ascending text T_m of H_m is not a text for any H_k with $k > \max(\{n, m\})$. The latter method gives the following result.

THEOREM 10. *Every infinite class $\mathcal{L}$ has an infinite subclass $\mathcal{H}$ such that $\mathcal{H}$ is not weakly reducible to any proper subclass of $\mathcal{H}$. In particular, $\mathcal{H}$ is not weakly mitotic.*

There is an easier example of a class which is not weakly mitotic. It is even an indexed family consisting only of finite sets.

Example 11. Assume that $\{L_0, L_1, L_2, \ldots\}$ is given as $L_0 = \{0, 1\}$ and $L_n = \{n\}$ for all $n \in \mathbb{N} - \{0\}$. Then $\{L_0, L_1, L_2, \ldots\}$ is not weakly mitotic.

Proof. Given any splitting $\mathcal{L}_0, \mathcal{L}_1$ of $\{L_0, L_1, L_2, \ldots\}$, one of these classes, say $\mathcal{L}_0$, contains at most one of the sets L_0, L_1. Then, for any given reduction (Θ, Ψ) from $\{L_0, L_1, L_2, \ldots\}$ to $\mathcal{L}_0$, $\Theta(\sigma)$ produces some string of nonempty content for some $\sigma \in 1\#^*$. Thus there are texts T_0, T_1 extending σ for L_0 and L_1, respectively, such that $\Theta(T_0)$ and $\Theta(T_1)$ are texts for different sets in $\mathcal{L}_0$ with a nonempty intersection. However, such sets do not exist, by choice of $\mathcal{L}_0$. □

Note that the class

$$\{\{0, 1, 2\}, \{1, 2\}, \{2\}, \{3\}, \{4\}, \{5\}, \ldots, \{n\}, \ldots\}$$

compared with the class from Example 11 has the following slight improvement. For any splitting $\mathcal{L}_0, \mathcal{L}_1$ of the class, one half of the splitting contains an ascending chain of two or three sets, while the other half contains only disjoint sets. Thus the two halves are not equivalent with respect to $\leq_{weak}$.

As these two examples show, it is more adequate to study the splitting of more restrictive classes like the inclusion-free classes. A special case of such classes are the finitely learnable classes. Here a class is finitely learnable [11] iff there is a learner which, on every text for a language to be learned, outputs only one hypothesis, which must be correct. For technical reasons, the learner keeps outputting a special symbol denoting the absence of a reasonable conjecture until it outputs its only hypothesis.

THEOREM 12. $\{L_0, L_1, L_2, \ldots\} \equiv_{strong} \{H_0, H_1, H_2, \ldots\}$ *whenever both classes are infinite indexed families which are finitely learnable. In particular, every such class is strongly mitotic.*

Proof. As $\{L_0, L_1, L_2, \ldots\}$ and $\{H_0, H_1, H_2, \ldots\}$ are infinite, one can without loss of generality assume that the underlying enumerations are one-to-one. Furthermore, they have exact finite learners M and N, respectively, which use the corresponding indexing. Now one translates $\{L_0, L_1, L_2, \ldots\}$ to $\{H_0, H_1, H_2, \ldots\}$ by mapping L_n to H_n; thus Ψ is the identity mapping, where in the domain the n stands for H_n and in the range the n stands for L_n. $\Theta(T) = \#^k T_n$, where k is the least number such that M outputs a hypothesis n on input $T[k]$ (that is, the first position where M conjectures a hypothesis) and T_n is the ascending text of H_n. This completes the proof of the first statement.

Given now an infinite finitely learnable class $\{L_0, L_1, L_2, \ldots\}$, one can split it into $\{L_0, L_2, L_4, \ldots\}$ and $\{L_1, L_3, L_5, \ldots\}$, which are the subclasses of languages with even and odd index, respectively. Both classes are also infinite indexed families which are

finitely learnable. Thus they are all equivalent by the above result. Furthermore, a classifier for splitting can be obtained by simulating the exact finite learner M for $\{L_0, L_1, L_2, \ldots\}$ on the input text, and then converging to 0 if the (only) grammar output by M on the input text is even and to 1 if the (only) grammar output by M on the input text is odd. □

4. Further splitting theorems. Another question is whether classes can be split into incomparable classes. So one would ask whether there is a parallel result to the Sacks splitting theorem [20]: Every recursively enumerable but nonrecursive set A is the disjoint union of two recursively enumerable sets A_0 and A_1 such that the Turing degrees of A_0 and A_1 are incomparable and strictly below that of A. The next example shows that there are classes where every splitting is of this form; thus these classes are not weakly mitotic. Furthermore, splittings exist, so the result is not making use of a pathological diagonalization against all classifiers.

Example 13. Let A be a maximal set. If $a \notin A$, then let $L_a = \{a\}$; else let $L_a = A$. Then $\{L_0, L_1, L_2, \ldots\}$ is recursively enumerable and finitely learnable, but any splitting $\mathcal{L}_0, \mathcal{L}_1$ of $\{L_0, L_1, L_2, \ldots\}$ satisfies $\mathcal{L}_0 \not\leq_{weak} \mathcal{L}_1$ and $\mathcal{L}_1 \not\leq_{weak} \mathcal{L}_0$.

Proof. Let $T_0, T_1, T_2, \ldots$ be a recursive enumeration of recursive texts for $L_0, L_1, L_2, \ldots$, respectively. Let $F(a)$ be the cardinality of $\{b < a \mid b \notin A\}$. It is easy to see that one can split $\{L_0, L_1, L_2, \ldots\}$ into $\{L_a \mid a \in A \vee F(a) \text{ is even}\}$ and $\{L_a \mid a \notin A \wedge F(a) \text{ is odd}\}$. Thus this class has a splitting; in fact there are infinitely many of them. Furthermore, $\{L_0, L_1, L_2, \ldots\}$ is finitely learnable by outputting an index for L_a for the first a occurring in a given text.

Assume now by way of contradiction that there is a splitting $\mathcal{L}_0, \mathcal{L}_1$ with $\mathcal{L}_0 \leq_{weak} \mathcal{L}_1$ via a reduction (Θ, Ψ). Now one defines the partial recursive function f which outputs on input a the first number occurring in $\Theta(T_a)$; if there occurs no number, then $f(a)$ is undefined. As $\mathcal{L}_0$ is infinite, there are infinitely many $a \notin A$ with $L_a \in \mathcal{L}_0$. For all but one of these, $\Theta(T_a)$ has to be a text for some set $L_b \neq A$ in $\mathcal{L}_1$. Then $L_b = \{b\}$ and $f(a) = b \notin A$ for these a's. It follows that, for every x, there is an $a > x$ with $a \notin A \wedge f(a) \notin A \wedge f(a) > x$. Then, by Remark 8, $f(a) = a$ for almost all $a \notin A$. As infinitely many of these a's belong to an $L_a \in \mathcal{L}_0$, one has that $\Theta(T_a)$ is a text for L_a and Θ translates some text for a set in $\mathcal{L}_0$ into a text for a set in $\mathcal{L}_0$ and not into a text for a set in $\mathcal{L}_1$. Thus $\mathcal{L}_0 \not\leq_{weak} \mathcal{L}_1$. By symmetry of the argument, $\mathcal{L}_1 \not\leq_{weak} \mathcal{L}_0$. □

While Example 13 showed that there are classes for which every splitting is a Sacks splitting, the next result shows that every explanatorily learnable recursively enumerable class has a Sacks splitting; but it might also have other splittings.

THEOREM 14. *Every infinite recursively enumerable and explanatorily learnable class $\{L_0, L_1, L_2, \ldots\}$ has a splitting into two infinite subclasses $\mathcal{L}_0, \mathcal{L}_1$ such that $\mathcal{L}_0 \not\leq_{weak} \mathcal{L}_1$ and $\mathcal{L}_1 \not\leq_{weak} \mathcal{L}_0$.*

Proof. Let M be a explanatory learner for $\{L_0, L_1, L_2, \ldots\}$ which satisfies the three conditions from Remark 5. Now one defines the following function F from $\mathbb{N}$ to $\{0, 1\}$, inductively using the oracle K for the halting problem.

Let $\Theta_0, \Theta_1, \Theta_2, \ldots$ be the enumeration of operators as given in Remark 2. Let T_n be a text for L_n, constructed effectively from n (as $\{L_0, L_1, L_2, \ldots\}$ is recursively enumerable, this can be done). Let U be the set of all minimal indices of languages in $\{L_0, L_1, L_2, \ldots\}$; that is, $n \in U$ iff for all $m < n$, $L_m \neq L_n$. Note that one can decide U relative to K, since $L_m = L_n$ iff M converges to the same index on both T_m and T_n. Let $F^*(n, a)$ be the number of $k \in U$ satisfying $k < n$ and $F(k) = a$. Note that $F^*(n, a)$ can be computed using oracle K (assuming F can be computed relative to

K, as will be shown later).

The value of $F(n)$ is defined by using the case below, with higher priority (reflected by the least number), for which the corresponding condition is true.

- Case with priority 0: $n \notin U$. Then there is an $m < n$ such that $L_m = L_n$. Let $F(n) = F(m)$ for the least such m.
- Case with priority $4e + 1$: There is an $m < n$ such that (a) M converges on $\Theta_e(T_m)$ and T_n to the same value; (b) $F(m) = 0$; and (c) there are no $i, j < n$ such that $F(i) = 0$, $F(j) = 0$, and M converges on $\Theta_e(T_i)$ to the same value as on T_j. Then let $F(n) = 0$.
- Case with priority $4e + 2$: There is an $m < n$ such that (a) M converges on $\Theta_e(T_m)$ and T_n to the same value; (b) $F(m) = 1$; and (c) there are no $i, j < n$ such that $F(i) = 1$, $F(j) = 1$, and M converges on $\Theta_e(T_i)$ to the same value as on T_j. Then let $F(n) = 1$.
- Case with priority $4e + 3$: $F^*(m, 0) < F^*(m, 1) + e$ for all $m \leq n$. Then let $F(n) = 0$.
- Case with priority $4e + 4$: $F^*(m, 1) < F^*(m, 0) + e$ for all $m \leq n$. Then let $F(n) = 1$.

More precisely, when defining $F(n)$ one searches for the least number k such that the case with priority k applies, and then defines F as described in that case. Note that the conditions for the cases with priorities $4e + 3$ and $4e + 4$ apply for all $e > n$, and thus there is always some case which applies.

Next it is shown that $F \leq_T K$. As M converges on T_m, for every m, the test whether the condition of the case with priority 0 applies for computing $F(n)$ (and the corresponding computation of $F(n)$) can be done relative to K: $L_m = L_n$ iff M converges to the same value on both T_m and T_n. For the test whether the condition of the case with priority $4e + 1$ applies, it needs to be checked whether M converges to the same value on both $\Theta_e(T_m)$ and T_n. This can be done using oracle K as follows. First compute the value d to which M converges on T_n. Then, as M satisfies the constraints in Remark 5, M on $\Theta_e(T_m)$ either converges to d or outputs d only finitely often. Thus one can check, using the oracle K, whether M converges on $\Theta_e(T_m)$ to d. Similarly, one can check, for any two numbers i and $j < n$, whether M converges to the same value on $\Theta_e(T_i)$ and T_j. Thus, one can test, using oracle K, whether the case with priority $4e + 1$ applies. Similarly, one can test whether the case with priority $4e + 2$ applies. The tests for the cases with priorities $4e + 3$ and $4e + 4$ are obviously doable relative to K as $U \leq_T K$ and the conditions refer only to statistics of previous values of F at places where the argument is in U.

So F can be computed in the limit. Having an approximation F_s to F, one defines a classifier C as $C(\sigma) = F_{|\sigma|}(m)$ for the least m such that $m = |\sigma| \vee M(T_m[|\sigma|]) = M(\sigma)$. Assume now that a text T of a language in $\{L_0, L_1, L_2, \ldots\}$ is given and n is the least index such that $L_n = \text{content}(T)$. Then, for all sufficiently large s, $C(T[s]) = F_s(n)$ (as M converges on T and T_n to the same index of L_n, but, for $m < n$, M converges on T_m to an index for the language L_m which is not equal to L_n). Thus C converges on T to $F(n)$.

Clearly the case with priority 0 is applied for computing $F(n)$ iff $n \notin U$ (this happens for infinitely many n). Now it is shown by induction that the case with priority $4e + c > 0$ (where $c \in \{1, 2, 3, 4\}$) applies for computing only finitely many $F(n)$. Assume by induction that, for some ℓ, all cases with priorities k strictly between 0 and $4e + c$ do not apply for computation of $F(n)$, $n \geq \ell$, and the case with priority $4e + c$ applies for computing $F(\ell)$. Now one makes a case-distinction depending on

which of the priorities $4e + c$ is applied for computing $F(\ell)$.

In the case of priority $4e+1$, for $n = \ell$, there is an $m < n$ such that (a) $F(m) = 0$, (b) $F(n) = 0$, and (c) M converges on $\Theta_e(T_m)$ and T_n to the same value. Now let $i = m$ and $j = \ell$. Then, for $n > \ell$, these values i, j are below n and ensure that the case with priority $4e + 1$ does not apply. So $n = \ell$ is the maximal n where $F(n)$ is defined according to this case.

In the case of priority $4e+3$, consider the set $\{n_0, n_1, n_2, \ldots, n_k\}$ of the least $k+1$ elements of $U \cap \{\ell, \ell + 1, \ell + 2, \ldots\}$, where $k = e + F^*(\ell, 1) - F^*(\ell, 0)$. One can now prove by induction for $u = 0, 1, 2, \ldots$ that

- for n_u with $u < k$, the case with priority $4e + 3$ applies and $F(n_u) = 0$;
- for n_u with $u \leq k$, $F^*(n_u, 0) = F^*(\ell, 0) + u$ and $F^*(n_u, 1) = F^*(\ell, 1)$.

Thus, $F^*(n_k, 0) = F^*(n_k, 1) + e$, and for $n \geq n_k$ the case with priority $4e + 3$ does not apply for computing $F(n)$.

The other two cases of priority $4e+2$ and $4e+4$ are symmetric to the two previous cases. Thus, one can conclude that there are only finitely many n where the case with priority $4e+c$ applies in the computation of $F(n)$. This completes the inductive step.

Let $\{L_0, L_1, L_2, \ldots\}$ be split into two classes as $\mathcal{L}_a = \{L_i : F(i) = a\}$. Now assume by way of contradiction that there is a reduction (Θ_e, Ψ) witnessing that $\mathcal{L}_0 \leq_{weak} \mathcal{L}_1$. Let ℓ be so large that the cases with priorities $1, 2, \ldots, 4e$ are not used to define any $F(n)$ with $n \geq \ell$. Due to the case with priority $4\ell + 7$, there is an $\ell' \in U$ with $F^*(\ell', 0) \geq F^*(\ell', 1) + \ell + 1$; note that $\ell' > \ell$. So more sets in $\mathcal{L}_0$ than in $\mathcal{L}_1$ have an index below ℓ'. Thus, there is an $m \leq \ell'$ such that $L_m \in \mathcal{L}_0$ and $\Theta_e(T_m)$ is not the text of any of the sets $L_0, L_1, \ldots, L_{\ell'}$. Let n be the minimal index of content$(\Theta_e(T_m))$; this index exists as Θ_e maps texts of languages in $\mathcal{L}_0$ to texts of languages in $\mathcal{L}_1$. It follows from the construction that either $F(n) = F(m) = 0$ and $L_n \in \mathcal{L}_0$ or there are $i, j < n$ with $F(i) = F(j) = 0$, $L_i, L_j \in \mathcal{L}_0$, and $\Theta_e(T_i)$ being a text for L_j. This contradicts the assumption that (Θ_e, Ψ) reduces $\mathcal{L}_0$ to $\mathcal{L}_1$. Hence $\mathcal{L}_0 \not\leq_{weak} \mathcal{L}_1$. Similarly one can show that $\mathcal{L}_1 \not\leq_{weak} \mathcal{L}_0$. □

For this reason, one cannot give a recursively enumerable class where all splittings $\mathcal{L}_0, \mathcal{L}_1$ satisfy either $\mathcal{L}_0 \leq_{strong} \mathcal{L}_1$ or $\mathcal{L}_1 \leq_{strong} \mathcal{L}_0$. Furthermore, complete classes have comparable splittings like before as they are mitotic and have even equivalent splittings. The next example gives a class where halves of some splittings are comparable but are never equivalent.

Example 15. Let A be a maximal set. For all $a \in \mathbb{N}$ and $b \in \{0, 1, 2\}$, let $L_{3a+b} = \{3a + b\}$ if $a \notin A$ and $L_{3a+b} = \{3c + b \mid c \in A\}$ if $a \in A$. Then $\{L_0, L_1, L_2, \ldots\}$ is not weakly mitotic as no halves of any splitting are equivalent with respect to $\leq_{weak}$, but $\{L_0, L_1, L_2, \ldots\}$ has a splitting $\mathcal{L}_0, \mathcal{L}_1$ with $\mathcal{L}_0 \leq_{strong} \mathcal{L}_1$.

Proof. If one takes the splitting $\mathcal{L}_0 = \{L_0, L_3, L_6, \ldots\}$ and $\mathcal{L}_1 = \{L_1, L_2, L_4, L_5, L_7, L_8, \ldots\}$, then it is easy to see that $\mathcal{L}_0 \leq_{strong} \mathcal{L}_1$ via (Θ, Ψ) such that Θ is based on translating every datum $3x$ to $3x + 1$ and Ψ is based on transforming every index e into an index for $\{3x \mid 3x + 1 \in W_e\}$. The details are left to the reader.

Given now a further splitting $\mathcal{L}_2, \mathcal{L}_3$ of $\{L_0, L_1, L_2, \ldots\}$, one of these two classes, say $\mathcal{L}_2$, must contain at least two of the sets $L_{3a}, L_{3a+1}, L_{3a+2}$ for infinitely many $a \notin A$. Assume by way of contradiction that (Θ, Ψ) would witness $\mathcal{L}_2 \leq_{weak} \mathcal{L}_3$. Now one defines the following functions f_b for $b = 0, 1, 2$ by letting $f_b(a)$ be the number x such that $3x$ or $3x + 1$ or $3x + 2$ occurs earliest in the text $\Theta((3a + b)^\infty)$. Now choose two different $b, b' \in \{0, 1, 2\}$ such that there are infinitely many $a \in \mathbb{N} - A$ with $L_{3a+b}, L_{3a+b'} \in \mathcal{L}_2$. Then one knows that, for every bound c, there are infinitely many $a \in \mathbb{N} - A$ such that $L_{3a+b} \in \mathcal{L}_2$ and $\Theta((3a+b)^\infty)$ is a text for some language in

$\mathcal{L}_3 - \{L_0, L_1, L_2, \ldots, L_c\}$. It follows by Remark 8 that $f_b(a) = a$ for almost all $a \notin A$. The same applies to $f_{b'}$. So there is an $a \notin A$ such that $L_{3a+b}, L_{3a+b'}$ are both in $\mathcal{L}_2$ and that Θ maps texts of both languages to texts of the sets L_{3a}, L_{3a+1}, or L_{3a+2}. As only one of these sets can be in $\mathcal{L}_3$, Θ has to map texts of different languages to texts of the same language, a contradiction. Thus $\mathcal{L}_2 \not\leq_{weak} \mathcal{L}_3$ and the class cannot be weakly mitotic. □

While in recursion theory a splitting A_0, A_1 of a recursively enumerable set A satisfies $A_0 \equiv_T A_1 \Rightarrow A_0 \equiv_T A$, the next example shows that the corresponding connection does not hold in inductive inference. The proof is very similar to Example 15 and is thus omitted.

Example 16. Let A be a maximal set. For all $a \in \mathbb{N}$ and $b \in \{0, 1\}$, let $L_{2a+b} = \{2a + b\}$ if $a \notin A$ and $L_{2a+b} = \{2c + b \mid c \in A\}$ if $a \in A$. Then $\{L_0, L_1, L_2, \ldots\}$ is not weakly mitotic, but the halves of the splitting $\{L_0, L_2, L_4, \ldots\}, \{L_1, L_3, L_5, \ldots\}$ are equivalent with respect to $\leq_{strong}$.

5. Beyond explanatory learning. One could, besides classes which are complete for explanatory learning, also consider classes which are complete for behaviorally correct learning [3, 5, 18] with respect to $\leq_{strong}$. Note that such a class $\mathcal{L}$ may no longer be explanatorily learnable, but $\mathcal{L}$ satisfies the following two properties:

- The class $\mathcal{L}$ is behaviorally correct learnable; that is, there is a learner which outputs, on every text T for a language in $\mathcal{L}$, an infinite sequence $e_0, e_1, e_2, \ldots$ of hypotheses such that $W_{e_n} = \text{content}(T)$ for almost all n.
- Every behaviorally correct learnable class $\mathcal{H}$ satisfies $\mathcal{H} \leq_{strong} \mathcal{L}$.

Note that the reduction $\leq_{strong}$ considered in this paper is always the same as defined for explanatory learning; reducibilities more adapted to behaviorally correct learning have also been studied [12, 13]. Completeness with respect to $\leq_{weak}$ is not considered in this section, so "complete" means "complete for $\leq_{strong}$" in this section.

It is easy to show that such complete classes exist. Consider as an example the class $\mathcal{L}$ of all sets $\{x\} \cup \{x + y + 1 \mid y \in L\}$, where the xth learner behaviorally correct learns the set L. So given any behaviorally correct learnable class and an index x of its learner, the translation $L \mapsto \{x\} \cup \{x + y + 1 \mid y \in L\}$ would translate all the sets behaviorally correct learned by this learner into sets in $\mathcal{L}$.

In the following let $\mathcal{L}$ be any class which is complete for behaviorally correct learning with respect to $\leq_{strong}$. Note that methods similar to those in Theorem 3 show that $\mathcal{L}$ is strongly mitotic. The next result shows that for any splitting $\mathcal{L}_0, \mathcal{L}_1$ of $\mathcal{L}$, one of these two classes is complete for behaviorally correct learning as well and therefore this class cannot be split into two incomparable subclasses.

THEOREM 17. *If $\mathcal{L}_0, \mathcal{L}_1$ are a splitting of a class which is complete for behaviorally correct learning with respect to $\leq_{strong}$, then either $\mathcal{L}_0 \equiv_{strong} \mathcal{L}_0 \cup \mathcal{L}_1$ or $\mathcal{L}_1 \equiv_{strong} \mathcal{L}_0 \cup \mathcal{L}_1$.*

Proof. First the theorem is shown for a special class $\mathcal{S}$ defined below. Let $\mathcal{H}$ be a class which is complete for behaviorally correct learning with respect to $\leq_{strong}$. Furthermore, let $C_0, C_1, \ldots$ be a list of all primitive recursive classifiers. One builds, for each x, a sequence $\tau_{x,0}, \tau_{x,1}, \ldots$ starting with $\tau_{x,0} = x$ as follows. If $\tau_{x,y}$ has been defined, then one takes $\tau_{x,y+1}$ to be the first extension of $\tau_{x,y}$ found, if any, such that

$$\{x, x + 1, x + 2, \ldots, x + y\} \subseteq \text{content}(\tau_{x,y+1}) \subseteq \{x, x + 1, x + 2, \ldots\}$$

and $C_x(\tau_{x,y+1}) \neq C_x(\tau_{x,y})$.

In the case that the above process terminates at some y, that is, if $\tau_{x,y+1}$ does not exist, then let $z = \max(\text{content}(\tau_{x,y}))$ and place in $\mathcal{S}$ the set

$$\{x, x+1, x+2, \ldots, z\} \cup \{z+u+1 \mid u \in H\},$$

for each $H \in \mathcal{H}$.

On the other hand, if the process does not terminate at any y, then $T = \lim_{y\in\mathbb{N}} \tau_{x,y}$ is a text for $\{x, x+1, x+2, \ldots\}$ on which C_x does not converge; in this case, one places the set

$$\{x, x+1, x+2, \ldots\}$$

into $\mathcal{S}$. Note that, in this case, C_x does not split $\mathcal{S}$ into two subclasses, as it diverges on a text for $\{x, x+1, x+2, \ldots\}$.

Now it is shown that $\mathcal{S}$ is behaviorally correct learnable. Suppose M is a behaviorally correct learner for $\mathcal{H}$. The new learner N for $\mathcal{S}$ works as follows. On input $T[n]$, it determines $x_n = \min(\text{content}(T[n]))$, $y_n =$ largest value of y such that $\tau_{x_n,y}$ is defined within n steps in the process given above, and $z_n = \max(\text{content}(\tau_{x_n,y_n}))$. The learner N then constructs a new string η, where it replaces every u in $T[n]$ by $u - z_n - 1$ if $u > z_n$, and by $\#$ if $u < z_n$. Then N conjectures the following set:

$$W_{N(\sigma)} = \begin{cases} \{x_n, x_n+1, x_n+2, \ldots, z_n\} & \\ \quad \cup\ \{u + z_n + 1 \mid u \in W_{M(\eta)}\} & \text{if } \tau_{x_n,y_n+1} \text{ does not get defined;} \\ \{x_n, x_n+1, x_n+2, \ldots\} & \text{if } \tau_{x_n,y_n+1} \text{ is defined.} \end{cases}$$

Note that the definition above is valid, as $\{x_n, x_n+1, x_n+2, \ldots, z_n\} \cup \{u+z_n+1 \mid u \in W_{M(\eta)}\} \subseteq \{x_n, x_n+1, x_n+2, \ldots\}$, and thus one can always switch to the second case when τ_{x_n,y_n+1} is defined. The verification that N indeed behaviorally correct learns $\mathcal{S}$ is straightforward; therefore the verification is omitted.

Now consider any classifier C_x which converges on every text for a language in $\mathcal{S}$. Then there is a maximal y such that $\tau_{x,y}$ is defined (since otherwise C_x does not converge on $T = \bigcup_{y\in\mathbb{N}} \tau_{x,y}$, which is a text for the language in $\mathcal{S}$). Let $z = \max(\text{content}(\tau_{x,y}))$. Therefore, the class $\mathcal{H}$ is strongly reducible to the subclass

$$\{\, \{x, x+1, x+2, \ldots, z\} \cup \{z+u+1 \mid u \in H\} \mid H \in \mathcal{H}\}$$

of $\mathcal{S}$. Furthermore, every set in this subclass has a text starting with $\tau_{x,y}$ and C_x converges on all such texts to $C_x(\tau_{x,y})$. Therefore this complete class is contained in one member of the splitting of $\mathcal{S}$ defined by C_x. Thus, one of these members is complete for behaviorally correct learning with respect to $\leq_{strong}$.

After dealing with this special class $\mathcal{S}$, consider any splitting $\mathcal{L}_0, \mathcal{L}_1$ of a class $\mathcal{L}$ which is complete for behaviorally correct learning with respect to $\leq_{strong}$. There is a reduction (Θ, Γ) from $\mathcal{S}$ to $\mathcal{L}_0 \cup \mathcal{L}_1$ due to completeness. Let C be the classifier which splits $\mathcal{L}$ into $\mathcal{L}_0, \mathcal{L}_1$. Let $\mathcal{S}_a$ consist of those L in $\mathcal{S}$ which are mapped to $\mathcal{L}_a$ by Θ. Note that this splitting of $\mathcal{S}$ can be effectively obtained by using Θ and C. As one of $\mathcal{S}_0$, $\mathcal{S}_1$ is complete for behaviorally correct learning (by the proof above), one of $\mathcal{L}_0$, $\mathcal{L}_1$ is complete for behaviorally correct learning with respect to $\leq_{strong}$. ∎

As just seen, any splitting $\mathcal{L}_0, \mathcal{L}_1$ of a class which is complete for behaviorally correct learning satisfies either $\mathcal{L}_0 \equiv_{strong} \mathcal{L}_1$ or $\mathcal{L}_0 <_{strong} \mathcal{L}_1$ or $\mathcal{L}_1 <_{strong} \mathcal{L}_0$. As the class is strongly mitotic, it can happen that the two halves of a splitting are

equivalent (although this is not always the case). The next result gives a class where the two halves of a splitting are always comparable but never equivalent.

THEOREM 18. *There is a recursively enumerable and behaviorally correct learnable class, which is not weakly mitotic, such that every splitting $\mathcal{L}_0, \mathcal{L}_1$ of the class satisfies either $\mathcal{L}_0 \leq_{strong} \mathcal{L}_1$ or $\mathcal{L}_1 \leq_{strong} \mathcal{L}_0$, but not $\mathcal{L}_0 \equiv_{weak} \mathcal{L}_1$.*

Proof. A minor modification of the construction of Post's simple set [19] gives the following: There is a recursive partition of the odd natural numbers into sets $I_0, I_1, I_2, \ldots$ and a recursively enumerable set L_1 of odd natural numbers such that (a) $1 \in L_1$, (b) $I_n \not\subseteq L_1$ for all n, and (c) L_1 intersects every recursively enumerable set which contains infinitely many odd natural numbers.

For $n \neq 1$, if n is even, then let $L_n = \{n\}$; else let $L_n = L_1 \cup \{n\}$.

Clearly $\{L_0, L_1, L_2, \ldots\}$ is behaviorally correct learnable: On input σ, the learner conjectures content(σ) if σ does not contain an odd number; otherwise it conjectures content(σ) $\cup L_1$.

Let $\mathcal{L}_0, \mathcal{L}_1$ be a splitting of $\{L_0, L_1, L_2, \ldots\}$. L_1 is in one of these classes, say in $\mathcal{L}_1$. Let C be the classifier witnessing the split. Then there is a stabilizing sequence σ for C on L_1, such that content(σ) $\subseteq L_1$ and $C(\tau) = 1$ for all extensions τ of σ with content(τ) $\subseteq L_1$. Let T be a text of L_1. Now for every $a \in L_1$ and every n, $C(\sigma a T[n]) = 1$. Since L_1 is simple, it follows that the set

$$D = \{a \mid a \text{ is odd and } \exists n\,[M(\sigma a T[n]) = 0]\}$$

is finite and thus $L_a \in \mathcal{L}_1$ for all odd $a \notin D$. Let n be such that $D \cap I_m = \emptyset$ for all $m \geq n$. Let $a_0, a_1, a_2, \ldots$ be an ascending enumeration of all even numbers plus members of D. Note that $\mathcal{L}_0 \subseteq \{L_{a_0}, L_{a_1}, L_{a_2}, \ldots\}$.

Let $b_m = \min(I_{n+m} - L_1)$—note that b_m is well defined since $I_{n+m} \not\subseteq L_1$ for all m. Note that $L_1 \cup \{b_m\}$ is in $\mathcal{L}_1$ for all m.

Now it is shown that $\mathcal{L}_0 \leq_{strong} \mathcal{L}_1$. Let Θ be an operator which maps texts for L_{a_m} to texts for $L_1 \cup \{b_m\}$. Note that such a Θ can easily be constructed. For Ψ, given a sequence E, one can determine, in the limit, the grammar e to which E converges (if any), and then determine, in the limit, the m such that e is a grammar for $L_1 \cup \{b_m\}$ (if there is any such m). Thus, one can construct a sequence which converges to a grammar for L_{a_m}, whenever such an m as above exists.

On the other hand, if an operator translates texts of sets in $\mathcal{L}_1$ into texts of sets in $\mathcal{L}_0$, then it has to map some text T of L_1 to some text of some L_{a_m}. There is an initial segment σ of T such that a_m appears on the output when σ is fed into the operator. There is only the set L_{a_m} in $\mathcal{L}_0$ containing a_m, but infinitely many sets in $\mathcal{L}_1$ that have a text starting with σ. Therefore the translation maps some texts of different sets in $\mathcal{L}_1$ to texts of L_{a_m}. Thus, the translation cannot be used for a weak reduction from $\mathcal{L}_1$ to $\mathcal{L}_0$. Hence $\mathcal{L}_1 \not\leq_{weak} \mathcal{L}_0$. □

6. Autoreducibility. Trakhtenbrot [23] defined that a set A is autoreducible iff one can reduce A to A such that $A(x)$ is obtained by accessing A only at places different from x. Ladner [15] showed that a recursively enumerable set is mitotic iff it is autoreducible. Ambos-Spies pointed out this result to the authors and asked whether the same holds in the setting of inductive inference. Unfortunately, this characterization fails for both of the major variants of autoreducibility. These variants are the ones corresponding to strong and weak reducibility.

DEFINITION 19. *A class $\mathcal{L}$ is strongly (weakly) autoreducible iff there is a strong (weak) reduction (Θ, Ψ) from $\mathcal{L}$ to $\mathcal{L}$ such that, for all sets $L \in \mathcal{L}$ and all texts T for L, $\Theta(T)$ is a text for a language in $\mathcal{L} - \{L\}$.*

Example 20. Let A be a maximal set and let $\mathcal{L}$ contain the following sets:

- $\{3x\}, \{3x+1\}, \{3x+2\}$ for all $x \notin A$;
- $\{3y : y \in A\}$, $\{3y+1 : y \in A\}$, $\{3y+2 : y \in A\}$.

Then the class $\mathcal{L}$ is neither strongly mitotic nor weakly mitotic. But $\mathcal{L}$ is autoreducible via some (Θ, Ψ) where Θ maps any text T to a text T' such that all elements of the form $3y$ in T have the form $3y+1$ in T', all elements of the form $3y+1$ in T have the form $3y+2$ in T', and all elements of the form $3y+2$ have the form $3y$ in T'.

So even the implication "strongly autoreducible $\Rightarrow$ weakly mitotic" fails. The remaining question is whether at least the converse direction is true in inductive inference. This is still unknown, but there is some preliminary result on sets which are complete for $\leq_{weak}$.

THEOREM 21. *If a class $\mathcal{L}$ is complete for $\leq_{weak}$, then it is weakly autoreducible.*

Proof. Let $\mathcal{L}$ be complete for $\leq_{weak}$ and let M be an explanatory learner for $\mathcal{L}$ which satisfies the conditions from Remark 5. As $\mathcal{L}$ is complete for $\leq_{weak}$, by Proposition 6, there is a reduction (Θ, Ψ) from the class $\mathcal{I}$ to $\mathcal{L}$ such that, for any set $I_x = \{0, 1, \ldots, x\} \in \mathcal{I}$ and any text T for I_x, $\Theta(T)$ is a text for a set on which M does not converge to an index less than or equal to x. Now, an autoreduction (Θ', Ψ') is constructed.

For this, one first defines Θ'' as follows and then concatenates it with Θ. The operator Θ'' translates every text T for a set L into a text for $I_{2^n(1+2m)}$, where m, n are chosen such that n is the value to which M converges on T and m is so large that all the elements put into $\Theta''(T)$, when following intermediate hypotheses of M on T, are contained in the set $I_{2^n(1+2m)}$. It is easy to verify that this can be done. Then Θ' is given as $\Theta'(T) = \Theta(\Theta''(T))$. The sequence $\Theta'(T)$ is a text for a set in $\mathcal{L}$ with the additional property that M converges on it to an index larger than $2^n(1+2m)$; this index is therefore different from n and $\text{content}(\Theta'(T)) \neq \text{content}(T)$.

The reverse operator Ψ' can easily be generated from Ψ. If E converges to an index for $\text{content}(\Theta'(T))$, then $\Psi(E)$ converges to some index for $I_{2^n(1+2m)}$. The number $2^n(1+2m)$ can be determined in the limit from this index by enumerating the corresponding finite set; thus Ψ' can translate E via $\Psi(E)$ to a sequence which converges to n. ∎

Example 22. The class $\mathcal{L}$ from Theorem 9 is complete for $\leq_{weak}$ and weakly autoreducible but not strongly autoreducible.

Proof. Let $\mathcal{L}$ and $a_0, a_1, a_2, \ldots$ be as in Theorem 9. Assume that (Θ, Ψ) witness that $\mathcal{L}$ is strongly autoreducible. Then Θ has to preserve inclusions and therefore map infinite sets in $\mathcal{L}$ to infinite sets. So, $\text{content}(\Theta(a_0\,(a_0+1)\,(a_0+2)\,\ldots))$ is an infinite set in $\mathcal{L}$ different from $\{a_0, a_0+1, a_0+2, \ldots\}$. By induction, one can show that

$$\text{content}(\Theta(a_n\,(a_n+1)\,(a_n+2)\,\ldots)) \subseteq \{a_{n+1}, a_{n+1}+1, a_{n+1}+2, \ldots\} \quad \text{and}$$
$$\text{content}(\Theta(a_n\,(a_n+1)\,(a_n+2)\,\ldots)) \subset \{a_n, a_n+1, a_n+2, \ldots\}.$$

But in Theorem 9 it was shown that no recursive operator has these properties. That $\mathcal{L}$ is complete for $\leq_{weak}$ was shown in Theorem 9, and that $\mathcal{L}$ is weakly autoreducible follows from Theorem 21. ∎

Acknowledgment. We would like to thank the anonymous referees for several helpful comments.

REFERENCES

[1] K. Ambos-Spies, *P-mitotic sets*, in Logic and Machines, Lecture Notes in Comput. Sci. 177, Springer, Berlin, 1984, pp. 1–23.

[2] D. Angluin, *Inductive inference of formal languages from positive data*, Inform. and Control, 45 (1980), pp. 117–135.

[3] J. Bārzdiņš, *Two theorems on the limiting synthesis of functions*, in Theory of Algorithms and Programs, Scientific Proceedings of Latvia State University 210, Riga, Latvia, 1974, pp. 82–88 (in Russian).

[4] L. Blum and M. Blum, *Toward a mathematical theory of inductive inference*, Inform. and Control, 28 (1975), pp. 125–155.

[5] J. Case and C. Lynes, *Inductive inference and language identification*, in Proceedings of the Ninth International Colloquium on Automata, Languages and Programming (ICALP) (Aarhus, Denmark, 1982), Lecture Notes in Comput. Sci. 140, Springer, Berlin, 1982, pp. 107–115.

[6] R. Freivalds, E. Kinber, and C. Smith, *On the intrinsic complexity of learning*, Inform. and Comput., 123 (1995), pp. 64–71.

[7] R. Friedberg, *Three theorems on recursive enumeration*, J. Symbolic Logic, 23 (1958), pp. 309–316.

[8] M. Fulk, *Prudence and other conditions on formal language learning*, Inform. and Comput., 85 (1990), pp. 1–11.

[9] C. Glaßer, A. Pavan, A. Selman, and L. Zhang, *Mitosis in computational complexity*, in Proceedings of the Third International Conference on Theory and Applications of Models of Computation (TAMC) (Beijing, China, 2006), Lecture Notes in Comput. Sci. 3959, Springer, Berlin, 2006, pp. 61–67.

[10] C. Glaßer, M. Ogihara, A. Pavan, A. Selman, and L. Zhang, *Autoreducibility, mitoticity and immunity*, in Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS) (Gdansk, Poland, 2005), Lecture Notes in Comput. Sci. 3618, Springer, Berlin, 2005, pp. 387–398.

[11] E. M. Gold, *Language identification in the limit*, Inform. and Control, 10 (1967), pp. 447–474.

[12] S. Jain, E. Kinber, and R. Wiehagen, *Language learning from texts: Degrees of intrinsic complexity and their characterizations*, J. Comput. System Sci., 63 (2001), pp. 305–354.

[13] S. Jain and A. Sharma, *The intrinsic complexity of language identification*, J. Comput. System Sci., 52 (1996), pp. 393–402.

[14] S. Jain and A. Sharma, *The structure of intrinsic complexity of learning*, J. Symbolic Logic, 62 (1997), pp. 1187–1201.

[15] R. Ladner, *Mitotic recursively enumerable sets*, J. Symbolic Logic, 38 (1973), pp. 199–211.

[16] P. Odifreddi, *Classical Recursion Theory*, North–Holland, Amsterdam, 1989.

[17] D. N. Osherson, M. Stob, and S. Weinstein, *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*, The MIT Press, Cambridge, MA, 1986.

[18] D. N. Osherson and S. Weinstein, *Criteria of language learning*, Inform. and Control, 52 (1982), pp. 123–138.

[19] E. Post, *Recursively enumerable sets of positive integers and their decision problems*, Bull. Amer. Math. Soc., 50 (1944), pp. 284–316.

[20] G. E. Sacks, *On the degrees less than* $0'$, Ann. of Math. (2), 77 (1963), pp. 211–231.

[21] C. H. Smith, R. Wiehagen, and T. Zeugmann, *Classifying predicates and languages*, Internat. J. Found. Comput. Sci., 8 (1997), pp. 15–41.

[22] R. Soare, *Recursively Enumerable Sets and Degrees*, Springer, Heidelberg, Germany, 1987.

[23] B. A. Trakhtenbrot, *On autoreducibility*, Soviet Mathematics Doklady (Dokl. Akad. Nauk SSSR), 11 (1970), pp. 814–817.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1300–1329

ON MONOTONE FORMULA COMPOSITION OF PERFECT ZERO-KNOWLEDGE LANGUAGES*

ALFREDO DE SANTIS†, GIOVANNI DI CRESCENZO‡, GIUSEPPE PERSIANO†, AND MOTI YUNG§

Abstract. We investigate structural properties of interactive perfect zero-knowledge (PZK) proofs. Specifically, we look into the closure properties of PZK languages under monotone boolean formula composition. This gives rise to new protocol techniques. We show that interactive PZK for random self-reducible (RSR) (and for co-RSR) languages is closed under monotone boolean formula composition. Namely, we present PZK proofs for monotone boolean formulae whose atoms are statements about membership in a PZK language which is RSR (or whose complement is RSR). We also discuss extensions, recent applications, and generalizations of the techniques.

Key words. zero knowledge, interactive proofs, random self-reducible languages

AMS subject classifications. 68Q15, 68Q10

DOI. 10.1137/S0097539798346123

1. Introduction.

A *zero-knowledge proof*, as introduced by Goldwasser, Micali, and Rackoff [42], is a proof procedure with the remarkable property of yielding nothing but the validity of the assertion. In [38] it was then shown that, under the assumption that one-way functions exist, all NP statements have computational zero-knowledge proofs, namely, proofs that are zero knowledge with respect to computationally bounded adversaries. Perfect zero-knowledge (PZK) proofs [42] are those proofs that are zero knowledge even with respect to a computationally unbounded adversary. The notion is important from both a practical and a theoretical perspective. We next review the practical and theoretical motivations of our work.

Practical motivations. PZK proof systems find their main application in the design of identification schemes, as first noted in [31]. An identification scheme is a protocol for two parties, called the user and the system, by which the user identifies himself to the system in a secure way; that is, a third party listening to the conversation cannot later impersonate the user. Typically, the user is trying to log into a system and has to identify himself as a legitimate user before the system grants him access. Identification schemes rely on the fact that certain PZK proofs are also "proofs of possession of knowledge of a witness" [31, 54, 3, 28] (a notion alluded to in [42, 32]).

In some applications it is desirable that the identity of the specific user trying to get access be kept secret from the system. For example, an investment firm accessing a stock market database would prefer not to reveal its identity to the database manager;

*Received by the editors October 14, 1998; accepted for publication (in revised form) September 9, 2007; published electronically July 23, 2008.
http://www.siam.org/journals/sicomp/38-4/34612.html

†Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84084 Fisciano (SA), Italy (ads@dia.unisa.it, giuper@dia.unisa.it). The first author's work was partially supported by research grants from MIUR and from the Università di Salerno. The third author's work was partially supported by the European Commission through the FP6 program under contracts FP6-1596 AEOLUS and IST-2002-507932 ECRYPT.

‡Telcordia Technologies, Piscataway, NJ 08854 (giovanni@research.telcordia.com). Part of the second author's work was done while the author was visiting the Università di Salerno.

§Google Inc. and Columbia University, New York, NY 10027 (moti@google.com, moti@cs.columbia.edu).

knowing which firm is interested in the stock of a given company is usually valuable information. At the same time, the system must make sure that the person trying to get access is a legitimate user (i.e., a subscriber to the service). The following "multifaceted identification scheme" is a possible solution to the above problem. As a concrete example, consider a system where user i knows a square root of x_i modulo an integer n which is the product of two primes. A user can identify himself as being a member of the group by proving that he knows a witness to the statement $(x_1 \in \mathrm{QR}_n) \vee (x_2 \in \mathrm{QR}_n) \vee \cdots \vee (x_m \in \mathrm{QR}_n)$ without disclosing his identity which is related to the specific x_i for which he knows a square root (QR_n is the set of elements of Z_n^* that admit a square root modulo n). On the other hand, at some other times user i can prove its individual (as opposed to group) identity by proving possession of a witness (square root) to the specific disjunct $(x_i \in \mathrm{QR}_n)$ associated with him. Other similar applications may employ a more complicated sharing of witnesses according to some given structure (typically called an access structure [44]), thus needing proof systems for general formulae (not just simply disjunction). In [24], a communication-efficient multifaceted identification scheme based on quadratic residuosity is presented.

Recently, the dual problem—consisting in hiding which record is accessed and not the identity of the user—has also been studied (see, e.g., [19]).

Theoretical motivations. From a theoretical prospective, PZK (as well as the related, less stringent notion of statistical ZK (SZK)) has been the subject of a number of early investigations. In [38] PZK proofs were given for the language of graph isomorphism as well as of graph nonisomorphism (which was the first language not known to be in NP shown to have an interactive proof). In [33, 2] it was proved that a complement of any SZK language has a 2-round proof system, and in [2] it was shown that any language with an SZK proof system also has a 2-round proof system. The round complexity and prover-power for SZK and PZK were studied in [5, 6, 7]. The honest verifier versus dishonest verifier and the private coin versus public coin problems have been studied, e.g., in [6, 49]. As PZK proofs do not depend on the properties of assumed hard problems (such as one-way functions) but only on the input language, they constitute a clean context for studying the intrinsic structural properties of the notion of a zero-knowledge proof and knowledge complexity of languages [42, 41, 40].

We remark that PZK/SZK proofs also motivated the useful notion of computationally sound proofs (in conjunction with the idea of zero-knowledge arguments) [17, 18, 47] and inspired the recent work on program checking and testing [14, 11].

PZK proofs are also the basic schemes used as preprocessing of further zero-knowledge proof systems; for example, in the noninteractive scenario [12, 13] and its applications (e.g., [48, 29, 28, 4]).

Given its practical use, its influence, and the complexity-theoretic investigations concerning PZK/SZK, we observe that relatively very few languages are known to be PZK. This, in turn, implies that only very few general protocol techniques are available for designing and utilizing the notion. Besides the proofs for quadratic residuosity and quadratic nonresiduosity given in the original paper [42], PZK proofs were given in [38, 34, 54, 37, 16, 25] and in [12, 22, 27] for the noninteractive zero-knowledge case. In [54] it was noted that all these languages have certain relations to random self-reducibility properties—either positively by claiming that the input is randomly reducible (by a group action) to some structure (e.g., one graph isomorphic to another), or negatively by claiming that the input is not reducible (e.g., a graph

is not isomorphic to another). This corresponds to the membership problem of a complement of an RSR language; see, e.g., [54]. All of their proof systems rely on certain algebraic properties associated with the random self-reducibility property.

The results of this paper suggest new protocol construction techniques, prove that a large class of logically constructible languages over atomic languages which are in PZK are themselves in PZK, and show that PZK is a larger class than previously known.

Summary of results. We prove that PZK for RSR languages (see [54]) is closed under monotone formula composition, thus extending the work of [42, 38] on PZK. More precisely, we present PZK proofs for all *monotone boolean formulae* whose atoms are statements about membership in an RSR language.

Let us give a concrete example. Fix an RSR language L (say graph isomorphism) and consider the following scenario. The prover and the verifier get as common input a monotone formula ϕ on m variables and strings $x_1, \ldots, x_m$. The prover wishes to convince the verifier that $\phi(\chi_1, \ldots, \chi_m)$ is true where $\chi_i = 1$ if $x_i \in L$ and $\chi_i = 0$ otherwise. We show that this can be done in PZK (and the interaction takes time polynomial in the size of ϕ and all the x_i's).

The above results are also shown for languages whose complement is RSR (such as nonisomorphism, nonresiduosity). Moreover, we prove that *threshold formulae* over membership statements for RSR languages and threshold formulae over *negated* membership statements for RSR languages also have PZK proofs.

Finally, we also show closure properties for a specific class of *nonmonotone* formulae.

Results in this paper for the special case of graph isomorphism and graph nonisomorphism languages have appeared in [26].

2. Notation and terminology. Given a boolean formula $\phi(v_1, \ldots, v_m)$, each truth assignment $t : \{v_1, \ldots, v_m\} \to \{0, 1\}$ naturally defines the validity t_ϕ of the formula ϕ under t. For convenience we assume that, scanning the formula from left to right, the variables $v_1, \ldots, v_m$ appear in this order. A *monotone formula* ϕ over the variables $v_1, \ldots, v_m$ is a boolean formula where each boolean operator is either an OR or an AND with two inputs and one output. A *threshold formula* is a boolean formula where each boolean operator computes a threshold function, where a threshold function $T_{k,m}$ takes m boolean values as input and outputs 1 if and only if at least k out of the m boolean values are 1. In this paper we will consider the following classes of boolean formulae: MON, the class of monotone formulae; T, the class of threshold formulae; and OR, the class of formulae that can be written as $((\phi) \vee (\neg\psi))$, where ϕ, ψ are monotone formulae.

Let L be a language, and let χ_L denote the indicator function for the language L (i.e., $\chi_L(x) = 1$ if and only if $x \in L$). Also, let B_m be a class of boolean formulae over m variables, $B = \cup_{m \geq 1} B_m$, and let $\vec{x} = (x_1, \ldots, x_m)$ be an m-tuple of binary strings. We define the language

$$CL(B; L) = \{(\phi; \vec{x}) \,|\, \text{for some } m > 0,\ \phi \in B_m; \phi(\chi_L(x_1), \ldots, \chi_L(x_m)) = 1\}.$$

We define the *size* of $(\phi, \vec{x})$ as the sum of the lengths of $\vec{x}$ and ϕ, according to their standard encoding (see, e.g., [35]).

Let us give a concrete example. Consider the language ISO of pairs of isomorphic graphs. Let ϕ be the monotone formula $(v_1 \wedge v_2) \vee (v_3 \wedge v_4)$, and let $\mathcal{G} = ((G_{10},$

$G_{11}), \ldots, (G_{40}, G_{41}))$ be a quadruple of pairs of graphs. Then, the pair $(\phi, \mathcal{G})$ belongs to the language CL(MON; ISO) if and only if the following statement is true:

$$((G_{10} \approx G_{11}) \wedge (G_{20} \approx G_{21})) \vee ((G_{30} \approx G_{31}) \wedge (G_{40} \approx G_{41})),$$

where the symbol $\approx$ denotes graph isomorphism.

By $\overline{L}$ we denote the complement of a language L. The symbol $\oplus$ denotes the (bitwise) xor logical operator. If $\vec{x} = (x_1, \ldots, x_l)$ and $\vec{y} = (y_1, \ldots, y_m)$ are sequences, by $\vec{x} \circ \vec{y}$ we denote the sequence $(x_1, \ldots, x_l, y_1, \ldots, y_m)$. If π_1 and π_2 are permutations, by $\pi_1 \circ \pi_2$ we denote the permutation obtained as a composition of π_1 and π_2, where π_2 is applied first.

Read-once formulae. Throughout this paper we will only consider read-once formulae, that is, formulae where each variable appears at most once. This is, however, without loss of generality, as we can always have a read-once formula which is still in the language by a small change in the representation (and length extension). Suppose that ϕ is a formula on m variables $v_1, \ldots, v_m$ and that v_i appears l_i times. By giving each variable a different name each time it appears, we construct a formula ϕ' on $m' = \sum_{i=1}^{m} l_i$ variables such that

$$\phi(v_1, \ldots, v_m) = \phi'(\underbrace{v_1, \ldots, v_1}_{l_1}, \ldots, \underbrace{v_m, \ldots, v_m}_{l_m}).$$

From this it follows that

$$(\phi, (x_1, \ldots, x_m)) \in \text{CL(MON;L)} \text{ if and only if } (\phi', (\underbrace{x_1, \ldots, x_1}_{l_1}, \ldots, \underbrace{x_m, \ldots, x_m}_{l_m})) \in \text{CL(MON;L)}.$$

2.1. Zero-knowledge proof systems.

DEFINITION 1. *Let* P*, the* prover*, be a probabilistic Turing machine, and let* V*, the* verifier*, be a probabilistic polynomial-time machine.* P *and* V *share the same input and can communicate with each other. By* $\text{P} \leftrightarrow \text{V}(x)$ *we denote* V*'s output after interacting with* P *on input* x*. A pair* (P,V) *constitutes an* interactive proof system *for the language* L *if the following hold:*

1. *(Completeness) If* $x \in L$*, then*

$$\Pr(\text{P} \leftrightarrow \text{V}(x) = \text{ACCEPT}) \geq 2/3.$$

2. *(Soundness) If* $x \notin L$*, then for all probabilistic Turing machines* P^*

$$\Pr(\text{P}^* \leftrightarrow \text{V}(x) = \text{ACCEPT}) \leq 1/2.$$

For simplicity, the above definition considers a constant error in the completeness and soundness requirements. Any protocol satisfying such definition can be easily transformed into a protocol with exponentially small error by using the standard technique of multiple independent repetitions. We define *View* $_{\text{V}}(x)$, V's view of the interaction with P on input x, as the random variable that assigns to pairs $(R; \textit{Trans})$ the probability that R is the portion of V's random tape used during the execution of the protocol and that *Trans* is the transcript of a conversation between P and V on input x given that V uses random string R.

DEFINITION 2. *An interactive proof system* (P,V) *for L is a* PZK proof system *for L (in short, a PZK proof system) if for each probabilistic polynomial-time machine* V^* *there exists a probabilistic machine S_{V^*} (called the simulator) running in expected polynomial time such that for all $x \in L$ the probability spaces $View_{V^*}(x)$ and $S_{V^*}(x)$ coincide.*

We denote by PZK the class of languages that have a PZK proof system.

3. Random self-reducible languages. The class of random self-reducible languages was introduced in [54, 1] and has since been referred to as RSR.

DEFINITION 3. *A* relation *is a subset of* $\{0,1\}^* \times \{0,1\}^*$.

Let $\mathcal{N}$ be a BPP language. A family of relations $\mathcal{R} = \{\mathcal{R}_x\}_{x \in \mathcal{N}}$ is a polynomial-time family of relations *if for an* instance z *and a* witness w *it is possible to check in time polynomial in $|x|$ whether $(z,w) \in \mathcal{R}_x$.*

DEFINITION 4. *A family of sets $\{\mathcal{S}_x\}_{x \in \mathcal{N}}$ is* efficiently samplable *if and only if there exists an algorithm running in expected polynomial time that, on input $x \in \mathcal{N}$, outputs a uniformly randomly chosen element of $\mathcal{S}_x$.*

DEFINITION 5 (RSR languages). *Let $\mathcal{N}$ be a BPP language and $\{\mathcal{S}_x\}_{x \in \mathcal{N}}$ be an efficiently samplable family of sets. Let $\{\mathcal{R}_x\}_{x \in \mathcal{N}}$ be a family of polynomial-time relations and define the* domain *of $\mathcal{R}_x$, $\mathrm{dom}_{\mathcal{R}}(x)$, as the set $\{z \mid \exists w$ such that $(z,w) \in \mathcal{R}_x\}$. If $(z,w) \in \mathcal{R}_x$, then we say that w is an x-*witness *for z. The language $L_{\mathcal{R}} = \{(x,z) | x \in \mathcal{N}$ and $z \in \mathrm{dom}_{\mathcal{R}}(x)\}$ is* RSR *if the following conditions hold:*

1. *There exists a polynomial-time algorithm $Sample_{\mathcal{R}}$ that, on input $(x,z) \in L_{\mathcal{R}}$ and $r \in \mathcal{S}_x$, outputs $y \in \mathrm{dom}_{\mathcal{R}}(x)$, and, if r is chosen at random from $\mathcal{S}_x$, then y is uniformly distributed over* $\mathrm{dom}_{\mathcal{R}}(x)$.
2. *There exists an algorithm that on input*
   2.1. *x, z, y, r such that $Sample_{\mathcal{R}}(x,z,r) = y$ and*
   2.2. *an x-witness for y*

   outputs an x-witness for z.
3. *There exists an algorithm $Generate_{\mathcal{R}}$ that, on input x, outputs in expected polynomial time a pair $(z,w) \in \mathcal{R}_x$ with z uniformly distributed over the domain of $\mathcal{R}_x$ and w uniformly distributed over all x-witnesses for z.*
4. *There exists a polynomial-time* reconstructing *algorithm $A_{\mathcal{R}}$ that, on input x, z, u, y and r', r'', satisfying $y = Sample_{\mathcal{R}}(x,z,r')$ and $u = Sample_{\mathcal{R}}(x,z,r'')$, returns r such that $y = Sample_{\mathcal{R}}(x,u,r)$. Moreover, if r'' is uniformly distributed over $\mathcal{S}_x$, then so is r.*

Remark 1 (the properties for RSR languages). Conditions 1 and 2 in the above definition for RSR languages are also contained in the definition given in [54]. Our results in section 4 use only conditions 1–3. In section 5 we show a protocol for monotone formulae over co-RSR languages that also uses condition 4 (which is, however, satisfied by all known RSR languages), and then in Remark 4 we briefly discuss a different construction that does not use this condition but results in a somewhat less efficient protocol.

Remark 2 (expected polynomial time versus strict polynomial time). Notice that we require only that the algorithm $Generate_{\mathcal{R}}$ and the algorithms to sample from the families $\{\mathcal{S}_x\}$ stop in expected polynomial time. We can use any of the known standard techniques and change the strategy of the verifier in order to keep the running time strictly bounded by a polynomial. This comes at the expense of increasing the soundness probability by a constant factor, and the zero-knowledge property of the protocol is not affected because we do not modify the prover's strategy.

We define the class co-RSR as the class of languages L such that $\overline{L} \in$ RSR.

Let us now give a few examples of RSR languages.

Square roots. Let $\mathcal{N}$ be the set of all positive integers x, and let $\mathcal{S}_x$ be the set Z_x^*. The relation Square_x is defined as $\text{Square}_x = \{(z,w)|w \in Z_x^* \text{ and } w^2 \equiv z \bmod x\}$. If $(z,w) \in \text{Square}_x$, then we say that z is a *square modulo* x and w is a *square root of* z *modulo* x. The language $L_{\text{Square}} = \{(x,z)|z \text{ is a square modulo } x\}$ is RSR. Indeed, conditions 1–4 of Definition 5 are satisfied.

1. Consider algorithm $Sample_{\text{Square}}$ that on input (x,z,r) outputs $y = zr^2 \bmod x$. If z is a square modulo x and r is chosen at random in Z_x^*, then y is a random square modulo x.
2. To see that condition 2 is met, we observe that, if x, y, z, and r are such that $y = zr^2 \bmod x$ and s is a square root of y modulo x, then $w = sr^{-1} \bmod x$ is a square root of z modulo x.
3. Algorithm $Generate_{\text{Square}}$ on input x chooses s at random from Z_x^*, sets $y = s^2 \bmod x$, and outputs (y,s).
4. The reconstructing algorithm takes as input values x, z, u, y, r', r'' satisfying $y = z(r')^2 \bmod x$ and $u = z(r'')^2 \bmod x$, and returns $r = r' \cdot (r'')^{-1} \bmod x$. It is easily verified that $y = ur^2 \bmod x$ and that r is uniformly distributed in Z_x^* if r' and r'' also are.

Graph isomorphism. We say that two graphs G and H on the same set V of vertices are *isomorphic* if there exists a permutation π of V such that (u,v) is an edge of G if and only if $(\pi(u), \pi(v))$ is an edge of H. In this case we write $H = \pi(G)$. Let $\mathcal{N}$ be the set of all graphs G. For a graph $G = (\text{V}, E)$, let $\mathcal{S}_G$ be the set of all permutations of V. The relation ISO_G is defined as $\text{ISO}_G = \{(H,\pi)|\pi(G) = H\}$. We next show that the language $L_{\text{ISO}} = \{(G,H)|G \text{ and } H \text{ are isomorphic}\}$ is RSR.

1. Algorithm $Sample_{\text{ISO}}(G,H,\pi)$ outputs the graph $M = \pi(H)$. When π is chosen uniformly over all permutations of V and G and H are isomorphic, the graph M is uniformly distributed over the set of graphs isomorphic to G.
2. To see that condition 2 is met, we observe that if $Sample_{\text{ISO}}(G,H,\pi) = M$ and τ is such that $M = \tau(G)$ (that is, τ is a G-witness for M), then $\pi^{-1} \circ \tau$ is a G-witness for H.
3. Algorithm $Generate_{\text{ISO}}(G)$ randomly selects a permutation τ of the vertices of G and outputs $M = \tau(G)$ and τ.
4. The *reconstructing* algorithm $A_{\mathcal{R}}$, on input G, H, M, L, π, and τ that satisfy $M = \pi(H)$ and $L = \tau(H)$, constructs the isomorphism μ such that $M = \mu(L)$ as $\mu = \pi \circ \tau^{-1}$. It is easily verified that μ is uniformly distributed over the set of all permutations of V if π and τ also are.

Discrete logarithm. Let $\mathcal{N}$ be the set of pairs (p,g), where p is a prime and g is an element of Z_p, and let $\mathcal{S}_{(p,g)}$ be the set Z_{p-1}. Define the relation $\text{SUB}_{(p,g)}$ as $\text{SUB}_{(p,g)} = \{(z,w)|z = g^w \bmod p\}$. If $(z,w) \in \text{SUB}_{(p,g)}$, then we say that z belongs to the *subgroup* of Z_p generated by g and that w is the (p,g)-*logarithm* of z. The language L_{SUB} defined as $L_{\text{SUB}} = \{((p,g),z)|\exists w \text{ such that } z = g^w \bmod p\}$ is RSR.

1. Algorithm $Sample_{\text{SUB}}$ on input $((p,g),z,r)$ outputs $y = g^r z \bmod p$. If z belongs to the subgroup of Z_p generated by g and r is a random element of Z_{p-1}, then y is a random element of the subgroup of Z_p generated by g.
2. Condition 2 is seen to be satisfied by observing that if s is the (p,g)-logarithm of y and $((p,g),z,r)$ are such that $Sample_{\text{SUB}}((p,g),z,r) = y$, then $z \equiv g^{s-r} \bmod p$ and thus $(s-r) \bmod (p-1)$ is the (p,g)-logarithm of z.
3. Algorithm $Generate_{\text{SUB}}$ on input (p,g) picks an element s at random from Z_{p-1}, sets $y = g^s \bmod p$, and outputs the pair (y,s).

4. The reconstructing algorithm $A_{\mathcal{R}}$ is defined as follows. On input x, z, y, u, r', r'' such that $y = zg^{r'} \bmod p$ and $u = zg^{r''} \bmod p$, it outputs $r = (r' - r'') \bmod (p-1)$. Simple algebra gives $y = ug^r \bmod p$, and we note that r is uniformly distributed over Z_{p-1} if r' and r'' also are.

Decisional Diffie–Hellman. Using the same definitions for the discrete logarithm problem, we define the relation $\mathrm{DDH}_{(p,g)}$ as consisting of the pairs $((z_a, z_b, z_c), (a, b, c))$ such that $z_a = g^a \bmod p, z_b = g^b \bmod p, z_c = g^c \bmod p$, and $c = ab \bmod (p-1)$. If $((z_a, z_b, z_c), (a, b, c)) \in \mathrm{DDH}_{(p,g)}$, then we say that $z = (z_a, z_b, z_c)$ is a *Diffie–Hellman triple* with respect to (p, g). The language L_{DDH} defined as $L_{\mathrm{DDH}} = \{((p, g), z) | z$ is a Diffie–Hellman triple with respect to $(p, g)\}$ is RSR.

1. Algorithm $Sample_{\mathrm{DDH}}$ on input $((p, g), (z_a, z_b, z_c), (r, s))$ outputs the triple $y = (y_a, y_b, y_c)$ defined as $y_a = g^r z_a \bmod p$, $y_b = g^s z_b \bmod p$, and $y_c = g^{rs} z_a^s z_b^r z_c \bmod p$. If z is a Diffie–Hellman triple with respect to (p, g) and r, s are random and independent elements of Z_{p-1}, then y is a random Diffie–Hellman triple with respect to (p, g).
2. Condition 2 is seen to be satisfied by observing that if a, b, c are the discrete logarithms of z_a, z_b, z_c, respectively, with $c = ab \bmod (p-1)$, and $Sample_{\mathrm{DDH}}((p, g), (z_a, z_b, z_c), (r, s)) = (y_a, y_b, y_c)$, then the discrete logarithms of y_a, y_b, y_c, respectively, are $a + r \bmod (p-1)$, $b + s \bmod (p-1)$, and $(a + r)(b + s) \bmod (p-1)$.
3. Algorithm $Generate_{\mathrm{DDH}}$ on input (p, g) picks elements a, b at random from Z_{p-1}; sets $y_a = g^a \bmod p$, $y_b = g^b \bmod p$, and $y_c = g^{ab} \bmod p$; and outputs the triple (y_a, y_b, y_c).
4. The reconstructing algorithm $A_{\mathcal{R}}$ is defined as follows. On input $p, g, z = (z_a, z_b, z_c)$, $y = (y_a, y_b, y_c)$, $u = (u_a, u_b, u_c)$, (r', s'), (r'', s'') such that $y_a = g^{r'} z_a \bmod p$, $y_b = g^{s'} z_b \bmod p$, $y_c = g^{r's'} z_a^{s'} z_b^{r'} z_c \bmod p$, and $u_a = g^{r''} z_a \bmod p$, $u_b = g^{s''} z_b \bmod p$, $u_c = g^{r''s''} z_a^{s''} z_b^{r''} z_c \bmod p$, it returns r, s such that $r = (r' - r'') \bmod (p-1)$ and $s = (s' - s'') \bmod (p-1)$. Simple algebra gives $y_a = g^r u_a \bmod p$, $y_b = g^s u_b \bmod p$, and $y_c = g^{rs} u_a^r u_b^s u_c \bmod p$, and we note that r, s are uniformly distributed over Z_{p-1} if r', s' and r'', s'' also are.

The following lemma gives a useful property of algorithm *Sample*.

LEMMA 6. *Let $L_{\mathcal{R}}$ be an RSR language and let $(x, z) \notin L_{\mathcal{R}}$. Then, for all $r \in \mathcal{S}_x$, $Sample_{\mathcal{R}}(x, z, r) \notin \mathrm{dom}_{\mathcal{R}}(x)$.*

Proof. Suppose there exists an r such that $y = Sample_{\mathcal{R}}(x, z, r) \in \mathrm{dom}_{\mathcal{R}}(x)$, and let w be such that $(y, w) \in \mathcal{R}_x$. By property 2 of the RSR languages, knowing x, z, y, r, and w, it is possible to compute an x-witness for z. This would imply that $(x, z) \in L_{\mathcal{R}}$, a contradiction. □

In [54], extending the protocol for quadratic residuosity of [42] and graph isomorphism of [38], it was proved that all RSR languages have PZK proof systems.

THEOREM 7 (Tompa and Woll [54]). *All RSR languages have a PZK proof system.*

4. Monotone formulae over RSR languages. In this section we prove that the language of all true monotone formulae whose atoms are statements about membership in an RSR language have a PZK proof system.

Before describing the proof system in its full generality we sketch an example for the simple cases of OR and AND of two statements about graph isomorphisms. These two simple proof systems will constitute the building block of our proof systems for general monotone formulae.

OR OF TWO GRAPH ISOMORPHISMS. We now briefly describe a zero-knowledge proof system for proving that at least one of two pairs of graphs (A_0, A_1) and (B_0, B_1) consists of two isomorphic graphs.

The prover randomly picks bits c_1 and c_2 and permutations τ_A and τ_B, constructs graphs $A = \tau_A(A_{c_1})$ and $B = \tau_B(B_{c_2})$, and sends the pair (A, B) to the verifier. The verifier picks a random bit b and sends it to the prover. The prover responds by sending random bits b_1 and b_2 such that $b = b_1 \oplus b_2$ and isomorphisms π_A and π_B such that $A = \pi_A(A_{b_1})$ and $B = \pi_B(B_{b_2})$.

For the completeness property, suppose that A_0 and A_1 are isomorphic. The prover sets $b_1 = b \oplus c_2$, $b_2 = c_2$, and $\pi_B = \tau_B$. Isomorphism π_A instead is computed in the following way. If b_1 is equal to c_1, then isomorphism π_A is set equal to τ_A. If b_1 is not equal to c_1, then, since A_0 and A_1 are isomorphic, there exists isomorphism μ such that $A_{c_1} = \mu(A_{b_1})$. Therefore isomorphism $\pi_A = \tau_A \circ \mu$ is such that $A = \pi_A(A_{b_1})$.

On the other hand, suppose A_0 and A_1 and B_0 and B_1 are not isomorphic. Then no matter how graphs A and B are constructed there exists at most one b_1 such that A is isomorphic to A_{b_1} and at most one b_2 such that B is isomorphic to A_{b_2}. Therefore there exists at most one value of b (namely, $b = b_1 \oplus b_2$) for which the prover can convince the verifier.

For the zero-knowledge property we consider a simulator that picks c_1 and c_2 at random and then simulates the exchange of messages until it happens that $b = c_1 \oplus c_2$. Since the value $c_1 \oplus c_2$ is independent from the pair of graphs (A, B), the simulator has probability $1/2$ of succeeding, and it can be verified that, when it succeeds, it produces a perfect simulation. Thus, on average the simulation has to be repeated two times.

AND OF TWO GRAPH ISOMORPHISMS. Suppose now that we wish to prove that A_0 is isomorphic to A_1 and B_0 is isomorphic to B_1. Obviously, we could prove the two statements sequentially using the proof system of [38]: first prove that A_0 is isomorphic to A_1 and then prove that B_0 is isomorphic to B_1. Instead we propose to perform the two proofs in parallel in the following way: the prover presents the verifier with two graphs A and B; the verifier picks *one* random bit c; and the prover shows isomorphism between A and A_c and B and B_c. As is immediately seen, completeness, soundness, and zero knowledge continue to hold.

The general case. Let us introduce a bit of notation. For the rest of this section, we let L be an RSR language. The *value* $Val(c_1, \ldots, c_m; \phi) \in \{0, 1, \perp\}$ of a formula ϕ with m variables with respect to the m-tuple $(c_1, \ldots, c_m) \in \{0,1\}^m$ is defined recursively in the following way. (Recall that we consider only read-once formulae.) If ϕ consists of a single variable v, then $Val(a; v) = a$. Else, suppose that $\phi = \phi_1 \wedge \phi_2$ and let $v_1, \ldots, v_{m_1}$ be the variables in ϕ_1 and $v_{m_1+1}, \ldots, v_m$ be the variables in ϕ_2. If $Val(c_1, \ldots, c_{m_1}; \phi_1) = Val(c_{m_1+1}, \ldots, c_m; \phi_2) = b \in \{0, 1, \perp\}$, then $Val(c_1, \ldots, c_m; \phi) = b$. If, instead, $Val(c_1, \ldots, c_{m_1}; \phi_1) \neq Val(c_{m_1+1}, \ldots, c_m; \phi_2)$, then $Val(c_1, \ldots, c_m; \phi)$ is set equal to $\perp$. On the other hand, suppose that $\phi = \phi_1 \vee \phi_2$. If $Val(c_1, \ldots, c_{m_1}; \phi_1)$ and $Val(c_{m_1+1}, \ldots, c_m; \phi_2)$ are both different from $\perp$, then $Val(c_1, \ldots, c_m; \phi) = Val(c_1, \ldots, c_{m_1}; \phi_1) \oplus Val(c_{m_1+1}, \ldots, c_m; \phi_2)$; otherwise we set $Val(c_1, \ldots, c_m; \phi) = \perp$. For example, $Val(c_1, c_2; v_1 \vee v_2) = c_1 \oplus c_2$ for $c_1, c_2 \in \{0, 1\}$. The above definition is depicted in Table 1.

If $Val(c_1, \ldots, c_m; \phi) \neq \perp$, then we say that the sequence of bits $(c_1, \ldots, c_m)$ is *well formed* with respect to ϕ. The following fact can be easily proved by induction on the length of the sequences.

FACT 8. *If $\vec{a} = (a_1, \ldots, a_m)$ and $\vec{c} = (c_1, \ldots, c_m)$ are two well-formed sequences with respect to ϕ, then so is sequence $\vec{b} = (a_1 \oplus c_1, \ldots, a_m \oplus c_m)$. Moreover, the number*

TABLE 1

Value of ϕ_1	Value of ϕ_2	Value of $\phi_1 \vee \phi_2$	Value of $\phi_1 \wedge \phi_2$
0	0	0	0
0	1	1	$\perp$
0	$\perp$	$\perp$	$\perp$
1	0	1	$\perp$
1	1	0	1
1	$\perp$	$\perp$	$\perp$
$\perp$	0	$\perp$	$\perp$
$\perp$	1	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

of sequences $\vec{a}$ such that $Val(\vec{a};\phi) = 0$ is equal to the number of sequences $\vec{a}$ such that $Val(\vec{a};\phi) = 1$.

DEFINITION 9 (opening). *Let ϕ be a monotone formula, and let $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$ be an m-tuple of pairs. We say that the m-tuple of quadruples $R = ((c_1, r_1, s_1, t_1), \ldots, (c_m, r_m, s_m, t_m))$ is a $(\phi, \mathcal{I})$-*opening as $b \in \{0,1\}$ with auxiliary sequence $S = ((s_1, t_1) \ldots, (s_m, t_m))$ of the m-tuple $Y = (y_1, \ldots, y_m)$ *if and only if*

1. *$r_i \in \mathcal{S}_{x_i}$, $c_i \in \{0,1\}$, and $(s_i, t_i) \in \mathcal{R}_{x_i}$ for $i = 1, \ldots, m$;*
2. *$Val(c_1, \ldots, c_m; \phi) = b$;*
3. *if $c_i = 0$, then $y_i = Sample_{\mathcal{R}}(x_i, z_i, r_i)$;*
4. *if $c_i = 1$, then $y_i = Sample_{\mathcal{R}}(x_i, s_i, r_i)$.*

Whenever ϕ and $\mathcal{I}$ are clear from the context we will simply say "opening" instead of $(\phi, \mathcal{I})$-opening.

The following two lemmas summarize the properties of the concept of an opening that we will exploit for the construction of our proof system. If $(\phi, \mathcal{I}) \in$ CL(MON;L), then every m-tuple that can be opened as 0 can also be opened as 1 and vice versa. If instead $(\phi, \mathcal{I}) \notin$ CL(MON;L), then if an m-tuple can be opened as 0, it cannot be opened as 1 and vice versa.

LEMMA 10. *Let ϕ be a formula with m variables and $\mathcal{I}$ be an m-tuple of pairs such that $(\phi, \mathcal{I}) \in$ CL(MON;L). Let Y be an m-tuple, and let R be a $(\phi, \mathcal{I})$-opening of Y as $b \in \{0,1\}$. Then, there exists a $(\phi, \mathcal{I})$-opening R' of Y as $1 - b$.*

Proof. The proof proceeds by induction on the number m of variables of ϕ.

Suppose that $m = 1$ and let $Y = (y)$ and $\mathcal{I} = ((x, z))$. As $(\phi, ((x, z))) \in$ CL(MON;L), it must be the case that $z \in \mathrm{dom}_{\mathcal{R}}(x)$. By property 1 of the RSR languages, it follows that, for any $s, y \in \mathrm{dom}_{\mathcal{R}}(x)$, there exists r such that $y = Sample_{\mathcal{R}}(x, s, r)$. Therefore if y can be opened as 0 (i.e., $y = Sample_{\mathcal{R}}(x, z, r)$ for some r), then $y \in \mathrm{dom}_{\mathcal{R}}(x)$, and, thus, it can also be opened as a 1: pick any $s \in \mathrm{dom}_{\mathcal{R}}(x)$ along with an x-witness t for s and r' such that $y = Sample_{\mathcal{R}}(x, s, r')$ and construct opening $R' = (1, r', s, t)$. Similarly, if y can be opened as 1, then it can also be opened as 0.

Suppose now that $m \geq 2$ and $\phi = \phi_1 \vee \phi_2$ and let R be a $(\phi, \mathcal{I})$-opening of Y as 0. We construct a $(\phi, \mathcal{I})$-opening of Y as 1. Let m_1 be the number of variables of ϕ_1 and split $\mathcal{I}$ in $\mathcal{I}_1 = ((x_1, z_1), \ldots, (x_{m_1}, z_{m_1}))$ and $\mathcal{I}_2 = ((x_{m_1+1}, z_{m_1+1}), \ldots, (x_m, z_m))$ and Y in $Y_1 = (y_1, \ldots, y_{m_1})$ and $Y_2 = (y_{m_1+1}, \ldots, y_m)$. As $(\phi, \mathcal{I}) \in$ CL(MON;L), it must be the case that at least one of $(\phi_1, \mathcal{I}_1)$ and $(\phi_2, \mathcal{I}_2)$, say $(\phi_1, \mathcal{I}_1)$, belongs to CL(MON;L). Let $a \in \{0,1\}$ be such that Y_2 can be $(\phi_2, \mathcal{I}_2)$-opened as a. (Such an a exists, for otherwise Y cannot be $(\phi, \mathcal{I})$-opened as 0.) Let R_2 be such an opening and let R_1 be a $(\phi_1, \mathcal{I}_1)$-opening of Y_1 as $1 \oplus a$. (Such an opening exists by the inductive hypothesis.) Then, as can be readily verified, $R_1 \circ R_2$ is a $(\phi, \mathcal{I})$-opening of Y as 1. If

Y can be $(\phi, \mathcal{I})$-opened as 1, a $(\phi, \mathcal{I})$-opening of Y as 0 is constructed in a similar way.

The case where $m \geq 2$ and $\phi = \phi_1 \wedge \phi_2$ is proved similarly. □

LEMMA 11. *Let ϕ be a formula with m variables and $\mathcal{I}$ be an m-tuple of pairs such that $(\phi, \mathcal{I}) \notin$ CL(MON;L). Then for each m-tuple Y there is at most one bit $b \in \{0, 1\}$ for which there exists a $(\phi, \mathcal{I})$-opening of Y as b.*

Proof. We proceed by induction on m.

Suppose $m = 1$ and let $Y = (y)$ and $\mathcal{I} = ((x, z))$. Since $(\phi, \mathcal{I}) \notin$ CL(MON;L), it follows that $(x, z) \notin L_{\mathcal{R}}$. If Y can be opened as 1, then $y \in \text{dom}_{\mathcal{R}}(x)$. By Lemma 6, for all $r \in \mathcal{S}_x$, it holds that $Sample_{\mathcal{R}}(x, z, r) \notin \text{dom}_{\mathcal{R}}(x)$, and thus Y cannot be $(\phi, \mathcal{I})$-opened as 0. If Y can be opened as 0, then there is $r \in \mathcal{S}_x$ such that $y = Sample_{\mathcal{R}}(x, z, r)$. Since $(x, z) \notin L_{\mathcal{R}}$, by Lemma 6, $y \notin \text{dom}_{\mathcal{R}}(x)$, and thus y cannot be opened as 1.

Suppose now that $m \geq 2$ and that $\phi = \phi_1 \vee \phi_2$. Moreover, let Y be an m-tuple that can be $(\phi, \mathcal{I})$-opened as 0. Let m_1 be the number of variables of ϕ_1 and split $\mathcal{I}$ in $\mathcal{I}_1 = ((x_1, z_1), \ldots, (x_{m_1}, z_{m_1}))$ and $\mathcal{I}_2 = ((x_{m_1+1}, z_{m_1+1}), \ldots, (x_m, z_m))$ and Y in $Y_1 = (y_1, \ldots, y_{m_1})$ and $Y_2 = (y_{m_1+1}, \ldots, y_m)$. As $(\phi, \mathcal{I}) \notin$ CL(MON;L), neither $(\phi_1, \mathcal{I}_1)$ nor $(\phi_2, \mathcal{I}_2)$ belongs to CL(MON;L). Suppose now that Y can be $(\phi, \mathcal{I})$-opened as both 0 and 1. This implies that at least one of Y_1 and Y_2 can be opened as both a 0 and a 1, which contradicts the inductive hypothesis.

The case $\phi = \phi_1 \wedge \phi_2$ is similar. □

The properties of the opening suggest the following simple interactive proof system (P,V) for CL(MON;L) (a formal description is found in Figure 2). Let $(\phi, \mathcal{I})$ be the input to the proof system. The prover chooses an m-tuple Y that can be $(\phi, \mathcal{I})$-opened as 0 and sends it to the verifier. The verifier picks a bit b at random and sends it to the prover. The prover then shows an opening of Y as b and sends it to the verifier. The verifier accepts if the prover complies with his request. Completeness follows from Lemma 10 and soundness from Lemma 11. To guarantee zero knowledge, we have the prover pick Y using the procedure *Construct-Opening* found in Figure 1.

The following lemma can be easily proved by induction on the length of Y.

LEMMA 12. *Let (Y, R) be a pair output by Construct-Opening on input $(\phi, \mathcal{I}, a)$. Then Y can be $(\phi, \mathcal{I})$-opened as a, and R is a $(\phi, \mathcal{I})$-opening of Y as a.*

In Figure 2, we formally describe proof system (P,V).

THEOREM 13. *The pair* (P,V) *is an interactive proof system for* CL(MON;L).

Proof. First observe that the verification step V.2 can be performed in polynomial time because both $Sample_{\mathcal{R}}$ and $\mathcal{R}$ are polynomial time.

Completeness. From Lemma 10, it follows that if $(\phi, \mathcal{I}) \in$ CL(MON;L), then the prover can always provide V with the requested opening of Y.

Soundness. Suppose that $(\phi, \mathcal{I}) \notin$ CL(MON;L). Let Y' be the m-tuple given to the verifier by prover P* and suppose that Y' can be opened as b. (If Y' cannot be opened at all, then the verifier will never accept.) With probability 1/2 the verifier asks the prover to open Y as $1 - b$, and the prover will not be able to satisfy the verifier's request. (See Lemma 11.) Therefore, the prover has probability at most 1/2 to make the verifier accept $(\phi, \mathcal{I})$. □

4.1. The simulator for (P,V). In this section we exhibit a simulator for (P,V). The simulator uses procedure *Construct-Opening* described in Figure 1 that on input $(\phi, \mathcal{I}, a)$ outputs a pair (Y, R) such that R is a $(\phi, \mathcal{I})$-opening as a of Y.

In the next lemma we prove that for (Y, R) output by procedure *Construct-Opening* on input $(\phi, \mathcal{I}, a)$ we have that

1. Y is independent from a given ϕ and $\mathcal{I}$;

Procedure *Construct-Opening*$(\phi, \mathcal{I}, a)$
Input:

- ϕ is a monotone formula, and let m be the number of variables of ϕ;
- $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$;
- $a \in \{0, 1\}$.

1. If ϕ consists of one variable v_1, then
   run $Generate_{\mathcal{R}}(x_1)$ obtaining (s_1, t_1);
   randomly choose $r_1 \in \mathcal{S}_{x_1}$;
   if $a = 0$, then set $y_1 = Sample_{\mathcal{R}}(x_1, z_1, r_1)$;
   if $a = 1$, then set $y_1 = Sample_{\mathcal{R}}(x_1, s_1, r_1)$;
   return$((y_1), (a, r_1, s_1, t_1))$;
2. If $\phi = \phi_1 \vee \phi_2$, then
   let m_1 be the number of variables of ϕ_1;
   let $\mathcal{I}_1 = ((x_1, z_1), \ldots, (x_{m_1}, z_{m_1}))$ and let $\mathcal{I}_2 = ((x_{m_1+1}, z_{m_1+1}), \ldots, (x_m, z_m))$;
   randomly choose $b \in \{0, 1\}$;
   obtain (Y_1, R_1) by running *Construct-Opening* on input $(\phi_1, \mathcal{I}_1, b)$;
   obtain (Y_2, R_2) by running *Construct-Opening* on input $(\phi_2, \mathcal{I}_2, a \oplus b)$;
   return$(Y_1 \circ Y_2, R_1 \circ R_2)$.
3. If $\phi = \phi_1 \wedge \phi_2$, then
   let m_1 be the number of variables of ϕ_1;
   let $\mathcal{I}_1 = ((x_1, z_1), \ldots, (x_{m_1}, z_{m_1}))$ and let $\mathcal{I}_2 = ((x_{m_1+1}, z_{m_1+1}), \ldots, (x_m, z_m))$;
   obtain (Y_1, R_1) by running *Construct-Opening* on input $(\phi_1, \mathcal{I}_1, a)$;
   obtain (Y_2, R_2) by running *Construct-Opening* on input $(\phi_2, \mathcal{I}_2, a)$;
   return$(Y_1 \circ Y_2, R_1 \circ R_2)$.

FIG. 1. *The procedure Construct-Opening.*

The Proof System (P,V).
Input: $(\phi, \mathcal{I}) \in$ CL(MON;L).

P.1 Obtain (Y, R) by running *Construct-Opening* on input $(\phi, \mathcal{I}, 0)$ and let S be the auxiliary sequence.
Send (Y, S) to V.
V.1 Randomly select $b \in \{0, 1\}$. Send b to P.
P.2 Select a $(\phi, \mathcal{I})$-opening R_b of Y as b according to the distribution induced on its second output by procedure *Construct-Opening* on input $(\phi, \mathcal{I}, b)$ conditioned on Y being the first element of the output pair and S being the auxiliary sequence.
Send R_b to V.
V.2 Verify that R_b is a $(\phi, \mathcal{I})$-opening of Y as b.

FIG. 2. *The proof system* (P,V) *for* CL(MON;L).

2. the auxiliary sequence S is independent from a given ϕ, $\mathcal{I}$, and Y.

LEMMA 14. *Let* $(\phi,\mathcal{I}) \in \mathrm{CL(MON;L)}$ *and denote by* $\mathrm{Prob}(Y,S,\phi,\mathcal{I},a)$ *the probability that Construct-Opening*$(\phi,\mathcal{I},a)$ *outputs the sequence* Y *and a* $(\phi,\mathcal{I})$*-opening* R *of* Y *as* a *with auxiliary sequence* S*. Then, for each* Y *and* S *we have*

$$\mathrm{Prob}(Y,S,\phi,\mathcal{I},0) = \mathrm{Prob}(Y,S,\phi,\mathcal{I},1).$$

Proof. The proof proceeds by induction on the number m of pairs of $\mathcal{I}$.

Base case: $m = 1$. In this case we have $\mathcal{I} = ((x,z))$, $Y = (y)$, and $S = (s,t)$.

First of all, observe that, independently of a, S is produced by running procedure *Generate*$_{\mathcal{R}}$. Moreover, since $(\phi,\mathcal{I}) \in \mathrm{CL(MON;L)}$, it must be the case that $z \in \mathrm{dom}_{\mathcal{R}}(x)$.

If $a = 0$, then y is the output of *Sample*$_{\mathcal{R}}$ on input x, z, and a random $r \in \mathcal{S}_x$ and thus, by property 1 of RSR languages and since $z \in \mathrm{dom}_{\mathcal{R}}(x)$, y is uniformly distributed over $\mathrm{dom}_{\mathcal{R}}(x)$.

Similarly, if $a = 1$, then y is the output of *Sample*$_{\mathcal{R}}$ on input x, s, and a random $r \in \mathcal{S}_x$. By property 1 of RSR languages and since $s \in \mathrm{dom}_{\mathcal{R}}(x)$, y is uniformly distributed over $\mathrm{dom}_{\mathcal{R}}(x)$. Therefore, independently from the values of a and s, y is a random element of $\mathrm{dom}_{\mathcal{R}}(x)$.

Finally, we observe that, by property 1 of RSR languages, given y, z, and s all belonging to $\mathrm{dom}_{\mathcal{R}}(x)$, there exist (the same number of) r' and r'' such that $y = \mathit{Sample}(x,z,r')$ and $y = \mathit{Sample}(x,s,r'')$.

Induction step: $m > 1$. Suppose that $\phi = \phi_1 \vee \phi_2$ and denote by m_1 the number of variables of ϕ_1 and let $\mathcal{I}_1 = ((x_1,z_1),\ldots,(x_{m_1},z_{m_1}))$ and $\mathcal{I}_2 = ((x_{m_1+1},z_{m_1+1}), \ldots,(x_m,z_m))$. Write $Y = (y_1,\ldots,y_m)$ and $S = ((s_1,t_1),\ldots,(s_m,t_m))$ and set $Y_1 = (y_1,\ldots,y_{m_1})$, $S_1 = ((s_1,t_1),\ldots,(s_{m_1},t_{m_1}))$, and $Y_2 = (y_{m_1+1},\ldots,y_m)$, $S_2 = ((s_{m_1+1},t_{m_1+1}),\ldots,(s_m,t_m))$. Then, we have

$$\begin{aligned}\mathrm{Prob}(Y,S,\phi,\mathcal{I},0) = \quad & \mathrm{Prob}(b=0 \text{ at step 2 of } \textit{Construct-Opening}) \\ & \cdot\mathrm{Prob}(Y_1,S_1,\phi_1,\mathcal{I}_1,0)\cdot\mathrm{Prob}(Y_2,S_2,\phi_2,\mathcal{I}_2,0) \\ + \quad & \mathrm{Prob}(b=1 \text{ at step 2 of } \textit{Construct-Opening}) \\ & \cdot\mathrm{Prob}(Y_1,S_1,\phi_1,\mathcal{I}_1,1)\cdot\mathrm{Prob}(Y_2,S_2,\phi_2,\mathcal{I}_2,1)\end{aligned}$$

and

$$\begin{aligned}\mathrm{Prob}(Y,S,\phi,\mathcal{I},1) = \quad & \mathrm{Prob}(b=0 \text{ at step 2 of } \textit{Construct-Opening}) \\ & \cdot\mathrm{Prob}(Y_1,S_1,\phi_1,\mathcal{I}_1,1)\cdot\mathrm{Prob}(Y_2,S_2,\phi_2,\mathcal{I}_2,0) \\ + \quad & \mathrm{Prob}(b=1 \text{ at step 2 of } \textit{Construct-Opening}) \\ & \cdot\mathrm{Prob}(Y_1,S_1,\phi_1,\mathcal{I}_1,0)\cdot\mathrm{Prob}(Y_2,S_2,\phi_2,\mathcal{I}_2,1).\end{aligned}$$

Since $(\phi,\mathcal{I}) \in \mathrm{CL(MON;L)}$ and $\phi = \phi_1 \vee \phi_2$, we can assume without loss of generality that $(\phi_1,\mathcal{I}_1) \in \mathrm{CL(MON;L)}$. Therefore, by the inductive hypothesis we have

$$\mathrm{Prob}(Y_1,S_1,\phi_1,\mathcal{I}_1,0) = \mathrm{Prob}(Y_1,S_1,\phi_1,\mathcal{I}_1,1)$$

from which the lemma follows.

The case $\phi = \phi_1 \wedge \phi_2$ is proved similarly. □

In Figure 3, we present a simulator S_{V^*} for each verifier V^*. In the formal description of S_{V^*}, we say that a random tape of appropriate length is picked for V^*. By this we mean that S_{V^*} picks $p(n)$ random bits for an input of length n, where $p(n)$ is a polynomial that upper bounds the number of steps performed by V^* on inputs of length n.

The simulator S_{V^*}
Input: $(\phi, \mathcal{I}) \in$ CL(MON;L).

1. Randomly choose a string *Random* of appropriate length and write it on the random tape of the verifier V^*.
2. Randomly choose $a \in \{0,1\}$ and run procedure *Construct-Opening* on input $(\phi, \mathcal{I}, a)$ obtaining (Y, R). Let S be the auxiliary sequence of R.
   Send (Y, S) to V^*.
3. Receive string σ from V^*, and let b be its first bit.
   If $a = b$, then **Output**$(Random; Y, S, b, R)$ and STOP; else Goto 1.

FIG. 3. *The simulator* S_{V^*}

LEMMA 15. *Assume* $(\phi, \mathcal{I}) \in$ CL(MON;L) *and let* V^* *be a verifier. Then, the simulator* S_{V^*} *runs in expected polynomial time. Moreover, the probability spaces* $S_{V^*}(\phi, \mathcal{I})$ *of the output by* S_{V^*} *on input* $(\phi, \mathcal{I})$ *and the probability space of* V^**'s view,* $View_{V^*}(\phi, \mathcal{I})$*, on input* $(\phi, \mathcal{I})$ *are identical.*

Proof. Since $Sample_{\mathcal{R}}$ is polynomial time, $Generate_{\mathcal{R}}$ is expected polynomial time, and $\mathcal{S}_x$ is efficiently samplable, *Construct-Opening* can be executed in expected polynomial time.

Observe that since $(\phi, \mathcal{I}) \in$ CL(MON;L), by Lemma 14, (Y, S) is independent from a. Therefore the probability that $b \neq a$ is exactly $1/2$; thus S_{V^*} runs in expected polynomial time.

The simulator produces strings of the form $(Random; Y, S, b, R)$, where *Random* is a string of bits, Y is an m-tuple, S is the auxiliary sequence of R, $b \in \{0,1\}$, and R is a random opening of Y as b, conditioned on S. Obviously, *Random* has the same distribution in both spaces.

In the output of S_{V^*} on input $(\phi, \mathcal{I})$, (Y, S) is picked according to the probability induced by *Construct-Opening* on input $(\phi, \mathcal{I}, 0)$ with probability $1/2$, and with probability $1/2$ according to the probability induced by *Construct-Opening* on input $(\phi, \mathcal{I}, 1)$. Therefore, by Lemma 14, (Y, S) is picked according to the probability induced by *Construct-Opening* on input $(\phi, \mathcal{I}, 0)$ which is exactly the distribution of (Y, S) in the view of V^*.

The bit b is, in both spaces, the output of V^* on input $(\phi, \mathcal{I})$ and (Y, S) computed using *Random* as source of random bits. Finally, R is in both spaces an opening of Y as b picked according to the distribution induced on R by *Construct-Opening* on input $(\phi, \mathcal{I}, b)$ conditioned on Y being the first element of the output pair and S being the auxiliary sequence. □

Thus, we obtain the following theorem.

THEOREM 16. *Let* L *be an RSR language. Then* CL(MON;L) $\in$ *PZK.*

Remark 3 (the properties of RSR languages used). Notice that the results of this section use only properties 1, 2, and 3 of the definition of RSR languages.

5. Monotone formulae over co-RSR languages. In this section we prove that the language of all true monotone formulae whose atoms are statements about membership in a co-RSR language has a PZK proof system.

We start from the following simple observation. De Morgan's law specifies a simple transformation DM by which it is possible to transform any monotone formula

ϕ into a monotone formula $\psi = DM(\phi)$ of essentially the same size and such that ϕ being evaluated with atoms $(x_i \in \bar{L})$ is equivalent to $\neg\psi$ being evaluated with atoms $(x_i \in L)$. In other words, proving that a monotone formula over a co-RSR language is true is equivalent to proving that a monotone formula over an RSR language is false. This can be done using the concept of opening in the following way.

Suppose the prover wants to prove that $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$, where L is an RSR language. The verifier first constructs formula $\psi = DM(\phi)$, runs procedure *Construct-Opening* on input $\psi, \mathcal{I}$ and a randomly selected bit c, obtains a pair (Y, R), and sends Y to the prover. The prover then has to guess the value of c. Completeness and soundness follow directly from Lemmas 11 and 14, and zero knowledge with respect to a honest verifier can also be easily argued.

In the formal description of the proof system, we denote by *co-Construct-Opening* the procedure that, on input $(\phi, \mathcal{I}, a)$, first constructs $\psi = DM(\phi)$ and then runs *Construct-Opening* on input $(\psi, \mathcal{I}, a)$. Also, we will say that the m-tuple of quadruples $R = ((c_1, r_1, s_1, t_1), \ldots, (c_m, r_m, s_m, t_m))$ is a $(\phi, \mathcal{I})$-*co-opening* as $b \in \{0, 1\}$ of the m-tuple $Y = (y_1, \ldots, y_m)$ if and only if R is a $(DM(\phi), \mathcal{I})$-opening as b of Y. From the discussion above and from the properties of an opening, it follows that if $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$, then any tuple Y which can be opened as a 0 cannot be opened as a 1 and vice versa. On the other hand, if $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$, then any tuple Y which can be opened as a 0 can also be opened as a 1 and vice versa.

Next we observe that, as a direct consequence of De Morgan's law, the procedure *co-Construct-Opening* can be obtained by swapping the roles of $\vee$ and $\wedge$ in the definition of *Construct-Opening*. Similarly, we define the *co-Val* of a sequence $(c_1, \ldots, c_m)$ with respect to a formula ϕ as the *Val* of $(c_1, \ldots, c_m)$ with respect to $DM(\phi)$. We also say that $(c_1, \ldots, c_m)$ is *co-well-formed* with respect to ϕ if *co-Val*$(c_1, \ldots, c_m; \phi) \neq \perp$, and, analogously to Fact 8, we obtain the following fact.

FACT 17. *If $\vec{a} = (a_1, \ldots, a_m)$ and $\vec{c} = (c_1, \ldots, c_m)$ are two co-well-formed sequences with respect to ϕ, then so is sequence $\vec{b} = (a_1 \oplus c_1, \ldots, a_m \oplus c_m)$. Moreover, the number of sequences $\vec{a}$ such that co-Val$(\vec{a}; \phi) = 0$ is equal to the number of sequences $\vec{a}$ such that co-Val$(\vec{a}; \phi) = 1$.*

5.1. Zero knowledge. Designing a zero-knowledge proof system (and not just an honest-verifier zero-knowledge system) requires some extra care. Following the suggestion of [42], we have the verifier give a "proof that he knows the value of c." Only after receiving such a "proof" does the prover give the verifier his guess for c. At the base of our construction is the notion of similarity that can be informally presented as follows. We say that the pair (u_0, u_1) is *similar* to the pair (x, z) if one of the elements of (u_0, u_1) is obtained by running $Sample_{\mathcal{R}}$ on input (x, s, r'), and the other is obtained by running $Sample_{\mathcal{R}}$ on input (x, z, r''), where $s \in \mathrm{dom}_{\mathcal{R}}(x)$ and $r', r'' \in \mathcal{S}_x$. The fact that pair (u_0, u_1) is similar to pair (x, z) can be proved by giving $s \in \mathrm{dom}_{\mathcal{R}}(x)$, an x-witness for s, and the random strings r', r'' used by $Sample_{\mathcal{R}}$ to generate (u_0, u_1). The following definition extends the concept of similarity to m-tuples of pairs with respect to a formula.

DEFINITION 18. *Let ϕ be a formula with m variables. We say that the m-tuple of pairs $\mathcal{H} = ((u_{10}, u_{11}), \ldots, (u_{m0}, u_{m1}))$ is* similar to the m-tuple of pairs $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$ with respect to ϕ *if and only if, for $i = 1, \ldots, m$, there exist $(s_i, t_i) \in \mathcal{R}_{x_i}$, $r'_i, r''_i \in \mathcal{S}_{x_i}$, and $a_i \in \{0, 1\}$ such that*

- *if $a_i = 0$, then $u_{i0} = Sample_{\mathcal{R}}(x_i, z_i, r'_i)$ and $u_{i1} = Sample_{\mathcal{R}}(x_i, s_i, r''_i)$;*
- *if $a_i = 1$, then $u_{i0} = Sample_{\mathcal{R}}(x_i, s_i, r'_i)$ and $u_{i1} = Sample_{\mathcal{R}}(x_i, z_i, r''_i)$;*
- *co-Val$(a_1, \ldots, a_m; \phi) \in \{0, 1\}$.*

If $\mathcal{H}$ is similar to $\mathcal{I}$ with respect to ϕ, we say that the vector $W = (w_1, \ldots, w_m)$, where $w_i = (s_i, t_i, r'_i, r''_i, a_i)$ for $i = 1, \ldots, m$, is a witness of the similarity of $\mathcal{H}$ *to* $\mathcal{I}$.

In the following whenever the formula ϕ is clear from the context, we will simply say "similar."

In Figure 4, we present the procedure *Construct-Similar* that can be used to construct similar tuples. We summarize the properties of *Construct-Similar* in the following lemma, whose proof is simple.

Procedure *Construct-Similar*$(\phi, \mathcal{I}, S, a)$.
Input:

- ϕ is a monotone formula on m variables;
- $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$;
- $S = ((s_1, t_1), \ldots, (s_m, t_m))$;
- $a \in \{0, 1\}$.

1. If ϕ consists of one literal v_1, then
   randomly choose $r', r'' \in \mathcal{S}_{x_1}$;
   if $a = 0$, then set $u_0 = \mathit{Sample}_{\mathcal{R}}(x_1, z_1, r')$ and $u_1 = \mathit{Sample}_{\mathcal{R}}(x_1, s_1, r'')$;
   if $a = 1$, then set $u_0 = \mathit{Sample}_{\mathcal{R}}(x_1, s_1, r')$ and $u_1 = \mathit{Sample}_{\mathcal{R}}(x_1, z_1, r'')$;
   return$((u_0, u_1), (s_1, t_1, r', r'', a))$.
2. If $\phi = \phi_1 \wedge \phi_2$, then
   let m_1 be the number of variables of ϕ_1;
   let $\mathcal{I}_1 = ((x_1, z_1), \ldots, (x_{m_1}, z_{m_1}))$ and $\mathcal{I}_2 = ((x_{m_1+1}, z_{m_1+1}), \ldots, (x_m, z_m))$;
   let $S_1 = ((s_1, t_1), \ldots, (s_{m_1}, t_{m_1}))$ and $S_2 = ((s_{m_1+1}, t_{m_1+1}), \ldots, (s_m, t_m))$;
   randomly choose $\tilde{a} \in \{0, 1\}$;
   obtain $(\mathcal{H}_1, W_1)$ by running *Construct-Similar* on input $(\phi_1, \mathcal{I}_1, S_1, \tilde{a})$;
   obtain $(\mathcal{H}_2, W_2)$ by running *Construct-Similar* on input $(\phi_2, \mathcal{I}_2, S_2, \tilde{a} \oplus a)$;
   return$(\mathcal{H}_1 \circ \mathcal{H}_2, W_1 \circ W_2)$.
3. If $\phi = \phi_1 \vee \phi_2$, then
   let m_1 be the number of variables of ϕ_1;
   let $\mathcal{I}_1 = ((x_1, z_1), \ldots, (x_{m_1}, z_{m_1}))$ and $\mathcal{I}_2 = ((x_{m_1+1}, z_{m_1+1}), \ldots, (x_m, z_m))$;
   let $S_1 = ((s_1, t_1), \ldots, (s_{m_1}, t_{m_1}))$ and $S_2 = ((s_{m_1+1}, t_{m_1+1}), \ldots, (s_m, t_m))$;
   obtain $(\mathcal{H}_1, W_1)$ by running *Construct-Similar* on input $(\phi_1, \mathcal{I}_1, S_1, a)$;
   obtain $(\mathcal{H}_2, W_2)$ by running *Construct-Similar* on input $(\phi_2, \mathcal{I}_2, S_2, a)$;
   return$(\mathcal{H}_1 \circ \mathcal{H}_2, W_1 \circ W_2)$.

FIG. 4. *The procedure Construct-Similar.*

LEMMA 19. *Let ϕ be a formula over m variables, let $(s_i, t_i) \in \mathcal{R}_{x_i}$ for $i = 1, \ldots, m$, and let $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$ and $S = ((s_1, t_1), \ldots, (s_m, t_m))$. Procedure Construct-Similar on input $\phi, \mathcal{I}, S$ and $a \in \{0, 1\}$ outputs in expected polynomial time an m-tuple $\mathcal{H} = ((u_{10}, u_{11}), \ldots, (u_{m0}, u_{m1}))$ similar to $\mathcal{I}$ along with*

a witness of similarity $W = ((s_1, t_1, r'_1, r''_1, a_1), \ldots, (s_m, t_m, r'_m, r''_m, a_m))$ *such that co-Val*$(a_1, \ldots, a_m; \phi) = a$. *Moreover, the ith pair* (u_{i0}, u_{i1}) *of the m-tuple* $\mathcal{H}$ *is constructed by running Sample*$_{\mathcal{R}}$ *twice: on input* x_i, z_i *and* $r'_i \in \mathcal{S}_{x_i}$ *and on input* x_i, s_i *and* $r''_i \in \mathcal{S}_{x_i}$.

An informal presentation. The zero-knowledge proof system for proving that $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$ can be informally presented in the following way. The verifier V randomly selects a bit c, computes an m-tuple $Y = (y_1, \ldots, y_m)$ that can be $(\phi, \mathcal{I})$-co-opened as c, and sends it to the prover P. The prover has to guess the value of c. By the definition of co-opening and the properties of an opening, we have that if $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$, then the prover has a way of guessing what c is; if instead $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$, then he can guess the value of c with probability at most $1/2$. However, to ensure the zero-knowledge property, before the prover sends his guess for c, the prover and the verifier perform the following protocol $l(|Y|)$ times in parallel, where $l(\cdot)$ is a polynomial to be specified later. V computes an m-tuple of pairs $\mathcal{H} = ((u_{10}, u_{11}), \ldots, (u_{m0}, u_{m1}))$ that is similar to $\mathcal{I}$ with respect to ϕ and has the following additional property: there exists a sequence $(b_1, \ldots, b_m)$ such that *co-Val*$(b_1, \ldots, b_m; \phi) \in \{0, 1\}$, and, for $i = 1, \ldots, m$, there exists $r_{y_i} \in \mathcal{S}_{x_i}$ such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{ib_i}, r_{y_i})$. The prover receives $\mathcal{H}$ and chooses at random to see either a witness of similarity of $\mathcal{H}$ to $\mathcal{I}$ or the sequence $(b_1, \ldots, b_m, r_1, \ldots, r_m)$. If his request is satisfied by the verifier, the prover gives his guess for the bit c.

A formal description of the proof system (P,V) is found in Figure 5.

V *runs in probabilistic polynomial time.* Let us now show that the verifier's program can be executed in probabilistic polynomial time.

At step V.0 the verifier constructs the m-tuple Y by running the procedure *co-Construct-Opening*. As seen from the description of *Construct-Opening* in Figure 1, by De Morgan's law and by the discussion in Remark 2, procedure co-Construct-Opening runs in polynomial time and returns the m-tuple Y and a $(\phi, \mathcal{I})$-co-opening $R = ((c_1, r_1, s_1, t_1), \ldots, (c_m, r_m, s_m, t_m))$ of Y. The m-tuples Y and R are such that, for each $i = 1, \ldots, m$, the element y_i is obtained by running $\mathit{Sample}_{\mathcal{R}}$ either on input x_i, z_i and $r_i \in \mathcal{S}_{x_i}$ or on input x_i, s_i and $r_i \in \mathcal{S}_{x_i}$.

At step V.1, the verifier executes procedure *Construct-Similar* on input $\phi, \mathcal{I}$, the m-tuple $S = ((s_1, r_1), \ldots, (s_m, r_m))$, and a random bit a (see Figure 4 for a formal description of procedure *Construct-Similar*). By Lemma 19, procedure *Construct-Similar* runs in polynomial time and outputs an m-tuple $\mathcal{H}$ similar to $\mathcal{I}$ along with a witness of similarity

$$W = ((s_1, t_1, r'_1, r''_1, a_1), \ldots, (s_m, t_m, r'_m, r''_m, a_m))$$

of $\mathcal{H}$ to $\mathcal{I}$.

Step V.2 is the only nontrivial step of V's program that remains to be specified. If $d = 0$, then it is enough for V to send the witness of similarity W to P obtained by running *Construct-Similar* at step V.1. Let us now examine the case $d = 1$. In this case the verifier sets $b_i = a_i \oplus c_i$. Now observe that, since the sequences $(a_1, \ldots, a_m)$ and $(c_1, \ldots, c_m)$ are, by construction, co-well-formed with respect to ϕ, by Fact 17 so is the sequence $(b_1, \ldots, b_m)$, and thus the first verification of step P.2 in the case $d = 1$ is always successfully passed. Moreover, this setting allows the verifier to use algorithm $A_{\mathcal{R}}$ to compute r_{y_i} such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{ib_i}, r_{y_i})$, for $i = 1, \ldots, m$, as follows. First, suppose that $a_i = 1$ (the case $a_i = 0$ is similar) and thus $u_{i0} = \mathit{Sample}_{\mathcal{R}}(x_i, s_i, r'_i)$ and $u_{i1} = \mathit{Sample}_{\mathcal{R}}(x_i, z_i, r''_i)$ (see step 1 in the procedure *Construct-Similar*). If $c_i = 0$, we have $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, z_i, r_i)$ and $b_i = 1$.

The Proof System (P,V).
Input: $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$

V.0 Uniformly select $c \in \{0,1\}$;
run procedure *co-Construct-Opening* on input $(\phi, \mathcal{I}, c)$ obtaining (Y, R);
write R as $R = ((c_1, r_1, s_1, t_1), \ldots, (c_m, r_m, s_m, t_m))$ and Y as $Y = (y_1, \ldots, y_m)$ and set $S = ((s_1, t_1), \ldots, (s_m, t_m))$;
send (Y, S) to P.

P↔V Perform $l(|Y|)$ parallel and independent executions of the following protocol.
begin

V.1 Uniformly select $a \in \{0,1\}$;
run procedure *Construct-Similar* on input $(\phi, \mathcal{I}, S, a)$ obtaining $(\mathcal{H}, W)$;
write $\mathcal{H}$ as $\mathcal{H} = ((u_{10}, u_{11}), \ldots, (u_{m0}, u_{m1}))$;
write W as $W = ((s_1, t_1, r'_1, r''_1, a_1), \ldots, (s_m, t_m, r'_m, r''_m, a_m))$;
send $\mathcal{H}$ to P.

P.1 Uniformly select $d \in \{0,1\}$ and send d to V.

V.2 If $d = 0$, then send W to P;
if $d = 1$, then
for $i = 1, \ldots, m$
set $b_i = a_i \oplus c_i$;
use algorithm $A_{\mathcal{R}}$ to compute r_{y_i} such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{ib_i}, r_{y_i})$;
send $T = (b_1, r_{y_1}, \ldots, b_m, r_{y_m})$ to P.

P.2 If $d = 0$, then check that W is a witness of similarity of $\mathcal{H}$ to $\mathcal{I}$;
if $d = 1$, then
check that $\mathit{co\text{-}Val}(b_1, \ldots, b_m; \phi) \in \{0,1\}$;
check that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{ib_i}, r_{y_i})$ for $i = 1, \ldots, m$.

end

P.3 If any of the verifications of the $l(|Y|)$ parallel executions is not satisfied, then STOP;
else compute $c' \in \{0,1\}$ such that Y can be $(\phi, \mathcal{I})$-co-opened as c'.
Send c' to V.

V.3 Verify that $c = c'$. If not, reject.

FIG. 5. *The proof system* (P,V) *for* $\mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$.

By the first part of property 4 of the RSR languages, the verifier can compute in polynomial time r_{y_i} such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{i1}, r_{y_i})$. On the other hand, if $c_i = 1$, we have $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, s_i, r_i)$ (see step 1 of procedure *Construct-Opening*) and $b_i = 0$. Again, by property 4 of the RSR languages the verifier can compute r_{y_i} such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{i0}, r_{y_i})$.

Completeness and soundness. The completeness property is easily proved and follows directly from Lemma 11 and from the discussion above.

For the soundness we assume that $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$. Under this assumption, we prove that the distribution of all that the prover sees (that is, the pair (Y, S), the $\mathcal{H}$'s, and the verifier's answers in the $l(n)$ execution of the subprotocol) is independent

from that of bit c. Thus, the probability that the prover correctly guesses the value of c by seeing V's answers to his challenges is at most $1/2$.

We start with pair (Y, S); the following lemma directly implies that the distribution of such a pair is independent from that of bit c when $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$.

LEMMA 20. *Let* $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ *and denote by* $\mathrm{Prob}(S, Y, \phi, \mathcal{I}, a)$ *the probability that co-Construct-Opening outputs the sequence* Y *and a* $(\phi, \mathcal{I})$*-opening* R *of* Y *as a with auxiliary sequence* S*. Then, for each* Y *and* S *we have*

$$\mathrm{Prob}(S, Y, \phi, \mathcal{I}, 0) = \mathrm{Prob}(S, Y, \phi, \mathcal{I}, 1).$$

Proof. Observe that $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ implies $(DM(\phi), \mathcal{I}) \in \mathrm{CL}(\mathrm{MON};\mathrm{L})$. The lemma thus follows from Lemma 14. □

For the jth parallel execution we denote by $\mathcal{H}^j$ the value sent by the verifier to the prover at step V.1, by d^j the bit sent by the prover to the verifier at step P.1, and by A^j the answer sent by the verifier at step V.2 for $j \in \{1, \ldots, l(|Y|)\}$.

We now see that when $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$, the distribution of tuple $\vec{\mathcal{H}} = (\mathcal{H}^1, \ldots, \mathcal{H}^{l(|Y|)})$, conditioned by the value of pair (Y, S), is independent from that of bit c.

LEMMA 21. *We let* $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ *and denote by* $\mathrm{Prob}(\vec{\mathcal{H}} \mid S, Y, \phi, \mathcal{I}, c)$ *the probability that Construct-Similar outputs the sequence* $\vec{\mathcal{H}}$ *given that co-Construct-Opening*$(\phi, \mathcal{I}, c)$ *returns* (Y, S) *with* S *as the auxiliary sequence in* R*. Then, for each* $\vec{\mathcal{H}}$ *we have that*

$$\mathrm{Prob}(\vec{\mathcal{H}} \mid S, Y, \phi, \mathcal{I}, 0) = \mathrm{Prob}(\vec{\mathcal{H}} \mid S, Y, \phi, \mathcal{I}, 1).$$

Proof. Note that $\vec{\mathcal{H}} = \mathcal{H}^1, \ldots, \mathcal{H}^{l(|Y|)}$. The lemma follows by observing that each $\mathcal{H}^j$ is generated as the first component of a pair returned by an independent execution of algorithm *Construct-Similar* on input $\phi, \mathcal{I}, S$ and a random and independent bit a, where we use the fact that the distribution of S is independent from that of c, as implied by Lemma 20. □

We now prove that, for $j = 1, \ldots, l(|Y|)$, the distribution of the answer A^j, conditioned on the values $Y, S, \vec{\mathcal{H}}$, $\vec{d} = (d^1, \ldots, d^{l(|Y|)})$, and $A^1, \ldots, A^{j-1}$, is independent from that of bit c. To this purpose, we distinguish two cases, according to the value of bit d^j.

First, consider the case $d^j = 0$. In this case A^j is generated as the second component W^j of the pair returned by an independent execution of algorithm *Construct-Similar* on input $\phi, \mathcal{I}, S$ and a random bit a. Here we can again use the fact that the distribution of S is independent from that of c, as implied by Lemma 20. Therefore, in this case, the distribution of A^j, conditioned on the values of $Y, S, \vec{\mathcal{H}}, \vec{d}, A^1, \ldots, A^{j-1}$, is independent from that of c.

The rest of the proof for the soundness property refers only to the case $d^j = 1$. In this case A^j is equal to sequence $T^j = (b_1^j, r_{y_1}, \ldots, b_m^j, r_{y_m})$. We first note that, even with further conditioning of the values of $b_1^j, \ldots, b_m^j$, each r_{y_i} is, by property 4 of the RSR languages, distributed as a random and independent value r such that $y_i = \mathit{Sample}(x_i, u_{iq}, r)$ for $q = b_i^j$. Therefore, to conclude the proof for this case, it is enough to prove that the distribution of sequence $(b_1^j, \ldots, b_m^j)$, conditioned on $Y, S, \vec{\mathcal{H}}, \vec{d}, A^1, \ldots, A^{j-1}$, is independent from c. We start the proof of this latter fact with some definitions.

DEFINITION 22. *Let $c \in \{0,1\}$, and let (Y,R) be the output of co-Construct-Opening on input $(\phi,\mathcal{I},c)$, where $Y=(y_1,\ldots,y_m)$, $R=((a_1,r_1,s_1,t_1),\ldots,(a_m,r_m,s_m,t_m))$, and $S=((s_1,t_1),\ldots,(s_m,t_m))$ denotes the auxiliary sequence. Also, let $\mathcal{H}=((u_{01},u_{11}),\ldots,(u_{0m},u_{1m}))$ be the first component of the pair returned by algorithm Construct-Similar on input $(\phi,\mathcal{I},S,a)$. Let sequence $(e_1,\ldots,e_m)\in\{0,1\}^m$ be co-well-formed with respect to ϕ; we say that $(e_1,\ldots,e_m)$ is*

- $(\phi,\mathcal{I},S,Y)$-consistent *if there exist $e\in\{0,1\}$ and $r_i\in\mathcal{S}_{x_i}$, for $i=1,\ldots,m$, such that the tuple $((e_1,r_1,s_1,t_1),\ldots,(e_m,r_m,s_m,t_m))$ is a $(\phi,\mathcal{I})$-co-opening of Y as e;*
- $(\phi,\mathcal{I},S,\mathcal{H})$-consistent *if there exist $e\in\{0,1\}$ and $r'_i,r''_i\in\mathcal{S}_{x_i}$, for $i=1,\ldots,m$, such that $\mathcal{H}$ is similar to $\mathcal{I}$ with respect to ϕ, and W is a witness of the similarity of $\mathcal{H}$ to $\mathcal{I}$, where $W=(w_1,\ldots,w_m)$, $w_i=(s_i,t_i,r'_i,r''_i,e_i)$, for $i=1,\ldots,m$, and co-Val$(e_1,\ldots,e_m)=e$;*
- $(\phi,\mathcal{I},S,Y,\mathcal{H})$-consistent *if there exists $e\in\{0,1\}$ such that co-Val$(e_1,\ldots,e_m)=e$, and, for $i=1,\ldots,m$, there exists r_{y_i} such that $y_i=$ Sample$_{\mathcal{R}}(x_i,u_{ie_i},r_{y_i})$.*

For any formula ϕ, and given $\mathcal{I},S,Y$, we denote by $\mathrm{Seq}(\phi,\mathcal{I},S,Y)$ *the set of sequences $(e_1,\ldots,e_m)$ that are $(\phi,\mathcal{I},S,Y)$-consistent; moreover, for any $j\in\{1,\ldots,l(|Y|)\}$, all $c\in\{0,1\}$, and given $\mathcal{H}^j$, we denote by* $\mathrm{Seq}(\phi,\mathcal{I},S,Y,\mathcal{H}^j)$ *(resp.,* $\mathrm{Seq}(\phi,\mathcal{I},S,\mathcal{H}^j)$*) the set of sequences $(e_1,\ldots,e_m)$ that are $(\phi,\mathcal{I},S,Y,\mathcal{H}^j)$-consistent (resp., $(\phi,\mathcal{I},S,\mathcal{H}^j)$-consistent).*

We will use the above definitions for the following three facts. First, we note that we can rephrase the analysis done while proving that V runs in polynomial time in step V.2 (in the case $d^j=1$) as follows.

FACT 23. *Let $j\in\{1,\ldots,l(|Y|)$ be such that $d_j=1$. If $(\phi,\mathcal{I})\notin\mathrm{CL}(\mathrm{MON};\bar{\mathrm{L}})$, then for any m-tuple $\vec{b}=(b_1,\ldots,b_m)\in\mathrm{Seq}(\phi,\mathcal{I},S,Y,\mathcal{H}^j)$, and any m-tuple $\vec{c}=(c_1,\ldots,c_m)\in\mathrm{Seq}(\phi,\mathcal{I},S,Y)$, there exists an m-tuple $\vec{a}=(a_1,\ldots,a_m)\in\mathrm{Seq}(\phi,\mathcal{I},S,\mathcal{H}^j)$ such that $a_i=b_i\oplus c_i$ for $i=1,\ldots,m$.*

Recall that by $\vec{c}=(c_1,\ldots,c_m)$ we denote the sequence of bits obtained from the output R returned by the execution of algorithm *co-Construct-Opening* in step V.0. We then note that, by definition of *co-Construct-Opening*, $\vec{c}$ is co-well-formed with respect to ϕ, and, by definition of step V.0, $\vec{c}\in\mathrm{Seq}(\phi,\mathcal{I},S,Y)$. For $c=0,1$, define $\mathrm{Seq}(\phi,\mathcal{I},S,Y,c)=\{\vec{c}\,|\,\vec{c}\in\mathrm{Seq}(\phi,\mathcal{I},S,Y),\text{co-Val}(\vec{c})=c\}$. Moreover, for $r=0,1,2$, define $\mathrm{Prob}_r[\,\vec{e}\,|\,tr,c,\ldots\,]$ as the probability that tuple $\vec{e}$ is computed by V during round V.r of the protocol, conditioned on the values $tr=(Y,S,\vec{\mathcal{H}},\vec{d},A^1,\ldots,A^{j-1})$, and by bit c chosen by V in step V.0. We now prove a useful fact about the distribution of sequence $\vec{c}$ with respect to the value of bit c chosen by V in step V.0.

FACT 24. *If $(\phi,\mathcal{I})\notin\mathrm{CL}(\mathrm{MON};\bar{\mathrm{L}})$, then*

$$\sum_{\vec{c}\in\,\mathrm{Seq}(\phi,\mathcal{I},S,Y,0)}\mathrm{Prob}_0[\,\vec{c}\,|\,tr,0\,]=\sum_{\vec{c}\in\,\mathrm{Seq}(\phi,\mathcal{I},S,Y,1)}\mathrm{Prob}_0[\,\vec{c}\,|\,tr,1\,].$$

Proof. As a first claim, we note that each tuple $\vec{c}$ co-well-formed with respect to ϕ has the same probability of being returned by algorithm *co-Construct-Opening* on input ϕ and a randomly chosen bit c. (This can be proved by simple induction over formula ϕ.) Using Fact 17, we note a second claim, stating that for any formula ϕ, the number of tuples $\vec{c}$ that can be returned by algorithm *co-Construct-Opening*, on input formula ϕ and c, is the same for both values of bit c.

Now, assume $(\phi,\mathcal{I})\notin\mathrm{CL}(\mathrm{MON};\bar{\mathrm{L}})$. By induction over formula ϕ, we can extend the above second claim to prove a third claim, saying that the number of tuples $\vec{c}$

belonging to set $\mathrm{Seq}(\phi, \mathcal{I}, S, Y, c)$ is the same for both values of bit c.

Specifically, the base case directly follows as an application of Lemma 10.

For the induction case, first consider the case $\phi = \phi_1 \vee \phi_2$, where ϕ_1 has m_1 variables. Then the claim follows by observing that the number of sequences $\vec{c}$ in $\mathrm{Seq}(\phi, \mathcal{I}, S, Y, c)$ is the product of the number of sequences $(c_1, \ldots, c_{m_1})$ in $\mathrm{Seq}(\phi_1, \mathcal{I}, S, Y, c)$ times the number of sequences $(c_{m_1+1}, \ldots, c_m)$ in $\mathrm{Seq}(\phi_2, \mathcal{I}, S, Y, c)$, and by applying the inductive hypothesis to both ϕ_1, ϕ_2, since our assumption implies that $(\phi_j, \mathcal{I}_j) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ for $j = 1, 2$.

Now, consider the case $\phi = \phi_1 \wedge \phi_2$, where ϕ_1 has m_1 variables. If $(\phi_j, \mathcal{I}_j) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ for $j = 1, 2$, then the claim follows similarly as for the previous case. Therefore, assume that $(\phi_1, \mathcal{I}_1) \notin \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ and $(\phi_2, \mathcal{I}_2) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ (the symmetric case being proved similarly). Note that Lemma 11 implies that, after conditioning on Y, there exists exactly one bit c_2 such that *co-Val*$(c_{m_1+1}, \ldots, c_m; \phi_2) = c_2$. Then the claim follows by applying the inductive hypothesis to ϕ_1 and by observing that the number of sequences $\vec{c}$ in $\mathrm{Seq}(\phi, \mathcal{I}, S, Y, c)$ is the product of the number of sequences $(c_1, \ldots, c_{m_1})$ in $\mathrm{Seq}(\phi_1, \mathcal{I}, S, Y, c \oplus c_2)$ times the number of sequences $(c_{m_1+1}, \ldots, c_m)$ in $\mathrm{Seq}(\phi_2, \mathcal{I}, S, Y, c_2)$, where we note that the latter number does not depend on c.

Then the fact follows by combining the first and third claims. □

Now, consider the jth execution of algorithm *Construct-Similar* in step V.2 while running the jth parallel iteration, and recall that we denote by $\vec{a} = (a_1, \ldots, a_m)$ the sequence of bits obtained from the output W returned by this execution. Here, we note that, by the definition of *Construct-Similar*, $\vec{a}$ is co-well-formed with respect to ϕ, and, moreover, using simple induction over formula ϕ, we obtain that $\vec{a} \in \mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$. For any subformula ψ of ϕ satisfying $(\psi, \mathcal{I}_\psi) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ and having leaves $i_1, \ldots, i_k$, we denote as $\mathcal{H}^j_\psi = ((u_{i_1,0}, u_{i_1,1}), \ldots, (u_{i_k,0}, u_{i_k,1}))$ the subsequence of $\mathcal{H}^j$, where each pair $(u_{i_j,0}, u_{i_j,1})$ is computed at step 1 of procedure *Construct-Similar* in correspondence to literal v_{i_j}. Note that an execution of algorithm *Construct-Similar* returning $\mathcal{H}^j$ implicitly defines a witness of similarity of $\mathcal{H}^j_\psi$ to $\mathcal{I}_\psi$, containing sequence $(a_{i_1}, \ldots, a_{i_k})$ such that *co-Val*$(a_{i_1}, \ldots, a_{i_k})$ is equal to a bit that we denote as $a_{\mathcal{H}^j,\psi}$. In the following fact we characterize set $\mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$ and the distribution of sequence $\vec{a}$ computed by V in step V.1.

FACT 25. *Let $j = 1, \ldots, l(|Y|)$ be such that $d_j = 1$; the following holds. Set $\mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$ is equal to the set of sequences $\vec{a}$ that are co-well-formed with respect to ϕ and such that for each subformula ψ of ϕ satisfying $(\psi, \mathcal{I}_\psi) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$ and having leaves $i_1, \ldots, i_k$, it holds that co-Val$(a_{i_1}, \ldots, a_{i_k}; \psi) = a_{\mathcal{H}^j,\psi}$. Moreover, the distribution of sequence $\vec{a}$ computed at step* V.1 *is uniform over set* $\mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$.

Proof. The definition of algorithm *Construct-Similar* already implies that set $\mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$ includes all co-well-formed sequences $\vec{a}$ satisfying *co-Val*$(a_{i_1}, \ldots, a_{i_k}; \psi) = a_{\mathcal{H}^j,\psi}$ for each subformula ψ of ϕ with leaves $i_1, \ldots, i_k$ and such that $(\psi, \mathcal{I}_\psi) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$. Now, consider all sequences $(a_1, \ldots, a_m)$ in $\mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$. By Definition 22, there exists a witness of the similarity between $\mathcal{H}^j$ and $\mathcal{I}$, and *co-Val*$(a_1, \ldots, a_m) \in \{0, 1\}$. This implies that for each subformula ψ of ϕ such that $(\psi, \mathcal{I}_\psi) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$, it holds that *co-Val*$(a_{i_1}, \ldots, a_{i_k})$ is equal to a bit that we denote as $a'_{\mathcal{H}^j,\psi}$. By Lemma 11, this bit is unique; that is, we have that $a'_{\mathcal{H}^j,\psi} = a_{\mathcal{H}^j,\psi}$. The claim on the distribution of sequence $\vec{a}$ follows by combining Fact 17 with the above characterization. □

We can now use the above facts to prove that the distribution of tuple $\vec{b}$ computed by V in step V.2, even conditioned over the view of P so far, is independent from bit

c chosen by V in step V.0. Specifically, for all $\vec{b} \in \mathrm{Seq}(\phi, \mathcal{I}, S, Y, \mathcal{H}^j)$, we have that

$$\begin{aligned}
\mathrm{Prob}_2[\,\vec{b}\,|\,tr, c\,] &= \sum_{\vec{c}} \mathrm{Prob}_0[\,\vec{c}\,|\,tr, c\,] \cdot \mathrm{Prob}_2[\,\vec{b}\,|\,tr, c, \vec{c}\,] \\
&= \sum_{\vec{c} \in \mathrm{Seq}(\phi,\mathcal{I},S,Y,c)} \mathrm{Prob}_0[\,\vec{c}\,|\,tr, c\,] \cdot \mathrm{Prob}_2[\,\vec{b}\,|\,tr, c, \vec{c}\,] \\
&= \sum_{\vec{c} \in \mathrm{Seq}(\phi,\mathcal{I},S,Y,c)} \mathrm{Prob}_0[\,\vec{c}\,|\,tr, c\,] \cdot \mathrm{Prob}_1[\,(b_1 \oplus c_1, \ldots, b_m \oplus c_m)\,|\,tr, c, \vec{c}\,] \\
&= \mathrm{Prob}_1[\,\vec{a}\,|\,tr\,] \cdot \sum_{\vec{c} \in \mathrm{Seq}(\phi,\mathcal{I},S,Y,c)} \mathrm{Prob}_0[\,\vec{c}\,|\,tr, c\,],
\end{aligned}$$

where the second equality follows from the fact that $\mathrm{Prob}_0[\,\vec{c}\,|\,tr, c\,] = 0$ when $\vec{c} \notin \mathrm{Seq}(\phi, \mathcal{I}, S, Y, c)$; the third equality follows from the fact that $\mathrm{Prob}_2[\,\vec{b}\,|\,tr, c, \vec{c}\,] = 0$ when $\vec{a} \neq (b_1 \oplus c_1, \ldots, b_m \oplus c_m)$ and from Fact 23; and the fourth equality follows by the definition of *Construct-Similar*, implying that the probability that tuple $(b_1 \oplus c_1, \ldots, b_m \oplus c_m)$ in $\mathrm{Seq}(\phi, \mathcal{I}, S, \mathcal{H}^j)$ is computed in step V.1 does not depend on the value of c or $\vec{c}$, as proved in Fact 25. Finally, we observe that even the factor $\sum_{\vec{c} \in \mathrm{Seq}(\phi,\mathcal{I},S,Y,c)} \mathrm{Prob}_0[\,\vec{c}\,|\,tr, c\,]$ does not depend on the value of c, as proved in Fact 24, and therefore neither does the value $\mathrm{Prob}_2[\,\vec{b}\,|\,tr, c\,]$. We obtain that from P's point of view, the distribution of sequence $\vec{b}$, even when conditioned from the transcript of the protocol so far, is independent from bit c chosen by V at step V.0. As discussed before, this implies the soundness property.

The simulator. Now we exhibit a simulator to prove the zero-knowledge property. The idea is to have the simulator run the protocol twice so as to get from the verifier both the witness of similarity and the two sequences $(b_1, \ldots, b_m)$ and $(r_{y_1}, \ldots, r_{y_m})$. As we shall prove next, this allows the simulator to compute the bit c. We give a formal description of the simulator in Figure 6.

LEMMA 26. *The simulator S_{V^*} runs in expected polynomial time.*

Proof. All the steps of S_{V^*} can be executed in probabilistic polynomial time except possibly step 7. For a random tape *Rand*, a formula ϕ, and a sequence $\mathcal{I}$, we denote by $\mathit{Good}(\mathit{Rand}, \phi, \mathcal{I})$ the set of l-bit sequences $(d_1, \ldots, d_l)$ for which V^* provides a correct answer and by $G = G(\mathit{Rand}, \phi, \mathcal{I})$ its size. Obviously, we have $0 \leq G \leq 2^l$. We distinguish three cases.

1. $G = 0$. In this case the simulator stops at step 6 for all values of $(d_1, \ldots, d_l)$.
2. $G = 1$. If $D \notin \mathit{Good}(\mathit{Rand}, \phi, \mathcal{I})$, then the simulator stops at step 6.
   If $D \in \mathit{Good}(\mathit{Rand}, \phi, \mathcal{I})$, which happens with probability 2^{-l}, then the simulator has to compute c such that Y can be co-opened as c.
   Therefore, if we let $l = l(|\phi| + |\mathcal{I}|)$, where $l(\cdot)$ is a polynomial such that the length of a $(\phi, \mathcal{I})$-co-opening of a sequence Y is at most $l(|\phi| + |\mathcal{I}|)$, and thus such a co-opening can be computed in time $O(\mathit{poly}(|\phi| + |\mathcal{I}|) \cdot 2^{l(|\phi|+|\mathcal{I}|)})$, the contribution of this case to the expected running time of the simulator is polynomial.
3. $G > 1$. The expected number of executions of step 7(a) is upper bounded by the probability that D is good (which equals $G/2^l$) times the expected number of trials until a good D' different from D is found (which equals $2^l/(G-1)$), for a total of $G/(G-1) \leq 2$. ∎

We now show that S_{V^*} is indeed a simulator for (P,V). Clearly, the random tape *Rand* and the sequence of bits $d_1, \ldots, d_m$ have the same distribution in the output of

The simulator S_{V^*}.
Input: $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \overline{\mathrm{L}})$, where ϕ is a formula with m variables.

1. Uniformly choose a string *Rand* (of appropriate length) and write it on the random tape of the verifier V^*.
2. Receive (Y, S) from V^* and denote $l(|Y|)$ simply by l.
3. Receive $\mathcal{H}_1, \ldots, \mathcal{H}_l$ from V^*.
4. Uniformly choose $D = (d_1, \ldots, d_l) \in \{0,1\}^l$. Send D to V^*.
5. Receive $mes = mes_1 \circ \ldots \circ mes_l$ from V^*.
6. If, for some i, mes_i is not a correct answer, then
   Output:$(Rand; Y, \mathcal{H}_1, \ldots, \mathcal{H}_l D, mes)$ and STOP.
7. Execute the following two steps (a) and (b) in parallel and continue to step 8 as soon as one of the two steps completes.
   (a) Repeatedly try to "extract" c from V^*
      (i) Rewind V^* to the state just after step V^*.1.
      (ii) Randomly choose sequence $D' = (d'_1, \ldots, d'_l)$.
      If $D = D'$, then go to step 7(a)(i) else send D' to V^*.
      (iii) Receive $mes' = mes'_1 \circ \cdots \circ mes'_l$ from V^*.
      (iv) If for some $j \in \{1, \ldots, l\}$, mes'_j is not a correct answer, then go to step 7(a)(i)
      (v) Let j be any index such that $d_j \neq d'_j$.
      (vi) If $d_j = 0$, then
         write mes_j as $mes_j = ((s_1, t_1, r'_1, r''_1, a_1), \ldots, (s_m, t_m, r'_m, r''_m, a_m))$;
         write mes'_j as $mes'_j = ((b_1, r_{y_1}), \ldots, (b_m, r_{y_m}))$;
      else
         write mes'_j as $mes'_j = ((s_1, t_1, r'_1, r''_1, a_1), \ldots, (s_m, t_m, r'_m, r''_m, a_m))$;
         write mes_j as $mes_j = ((b_1, r_{y_1}), \ldots, (b_m, r_{y_m}))$.
      (vii) Compute c as $c = co\text{-}Val(a_1 \oplus b_1, \ldots, a_m \oplus b_m; \phi)$.
   (b) Run an exhaustive search algorithm for computing bit c such that Y can be co-opened as c.
8. **Output:** $(Rand; Y, \mathcal{H}_1, \ldots, \mathcal{H}_l, D, mes, c)$ and STOP.

FIG. 6. *The simulator for verifier* V^*.

S_{V^*} as in the view of V^*. The next lemma shows that the bit c computed at step 7(a) is such that Y can be co-opened as c.

LEMMA 27. *Let* $W = ((s_1, t_1, r'_1, r''_1, a_1), \ldots, (s_m, t_m, r'_m, r''_m, a_m))$ *be a witness of the similarity of the* m*-tuple* $\mathcal{H} = ((u_{10}, u_{11}), \ldots, (u_{m0}, u_{m1}))$ *to the* m*-tuple* $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$*, let* $(b_1, \ldots, b_m)$ *be a well-formed sequence with respect to the formula* ϕ*, and let* $Y = (y_1, \ldots, y_m)$ *be a sequence.*

Then there exists a polynomial-time algorithm that, on input W *and* $r_i \in \mathcal{S}_{x_i}$ *such that* $y_i = Sample_{\mathcal{R}}(x_i, u_{ib_i}, r_i)$ *for* $i = 1, \ldots, m$*, computes a* $(\phi, \mathcal{I})$*-co-opening of* Y *as* $c = co\text{-}Val(a_1 \oplus b_1, \ldots, a_m \oplus b_m; \phi)$.

Proof. Let $c_i \stackrel{\text{def}}{=} a_i \oplus b_i$. We show that it is possible to compute in polynomial time, for $i = 1, \ldots, m$, r_{y_i} such that if $c_i = 0$, then $y_i = Sample_{\mathcal{R}}(x_i, z_i, r_{y_i})$, and if $c_i = 1$, then $y_i = Sample_{\mathcal{R}}(x_i, s_i, r_{y_i})$. The sequence $((c_1, r_{y_1}), \ldots, (c_m, r_{y_m}))$ is a co-opening of Y as c.

We consider only the case $c_i = 0$, the case $c_i = 1$ being similar. Assume $a_i = 0$ and $b_i = 0$. Then we have $u_{i0} = \mathit{Sample}_{\mathcal{R}}(x_i, z_i, r'_i)$, $u_{i1} = \mathit{Sample}_{\mathcal{R}}(x_i, s_i, r''_i)$, and $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{i0}, r_i)$. Using property 4 of RSR languages we have that it is possible to compute in polynomial time r_{y_i} such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, z_i, r_{y_i})$. If instead $a_i = 1$ and $b_i = 1$, we have that $u_{i0} = \mathit{Sample}_{\mathcal{R}}(x_i, s_i, r'_i)$, $u_{i1} = \mathit{Sample}_{\mathcal{R}}(x_i, z_i, r''_i)$, and $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, u_{i1}, r_i)$. Again by property 4 of the RSR languages it follows that it is possible to compute in polynomial time r_{y_i} such that $y_i = \mathit{Sample}_{\mathcal{R}}(x_i, z_i, r_{y_i})$. □

LEMMA 28. *The probability distributions* $S_{\mathrm{V}^*}(\phi, \mathcal{I})$ *and* $\mathit{View}_{\mathrm{V}^*}(\phi, \mathcal{I})$ *are identical for all* $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$.

Proof. Both distributions consist of a random *Rand*, sequences Y and $\mathcal{H}_1, \ldots, \mathcal{H}_l$ (obtained for both distributions by applying V^* to *Rand* and $(\phi, \mathcal{I})$), a random sequence $D = d_1, \ldots, d_l$, and, in case $D \in \mathit{Good}(\mathit{Rand}, \phi, \mathcal{I})$, messages from the verifier (obtained for both distributions by applying V^* to *Rand*, D, and $(\phi, \mathcal{I})$), and a bit c. By Lemma 11, c is the only value for which Y can be $(\phi, \mathcal{I})$-co-opened as c. □

We have thus proved the following theorem.

THEOREM 29. *Let L be an RSR language. Then,* $\mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}}) \in PZK$.

Remark 4 (a variation on the protocol for monotone formulae over co-RSR languages). Next we show that it is possible to design a zero-knowledge proof system for any monotone formula over co-RSR languages, which, although being less efficient than the one already presented, does not use condition 4 in the definition of RSR languages.

The only variation from the protocol described in Figure 5 is in the subprotocol in which the verifier gives a "proof that he knows what c is." Here, the verifier considers the NP statement "there exist a string R and a bit c such that (Y, R) is a $(\phi, \mathcal{I})$-co-opening as c of Y." Using standard polynomial-time reductions (see [35]), this statement can be reduced in polynomial time to a statement about the Hamiltonicity of graph G. We can assume without loss of generality that the used reduction is witness-preserving (i.e., there exists an efficient algorithm that, on input a witness R for the former statement, allows us to compute a witness π for the latter), since so are many such reductions in the literature. Now, the verifier will compute a witness π for the latter statement and prove its knowledge using a parallel execution of a modification of the zero-knowledge proof system for graph Hamiltonicity given in [10]. The modification is the following. Recall that the protocol in [10] uses a bit commitment scheme based on one-way functions to encode an isomorphic copy of the input graph G. We propose instead to use the procedure *co-Construct-Opening* as a bit commitment scheme: to commit to a bit b run *co-Construct-Opening* on input $\phi, \mathcal{I}$ and b. The commitment scheme described has the following properties. If $(\phi, \mathcal{I}) \in \mathrm{CL}(\mathrm{MON}, \bar{\mathrm{L}})$, the verifier is information theoretically bound to the committed bit; this allows us to extract the bit c and thus to prove the zero-knowledge property. On the other hand, if $(\phi, \mathcal{I}) \notin \mathrm{CL}(\mathrm{MON}, \bar{\mathrm{L}})$, the committed bit is perfectly hidden to the prover, and thus soundness can be easily proved.

The described technique is based on ideas that first appeared in [45].

6. Threshold formulae over RSR languages. In this section we give a PZK proof system for the language $\mathrm{CL}(T; L)$ of all true threshold formulae whose atoms are statements about membership in an RSR language L. As MON is a subset of T, the results in this section subsume the one in section 4. However, we note that the proof system presented here is more complicated than that presented in section 4. On the other hand, the threshold function has polynomial-size monotone boolean formulae (see [55]), and thus we could use the proof system of section 4 for the monotone

formula of size $O(n^{5.3})$ for the threshold function on n arguments given by Valiant in [55]. We show, however, a much more efficient construction that is constructive. (Valiant's proof that the threshold function has a monotone formula does not provide an effective construction.)

6.1. Threshold schemes. In the construction of our proof system we will use the notion of *threshold scheme*, introduced by Shamir [52] and Blackley [9]. A (k, m)-threshold scheme is an efficient probabilistic algorithm that, on input a datum D, outputs m *pieces* $sh_1, \ldots, sh_m$, such that

- knowledge of any k or more pieces sh_i makes D easily computable;
- knowledge of any $k-1$ or fewer pieces sh_i leaves D completely undetermined (i.e., all its possible values are equally likely).

Shamir [52] shows how to construct such threshold schemes using interpolation of polynomials in the following way. Let $(\mathcal{E}, +, \cdot)$ be a finite field with more than m elements and let $D \in \mathcal{E}$ be the value to be shared; we will consider $\mathcal{E} = \mathrm{GF}(2^l)$ (which we identify with $\{0, 1, \ldots, 2^l - 1\}$). Choose at random $a_1, \ldots, a_{k-1} \in \mathcal{E}$, construct the polynomial $q(x) = D + a_1 \cdot x + \cdots + a_{k-1} \cdot x^{k-1}$, and output $sh_i = q(i) \in \mathcal{E}$ (all operations are performed in $\mathcal{E}$) for $i = 1, \ldots, m$.

We say that a sequence $(sh_1, \ldots, sh_m)$ is a (k, m)-*sequence of admissible shares* (we will simply call it a *sequence of admissible shares* when k and m are clear from the context) if there exists a polynomial $q(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$ with coefficients in $\mathcal{E}$, such that $sh_i = q(i)$ for $i = 1, \ldots, m$. We say that a sequence $(sh_1, \ldots, sh_m)$ is a (k, m)-*sequence of admissible shares* for D if there exists a polynomial $q(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$ with coefficients in $\mathcal{E}$, such that $a_0 = D$ and $sh_i = q(i)$ for $i = 1, \ldots, m$.

Now we recall some facts about threshold schemes that will be important for the construction of our proof system. Let $I \subseteq \{1, \ldots, m\}$ and suppose $|I| < k$. Given D and a sequence $(sh_i | i \in I)$ of values, it is always possible to efficiently generate values sh_i, $i \notin I$, such that $(sh_1, \ldots, sh_m)$ is a sequence of admissible shares for D and the sh_i's for $i \notin I$ are uniformly distributed among the sh_i's such that $(sh_1, \ldots, sh_m)$ is a sequence of admissible shares for D. If $|I| \geq k$, then a sequence $(sh_i | i \in I)$ of values uniquely determines at most one value D and values sh_i for $i \notin I$ such that $(sh_1, \ldots, sh_m)$ is a sequence of admissible shares for D. Also, for $|I| < k$, given a sequence $(sh_i | i \in I)$ of values, if the values sh_i for $i \notin I$ are chosen with uniform distribution among the sh_i's such that $(sh_1, \ldots, sh_m)$ is a sequence of admissible shares, then D is uniformly distributed in $\mathcal{E}$.

6.2. The construction. The construction of a proof system for $\mathrm{CL}(T; L)$ is based on an ad hoc modification of the concepts of value and opening and will be obtained as for the proof system for monotone formulae over RSR languages.

The *value* $t\text{-}Val(sh_1, \ldots, sh_m; \phi) \in \mathcal{E} \cup \{\perp\}$ of an m-tuple $(sh_1, \ldots, sh_m) \in \mathcal{E}^m$ with respect to a threshold formula ϕ with m variables is defined recursively in the following way. If ϕ consists of a single variable v_i, then $t\text{-}Val(sh_i; v_i) = sh_i$. Otherwise, assume ϕ is a threshold formula $T_{k,n}$, over subformulae $\phi_1, \ldots, \phi_n$ for some integers k, n; also, let $m_0 = 0$, let m_i be the number of variables in $\phi_1, \ldots, \phi_i$, for $i = 1, \ldots, n$, and let $s_i = t\text{-}Val(sh_{m_{i-1}+1}, \ldots, sh_{m_i}, \phi_i)$. Then it holds that $t\text{-}Val(sh_1, \ldots, sh_m; \phi)$ is equal to $D \in \mathcal{E}$ if $(s_1, \ldots, s_n)$ is a $(n - k + 1, n)$-sequence of admissible shares for D, or equal to $\perp$ otherwise. Now we can define the notion of threshold-opening.

DEFINITION 30 (threshold-opening). *Let ϕ be a threshold formula, let $D \in \mathrm{GF}(2^l)$, and let $\mathcal{I} = ((x_1, z_1), \ldots, (x_m, z_m))$ be an m-tuple of pairs.*

We say that

$$R = (((c_{11}, r_{11}, s_{11}, t_{11}), \ldots, (c_{1l}, r_{1l}, s_{1l}, t_{1l})), \ldots, ((c_{m1}, r_{m1}, s_{m1}, t_{m1}), \ldots, (c_{ml}, r_{ml}, s_{ml}, t_{ml})))$$

is a $(\phi, \mathcal{I})$*-threshold-opening as* D *of* $Y = ((y_{11}, \ldots, y_{1l}), \ldots, (y_{m1}, \ldots, y_{ml}))$ *if and only if the following hold:*

1. $r_{i,j} \in \mathcal{S}_{x_i}$, $c_{i,j} \in \{0,1\}$, *and* $(s_{i,j}, t_{i,j}) \in \mathcal{R}_{x_i}$ *for* $i = 1, \ldots, m$ *and* $j = 1, \ldots, l$;
2. $t\text{-}Val(sh_1, \ldots, sh_m, \phi) = D$, *where* $sh_i = c_{i1} \circ \cdots \circ c_{il}$;
3. *if* $c_{i,j} = 0$, *then* $y_{ij} = Sample_{\mathcal{R}}(x_i, z_i, r_{ij})$;
4. *if* $c_{i,j} = 1$, *then* $y_{ij} = Sample_{\mathcal{R}}(x_i, s_i, r_{ij})$.

Using the above properties of threshold schemes and the properties of RSR languages, one can see that if $(\phi, \mathcal{I}) \in \mathrm{CL}(T; L)$, then any m-tuple $\vec{Y}$ can be threshold-opened as a sequence of admissible shares of D for all $D \in \{0,1\}^l$. Moreover, if $(\phi, \mathcal{I}) \notin \mathrm{CL}(T; L)$, then any m-tuple can be threshold-opened as a sequence of admissible shares for at most one D.

Therefore, similarly as before, we have the following two lemmas describing the properties of the concept of threshold-opening.

LEMMA 31. *Let* $(\phi, \mathcal{I}) \in \mathrm{CL}(T; L)$, *let* $D, D' \in \{0,1\}^l$, *let* Y *be a tuple, and let* R *be a* $(\phi, \mathcal{I})$*-threshold-opening as* D *of the* m*-tuple* Y. *Then, there exists a tuple* R' *that is a* $(\phi, \mathcal{I})$*-threshold-opening as* D' *of* Y.

LEMMA 32. *Let* $(\phi, \mathcal{I}) \notin \mathrm{CL}(T; L)$ *and let* Y *be a tuple. Then, there exists at most one element* D *for which there exists a* $(\phi, \mathcal{I})$*-threshold-opening as* D *of* Y.

Similarly to procedure *Construct-Opening*, we can design a procedure *Threshold-Construct-Opening* that on input $(\phi, \mathcal{I}, a)$ with $a \in \{0,1\}$ outputs a sequence Y and a $(\phi, \mathcal{I})$-threshold-opening as a of the m-tuple Y. Moreover, if $(\phi, \mathcal{I}) \in \mathrm{CL}(T; L)$, the distribution induced on Y is independent of a.

A PZK proof system (P,V) for language $\mathrm{CL}(T; L)$ follows from the above two properties and can be found in Figure 7.

The Proof System (P,V).
Input:$(\phi, \mathcal{I}) \in \mathrm{CL}(T; L)$.

- P.1 Obtain (Y, R) by running *Threshold-Construct-Opening* on input $\phi, \mathcal{I}$ and 0. Send Y to V.
- V.1 Randomly select $b \in \{0,1\}$. Send b to P.
- P.2 Randomly select a $(\phi, \mathcal{I})$-threshold-opening R of Y as $D' = b$ according to the distribution induced by procedure *Threshold-Construct-Opening* on its second output conditioned on Y being the first element of the output pair. Send R to V.
- V.2 Verify that R is a $(\phi, \mathcal{I})$-threshold-opening of Y as b.

FIG. 7. *The proof system* (P,V) *for* $\mathrm{CL}(T; L)$.

The completeness of (P,V) follows from Lemma 31 and the soundness from Lemma 32. The simulator is designed analogously to the one in section 4.1, using procedure *Threshold-Construct-Opening*.

THEOREM 33. *Let* L *be an RSR language. The above protocol* (P,V) *is a PZK proof system for* $\mathrm{CL}(T; L)$.

6.3. Threshold formulae over co-RSR languages. Similarly to what we did for the monotone formulae, we can obtain a proof system for threshold formulae over co-RSR languages using DeMorgan's law and observing that a threshold formula over statements $x_i \notin L$ is equivalent to the negation of a threshold formula $x_i \in L$ with a (k, n)-threshold becoming an $(n - k + 1, n)$-threshold.

7. Nonmonotone formulae over RSR languages. In this section we prove that a class of nonmonotone formulae over RSR languages is in PZK. Specifically, we consider the language CL(OR;L), where L is an RSR language. Recall that such language is defined as the set of all tuples $((\phi, \mathcal{I}_\phi), (\psi, \mathcal{I}_\psi))$, where ϕ and ψ are monotone formulae, such that at least one of ϕ, ψ is true, and the atoms of ϕ (resp., ψ) are statements about membership in an RSR language L (resp., nonmembership in L). That is, $(\phi, \mathcal{I}_\phi) \in \mathrm{CL}(\mathrm{MON}; \mathrm{L})$ or $(\psi, \mathcal{I}_\psi) \in \mathrm{CL}(\mathrm{MON}; \bar{\mathrm{L}})$.

Before explaining the proof system in its generality, we briefly discuss an example relative to graph isomorphism.

A simple example. Let us consider the following proof system for proving that, given two pairs of graphs (A_0, A_1) and (B_0, B_1), A_0 is isomorphic to A_1 or B_0 is not isomorphic to B_1. The verifier picks a random bit b and encodes it by constructing and sending the prover graph B isomorphic to B_b. The prover then constructs and sends the verifier a graph A. Then the verifier reveals b by giving an isomorphism between B and B_b. Finally, the verifier accepts if the prover shows an isomorphism between A and A_b.

For the completeness, first consider the case in which A_0 and A_1 are isomorphic. Then A can be constructed isomorphic to A_0, and then, no matter what the value of b is, the prover can always exhibit an isomorphism between A and A_b. Suppose instead that A_0 and A_1 are not isomorphic and thus B_0 and B_1 are not isomorphic either. Then, B is isomorphic to exactly one of B_0 and B_1, and thus the prover can compute the value of b and construct A isomorphic to A_b.

For the soundness, observe that if A_0 and A_1 are not isomorphic, then A is isomorphic to exactly one of the two. Moreover, since B_0 and B_1 are isomorphic, the prover does not learn the value of b until after A has been given to the verifier. Therefore, the verifier accepts with probability at most 1/2.

Finally, it is also easy to argue that the proof system described is honest-verifier zero-knowledge. Indeed the simulator, after receiving graph B from the verifier, picks a random $\hat{b}$ and constructs A isomorphic to $A_{\hat{b}}$. Then if $b = \hat{b}$, the simulation is completed by giving the isomorphism between A_b and A. If instead $b \neq \hat{b}$, then the simulation is started again from scratch. Since b and $\hat{b}$ are uniformly distributed (and independent), the simulation will have to be repeated on average two times. To make it zero-knowledge, we need to augment it as we have done in section 5.

The general case. The construction of a proof system for CL(OR;L) is based on the concepts of value, co-value, opening, and co-opening.

Now we informally describe our proof system (P,V) for CL(OR;L). Let $((\phi, \mathcal{I}_\phi), (\psi, \mathcal{I}_\psi))$ be the input to (P,V). The first three rounds are precisely the first three rounds of protocol (P,V) in section 5 (i.e., steps labeled as V.0, V.1, P.1, V.2, P.2), when run on input the formula ψ and the m-tuple $\mathcal{I}_\psi$. That is, V sends a pair (Y_ψ, S) to P and proves that he knows a $(\psi, \mathcal{I}_\psi)$-co-opening of Y_ψ and that S has been correctly constructed. The fourth step of our proof system is the following: P checks that V knows a $(\psi, \mathcal{I}_\psi)$-co-opening of Y and that S has been correctly constructed. If this check is not satisfied, he stops. Otherwise, he computes whether the formula ψ is true. If it is, he computes the bit b such that Y_ψ can be $(\psi, \mathcal{I}_\psi)$-co-opened as b,

and then he sets $a = b$. If the formula ψ is false, he picks the value of a at random from $\{0, 1\}$. Then he sends an m-tuple Y_ϕ to V that can be $(\phi, \mathcal{I}_\phi)$-opened as a. Now, V reveals the bit b that he chose in his first step and the $(\psi, \mathcal{I}_\psi)$-co-opening of Y_ψ as b to P. Finally P computes and sends to V a $(\phi, \mathcal{I}_\phi)$-opening of Y_ϕ as bit b, and V verifies that he receives a $(\phi, \mathcal{I}_\phi)$-opening of Y_ϕ as b.

The following theorem holds.

THEOREM 34. *Let L be an RSR language; then the language* CL(OR;L) *is in PZK.*

Proof. We show that the above protocol (P,V) is a PZK proof system for CL(OR;L).

Completeness. Suppose that $((\phi, \mathcal{I}_\phi), (\psi, \mathcal{I}_\psi)) \in$ CL(OR;L).

In the first case $(\psi, \mathcal{I}_\psi) \in$ CL(MON; $\bar{\text{L}}$). Then, by the properties of a co-opening, the prover can always find exactly one bit b such that Y_ψ can be $(\psi, \mathcal{I}_\psi)$-co-opened as b and pick a equal to the bit b chosen by V. Also, as in section 5.1, one can show that V can always perform the "proof of knowledge" in polynomial time. Then we have that P can compute an m-tuple Y_ϕ such that Y_ϕ can be $(\phi, \mathcal{I}_\phi)$-opened as b, and he can later reveal an opening for Y_ϕ.

Assume now that $(\phi, \mathcal{I}_\phi) \in$ CL(MON; L). Then, by Lemma 10, the m-tuple Y_ϕ can be opened both as 0 and as 1, and P can compute a $(\phi, \mathcal{I}_\phi)$-opening as the bit b sent by V. In both cases V's verifications are satisfied with probability 1.

Soundness. Suppose that $((\phi, \mathcal{I}_\phi), (\psi, \mathcal{I}_\psi)) \notin$ CL(OR; L). In this case, the m-tuple Y_ψ sent by the verifier can be $(\psi, \mathcal{I}_\psi)$-co-opened both as 0 and as 1. Moreover, observe that, using an argument similar to the one used for proving the soundness of the proof system of section 5, it can be shown that no information about b is revealed by the verifier in his proof of knowledge. Also, by Lemma 11 any m-tuple Y_ϕ can be $(\phi, \mathcal{I}_\phi)$-opened as at most one bit a. Thus the probability that bit a is equal to the bit b chosen by the verifier is at most $1/2$. Thus the verifications of the verifier are satisfied with probability at most $1/2$.

Perfect zero knowledge. Now we only sketch a simulator S_{V^*} for (P,V) as the simulator is based on the same strategy as the simulator for the proof system of section 5. The verifier is run twice, and from the answers of the verifier it is possible to reconstruct the bit b. Finally, once b has been extracted, the simulator uses the procedure *Construct-Opening* to compute an m-tuple Y_ϕ that can be $(\phi, \mathcal{I}_\phi)$-opened as b along with an opening. □

8. Conclusions, recent works, and open problems. We have shown closure properties under composition of (polynomial-size) monotone formulae and some classes of nonmonotone formulae for the class PZK when restricted to RSR languages. In the context of PZK, even for this set of languages, showing closure with respect to any boolean formula, or, more generally, any boolean circuit, is still open.

Related works include closure results in [22, 23] for PZK proofs on quadratic residuosity in the noninteractive model, as well as some closure techniques in [20] for witness hiding and witness indistinguishable proofs over a class of languages containing RSR languages.

We notice that the techniques of this paper have played an important role in some recent results. Specifically, closure under some clustering-based type of composition has been considered in [25] based on the techniques herein. Restricted to the class SZK of statistical zero-knowledge proofs (see [42] for definitions), our techniques were also applied recently in [51] as probability amplification techniques for proving some closure properties over general SZK statements proved to an honest verifier (the techniques can, in turn, be applied to achieve monotone formula com-

position of the resulting general SZK proofs constructed there using results in [49]). Furthermore, our techniques were used to augment the set of known languages having other types of PZK protocols [30], such as PZK proofs of decision power or PZK and result-indistinguishable transfers of decision.

Acknowledgments. We would like to thank Erez Petrank for very useful discussions, and Mihir Bellare, Oded Goldreich, and Johan Håstad for their remarks, corrections, and valuable suggestions on early manuscripts. Finally, we thank the referees for their careful reading of the manuscript and for several suggestions that have significantly improved the presentation of our results.

REFERENCES

[1] M. Abadi, J. Feigenbaum, and J. Kilian, *On hiding information from an oracle,* J. Comput. System Sci., 39 (1989), pp. 21–50.

[2] W. Aiello and J. Håstad, *Statistical zero knowledge can be recognized in two rounds*, J. Comput. System Sci., 42 (1991), pp. 327–345.

[3] M. Bellare and O. Goldreich, *On defining proofs of knowledge*, in Advances in Cryptology—CRYPTO '92, Lecture Notes in Comput. Sci. 740, E. Brickell, ed., Springer-Verlag, Berlin, 1993, pp. 390–420.

[4] M. Bellare and S. Goldwasser, *A new paradigm for digital signatures and message authentication based on non-interactive zero-knowledge proofs*, in Advances in Cryptology—CRYPTO '89, Lecture Notes in Comput. Sci. 425, G. Brassard, ed., Springer-Verlag, New York, 1989, pp. 194–211.

[5] M. Bellare, S. Micali, and R. Ostrovsky, *Perfect zero knowledge in constant rounds,* in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 482–493.

[6] M. Bellare, S. Micali, and R. Ostrovsky, *The (true) complexity of statistical zero knowledge*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 494–502.

[7] M. Bellare and E. Petrank, *Making zero-knowledge provers efficient,* in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 711–722.

[8] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, S. Micali, and P. Rogaway, *Everything provable is provable in zero-knowledge*, in Advances in Cryptology—CRYPTO '88, Lecture Notes in Comput. Sci. 403, Springer-Verlag, Berlin, 1988, pp. 37–56.

[9] G. R. Blackley, *Safeguarding cryptographic keys*, in Proceedings of the 1979 National Computer Conference, American Federation of Information Processing Societies, 1979, pp. 313–317.

[10] M. Blum, *How to prove a theorem so no one else can claim it*, in Proceedings of the International Congress of Mathematicians, Berkeley, CA, 1986, pp. 1444–1451.

[11] M. Blum, *Program result checking: A new approach to making programs more reliable*, in Proceedings of the 20th Annual International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 700, Springer-Verlag, London, UK, 1993, pp. 1–14.

[12] M. Blum, A. De Santis, S. Micali, and G. Persiano, *Noninteractive zero-knowledge*, SIAM J. Comput., 20 (1991), pp. 1084–1118.

[13] M. Blum, P. Feldman, and S. Micali, *Non-interactive zero-knowledge and applications*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 103–112.

[14] M. Blum and S. Kannan, *Designing programs that check their work,* J. ACM, 42 (1995), pp. 269–291.

[15] R. Boppana, J. Håstad, and S. Zachos, *Does co-NP have short interactive proofs?*, Inform. Process. Lett., 25 (1987), pp. 127–132.

[16] J. Boyar, K. Friedl, and C. Lund, *Practical zero-knowledge proofs: Giving hints and using deficiencies*, J. Cryptology, 4 (1991), pp. 185–206.

[17] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp. 156–189.

[18] G. Brassard, C. Crépeau, and M. Yung, *Perfect zero-knowledge computationally convincing proofs for NP in constant rounds*, Theoret. Comput. Sci., 84 (1991), pp. 23–52.

[19] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, *Private information retrieval*, J. ACM, 45 (1998), pp. 965–981.

[20] R. Cramer, I. Damgard, and B. Schoenmakers, *Proofs of partial knowledge and simplified design of witness hiding protocols*, CWI Quarterly, 8 (1995), pp. 111–127.

[21] I. Damgard, *Interactive hashing can simplify zero-knowledge protocol design without computational assumptions*, in Advances in Cryptology—CRYPTO '93, Lecture Notes in Comput. Sci. 773, D. Stinson, ed., Springer-Verlag, Berlin, 1994, pp. 100–109.

[22] A. De Santis, G. Di Crescenzo, and G. Persiano, *The knowledge complexity of quadratic residuosity languages*, Theoret. Comput. Sci., 132 (1994), pp. 291–317.

[23] A. De Santis, G. Di Crescenzo, and G. Persiano, *On NC^1 Boolean circuit composition of non-interactive perfect zero-knowledge*, in Proceedings of the 29th Annual International Symposium on Mathematical Foundations of Computer Science 2004 (MFCS 2004), Lecture Notes in Comput. Sci. 3153, Springer-Verlag, Berlin, 2004, pp. 356–367.

[24] A. De Santis, G. Di Crescenzo, and G. Persiano, *Communication-efficient anonymous group identification*, in Proceedings of the Fifth Annual ACM Conference on Computer and Communication Security (CCS 98), 1998, pp. 73–82.

[25] A. De Santis, G. Di Crescenzo, O. Goldreich, and G. Persiano, *The graph-clustering problem has a perfect zero-knowledge proof system*, Inform. Process. Lett., 69 (1999), pp. 201–206.

[26] A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung, *On monotone formula closure of SZK*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS 94), Santa Fe, NM, 1994, pp. 453–464.

[27] A. De Santis, S. Micali, and G. Persiano, *Non-interactive zero-knowledge proof systems*, in Advances in Cryptology—CRYPTO '87, Lecture Notes in Comput. Sci. 293, C. Pomerance, ed., Springer-Verlag, Berlin, 1998, pp. 52–72.

[28] A. De Santis and G. Persiano, *Zero-knowledge proofs of knowledge without interaction,* in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 427–437.

[29] A. De Santis and M. Yung, *Cryptographic applications of the non-interactive metaproof and many-prover systems,* in Advances in Cryptology—CRYPTO '90, Lecture Notes in Comput. Sci. 537, A. Menezes and S. Vanstone, eds., Springer-Verlag, Berlin, 1991, pp. 366–377.

[30] G. Di Crescenzo, K. Sakurai, and M. Yung, *On zero-knowledge proofs: From membership to decision*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2000), Portland, OR, 1999, pp. 255–264.

[31] U. Feige, A. Fiat, and A. Shamir, *Zero-knowledge proofs of identity*, J. Cryptology, 1 (1988), pp. 77–94.

[32] M. Fischer, S. Micali, and C. Rackoff, *A secure protocol for the oblivious transfer,* J. Cryptology, 9 (1996), pp. 191–195.

[33] L. Fortnow, *The complexity of perfect zero-knowledge*, in Randomness and Computation, Advances in Computing Research 5, JAI Press, Greenwich, CT, 1989, pp. 327–343.

[34] Z. Galil, S. Haber, and M. Yung, *Minimum-knowledge interactive proofs for decision problems*, SIAM J. Comput., 18 (1989), pp. 711–739.

[35] M. Garey and D. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1983.

[36] O. Goldreich and H. Krawczyk, *On the composition of zero-knowledge proof systems*, SIAM J. Comput., 25 (1996), pp. 169–192.

[37] O. Goldreich and E. Kushilevitz, *A perfect zero knowledge proof for a problem equivalent to the discrete logarithm,* J. Cryptology, 6 (1993), pp. 97–116.

[38] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.

[39] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems*, J. Cryptology, 7 (1994), pp. 1–32.

[40] O. Goldreich, R. Ostrovsky, and E. Petrank, *Computational complexity and knowledge complexity*, SIAM J. Comput., 27 (1998), pp. 1116–1141.

[41] O. Goldreich and E. Petrank, *Quantifying knowledge complexity*, Comput. Complexity, 8 (1999), pp. 50–98.

[42] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[43] R. Impagliazzo and M. Yung, *Direct minimum knowledge computations*, in Advances in Cryptology—CRYPTO '87, Lecture Notes in Comput. Sci. 293, C. Pomerance, ed., Springer-Verlag, London, 1987, pp. 40–51.

[44] M. Ito, A. Saito, and T. Nishizeki, *Multiple assignment scheme for sharing secret*, J. Cryptology, 6 (1993), pp. 15–20.
[45] T. Itoh, Y. Ohta, and H. Shizuya, *A language-dependent cryptographic primitive*, J. Cryptology, 10 (1997), pp. 37–49.
[46] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.
[47] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, *Perfect zero-knowledge arguments for NP using any one-way permutation*, J. Cryptology, 11 (1998), pp. 87–108.
[48] M. Naor and M. Yung, *Public-key cryptosystems provably secure against chosen ciphertext attacks*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 427–437.
[49] T. Okamoto, *On relationships between statistical zero-knowledge proofs*, J. Comput. System Sci., 60 (2000), pp. 47–108.
[50] E. Petrank and G. Tardos, *On the knowledge complexity of NP*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 494–503.
[51] A. Sahai and S. Vadhan, *A complete promise problem for statistical zero-knowledge*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, p. 448.
[52] A. Shamir, *How to share a secret*, Comm. ACM, 22 (1979), pp. 612–613.
[53] A. Shamir, *IP = PSPACE*, J. ACM, 39 (1992), pp. 869–877.
[54] M. Tompa and H. Woll, *Random self-reducibility and zero-knowledge interactive proofs of possession of information*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 472–482.
[55] L. Valiant, *Short monotone formulae for the majority function*, J. Algorithms, 5 (1984), pp. 363–366.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1330–1346

NETWORK FAILURE DETECTION AND GRAPH CONNECTIVITY*

JON KLEINBERG†, MARK SANDLER‡, AND ALEKSANDRS SLIVKINS§

Abstract. We consider a model for monitoring the connectivity of a network subject to node or edge failures. In particular, we are concerned with detecting (ϵ, k)-*failures*: events in which an adversary deletes up to k network elements (nodes or edges), after which there are two sets of nodes A and B, each at least an ϵ fraction of the network, that are disconnected from one another. We say that a set D of nodes is an (ϵ, k)-detection set if, for any (ϵ, k)-failure of the network, some two nodes in D are no longer able to communicate; in this way, D "witnesses" any such failure. Recent results show that for any graph G, there is an (ϵ, k)-detection set of size bounded by a polynomial in k and ϵ, independent of the size of G. In this paper, we expose some relationships between bounds on detection sets and the edge-connectivity λ and node-connectivity κ of the underlying graph. Specifically, we show that detection set bounds can be made considerably stronger when parameterized by these connectivity values. We show that for an adversary that can delete $k\lambda$ edges, there is always a detection set of size $O(\frac{k}{\epsilon}\log\frac{1}{\epsilon})$ which can be found by random sampling. Moreover, an (ϵ, λ)-detection set of minimum size (which is at most $\frac{1}{\epsilon}$) can be computed in polynomial time. A crucial point is that these bounds are independent not just of the size of G but also of the value of λ. Extending these bounds to node failures is much more challenging. The most technically difficult result of this paper is that a random sample of $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ nodes is a detection set for adversaries that can delete a number of nodes up to κ, the node-connectivity. For the case of edge-failures we use VC-dimension techniques and the cactus representation of all minimum edge-cuts of a graph; for node failures, we develop a novel approach for working with the much more complex set of all minimum node-cuts of a graph.

Key words. network failures, detection sets, connectivity, minimal cuts, cactus representation, VC-dimension

AMS subject classifications. Primary, 68Q25; Secondary, 68R10, 05C40

DOI. 10.1137/070697793

1. Introduction.

Monitoring network connectivity. As links or nodes fail in a network, it is important to maintain information about basic properties such as connectivity. For large, unstructured networks, this is often done by recourse to sampling and other approximate measurements; performing such measurements in a robust and accurate way is an active research topic (e.g., [4, 5, 19, 21, 20]). A general problem here is to minimize the cost of network monitoring and measurement, in terms of communication, computation, and resource usage.

Here we consider a model proposed by the first author for monitoring network connectivity [16]. We are given a connected node graph G on n nodes, and we want to detect "failure events" in which at most k network elements (nodes or edges) are deleted, after which there are two sets of nodes A and B, each of size $\geq \epsilon n$, such that no node in A has a path to any node in B. We will call such a pair of sets *separated*,

*Received by the editors July 20, 2007; accepted for publication April 2, 2008; published electronically August 6, 2008. A preliminary version of this paper appeared in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004. This work was completed while the second and third authors were graduate students at Cornell University. This work was supported in part by a David and Lucile Packard Foundation Fellowship and NSF ITR/IM grant IIS-0081334 of Jon Kleinberg.

http://www.siam.org/journals/sicomp/38-4/69779.html

†Department of Computer Science, Cornell University, Ithaca, NY 14853 (kleinber@cs.cornell.edu).

‡Google Inc., New York, NY 10011 (sandler@google.com).

§Microsoft Research, Mountain View, CA 94043 (slivkins@microsoft.com).

and we will call such an event an (ϵ, k)-*failure*. (To reflect the fact that the k node or edge failures can be arbitrary, we will sometimes speak of them as being selected by an adversary.)

To detect such failures, we consider the strategy of placing "detectors" at a subset D of the nodes of G. Now, if we find that two detectors are unable to communicate—either because there is no path between them, or because one has been deleted—we can record a fault in the network. We would like our set D to have the property that *whenever* an (ϵ, k)-failure occurs, some two detectors are unable to communicate; we will refer to such a set D as an (ϵ, k)-*detection set.* Note the nature of this condition: D must detect all possible (ϵ, k)-failures, so we imagine D as being chosen *before* the adversary selects a set of k network elements to delete. The emphasis in [16] was on finding a bound on the number of nodes that must be randomly selected from a graph G in order to obtain an (ϵ, k)-detection set with high probability. Improvements to these bounds were obtained by [7].[1]

In this paper, we adopt a somewhat different approach to this issue, by exposing some interesting and nontrivial connections between the size of the smallest detection set for a graph G and the values of its edge- and node-connectivity. (The edge-connectivity of G, denoted $\lambda(G)$, is the smallest number of edges that must be deleted in order to disconnect G. The node-connectivity of G, denoted $\kappa(G)$, is the analogous quantity for node deletions.) We show that stronger bounds on detection set size can be obtained if we parameterize these bounds by the connectivity values λ and κ, and, for some cases, we use this relationship with connectivity to provide the first *per-instance* guarantees for detection sets.

Because our results are different depending on whether the adversary is deleting edges or nodes, we consider these two cases separately.

Detection sets for edge failures. We begin with adversaries that can delete up to k edges; as such, we will be concerned with (ϵ, k)-*edge-failures*, which are (ϵ, k)-failures in which only edges are deleted. It is known that a random set of $O(\frac{k}{\epsilon} \log \frac{1}{\epsilon})$ nodes is an (ϵ, k)-detection set for edge-failures with high probability [16], and that every graph contains an (ϵ, k)-detection set for edge-failures of size $O(\frac{k}{\epsilon})$ [7]; note that both bounds are independent of the size of the graph G. It is not difficult to show that both bounds are tight, and so there is no prospect of obtaining an improvement that applies to all graphs. However, it makes sense to ask whether better bounds are possible in terms of natural parameters of the graph G.

An obvious parameter to consider here is the edge-connectivity λ; indeed, there can be no (ϵ, k)-edge-failures in G if $k < \lambda$. Our first main result establishes that λ is indeed a natural way to parameterize the problem; we show that every graph G has an (ϵ, λ)-detection set for edge-failures of size at most $\frac{1}{\epsilon}$. Note that there is no leading constant in this bound and that it is independent not just of the size of G but also of the value of λ. Extending this result, we show further that an (ϵ, λ)-detection set for edge-failures of *minimum* size for a graph G can be computed in polynomial time. The algorithms used to establish these results are based on the cactus representation of all minimum edge-cuts of G [6, 9].

Given that strong bounds are possible for detecting an adversary that can delete one minimum cut's worth of edges, it is natural to ask what can be said about an adversary capable of deleting a number of edges equal to k times the size of a minimum cut. We show that a random set of $O(\frac{k}{\epsilon} \log \frac{1}{\epsilon})$ nodes is a $(k\lambda, \epsilon)$-detection set for edge-

[1] Following the publication of the conference version of this paper, further improvements have been obtained in [8].

failures with high probability. This is essentially a factor of λ times stronger than the bounds of [7, 16], which did not take edge-connectivity into account. Our proof of this result uses a VC-dimension argument in the style of [16]; the bound on the VC-dimension is obtained using a result of Mader [18, 10] that extends results of Lovász [17] and of Cherkasskij [3] (also see [10]) on edge-disjoint paths in graphs.

Detection sets for node failures. We now consider adversaries that can delete up to k nodes. By analogy with our results for edge-failures, we consider the size of detection sets in terms of the node-connectivity κ. Our main result here is that every graph G (with $\kappa = O(\epsilon^2 n)$) has an (ϵ, κ)-detection set for node-failures of size $O(\frac{1}{\epsilon})$; moreover, a random set of $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ nodes forms an (ϵ, κ)-detection set for node-failures with high probability. Again, note that these bounds are independent not just of the size of G but also of the value of κ. Extending our results to adversaries that delete $k\kappa$ nodes for $k > 1$ is a very interesting and apparently difficult open question.

We note the distinction, raised by Gupta [13] (see section 1.4.2 of [7]), between *strong* and *weak* detection sets for node-failures. A strong detection set D has the property that, after any (ϵ, k)-node-failure, two nodes of D lie in different connected components. A weak detection set D' has the property that, after any (ϵ, k)-node-failure, two nodes of D lie in different connected components *or* an element of D has been deleted. Either of these definitions arguably forms a plausible definition of network failure detection. Improving a bound of [16], Fakcharoenphol showed that a random set of $O(\frac{k}{\epsilon} \log k \log \frac{1}{\epsilon})$ nodes is a strong (ϵ, k)-detection set for node-failures [7], and Gupta showed that every graph has a weak (ϵ, k)-detection set for node-failures of size $O(\frac{k}{\epsilon})$. As we note in section 4, weak detection appears to be a more useful notion when the problem is parameterized by node-connectivity; in particular, our main result is about weak detection sets. Henceforth, we will assume that all detection sets for node-failures are weak unless otherwise specified.

Our analysis for node-failures is significantly more complicated than for edge-failures, and this is not surprising; not only is no analogue of the cactus representation known for min-node-cuts, but this appears to be intrinsic due to the $\#P$-completeness of even counting the number of min-node-cuts [2]. Indeed, given the lack of tractable representations for min-node-cuts, we believe that our analysis develops some useful properties of their structure. We begin by constructing a detection set of minimum size for adversaries that can delete *shredders* [2, 15]—min-node-cuts whose deletion produces at least three components. The construction of the detection set then proceeds by greedily isolating a maximal collection of relatively balanced min-node-cuts that produce just two components, and whose "small sides" are disjoint; the small side of each such cut is required to have at least $\frac{\epsilon n}{10}$ nodes. We then show that by placing detectors so that one lies on the small side of each of these cuts, there is no way for a min-node-cut producing two components of size at least ϵn each to avoid being detected.

Further discussion. A simple calculation based on Karger's algorithm gives an upper bound of $O(\frac{k}{\epsilon} \log n)$ on a random sample of nodes that forms an $(\epsilon, k\lambda)$-detection set for edge-failures.[2] However, our goal in this paper is to find bounds that do not depend on the size of the graph.

Following [16], we can extend our results to a model in which the nodes of the network G are partitioned into two sets—a set V_0 of *end nodes* and a set V_1 of *internal*

[2]Note that no such simple bound is available for the case of node-failures, which is yet another evidence of its difficulty.

nodes. We assume that we are allowed only to place detectors at end nodes, and correspondingly are interested only in monitoring the connectivity of the end nodes. Specifically, we redefine (ϵ, k)-failures as failures of $\leq k$ network elements, after which two disjoint subsets of V_0, each of size $\geq \epsilon|V_0|$, are separated from each other. We can show that the bounds obtained above carry over to this more general setting; we omit further discussion of the generalization from this version of the paper.

Our work is similar in spirit to some of the work on vertex connectivity and augmentation thereof, e.g., [14, 2, 15, 12]. The actual technical issues are quite different, however, since we are interested only in balanced cuts. In general one could view our work here as integrating notions from edge- and node-connectivity with the problem of balanced separators of graphs—two topics that have traditionally been approached separately due to their great differences in tractability.

Notation. In this paper all graphs are assumed undirected; our standard notation for a graph is $G = (V, E)$. An edge(node)-cut is a set X of edges (nodes) such that $G \setminus X$ is disconnected.

A min-edge(node)-cut is an edge(node)-cut of minimum size. This size is also known as edge(node)-*connectivity* and is denoted by λ and κ, respectively. We will write *min-cut* when it is clear whether we are talking about edge-cuts or node-cuts. A set of nodes is *tight* if it is a union of some (but not all) components of a min-cut. A cut X is called ϵ-*balanced* if there are two sets of vertices of size $\geq \epsilon n$ that are disconnected from one another in $G \setminus X$. An ϵ-balanced cut of $\leq k$ edges(nodes) is called an (ϵ, k)-cut.

If sets X, Y have a nonempty intersection, we say X *meets* Y. To help clarify the notation in places, we will sometimes write $X + Y$ to denote the union of disjoint sets X and Y, and $X - Y$ to denote the difference of sets X and Y for which $Y \subseteq X$.

2. Detection sets for edge failures. In this section we present our results on edge-failures. For edge-failures that correspond to min-edge-cuts, our algorithms are based on the *cactus representation* of all min-cuts in a graph [6, 9]. We include a self-contained review of the relevant definitions and facts. Our result for the general edge-failure proof uses a VC-dimension argument in the style of [16]; the bound on the VC-dimension is obtained using a result of Mader [18, 10] on edge-disjoint paths in graphs.

Throughout the section, all cuts are edge-cuts, and all detection sets are for edge-failures. Let D be a set of nodes, representing the locations of our detectors. We say that D *detects* a cut X if some pair of detectors is separated in $G \setminus X$. We call D an (ϵ, k)-*detection set* if it detects every (ϵ, k)-edge-cut.

In section 2.1 we review the cactus representation. Section 2.2 is on min-edge-cuts: we construct a smallest (ϵ, λ)-detection set and prove that it has size $\leq \frac{1}{\epsilon}$. Section 2.3 is on general edge-failures: we prove that a set of $O(\frac{k}{\lambda\epsilon} \log \frac{1}{\epsilon})$ randomly sampled nodes is an (ϵ, k)-detection set with high probability.

2.1. Review: Cactus representation. Edges will be viewed as cycles of length 2; cycles of length 3 or more are called *proper*. A *cactus* is a connected graph such that any two of its cycles have at most one vertex in common. An arbitrary cactus can be obtained starting from a cycle and recursively adding new cycles that share a single vertex with the existing graph. In a cactus, some edges are contained in a proper cycle (*cycle edges*), and some are not (*path edges*). Each cycle edge has capacity $\frac{1}{2}$, and each path edge has capacity 1. It follows that min-cuts of a cactus have capacity 1. We can characterize them as follows.

FACT 2.1. *Consider a cactus T. Then* (a) *each path edge is a min-cut,* (b) *any pair of cycle edges from the same cycle is a min-cut, and* (c) *there are no other min-cuts.*

Proof. Clearly any cut in T has capacity at least 1. For part (a), let uv be a path edge. If there exists a uv-path p not containing the edge uv, then $p + uv$ is a cycle, contradicting the definition of a path edge. Therefore u and v are separated in $T - uv$. So uv is a cut in T, hence a min-cut.

For part (b), let e_1, e_2 be cycle edges from the same cycle C. $e_1 + e_2$ splits C into two arcs; call them C_1 and C_2. Suppose C_1 and C_2 are connected in $T - e_1 - e_2$. Then there exist vertices $u \in C_1$, $v \in C_2$ such that there is a uv-path p that does not intersect with C except for the endpoints. Let C' be the uv-arc of C that contains e_1. Then $p + C'$ is a cycle in T that shares ≥ 2 vertices with C, which is a contradiction. So C_1 and C_2 are not connected in $T - e_1 - e_2$. Therefore $e_1 + e_2$ is a cut in T, hence a min-cut.

For part (c), suppose X is a min-cut of T that is neither a path edge nor a pair of cycle edges from the same cycle. Since the capacity of X is ≤ 1, it consists of one or two cycle edges. Thus there is a (proper) cycle C such that X contains exactly one edge $uv \in C$. Since X is a min-cut, it must separate u and v. However, they are connected by $C - uv$, which is a contradiction. □

In a cactus, nodes of degree one will be called *leaves*, nodes of degree two that are contained in a cycle will be called *cycle nodes*, and all other nodes will be called *branch nodes*.

FACT 2.2. *Let v be a branch node of a cactus T. Then the cycles that contain v are pairwise disconnected in $T - v$.*

Proof. Let C, C' be cycles that contain v. Let uv, $u'v$ be edges in C, C', respectively, Suppose u and u' are connected in $T - v$; i.e., there is a uu'-path p not containing v. Then $p + uv + vu'$ is a cycle that shares ≥ 2 vertices with C (and C'), a contradiction. Thus C and C' are disconnected in $T - v$. □

Consider a branch node v of a cactus T. It connects two or more cycles. By Fact 2.2, the removal of v splits T into two or more connected components (v-*components*). Each v-component X is tight: for some cycle C containing v, it is obtained by removing any edge of C that is adjacent to v.

FACT 2.3. *Suppose S is a tight set in cactus T, and v is a branch node. Then*

(a) *if $v \in S$, then S contains at least one v-component;*
(c) *if $v \notin S$, then S is contained in a v-component;*
(c) *for any v-component X of T, either $X \subset S$, or $S \subset X$, or $X \subset V - S$, or $V - S \subset X$.*

Proof. For part (a), let S be a component of a min-cut C. By Fact 2.1 C is contained in a cycle, so $C \subset T[X + v]$ for some v-component X. Therefore if Y is any other v-component, then $Y + v$ is connected in $T \setminus C$. $Y \subset S$ follows since $v \in S$ and S is connected in $T \setminus C$.

For part (a), suppose S meets two v-components. Then they are connected in $T - v$ (via S), a contradiction.

For part (c), suppose X meets both S and $V - S$. Then by part (b) if $v \in S$, then $V - S \subset X$; else we have $S \subset X$. □

Let G be a weighted graph on n vertices. A *cactus-pair* of G is a pair (T, π) where T is a cactus and π is a mapping from $V(G)$ to $V(T)$ such that if M is a tight set in T, then $\pi^{-1}(M)$ is a tight set in G. For each tight set M of T, say that (T, π) *represents* the min-cut C of G such that $\pi^{-1}(M)$ is a C-component. A *cactus representation*

of G is a cactus-pair of G that represents all min-cuts of G. Dinits, Karzanov, and Lomonosov [6] proved that every capacitated graph has a cactus representation of size $O(n)$. Further results show that a cactus representation of size $O(n)$ can be efficiently constructed. See the introduction of [9] for a discussion.

2.2. Detection sets for min-edge-cuts. Here we are interested only in ϵ-balanced min-cuts, and so the cactus representation is too general for our purposes. This motivates the following definitions.

DEFINITION 2.4. *Let an* ϵ-cactus-pair *be a cactus-pair that represents all* ϵ-*balanced min-cuts. Let an* ϵ-cactus *be the cactus in such a cactus-pair (if the mapping is clear). A subset* S *of vertices of a cactus is* heavy *if* $|\pi^{-1}(S)| \geq \epsilon n$. *Call a cactus-pair* reduced *if every* v*-component is heavy.*

A reduced ϵ-cactus-pair can be efficiently computed from a standard cactus representation by consecutively applying the following reduction.

LEMMA 2.5. *Suppose* T *is an* ϵ*-cactus,* v *is a branch node, and* X *is a* v*-component that is not heavy. Let* T' *be* T *with* X *contracted into* v. *Then* T' *is also an* ϵ*-cactus.*

Proof. For each ϵ-balanced min-cut C of G there is a min-cut C' of T that represents it. By Fact 2.3(c) there is a component S of C' such that $X \subset S$ or $S \subset X$. Since S is heavy and X is not, it must be the case that X is a proper subset of S. Then $v \in S$, so C' is a min-cut in T', too. Therefore T' represents C. □

Let G be a capacitated graph. Let (T, π) be a reduced ϵ-cactus-pair of G. We will characterize (ϵ, λ)-detection sets of minimum size in terms of T.

Let a *subcycle* be a set of consecutive cycle nodes of a (proper) cycle in T. Consider the nondegenerate case when there is at least one branch node. Then the weight $|\pi^{-1}(\cdot)|$ of each leaf and each subcycle is at most $(1-\epsilon)n$. Let a *canonical subcactus* be a set of nodes of T that contains each leaf, has an element in every heavy subcycle, and contains no branch nodes. Let $D \subset V(G)$ be a set of detectors (not necessarily an (ϵ, λ)-detection set). Say D is T*-canonical* if $\pi(D)$ is a canonical subcactus and at most one detector is mapped to each node of T. The following two lemmas show that any smallest (ϵ, λ)-detection set is in fact a smallest T-canonical set.

Call $S \subset V$ *heavy* if $|S| \geq \epsilon n$, and *balanced* if both S and $V \setminus S$ are heavy. Call $S' \subset V(T)$ *balanced* if $\pi^{-1}(S')$ is balanced. For each balanced tight set S of G let $\pi'(S)$ be a (balanced) tight set S' of T such that $S = \pi^{-1}(S')$.

LEMMA 2.6. *Any smallest* (ϵ, λ)*-detection set is* T*-canonical.*

Proof. Let D be a smallest (ϵ, λ)-detection set. Call elements of D *detectors*. We need to show that (1) at most one detector is mapped to each node of T, (2) there is a detector mapped to each leaf and each heavy subcycle of T, and (3) no detectors are mapped to branch nodes of T. (See Figure 1.) Let us prove these three statements in order.

(1) Suppose two detectors d_1, d_2 map to a node v of T. To obtain a contradiction it suffices to show an (ϵ, λ)-detection set smaller than D. We claim that $D - d_1$ is also an (ϵ, λ)-detection set. Suppose not. Then there is a balanced tight set S of G that contains $D - d_1$. Obviously $d_1 \notin S$. Let $S' = \pi'(S)$. Since $d_2 \in S$, $v = \pi(d_2) \in S'$, and thus $d_1 \in S$, too, a contradiction.

(2) There is a detector mapped to each heavy tight set of T, in particular, to each leaf and each heavy subcycle.

(3) Suppose a detector d is mapped to a branch node v of T. By analogy with (1), we claim that $D - d$ is also an (ϵ, λ)-detection set. For suppose not. Then $D - d$ is disjoint with some balanced tight set S. Let $S' = \pi'(S)$. Since D is an (ϵ, λ)-detection

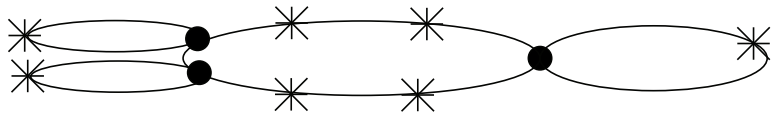


FIG. 1. *An ϵ-cactus with detectors. Branch nodes are denoted by "•", detectors by "*". In the central cycle, there are three subcycles between the branch nodes. The smallest of them is not heavy, and hence does not contain a detector. The other two are big enough so that they need two detectors each. Each of the three smaller cycles is heavy (even without its branch node), since otherwise it would have been contracted.*

set, $d \in S$, and thus $v \in S'$. Therefore by Fact 2.3(a) S' contains some v-component S''. Since T is reduced, S'' is heavy, so there is a detector mapped to it. Thus S contains a detector other than d, a contradiction. □

LEMMA 2.7. *Any T-canonical set is an (ϵ, λ)-detection set.*

Proof. Suppose $D \subset V$ and $\pi(D)$ meets each leaf and each heavy subcycle of T. We need to prove that $\pi(D)$ meets each heavy tight set of T. To show this we claim that any heavy tight set S of T contains a leaf or a heavy subcycle.

We will use induction on the size of S. The base case corresponds to an S that consists of one vertex, say v. By Fact 2.3(a) v cannot be a branch node. So either v is a leaf or it is a heavy subcycle consisting of a single cycle node.

For the induction step, note that if S contains a branch node v, then by Fact 2.3(a) S contains some (heavy) v-components S', to which the induction hypothesis applies. If S does not contain any branch nodes, then it lies within a single cycle, so S is a (heavy) subcycle. The claim follows. □

THEOREM 2.8. *A smallest (ϵ, λ)-detection set is of size at most $\frac{1}{\epsilon}$. There is a polynomial-time algorithm to construct it.*

Proof. Let (T, π) be a reduced ϵ-cactus-pair of G. We have seen that smallest (ϵ, λ)-detection sets are (mapped to) smallest canonical subcacti of T. Therefore it suffices to compute a smallest canonical subcactus of T.

Let S be a subset of a proper cycle C in T. Call S a C-detection set if S does not contain any branch nodes and every heavy subcycle of C contains an element of S. By definition, if there are no heavy subcycles in C, then an empty set is a C-detection set. Obviously, a subset of T is a canonical subcactus if and only if it is a union of leaves of T and (disjoint) C-detection sets, one for each proper cycle of T. Therefore to compute a smallest canonical subcactus of T it suffices to construct a smallest C-detection set for each proper cycle C of T.

The construction is as follows. Assuming T consists of more than one cycle, C contains one or more branch nodes. Assuming C contains cycle nodes, pick any branch node v_b followed by a cycle node v. Start with v. In the iterative step, start with a cycle node and move clockwise along C till a heavy subcycle is detected (call this subcycle *selected*) or a branch node is reached. Start a new step with the next cycle node. Stop when v_b is reached. Let S be the set of the last nodes (clockwise) of selected subcycles.

Obviously S is a C-detection set. S is a smallest such set by the following observation. Let S' be a C-detection set. Let $v \in C$ be a branch node or an element of S'. Let v' be the next node clockwise. Let C' be the smallest heavy subcycle starting with v' if it exists. Let w be the last node of C'. Then C' contains at least one element of S. The observation is that $S' - C' + w$ is a C-detection set with the same or smaller number of elements. Consecutively applying this observation, we can transform S' to

S without increasing the number of detectors.

Our construction puts one detector into each leaf of T and each selected sub cycle. Since leaves of T are heavy and selected subcycles are heavy and disjoint, our construction covers at least ϵn weight with each detector. Since the total weight of (nodes of) T is n, the total number of detectors is at most $\frac{1}{\epsilon}$. □

2.3. Smaller detection sets for edge-failures. A set S of nodes is *k-edge-separable* if there exists a set Z of $\leq k$ edges such that S is a union of components of $G \setminus Z$. Let $\mathcal{F}$ be the family of all k-edge-separable sets. We say that $A \subseteq V$ is *shattered* by $\mathcal{F}$ if for all $B \subseteq A$ there exists an $F \in \mathcal{F}$ such that $B = A \cap F$. The *VC-dimension* of $\mathcal{F}$ is defined to be the maximum cardinality of a subset of V that is shattered by $\mathcal{F}$.

In [16], it was shown that one can connect the VC-dimension d of $\mathcal{F}$ with (ϵ, k)-detection sets via the notion of an *ϵ-net*, which is a set that meets each $F \in \mathcal{F}$ of size $\geq \epsilon n$. Specifically, a theorem by [1] says that a set of $O(\frac{d}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta})$ randomly sampled nodes is an ϵ-net for $\mathcal{F}$ with probability at least $1 - \delta$.[3] Moreover, it is easy to show [16] that an ϵ-net for $\mathcal{F}$ is an (ϵ, k)-detection set.

In [16], it was shown that the VC-dimension of $\mathcal{F}$ is at most $2k + 1$, yielding a bound of $O(\frac{k}{\epsilon} \log \frac{1}{\epsilon})$ on the size of an (ϵ, k)-detection set. In this section, we strengthen the VC-dimension bound on $\mathcal{F}$ to $O(\frac{k}{\lambda})$. Therefore, we obtain the following theorem.

THEOREM 2.9. *A set of $O(\frac{k}{\lambda\epsilon} \log \frac{1}{\epsilon})$ randomly sampled nodes is an (ϵ, k)-detection set with high probability.*

We now turn to the new bound on the VC-dimension; to prove it, we will use the following theorem by Mader [18] on edge-disjoint paths between elements of a given set of vertices. Let R be a subset of V of size r. Let $d(R)$ be the number of edges leaving R. Let $q(R)$ be the number of components C of $G - R$ for which $d(C)$ is odd. Let an *R-path* be a path connecting distinct elements of R.

THEOREM 2.10 (Mader [18]). *The maximal number of edge-disjoint R-paths is $\frac{1}{2} \min(\sum d(V_i) - q(\cup V_i))$, where the minimum is taken over all collections of disjoint subsets of vertices $V_1, V_2, \ldots, V_r$ such that $|V_i \cap R| = 1$.*

COROLLARY 2.11. *There are $\Omega(r\lambda)$ edge-disjoint R-paths.*

Proof. Consider a collection of disjoint subsets of vertices $V_1, V_2, \ldots, V_r$ such that $|V_i \cap R| = 1$. Let $d = \sum d(V_i)$, $q = q(\cup V_i)$. By the above theorem it suffices to prove that $d - q = \Omega(r\lambda)$.

Note that $d \geq r\lambda$ since $d(V_i) \geq \lambda$. Let $C_1 \ldots C_q$ be the components C of $G - \cup V_i$ such that $d(C)$ is odd. All edges exiting each C_i are to $\cup V_i$. So $d \geq d(\cup V_i) \geq \sum d(C_i) \geq q\lambda$. If $r \geq q$, then $d - q \geq r\lambda - q \geq r(\lambda - 1)$. If $r < q$, then $d - q \geq q\lambda - q \geq r(\lambda - 1)$. Therefore $d - q = \Omega(r\lambda)$. □

The following is a well-known application of the probabilistic method.

LEMMA 2.12. *Let (R, F) be a multigraph on R. Then there exists a partition of R into sets R_1, R_2 such that there are at least $\frac{1}{2}|F|$ edges between R_1 and R_2.*

LEMMA 2.13. *The VC-dimension of $\mathcal{F}$ is $O(\frac{k}{\lambda})$.*

Proof. Let R be a subset of V of size r. By Corollary 2.11 there exists a family $\mathcal{P}$ of $\Omega(rc)$ edge-disjoint R-paths. Let (R, F) be a multigraph on R such that there is a 1-1 correspondence between uv-paths in $\mathcal{P}$ and edges $uv \in F$. By Lemma 2.12 there exists a partition of R into sets R_1, R_2 such that (in the original graph) there

[3] Both [16] and [7] used a slightly weaker theorem, with a corresponding bound of $O(\frac{d}{\epsilon} \log \frac{d}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta})$.

are $\Omega(r\lambda)$ edge-disjoint paths between R_1 and R_2. We can choose $r = \Theta(\frac{k}{\lambda})$ so that there is guaranteed to be a family $\mathcal{P}'$ of (at least) $k+1$ edge-disjoint paths between R_1 and R_2.

We claim that R cannot be shattered by $\mathcal{F}$. Suppose not. Then there exists $X \in \mathcal{F}$ such that $X \cap R = R_1$. X is a union of components of some cut Z of k or fewer edges. Z is disjoint with (at least) one path $p \in \mathcal{P}'$. The ends of p are in the same Z-component, so they are either both in X, or both not in X. In both cases this contradicts $X \cap R = R_1$. Thus, the claim is proved, and it follows that the VC-dimension of $\mathcal{F}$ is $r = O(\frac{k}{\lambda})$. □

3. Detection sets for node failures. The main theorem of this section (Theorem 3.6) is that for $\kappa < O(\epsilon^2 n)$ a set of $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ randomly sampled nodes is a weak (ϵ,κ)-detection set with high probability. We rely on a special case of ϵ-shredders, which is a corollary of our result on *strong* detection thereof (Theorem 3.1). We also present a partial result (Theorem 3.15) on extending strong detection sets for ϵ-shredders to those for general (ϵ,κ)-cuts.

Before we proceed, let us review the definitions. In this section all cuts are node-cuts, and all detection sets are for node failures. A cut X is called *two-way* if $G \setminus X$ has exactly two connected components, called the *sides* of X. A *shredder* is a min-cut with three or more components. An *ϵ-shredder* is an ϵ-balanced shredder. A set D of nodes *strongly detects* a cut X if some pair of detectors is separated in $G \setminus X$. If D either meets or strongly detects X, we say D *weakly* detects X. D detects (is a detection set for) a family of cuts if it detects every cut in the family.

The rest of this section is organized as follows. In section 3.1 we show how to find a strong detection sets for ϵ-shredders. In section 3.2 we use shredders to get a detection set for two-way ϵ-balanced min-cuts. In section 3.3 we combine these two results and obtain the main theorem. Finally, in section 3.4 we present our partial result on strong detection sets.

3.1. Strong detection sets for shredders. It is a well-known fact that there can be exponentially many min-cuts. Furthermore, even *counting* min-cuts is #P-complete [2]. However, there can be only $O(n)$ shredders [15], with a polynomial-time enumeration algorithm [2]. We start by stating the main result of this subsection.

THEOREM 3.1. *Suppose $\kappa < \epsilon n$. Then a set of $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon\delta})$ randomly sampled nodes is a strong detection set for ϵ-shredders with probability at least $1-\delta$. Moreover, a* smallest *strong detection set for ϵ-shredders has size $\le \frac{1}{\epsilon}$ and can be constructed in polynomial time.*

Before we prove this theorem we need to establish some basic facts about min-cuts. For a cut X the connected components of $G \setminus X$ are also called *X-components*. Let S, T be min-cuts. Say *S meshes T* if S meets at least two T-components. By [2, Lemma 4.3(1)], if S meshes T, then T meets every S-component. Thus meshing is a symmetric relation. If S meshes T (and T meshes S), the two cuts are *meshing*. Else S and T are *nonmeshing*.

LEMMA 3.2 (see [2, Lemma 4.3(2)]). *If min-cuts S and T are meshing, then there is a component Q of either S or T such that Q contains $V - S - T$.*

COROLLARY 3.3. *If $\kappa < \epsilon n$, then any two ϵ-shredders are nonmeshing.*

LEMMA 3.4. *Let S and T be nonmeshing shredders. Let C be the S-component that meets T. Then C contains all T-components but one, call it C'. Moreover, C' contains $V - S - C$, i.e., all S-components other than C.*

Proof. Pick any $v \in S - T$. By minimality of S, v has edges to each S-component (else, $S - v$ is a cut). Thus, $V - S - C + \{v\}$ is connected. Since $T \subset S \cup C$, $V - T - C$

is connected and hence lies in a T-component C'. So all other T-components are contained in C and $V - S - C \subset V - T - C \subset C'$. □

For a family $\mathcal{F}$ of ϵ-shredders, we call a component of a shredder an *$\mathcal{F}$-head* if it meets at least one shredder in $\mathcal{F}$. Now, suppose we have an (ϵ, k)-detection set for shredders, and S is an ϵ-shredder with an $\mathcal{F}$-head H. Then there exists $T \in \mathcal{F}$ that meets H; so by Lemma 3.4, H contains all T-components but one and hence contains a detector. This gives the following lemma.

LEMMA 3.5. *Let $\mathcal{F}$ be a family of ϵ-shredders, with $\kappa < \epsilon n$, and let S be an ϵ-shredder with an $\mathcal{F}$-head H. Then any detection set for $\mathcal{F}$ meets H.*

Proof of Theorem 3.1. Let $\mathcal{F}_0$ be the family of all ϵ-shredders. Start with $\mathcal{F} = \mathcal{F}_0$. While there exists an ϵ-shredder $S \in \mathcal{F}$ with two or more $\mathcal{F}$-heads, delete S from $\mathcal{F}$. Let $\mathcal{F}_1$ be the resulting family of shredders. By Lemma 3.5 any strong detection set for $\mathcal{F}_1$ is a strong detection set for $\mathcal{F}_0$.

Let $S \in \mathcal{F}_1$. Let the *head* H of S be the (single) $\mathcal{F}_1$-head of S. Let the *tail* of S be $V - S - H$. Note that by Lemma 3.4 for any $S, T \in \mathcal{F}_1$ the tail of S is contained in the head of T (and vice versa). In particular, tails are pairwise disjoint. Since each head contains someone else's tail, a set D of nodes is a detection set for $\mathcal{F}_1$ if and only if D meets the tail of each $S \in \mathcal{F}_1$. Therefore, a smallest detection set for $\mathcal{F}_1$ has size $|\mathcal{F}_1|$. Since tails are of size $\geq \epsilon n$ each, $|\mathcal{F}_1| \leq \frac{1}{\epsilon}$. The random sampling result follows by a simple probabilistic computation. □

3.2. Detecting two-way min-cuts. In this subsection we construct a weak detection set for two-way (ϵ, κ)-cuts. First we give a nonefficient deterministic construction. We consider $(\frac{\epsilon}{10}, \kappa)$-cuts and use a greedy-type algorithm to construct a "maximal" family of two-way $(\frac{\epsilon}{10}, \kappa)$-cuts with sides A_i and B_i such that $A_i \subseteq B_j$ for all $i \neq j$. In particular A_i's are pairwise disjoint, so there are at most $\frac{10}{\epsilon}$ of them. It turns out that, if $\kappa < O(\epsilon^2 n)$, then putting a detector into each A_i suffices. More precisely we show (Theorem 3.8) that these detectors together with any weak detection set for shredders give a weak (ϵ, κ)-detection set. Then a simple probabilistic argument yields a randomized result stated below.

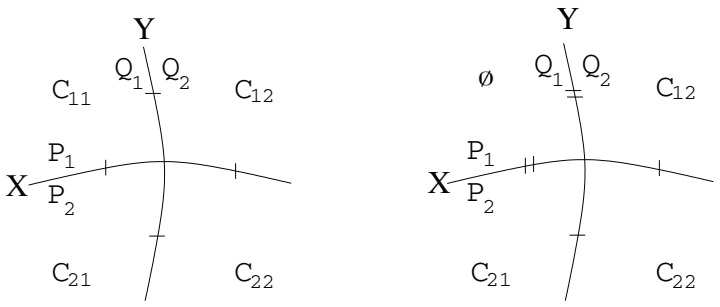
THEOREM 3.6. *Suppose $\kappa < \frac{\epsilon^2 n}{20}$. Then a set of $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon\delta})$ randomly sampled nodes is a weak (ϵ, κ)-detection set with probability at least $1 - \delta$.*

We start with some notation and a simple but very useful lemma about crossing min-cuts. Let S be a set of nodes. Call S *connected* if the subgraph of G induced by S is connected. Else say S is *disconnected*. Say a cut X *preserves* S if X is disjoint with S and S lies in one component of $G \setminus X$. Note that a connected set of nodes is preserved by X if and only if it is disjoint with X. $N(S)$ denotes the set of neighbors of S, i.e., the set of all nodes in $V - S$ that have an edge to S. Note that if $V - S - N(S)$ is nonempty, then $N(S)$ is a cut.

Say two-way min-cuts X and Y are *strongly crossing* if each side of X meets each side of Y. Say X and Y are *weakly crossing* if X meets both sides of Y and vice versa.[4] It is easy to see that strong crossing implies weak crossing, but not the other way round.

To formulate the promised lemma, we will use the following notation. The sides of X and Y are, respectively, P_1, P_2 and Q_1, Q_2. Their intersections ("quarters") are $C_{ij} = P_i \cap Q_j$. Also let $X_i = Q_i \cap X$, $Y_i = P_i \cap Y$, and $X \cap Y = S$.

[4]Note that if X meets both sides of Y, say at v_1 and v_2, respectively, then Y meets both sides of X. Indeed, for the sake of contradiction suppose Y does not meet a side P_1 of X. Then, since any node in X has at least one edge to P_1 and P_1 is connected, there is a v_1v_2-path in G/Y, a contradiction.



(a) X and Y are strongly crossing (b) X and Y are weakly crossing

FIG. 2. *Two applications of the two-quarter lemma.*

LEMMA 3.7 (the two-quarters lemma). *Suppose two-way min-cuts X and Y are weakly crossing so that the two quarters C_{21} and C_{12} are nonempty. Then*

(a) $|X_1| = |Y_1|$ *and* $|Y_2| = |X_2|$,

(b) C_{21} *and* C_{12} *are tight, with* $N(C_{ij}) = Y_j + X_i + S$,

(c) $V - C_{21} - N(C_{21})$ *is connected, same for* C_{12}.

Proof. $T = X_1 + Y_2 + S$ and $U = X_2 + Y_1 + S$ separate C_{21} and C_{12}, respectively, from the rest of the graph. It follows that $Y_2 \geq X_2$ (else $|T| < |X|$), $X_1 \geq Y_1$ (else $|T| < |Y|$), $X_2 \geq Y_2$ (else $|U| < |Y|$), and $Y_1 \geq X_1$ (else $|U| < |X|$). Therefore $|X_1| = |Y_1|$ and $|X_2| = |Y_2|$, so U and T are min-cuts and C_{12} and C_{21} are tight. Finally, $V - C_{21} - N(C_{21})$ is connected as a union of two connected sets (Q_1 and P_2) with a nonempty intersection (C_{12}). ∎

This lemma is similar to the result of Jordán [14] on intersecting tight sets. Note that if X and Y are strongly crossing, our lemma yields $|X_1| = |X_2| = |Y_1| = |Y_2|$ (Figure 2(a)). We will also use it for $\frac{\epsilon}{10}$-balanced min-cuts that are crossing weakly but not strongly. Then one of the "quarters," say C_{11}, is empty, so, assuming $\kappa < \frac{\epsilon n}{10}$, C_{21} and C_{12} are not (Figure 2(b)).

Now we are ready to describe the construction.

CONSTRUCTION.

1. Let $\mathcal{F}$ denote the family of all $\frac{\epsilon}{10}$-balanced two-way min-cuts, and let $\mathcal{A}(\mathcal{F})$ denote the family of the sides of all $F \in \mathcal{F}$. Stop if $\mathcal{F}$ is empty.
2. Choose any inclusionwise minimal component A_0 from $\mathcal{A}(\mathcal{F})$, let $X_0 = N(A_0)$ be the corresponding cut, and let B_0 be the second component of X_0. Put detectors in A_0 and B_0.
3. Delete from $\mathcal{F}$ all cuts which do not preserve A_0. For $X \in \mathcal{F}$, let $A(X)$ be the side of X that *does not* contain A_0.
4. Start with the first iteration. For the ith iteration choose a cut $X_i \in \mathcal{F}$ so that $A(X_i)$ does not contain any other $A(X)$ for $X \in \mathcal{F}$. Let $A_i = A(X_i)$. Let B_i be the other side of X_i. (See Figure 3.)
5. Put a detector into A_i. Remove from $\mathcal{F}$ all cuts which do not preserve $A_0 \cup A_1 \cup \cdots \cup A_i$. Stop if $\mathcal{F}$ is empty; else iterate.

By construction all A_i's are pairwise disjoint, and each $A_i \geq \frac{\epsilon}{10}$. Therefore our algorithm will terminate after at most $\frac{10}{\epsilon}$ steps after putting at most $\frac{10}{\epsilon}$ detectors. Denote this set of detectors by $\mathcal{D}_2$. Let $\mathcal{D}_1$ be any weak detection set for shredders, $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$.

THEOREM 3.8. *Suppose $\kappa \leq \frac{\epsilon^2}{20} n$. Then any ϵ-balanced two-way min-cut is weakly detected by $\mathcal{D}$.*

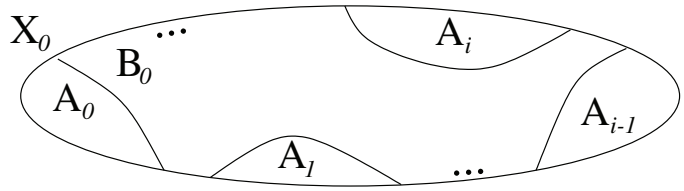


FIG. 3. *Partitioning of the graph after the ith iteration of the algorithm.*

Let us start with some simple properties of the construction.

LEMMA 3.9. *For all $i \neq j$, $A_j \subseteq B_i$. In particular, X_i is disjoint with A_j.*

Proof. We will prove that, for any $i \neq j$, X_i is disjoint with A_j (which would immediately imply $A_j \subseteq B_i$). If $j < i$, then by construction X_i is disjoint with all A_j for $j \leq i$ and $A_j \subseteq B_i$. On the other hand, if $j > i$, then B_j contains A_i, and, supposing $A_j \cap X_i \neq \emptyset$, then $v \in A_j \cap X_i$ has at least one edge to A_i and thus to B_j. Thus A_j and B_j are not separated, a contradiction. ∎

COROLLARY 3.10. *Each B_i contains at least one detector.*

LEMMA 3.11. *If a tight set $A \subset A_i$ is of size $\geq \frac{\epsilon n}{10}$, then the cut $N(A)$ is a shredder.*

Proof. Suppose not. Then $N(A)$ is a two-way $(\frac{\epsilon}{10}, \kappa)$-cut preserving B_i and hence $\bigcup_{j=0}^{i-1} A_j$. Thus $N(A)$ was not deleted from $\mathcal{F}$ until iteration i, so it should have been chosen instead of X_i, a contradiction. ∎

In what follows we assume $\kappa \leq \frac{\epsilon^2}{20} n$. The next lemma shows how $\mathcal{D}_1$ (a detection set for shredders) helps to detect two-way min-cuts.

LEMMA 3.12. *Let Y be an $\frac{\epsilon}{10}$-balanced two-way min-cut with sides C and D. Suppose D contains a set W of size at least $\frac{\epsilon n}{10}$ such that $N(W)$ is a shredder. Then $D + Y$ contains at least one detector from $\mathcal{D}_1$.*

Proof. The shredder $Z = N(W)$ is $\frac{\epsilon}{10}$-balanced, so it is weakly detected by $\mathcal{D}_1$. Since Y is a cut, there are no edges between W and C; i.e., Z lies in $D + Y$. It follows that C is connected in $G \setminus Z$, and hence lies in a single connected component thereof. Thus at least one detector from $\mathcal{D}_1$ is not in C, so it is in $D + Y$. ∎

Now we are ready to sketch the proof of Theorem 3.8; the details are in the next subsection.

Proof sketch of Theorem 3.8. Let Y be an ϵ-balanced two-way min-cut with sides C and D. We need to show that $\mathcal{D}$ meets Y or both sides thereof. For the sake of contradiction suppose it is not so. Then without loss of generality $\mathcal{D} \subset C$, which implies that C meets every A_i and B_i. Clearly then $A_i \not\subseteq D + Y$, for every i. Also note that by Lemma 3.12 D cannot contain disconnected tight sets larger than $\frac{\epsilon n}{10}$.

There are now three cases to consider, depending on the relation of Y to the sets X_i. First, suppose Y does not strongly cross any X_i. We show that $N(D \setminus \cup X_i)$ is a two-way $\frac{\epsilon}{10}$-balanced cut that was not excluded from $\mathcal{F}$ (see Figure 4(a)), and this contradicts the stopping condition of the algorithm. If Y strongly crosses exactly one X_i, then we replace Y by the cut $Y' = N(D \cap B_i)$ (see Figure 4(b)). Y' does not strongly cross any X_i, so we apply the argument from the case above to show that Y' is detected. Therefore there is at least one detector in set D, which contradicts our assumption. Finally, if neither of these two cases applies, then Y strongly crosses at least two sets among $\{X_i\}$, say X_i and X_j. An argument using the two-quarters lemma then shows that X_i and X_j partition Y into the same subsets (see Figure 4c). We then prove that X_i and X_j cut off a large connected subset D' of D such that

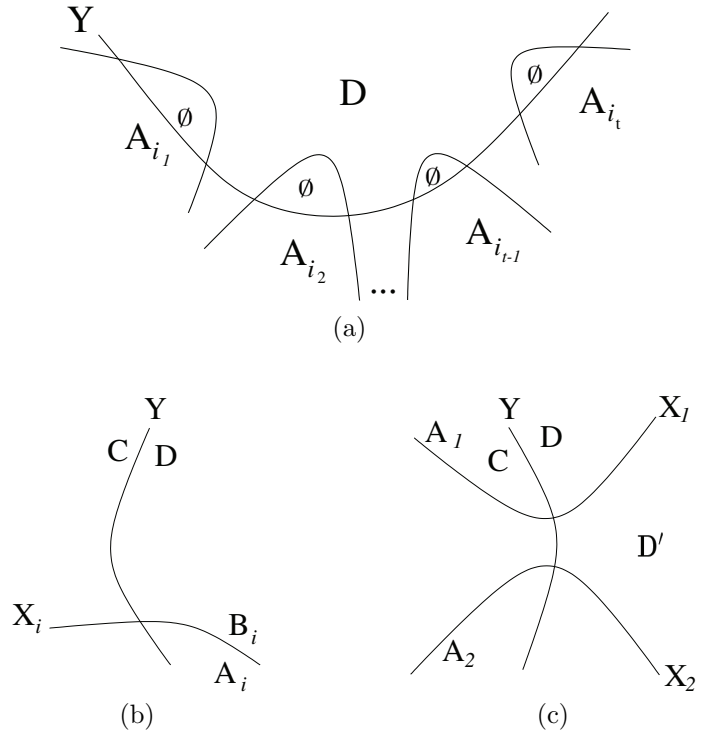


FIG. 4. *Three different options of how Y can interact with X_i's. For* (c) *we prove that the portion of Y between cuts X_1 and X_2 shrinks to an empty set, and $X_1 \cap Y = X_2 \cap Y$.*

$N(D')$ is a two-way $(\frac{\epsilon}{10}, \kappa)$-cut not deleted from $\mathcal{F}$, which thus violates the stopping condition. □

3.3. Full proof of Theorem 3.8.

LEMMA 3.13. *Suppose Y is ϵ-balanced and A_i meets D. Then either there is a detector in $D + Y$ or the following conditions hold:*

(a) *Y strongly crosses X_i, and*

(b) *$N(D \cap B_i)$ is a two-way $\frac{8\epsilon}{10}$-balanced min-cut.*

Proof. Suppose there is no detector in $D + Y$. Since A_i and B_i each contain a detector, they meet C. Now we can invoke the two-quarters lemma to quarters $B_i \cap C$ and $A_i \cap D$ and conclude that $A_i \cap D$ is tight. We claim that $|B_i \cap D| \geq \frac{8\epsilon}{10}\, n$. Indeed, otherwise $|A_i \cap D| \geq \frac{\epsilon n}{10}$; thus, by Lemma 3.12, $N(A_i \cap D)$ is a two-way cut, which contradicts Lemma 3.11. The claim is proved.

This proves (a) and shows that $N(B_i \cap D)$ is an $\frac{8\epsilon}{10}$-balanced cut. To complete (b), note that $B_i \cap D$ is tight by the two-quarters lemma; thus, by Lemma 3.12, $N(B_i \cap D)$ is two-way. □

Let Y be an ϵ-balanced two-way min-cut with sides C and D. We need to show that $\mathcal{D}$ meets Y or both sides thereof. For the sake of contradiction suppose it is not so. Then without loss of generality $\mathcal{D} \subset C$, which implies that C meets every A_i and B_i. Clearly then $A_i \not\subseteq D + Y$, for every i. Also note that, by Lemma 3.12, D cannot contain disconnected tight sets larger than $\frac{\epsilon n}{10}$. There are three possible cases which we prove separately: (1) cut Y does not strongly cross any X_i, (2) cut Y strongly crosses exactly one X_i, and (3) cut Y strongly crosses at least two X_i's.

Case 1. Cut Y does not strongly cross any X_i.

To reuse this proof for the second case, we will assume that Y is only $\frac{8\epsilon}{10}$-balanced, rather than ϵ-balanced,

Since we assumed that X_i does not strongly cross Y, by Lemma 3.13 we have that all A_i's are disjoint with D. Using this fact, we show that each X_i excises a small piece of size at most κ from D, and finally we show that $D \setminus \cup X_i$ is large, tight, and connected and preserves $\cup A_i$; thus our algorithm could have made at least one more step.

Let $X_{i_1}, X_{i_2}, \ldots, X_{i_t}$ be all cuts which are intersecting with D. Let $D_j = D - D \cap \bigcup_{h=1}^{j} X_{i_h}$, $Y_j = N(D_j)$, and $C_j = V - Y_j - D_j$. First,

$$|D_j| \geq |D| - \sum_{h=1}^{j} |X_{i_h}| \geq \frac{8\epsilon}{10}\, n - \kappa \frac{10}{\epsilon} \geq \frac{3\epsilon}{10}\, n.$$

The last transition is because $\kappa \leq \frac{\epsilon^2}{20} n$.

We will prove by induction that each D_j is tight and connected and that corresponding cut $Y_j = N(D_j)$ is two-way for every $0 \leq j \leq t$.

Suppose we did that; then D_t by its construction is disjoint with any X_i, and thus all A_i's are disjoint with Y_t and hence lie in $V - D_t - Y_t$. Therefore Y_t preserves $\bigcup A_i$ (because Y_t is a two-way cut). On the other hand, $|D_t| \geq \frac{2\epsilon}{10}\, n$ and $|C_t| \geq |C| \geq \epsilon N$. So Y_t is a $\frac{2\epsilon}{10}$-balanced two-way min-cut and preserves $\bigcup A_i$; thus our algorithm could have made one more step, and so we come to a contradiction.

Now we have to prove our claim. Clearly D_0 is tight and connected, and $N(D_0) = Y$ is two-way by our definition of Y and D. Supposing the claim holds for D_{j-1}, we now prove it for D_j. We have

$$D_j = D_{j-1} - D_{j-1} \cap X_{i_j} = B_{i_j} \cap D_{j-1}.$$

If D_j is disjoint with X_{i_j} then $D_j = D_{j-1}$ and we are immediately done. Otherwise, Y_{j-1} weakly crosses X_{i_j}. (Indeed, D_{j-1} is not preserved by X_{i_j}. Moreover, C_{j-1} contains C and hence meets both A_{i_j} and B_{i_j}. It follows that C_{j-1} is not preserved by X_{i_j}.) But then we satisfy conditions of the two-quarters lemma, where $A_{i_j} \cap C_{j-1}$ and $B_{i_j} \cap D_{j-1}$ is not empty, and thus $D_j = B_{i_j} \cap D_{j-1}$ is tight. Therefore, by Lemma 3.12, D_j is connected and $N(D_j)$ is a two-way cut. This proves the claim.

Case 2. Cut Y strongly crosses exactly one X_i.

Indeed, consider set $D' = D \cap B_i$. By Lemma 3.13 and our assumption that there were no detectors in $D + Y$, it has size at least $\frac{8\epsilon}{10}\, n$ and is tight, and corresponding cut $Y' = D \cap X_i + X_i \cap Y + Y \cap B_i$ is a two-way min-cut.

Since $D' \subseteq D$ and only one A_i meets D (and it does not meet with D' by our construction), no A_i meets with D'. Therefore, by Lemma 3.13, Y' does not strongly cross any X_i, and thus by the case (1) Y' is detected by $\mathcal{D}$. This proves that there is at least one detector in $Y' + D'$, and by construction $Y' + D' \subseteq D + Y$; therefore there is at least one detector in $D + Y$, a contradiction.

Case 3. Cut Y strongly crosses at least two X_i's. We need to prove that either $D + Y$ contains at least one detector from $\mathcal{D}$ (thus contradicting our assumption), or we could have done one more iteration of the construction in section 3.2. Without loss of generality, Y strongly crosses X_1 and X_2 (see Figure 4(c)).

First we prove that each of the triples (A_1, X_1, B_1) and (A_2, X_2, B_2) partitions set Y into the same subsets.

CLAIM 3.14. *$X_1 \cap Y = X_2 \cap Y$, $A_1 \cap Y = B_2 \cap Y$, and $B_1 \cap Y = A_2 \cap Y$.*

Proof. Note that $Y = Y \cap A_i + Y \cap X_i + Y \cap B_i$. Since $A_1 \subseteq B_2$, we have that $Y \cap A_1 \subseteq Y \cap B_2$, and analogously $Y \cap A_2 \subseteq Y \cap B_1$, but by the two-quarters lemma we have $|Y \cap A_1| = |Y \cap B_1|$ and $|Y \cap A_2| = |Y \cap B_2|$ and thus $Y \cap A_1 = Y \cap B_2$ and $Y \cap A_2 = Y \cap B_1$. Thus $X_1 \cap Y = X_2 \cap Y$. ∎

We will prove that either there is a leftover part D' in D, which could have been used for the next step of the algorithm, or Y is detected.

For each $i = 1, 2$, since X_i strongly crosses Y, set D is partitioned by X_i into three nonempty parts $D'_i = D \cap B_i$, $D''_i = D \cap A_i$, and $D'''_i = D \cap X_i$. Now, by Lemma 3.13 and our assumption that $\mathcal{D} \cap (D + Y) = \emptyset$, we conclude that D'_i is tight, its cardinality is at least $\frac{8\epsilon}{10} n$, and $N(D'_i)$ is two-way min-cut.

Consider $D' = D'_1 \cap D'_2$. We claim that the corresponding cut $Z = N(D')$ is a two-way $(\frac{\epsilon}{10}, \kappa)$-cut that preserves $\cup A_i$. This contradicts the stopping condition of the algorithm: it could have made one more iteration. Therefore it remains to prove the claim.

First, D' is tight by the two-quarters lemma applied to cuts $N(D'_1)$ and $N(D'_2)$. Its size is

$$|D'| = |D'_1 \cap D'_2| = |D - (D''_1 + D'''_1) \cup (D''_2 + D'''_2)| \geq \epsilon n - 2\left(\frac{\epsilon n}{10} + \kappa\right) \geq \frac{6\epsilon}{10} n,$$

so Z is $\frac{\epsilon}{10}$-balanced, and, moreover, D' is connected (this is by Lemma 3.12 and the assumption that $\mathcal{D}$ is disjoint with $Y + D$). Since $Z = (X_1 \cup X_2) \cap (D \cup Y)$, we conclude that (1) Z is two-way, since $V - D' - Z$ is connected as a union of three nondisjoint connected subsets C, A_1, and A_2, and (2) Z is disjoint with $\cup A_i$ by Lemma 3.9.

To prove that Z preserves $\cup A_i$ it remains to show that all A_i's are disjoint with D'. Indeed, suppose some A_i meets D'. It cannot be properly contained in D, and hence not in D'. So, since A_i is connected, it meets Z, a contradiction. The claim is proved. This completes the proof of Theorem 3.8. ∎

3.4. Strong detection sets. We present a partial result on extending strong detection sets for ϵ-shredders to those for general (ϵ, κ)-cuts. Essentially, we show that it suffices to have a strong (ϵ, κ)-detection set D' for some subgraph $G' = (V, E')$ of G of the same connectivity κ. In particular, we can without loss of generality assume that G is *minimally* k-connected.

THEOREM 3.15. *Suppose $\kappa < \epsilon n$ and we have a strong (ϵ, κ)-detection set D' for a κ-connected subgraph $G' = (V, E')$ of G. Then we can use D' to construct a strong (ϵ, κ)-detection set for G. Specifically, for a high-probability result it suffices to add $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ randomly sampled detectors. Alternatively, it suffices to add at most $\frac{2}{\epsilon}$ detectors, and there is a polynomial-time algorithm to construct them.*

Proof. Let D'' be a smallest detection set for ϵ-shredders of G. Let S be an (ϵ, κ)-cut G. Then S is an (ϵ, κ)-cut in G' such that each S-component in G is a union of S-components in G'. Obviously, if S-components are the same in G and in G', then D' detects S. Therefore, if $D' \cup D''$ does not detect S, then S is a two-way (ϵ, κ)-cut in G but a shredder in G'. Call such cuts *evil*. Therefore it suffices to detect all evil cuts.

For an evil cut S, the two components of S in G are called *S-shores*. We need to put a detector in each S-shore. For the rest of the proof we can forget about G. We operate (only) on G' and treat S-shores as unions of components of S in G'. The proof is similar to that of Theorem 3.1.

Evil cuts are ϵ-shredders in G', so there are at most n of them and they can be efficiently listed. Let $\mathcal{F}_0$ be the family of *all* evil cuts. Start with $\mathcal{F} = \mathcal{F}_0$. While

there exists $S \in \mathcal{F}$ such that each S-shore contains an $\mathcal{F}$-head of S, delete S from $\mathcal{F}$ (because by Lemma 3.5 S is detected by D'). Let $\mathcal{F}_1$ be the resulting family of evil cuts. Clearly if D is a detection set for $\mathcal{F}_1$, then $D \cup D'$ is a detection set for $\mathcal{F}_0$.

Say $H \subset V$ is a *head* of S if H is an $\mathcal{F}_1$-head of S. Let the *tail shore* of $S \in \mathcal{F}_1$ be the S-shore that does not contain any heads of S (such a shore exists by construction of $\mathcal{F}_1$). Observe that for any two $S, T \in \mathcal{F}_1$ the tail shore of T is contained in a head of S (and vice versa). Why? T meets exactly one component of S, say H (so H is a head). By Lemma 3.4 H contains all T-components but one, call it C. C meets S; thus C is a head. Therefore, the tail shore of T is contained in H.

By the observation above, the tail shores of cuts in $\mathcal{F}_1$ are pairwise disjoint, and, moreover (assuming $\mathcal{F}_1$ consists of at least two cuts), putting a detector in each of these shores strongly detects $\mathcal{F}_1$. Since the tail shores have size $\geq \epsilon n$ each, $|\mathcal{F}_1| \leq \frac{1}{\epsilon}$. Therefore we need $\frac{1}{\epsilon}$ detectors for F_0, which together with D'' is at most $\frac{2}{\epsilon}$ detectors.

For the random sampling result note that it suffices to augment D' by a hitting set for the tail shores of $\mathcal{F}_1$ and the tails of ϵ-shredders of G, as defined in the proof of Theorem 3.1. □

4. Extensions and further directions. There are a number of natural questions left open by this work. One is to investigate whether an (ϵ, κ)-detection set for node failures of minimum size can be computed in polynomial time for a given graph G; this would parallel the per-instance result we obtain for edge failures. We note that section 3.1 provides such an optimality result for node failures when the adversary is restricted to deleting a shredder.

We believe it would be interesting to extend our results on node failures to obtain bounds for strong detection sets. In fact, our bounds for shredders apply already to the case of strong detection, and in Theorem 3.15 we provide a further step in this direction, proving that we can without loss of generality assume that G is *minimally* k-connected.

It would clearly be interesting to obtain results on detection sets with respect to adversaries that can delete a number of nodes equal to a constant times the node-connectivity, by analogy with our results for edge-connectivity. To obtain detection set bounds here that are independent of the value of κ, it is not difficult to see that we need to focus on weak detection; indeed, there exist graphs in which we would need at least $k - \kappa$ nodes in any strong (ϵ, k)-detection set for node failures.

Finally, the problem of deciding whether a given set D is an (ϵ, k)-detection set provides another clear connection to the problem of balanced separators in graphs: indeed, deciding whether the empty set is an (ϵ, k)-detection set is coNP-complete because of its equivalence to a balanced separator problem. On the other hand, using techniques from [11, 22], we can obtain a polynomial-time algorithm for deciding whether D is an (ϵ, k)-detection set for node failures when $k = \kappa$; this is nontrivial due to the fact that there can be exponentially many min-node-cuts.

Acknowledgment. It is our pleasure to acknowledge the contribution of Laszlo Lovász; discussions with him about the prospect of parameterizing detection sets by the minimum cut size provided a portion of the motivation for this work and also led to the results described in section 2.3.

REFERENCES

[1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, *Learnability and the Vapnik-Chervonenkis dimension*, J. ACM, 36 (1989), pp. 929–965.

[2] J. Cheriyan and R. Thurimella, *Fast algorithms for k-shredders and k-node connectivity augmentation*, J. Algorithms, 33 (1999), pp. 15–50.
[3] B. Cherkasskij, *Solution of a problem on multicommodity flows in a network*, Ekon. Mat. Metody, 13 (1977), pp. 143–151 (in Russian).
[4] F. Chung, M. Garrett, R. Graham, and D. Shallcross, *Distance realization problems with applications to Internet tomography*, J. Comput. System Sci., 63 (2001), pp. 432–448.
[5] K. Claffy, T. Monk, and D. McRobb, *Internet Tomography*, Nature, Web Matters, January 7, 1999, http://www.nature.com/nature/webmatters/tomog/tomog.html.
[6] E. Dinits, A. Karzanov, and M. Lomonosov, *On the structure of a family of minimal weighted cuts in a graph*, in Studies in Discrete Optimization, A. Fridman, ed., Nauka, Moscow, 1976, pp. 290–306 (in Russian); English translation available from National Translations Center, Library of Congress, Cataloging Distribution Service, Washington, DC (NTC 89-20265).
[7] J. Fakcharoenphol, *An Improved VC-Dimension Bound for Finding Network Failures*, Master's thesis, Department of EECS, University of California-Berkeley, Berkeley, 2001.
[8] U. Feige and M. Mahdian, *Finding small balanced separators*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006, pp. 375–384.
[9] L. Fleischer, *Building chain and cactus representation of all minimum cuts from Hao-Orlin in the same asymptotic run time*, J. Algorithms, 33 (1999), pp. 51–72.
[10] A. Frank, *Packing paths, circuits, and cuts—a survey*, in Paths, Flows and VLSI-Layouts, B. Korte, L. Lovász, H-J. Prömel, and A. Schrijver, eds., Springer-Verlag, Berlin, 1990, pp. 47–100.
[11] H. Gabow, *Centroids, representations, and submodular flows*, J. Algorithms, 18 (1995), pp. 586–628.
[12] H. Gabow, *Using expander graphs to find vertex connectivity*, J. ACM, 53 (2006), pp. 800–844.
[13] A. Gupta, *private communication*, 2000.
[14] T. Jordán, *On the optimal vertex-connectivity augmentation*, J. Combin. Theory Ser. B, 63 (1995), pp. 8–20.
[15] T. Jordán, *On the number of shredders*, J. Graph Theory, 31 (1999), pp. 195–200.
[16] J. M. Kleinberg, *Detecting a network failure*, Internet Math., 1 (2003), pp. 37–55.
[17] L. Lovász, *On some connectivity properties of Eulerian graphs*, Acta Math. Hungar., 28 (1976), pp. 129–138.
[18] W. Mader, *Über die Maximalzahl kantendisjunkter A-Wege*, Arch. Math. (Basel), 30 (1978), pp. 325–336.
[19] J. Mahdavi and V. Paxson, *IPPM Metrics for Measuring Connectivity*, RFC 2678, The Internet Engineering Task Force (IETF), 1999.
[20] S. Muthukrishnan, T. Suel, and R. Vingralek, *Inferring tree topologies using flow tests*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 828–829.
[21] V. Paxson, *Towards a framework for defining internet performance metrics*, in Proceedings of the INET, 1996.
[22] J.-C. Picard and M. Queyranne, *On the structure of all minimum cuts in a network and applications*, Math. Programming Stud., 13 (1980), pp. 8–16.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1347–1363

RESOLUTION IS NOT AUTOMATIZABLE UNLESS W[P] IS TRACTABLE*

MICHAEL ALEKHNOVICH† AND ALEXANDER A. RAZBOROV‡

Abstract. We show that neither resolution nor tree-like resolution is automatizable unless the class **W[P]** from the hierarchy of parameterized problems is fixed-parameter tractable by randomized algorithms with one-sided error.

Key words. proof complexity, resolution, automatizability

AMS subject classifications. 03F20, 03D15

DOI. 10.1137/06066850X

1. Introduction.

Analysis of the usefulness of proof search heuristics and automated theorem proving procedures based on a proof system P amounts (on the theoretical level) to the following two basic questions:

Question 1. Which theorems in principle possess efficient P-proofs?

Question 2. How can one find the optimal (or, at least, a nearly optimal) proof of a given theorem in P?

Traditional proof complexity mostly dealt, and still deals, with the first question. However, there has been a growing interest in the second one, too. An additional motivation for studying the complexity of finding optimal proofs comes from deep connections with efficient interpolation theorems; we refer the reader to the surveys [9, 19, 22] for more details. These surveys also serve as a good starting point for learning more about propositional proof complexity in general.

One convenient framework for the theoretical study of Question 2 was proposed by Bonet, Pitassi, and Raz in [13]. Namely, they called a proof system P *automatizable* if there exists a deterministic algorithm A which, given a tautology τ, returns its P-proof in time polynomial *in the size of the shortest P-proof of τ*. The definition of a quasi-automatizable proof system is given in the same way, but we only require algorithm A to run in time which is quasi-polynomial (in the same parameter).

One advantage of this definition is that it allows us to completely disregard the first basic question on the *existence* of efficient P-proofs and to indeed concentrate on *finding* efficient proofs *provided* they exist. In particular, the notion of automatizability makes perfect sense for those (weak) proof systems for which hard tautologies are already known. Moreover, the weaker our system is, the more likely it seems to be automatizable. One possible explanation of this phenomenon comes from the connection between automatizability and efficient interpolation (every automatizable proof system has efficient interpolation, and the property of having efficient interpolation is indeed antimonotone w.r.t. the strength of the system). Anyway, given

*Received by the editors August 29, 2006; accepted for publication (in revised form) April 14, 2008; published electronically August 6, 2008. A preliminary version of this paper appeared in the *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, (Las Vegas, NV)*, 2001.

http://www.siam.org/journals/sicomp/38-4/66850.html

†This author is deceased. Former address: Department of Mathematics, University of California at San Diego, La Jolla, CA 92093.

‡Department of Computer Science, University of Chicago, Chicago, IL 60637 (razborov@cs.uchicago.edu). This author's research was supported by the State of New Jersey and NSF grant CCR-9987077.

this connection, the results from [20, 13] imply that extended Frege and TC^0-Frege proof systems, respectively, are not automatizable under some widely believed cryptographic assumptions. Bonet et al. [11] extended the latter result to bounded-depth Frege but under a much stronger assumption.

In this paper we are primarily interested in the automatizability of resolution and tree-like resolution. It is worth noting that both systems possess efficient interpolation, and therefore their nonautomatizability cannot be proved via techniques similar to those in [20, 13, 11]. Nonetheless, [18] proved that it is **NP**-hard to find the shortest resolution refutation. Alekhnovich et al. [3] proved that if $\mathbf{P} \neq \mathbf{NP}$, then the length of the shortest resolution refutation cannot be approximated to within a constant factor (both for general and tree-like resolution). Under the stronger assumption $\mathbf{NP} \not\subseteq \mathbf{QP}$, they were able to improve the ratio from an arbitrary constant to $2^{\log^{1-\epsilon} n}$ (later, Dinur and Safra [16] obtained a better probabilistically checkable proofs (PCP) characterization of **NP** that allows one to prove the same bound for arbitrary $\epsilon \to 0$ modulo $\mathbf{P} \neq \mathbf{NP}$).

In the opposite direction, Beame and Pitassi [8] observed that tree-like resolution is quasi-automatizable. Thus, it is unlikely to show that this system is not automatizable modulo $\mathbf{P} \neq \mathbf{NP}$, because it would imply quasi-polynomial algorithms for **NP** (in case of general resolution this goal seems also tricky at the moment because there is only one[1] known example [12] for which the proof search algorithm of [10] requires more than quasi-polynomial time). Therefore, any result establishing nonautomatizability of tree-like resolution needs to be formulated within a complexity framework in which the asymptotics $n^{O(1)}$ and $n^{\log n}$ are essentially different.

One natural example of such a framework is *parameterized complexity* introduced by Downey and Fellows (see [17]) in which algorithms working in times $f(k)n^{O(1)}$ and n^k are considered different from the point of view of effectiveness (here k is an integer input parameter that should be thought of as an "arbitrarily large" constant). In this paper we prove that neither resolution nor tree-like resolution is automatizable unless the class $\mathbf{W}[\mathbf{P}]$ (lying very high in the hierarchy of parameterized problems) is fixed-parameter tractable by a randomized algorithm with one-sided error (Theorem 2.7). Our proof goes by a reduction from the optimization problem MINIMUM MONOTONE CIRCUIT SATISFYING ASSIGNMENT (MMCSA) whose decision version is complete for the class $\mathbf{W}[\mathbf{P}]$. An alternative hardness assumption is that there is no *deterministic* fixed-parameter algorithm which *approximates* MMCSA within any constant factor (Theorem 2.5). It is worth noting in this connection that we were able to relate to each other the hardness of finding *exact* and *approximate* solutions for MMCSA without using the PCP theorem (see the proof of Theorem 2.7 given in section 4). This result can be interesting on its own.

The paper is organized as follows. Section 2 contains necessary preliminaries and definitions. In section 3 we present our core reduction from MMCSA to automatizability of resolution, and in section 4 we use (sometimes nontrivial) self-improving techniques to prove our main results, Theorems 2.5 and 2.7. The paper is concluded with some open problems in section 5.

1.1. Recent developments. Since the preliminary version of this paper was released, the following related developments have occurred.

Atserias and Bonet [6] studied a slightly different variant of automatizability that they called *weak automatizability*. Using their techniques, they were also able

[1] See, however, section 1.1.

to produce more examples of poly-size tautologies for which the width-based proof search algorithm from [10] requires more than quasi-polynomial time. In the opposite direction, Alekhnovich and Razborov [4] introduced an enhancement of that algorithm which they called BWBATP (branch-width based automated theorem prover). This algorithm performs better than the width-based algorithm for several important classes of tautologies, and for at least one such class it even achieves complete (that is, polynomial) automatization. Finally, quite unexpectedly our techniques turned out to be useful in the totally different area of computational learning, where they inspired a number of strong hardness results for the so-called model of proper learning [2].

2. Preliminaries and main results.

2.1. Resolution and automatizability. Let x be a Boolean variable, i.e., a variable that ranges over the set $\{0,1\}$. A *literal* of x is either x (denoted sometimes as x^1) or $\bar{x}$ (denoted sometimes as x^0). A *clause* is a disjunction of literals. A *CNF* (conjunctive normal form) is a conjunction of pairwise different clauses.

Let $f(x_1,\dots,x_n)$ be an arbitrary function (possibly, partial) from $\{0,1\}^n$ to some finite domain D. An *assignment to* f is a mapping $\alpha : \{x_1,\dots,x_n\} \to \{0,1\}$. A *restriction of* f is a mapping $\rho : \{x_1,\dots,x_n\} \to \{0,1,\star\}$. We denote by $|\rho|$ the number of assigned variables, $|\rho| \stackrel{\text{def}}{=} |\rho^{-1}(\{0,1\})|$. The *restriction of a function* f *or CNF* τ *by* ρ, denoted by $f|_\rho$ [$\tau|_\rho$], is the function [CNF] obtained from f [τ, respectively] by setting the value of each $x \in \rho^{-1}(\{0,1\})$ to $\rho(x)$, and leaving each $x \in \rho^{-1}(\star)$ as a variable.

The general definition of a propositional proof system was given in the seminal paper [15]. But since we are interested only in resolution (which is one of the simplest and most widely studied concrete systems), we prefer to skip this general definition. *Resolution* operates with clauses and has one rule of inference called *resolution rule*:

$$\frac{A \vee x \qquad B \vee \bar{x}}{A \vee B}.$$

A resolution proof is *tree-like* if its underlying graph is a tree. A *resolution refutation* of a CNF τ is a resolution proof of the empty clause from the clauses appearing in τ.

The *size* of a resolution proof is the overall number of clauses in it. For an unsatisfiable CNF τ, $S(\tau)$ [$S_T(\tau)$] is the minimal size of its resolution refutation (tree-like resolution refutation, respectively). Clearly, $S(\tau) \le S_T(\tau)$.

The *width* $w(C)$ *of a clause* C is the number of literals in C. The *width* $w(\tau)$ *of a set of clauses* τ (in particular, the width of a resolution proof) is the maximal width of a clause appearing in this set. For a CNF τ, let $n(\tau)$ be the overall number of distinct variables appearing in it, and let $|\tau|$ be the overall number of occurrences of variables in τ, i.e., $|\tau| \stackrel{\text{def}}{=} \sum_{C\in\tau} w(C)$. For an unsatisfiable CNF τ, $w(\tau \vdash \emptyset)$ will denote the minimal width of its resolution refutation.

For a nonnegative integer n, let $[n] \stackrel{\text{def}}{=} \{1,2,\dots,n\}$, and let $[n]^k \stackrel{\text{def}}{=} \{\, I \subseteq [n] \mid |I| = k \,\}$.

We will recall the general definition of automatizability from [13] for the special cases of resolution and tree-like resolution.

DEFINITION 2.1. *Resolution (tree-like resolution) is* (quasi-)automatizable *if there exists a deterministic algorithm* A *which, given an unsatisfiable CNF* τ*, returns its resolution refutation (tree-like resolution refutation, respectively) in time which is (quasi-)polynomial in* $|\tau| + S(\tau)$ *(*$|\tau| + S_T(\tau)$*, respectively).*

Remark 1. Note that we do not require that all clauses from τ must necessarily appear in the refutation, and therefore, we cannot a priori expect the inequality $n(\tau) \cdot S(\tau) \geq |\tau|$. This is why we must introduce the term $|\tau|$ into the bound on the running time of A when adapting the general definition of automatizability from [13] to the case of resolution.

2.2. Parameterized complexity and the MMCSA problem. We refer the reader to [17] for a good general introduction to the topic of parameterized complexity.

DEFINITION 2.2 (see [17, Definition 2.4]). *The class* **FPT** (fixed-parameter tractable) *of parameterized problems consists of all languages $L \subseteq \Sigma^* \times \mathbb{N}$ for which there exists an algorithm Φ, a constant c, and a recursive function $f : \mathbb{N} \longrightarrow \mathbb{N}$ such that*

1. *the running time of $\Phi(\langle x, k\rangle)$ is at most $f(k) \cdot |x|^c$;*
2. *$\langle x, k\rangle \in L$ iff $\Phi(\langle x, k\rangle) = 1$.*

Thus, an algorithm is considered to be feasible if it works in time polynomial in $n \stackrel{\text{def}}{=} |x|$ and $f(k)$, where k should be thought of as much smaller than n, and f is an arbitrarily large (recursive) function. A similar feasibility requirement arises in the theory of polynomial-time approximation schemes (PTAS) for **NP**-hard problems: assume that we have an algorithm that approximates a given problem within arbitrary error $\epsilon > 0$ working in time $n^{O(1/\epsilon)}$. Is it possible to get rid of $1/\epsilon$ in the exponent and do it in time $f(1/\epsilon)n^{O(1)}$? (The algorithms which obey the latter bound on the running time are called EPTAS, efficient polynomial-time approximation schemes.)

It turns out that this question is tightly related to the fixed-parameter tractability. Namely, the existence of EPTAS for a given problem implies an *exact* algorithm for the corresponding fixed-parameter version (see [7, 14]).

To study the complexity of parameterized problems, the following *parameterized reduction* (that preserve the property of being in **FPT**) is used.

DEFINITION 2.3 (see [17, Definition 9.3]). *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$* reduces *to another parameterized problem $L' \subseteq \Sigma^* \times \mathbb{N}$ if there exist (arbitrary!) functions $f, g : \mathbb{N} \longrightarrow \mathbb{N}$, and a function $h : \Sigma^* \times \mathbb{N} \longrightarrow \Sigma^*$ such that $h(x, k)$ is computable in time $f(k)|x|^{O(1)}$, and $\langle x, k\rangle \in L$ iff $\langle h(x, k), g(k)\rangle \in L'$.*

For any integer t, the parameterized problem WEIGHTED t-NORMALIZED SATISFIABILITY is defined by restricting the ordinary SATISFIABILITY to a certain class of Boolean formulas depending on t (we omit the exact definition since it is a little bit technical and not needed for our results), and the parameter k bounds the Hamming weight of the satisfying assignment we are searching for. The complexity class $\mathbf{W[t]}$ consists of all problems that can be reduced to WEIGHTED t-NORMALIZED SATISFIABILITY via parameterized reduction, and the class $\mathbf{W[P]}$ (where P stands for polynomial) includes all problems reducible to WEIGHTED CIRCUIT SATISFIABILITY described as follows:

WEIGHTED CIRCUIT SATISFIABILITY:
Instance: A circuit C.
Parameter: A positive integer k.
Question: Does C have a satisfying assignment of Hamming weight (defined as the number of ones) k?

These definitions lead to the following *parameterized hierarchy*, in which every inclusion is believed to be strict:

$$\mathbf{FPT} \subseteq \mathbf{W[1]} \subseteq \mathbf{W[2]} \cdots \subseteq \mathbf{W[P]}.$$

In our paper we construct a randomized parameterized reduction from the automatizability of resolution to the following optimization problem (MMCSA in what follows) that was introduced in [3].

Monotone minimum circuit satisfying assignment:
Instance: A monotone circuit C in n variables over the basis $\{\wedge, \vee\}$.
Solution: An assignment $a \in \{0,1\}^n$ such that $C(a) = 1$.
Objective function: $k(a)$, defined as its Hamming weight.

By $k(C)$ we will denote the minimal value $k(a)$ of a solution a for an instance C of MMCSA.

The following easy observation was made in [3] ("self-improvement").

PROPOSITION 2.4. *For every fixed integer $d > 0$ there exists a polynomial-time computable function π which maps monotone circuits into monotone circuits and such that $k(\pi(C)) = k(C)^d$ for all C.*

Our first result can be now formulated as follows.

THEOREM 2.5. *If either resolution or tree-like resolution is automatizable, then for any fixed $\epsilon > 0$ there exists an algorithm Φ receiving as inputs monotone circuits C which runs in time $\exp\left(k(C)^{O(1)}\right) \cdot |C|^{O(1)}$ and approximates the value of $k(C)$ to within a factor $(1+\epsilon)$.*

The decision version of MMCSA was considered in [17] (under the name WEIGHTED MONOTONE CIRCUIT SATISFIABILITY) in the context of parameterized complexity and was shown to be complete in the class **W[P]**.

In order to formulate our second (and main) result, we need to introduce the obvious hybrid of the classes **R** and **FPT**.

DEFINITION 2.6. *The class* **FPR** *(fixed-parameter randomized) of parameterized problems consists of all languages $L \subseteq \Sigma^* \times \mathbb{N}$ for which there exists a probabilistic algorithm Φ, a constant c, and a recursive function $f : \mathbb{N} \longrightarrow \mathbb{N}$ such that*

1. *the running time of $\Phi(\langle x, k\rangle)$ is at most $f(k) \cdot |x|^c$;*
2. *if $\langle x, k\rangle \in L$, then $\mathbf{P}[\Phi(\langle x, k\rangle) = 1] \geq 1/2$;*
3. *if $\langle x, k\rangle \notin L$, then $\mathbf{P}[\Phi(\langle x, k\rangle) = 1] = 0$.*

Then we have the following.

THEOREM 2.7. *If either resolution or tree-like resolution is automatizable, then* **W[P]** $\subseteq$ *co-*__FPR__.

3. Main reduction from MMCSA to automatizability of resolution. This section is entirely devoted to the proof of the following technical lemma.

LEMMA 3.1. *There exists a polynomial-time computable function τ which maps any pair $\langle C, 1^m\rangle$, where C is a monotone circuit and m is an integer, to an unsatisfiable CNF $\tau(C, m)$ such that*

$$S_T(\tau(C,m)) \leq |C| \cdot m^{O(\min\{k(C),\ \log m\})}$$

and

$$S(\tau(C,m)) \geq m^{\Omega(\min\{k(C),\ \log m\})}. \tag{1}$$

We begin the proof of Lemma 3.1 by describing CNFs, which form the main building block $\tau(C, m)$, and establishing their necessary properties. From now on fix a monotone circuit C in n variables $p_1, \ldots, p_n$. Let $\mathcal{A} \subseteq \{0,1\}^m$. We will call vectors from $\mathcal{A}$ (usually represented as columns) *admissible* and call a 0-1 matrix with m rows $\mathcal{A}$-*admissible* if all its columns are so. Consider the following combinatorial principle

$\mathcal{P}_{C,\mathcal{A}}$ (that may be true or false, depending on the choice of C and $\mathcal{A}$):

$\mathcal{P}_{C,\mathcal{A}}$: *every* $(m \times n)$ 0-1 $\mathcal{A}$*-admissible matrix* $A = (a_{ij})$ *contains a row* $i \in [m]$ *such that* $C(a_{i1}, a_{i2}, \ldots, a_{in}) = 1$.

Let us formulate one sufficient condition for $\mathcal{P}_{C,\mathcal{A}}$ to be true (regardless of proof complexity considerations).

DEFINITION 3.2. *$d_1(\mathcal{A})$ is the maximal d such that for every d vectors from $\mathcal{A}$ there exists a position $i \in [m]$ in which all these vectors have* 1.

$d_1(\mathcal{A})$ can also be easily characterized in terms of minimum covers. Namely, if we associate with every

$$\begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \in \mathcal{A}$$

the subset $\{\, i \in [m] \mid a_i = 0 \,\}$ of $[m]$, then $d_1(\mathcal{A}) + 1$ is exactly the minimal number of such sets needed to cover the whole $[m]$.

LEMMA 3.3. *If $k(C) \le d_1(\mathcal{A})$, then $\mathcal{P}_{C,\mathcal{A}}$ is true.*

Proof. Let A be an $(m \times n)$ 0-1 $\mathcal{A}$-admissible matrix. Let $a = (a_1, \ldots, a_n)$ be such that $C(a_1, \ldots, a_n) = 1$ and $k(a) = k(C)$. Let

$$\mathcal{A}_0 \stackrel{\text{def}}{=} \left\{ \left(\begin{array}{c} a_{1j} \\ \vdots \\ a_{mj} \end{array} \right) \middle| \; a_j = 1 \right\}$$

be the set of all columns in A corresponding to those positions j for which $a_j = 1$. Since $|\mathcal{A}_0| \le k(a) = k(C) \le d_1(\mathcal{A})$, there exists $i \in [m]$ such that $a_{ij} = 1$ whenever $a_j = 1$. This means $a_{ij} \ge a_j$ for all $j \in [n]$ and implies $C(a_{i1}, \ldots, a_{in}) = 1$ since C is monotone. □

The proof of Lemma 3.3 suggests that if C and $\mathcal{A}$ with the property $k(C) \le d_1(\mathcal{A})$ are "generic enough," then the optimal propositional proof of the principle $\mathcal{P}_{C,\mathcal{A}}$ should exhaustively search through all $|\mathcal{A}|^{k(C)}$ possible placements of admissible vectors to the columns $\{\, j \mid a_j = 1 \,\}$ and thus have size roughly $|\mathcal{A}|^{k(C)}$. Our task is to find an encoding of (the negation of) $\mathcal{P}_{C,\mathcal{A}}$ as a CNF so that we can prove tight upper and lower bounds on $S_T(\tau(C, \mathcal{A}))$ and $S(\tau(C, \mathcal{A}))$ of (roughly) this order. This encoding is somewhat technical and involves several auxiliary functions (see Definition 3.4 below). In order to convey why we need all of these, let us briefly discuss two "naive" attempts at a simpler proof.

Attempt 1 (no encoding at all). Suppose that we simply enumerate elements of $\mathcal{A}$ by binary strings of length $\log |\mathcal{A}|$ and introduce propositional variables expressing their bits. The main problem with this encoding is that it does not behave well with respect to (random) restrictions. The standard width-reducing argument from [8] that we use in part (c) of Lemma 3.8 below assumes a "reasonably uniform" distribution on the set of those restrictions that "reasonably preserve" the complexity of the tautology. But with the straightforward encoding, any restriction of propositional variables used for enumerating the set $\mathcal{A}$ results in shrinking this set and completely destroys its useful properties.

We circumvent this in a standard way by using "excessive encodings" $F_1, \ldots, F_n : \{0,1\}^s \longrightarrow \mathcal{A}$, where $F_i(x_1, \ldots, x_s)$ are surjective and remain so after restricting not too many variables (Definition 3.5). It is worth noting that even if we may have assumed in our definition of $\tau(C, \mathcal{A})$ that $F_1 = F_2 = \cdots = F_n$, this property will *not*

be invariant under restrictions (and this is why it is more convenient not to make this assumption).

Attempt 2 (same encoding for $\mathcal{A}$ and C). The naive encoding of the circuit C (that is, by propositional variables z_{iv} encoding the intermediate result of the computation by the circuit C at the gate v when its input is the ith row of A) suffers from the same drawback as above, which is that we do not want the values of z_{iv} to be exposed by a random restriction. But why do we not apply to the variables z_{iv} just the same excessive encodings we used above for the elements of $\mathcal{A}$?

It turns out that with this "lighter" version our lower bounds already go through, and the upper bound holds for *general* resolution (in particular, the reader interested in only this case can safely assume this simplification). In the tree-like case, however, the upper bound becomes problematic. Namely, when formalizing the proof of Lemma 3.3, we need to prove the fact $C(a_{i_1}, \dots, a_{i_n}) = 1$, and the natural way of doing this in *tree-like* resolution assumes full access to clauses of the form $(z_{i,v_1} \wedge \dots \wedge z_{i,v_\mu} \supset z_{i,v})$ (cf. the proof of part (a) of Lemma 3.8). This is not a problem if the variables $z_{i,v}$ are not encoded, but if we encode them in a nontrivial way, then we no longer will have a resolution proof.

In order to balance between these two conflicting requirements, we introduce a more sophisticated encoding scheme that intuitively looks as follows. Imagine that we have many independent copies $C_1, \dots, C_r$ of the circuit C; indices $c \in \{1, 2, \dots, r\}$ will be called *controls*. The (unencoded!) variables $z^c_{i,v}$ will again express the protocol of computing the value $C_c(a_{i_1}, \dots, a_{i_n})$. But for every individual row $i \in [m]$, our axioms will require this protocol to be valid only for *one* of these r circuits (say, C_{c_i}), and the values c_i *are* excessively encoded by surjective mappings $f_1, \dots, f_m : \{0,1\}^s \longrightarrow [r]$ in the same way as we did with the elements of $\mathcal{A}$.

In order to not obstruct the proof with irrelevant details, we will define our CNFs $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ and establish their necessary properties in a situation which is more general than what will be actually needed for completing the proof of Lemma 3.1. If the reader prefers, he/she may think during the course of the proof that $\mathcal{A}$ is an arbitrary set of vectors such that $d_1(\mathcal{A}) \geq \Omega(\log m)$ and (see Definition 3.6) $d_0(\mathcal{A}) \geq \Omega(\log m)$. Furthermore, $r = \log m$, $s = O(\log m)$, and F_j, f_i will be $(\log m)$-surjective in the sense of Definition 3.5.

DEFINITION 3.4. *Let $C(p_1, \dots, p_n)$ be a monotone circuit, $\mathcal{A} \subseteq \{0,1\}^m$ be a set of vectors, and $F_1, \dots, F_n : \{0,1\}^s \longrightarrow \mathcal{A}$, $f_1, \dots, f_m : \{0,1\}^s \longrightarrow [r]$ be surjective functions, where f_is are possibly partial. For every $j \in [n]$ and $\nu \in [s]$ we introduce a propositional variable x^ν_j, for every $i \in [m]$ and $\nu \in [s]$ introduce a variable y^ν_i, and for every $i \in [m]$, every $c \in [r]$ (elements of this set will be sometimes referred to as* controls*), and every vertex v of the circuit C introduce a variable z^c_{iv}.*

For $j \in [n]$ and $\vec{a} \in \mathcal{A}$, let us denote by $[Column_j = \vec{a}]$ the predicate $F_j(x^1_j, \dots, x^s_j) = \vec{a}$. Likewise, for $i \in [m]$ and $c \in [r]$, let $[Control_i = c]$ denote the predicate "$f_i(y^1_i, \dots, y^s_i)$ is defined and $f_i(y^1_i, \dots, y^s_i) = c$."

The CNF $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ consists of all clauses that result from the expansion of the following Boolean predicates as CNFs:

$$(y^1_i, \dots, y^s_i) \in dom(f_i) \text{ for all } i \in [m]; \tag{2}$$

$$\left.\begin{array}{c} ([Column_j = \vec{a}] \wedge [Control_i = c]) \supset z^c_{i,p_j} \\ \text{for all } \vec{a} \in \mathcal{A},\ i \in [m] \text{ such that } a_i = 1 \text{ and all } j \in [n],\ c \in [r]; \end{array}\right\} \tag{3}$$

$$\left.\begin{array}{c} \big([\mathit{Control}_i = c] \wedge (z^c_{i,v'} * z^c_{i,v''})\big) \supset z^c_{iv} \\ \text{for all } i \in [m],\ c \in [r] \text{ and all internal nodes } v \\ \text{corresponding to the instruction } v \leftarrow v' * v'',\ * \in \{\wedge, \vee\}; \end{array}\right\} \tag{4}$$

$$[\mathit{Control}_i = c] \supset \bar{z}_{i,v_{\mathrm{fin}}},\ \text{where } v_{\mathrm{fin}} \text{ is the output node of } C. \tag{5}$$

It is easy to see that $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ is unsatisfiable (for arbitrary surjective $\vec{F}, \vec{f}$) iff $P_{C,\mathcal{A}}$ is true. Also, as we already mentioned, the only thing we need from $\vec{F}, \vec{f}$ is that they remain surjective after restricting a few variables.

DEFINITION 3.5. *We say that an onto (possibly partial) function $g : \{0,1\}^s \longrightarrow D$ is* r-surjective *if for any restriction ρ with $|\rho| \le r$ the function $g|_\rho$ is still onto.*

Finally, for the lower bound we need a notion dual to $d_1(\mathcal{A})$.

DEFINITION 3.6. *$d_0(\mathcal{A})$ is the maximal d such that for every d positions $i_1, \ldots, i_d \in [m]$ there exists $\vec{a} \in \mathcal{A}$ such that $a_{i_1} = \cdots = a_{i_d} = 0$.*

Now we are ready to formulate our main technical lemma that provides upper and lower bounds on the size of optimal resolution refutations of $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$. Like many similar proofs in the area, the lower bound is naturally split into two fairly independent parts. The first part provides lower bounds on $w(\tau(C, \mathcal{A}, \vec{F}, \vec{f}) \vdash \emptyset)$, but for technical reasons we need a slightly stronger statement based on a modified notion of width.

DEFINITION 3.7. *For a clause D in the variables of the CNF $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$, let $w_x(D), w_y(D)$, and $w_c(D)$ ($c \in [r]$) be the numbers of x-variables, y-variables, and z-variables of the form $z^c_{i,v}$, respectively, appearing in D. We define the* controlled width $\widetilde{w}(D)$ *as*

$$\widetilde{w}(D) \stackrel{\mathrm{def}}{=} w_x(D) + w_y(D) + r \cdot \min_{c \in [r]} w_c(D).$$

The minimal controlled width $\widetilde{w}(\tau(C, \mathcal{A}, \vec{F}, \vec{f}) \vdash \emptyset)$ *is defined similarly to the minimal refutation width.*

Clearly, $\widetilde{w}(D) \le w(D)$ for any clause D, and thus $\widetilde{w}(\tau(C, \mathcal{A}, \vec{F}, \vec{f}) \vdash \emptyset) \le w(\tau(C, \mathcal{A}, \vec{F}, \vec{f}) \vdash \emptyset)$.

LEMMA 3.8. *Let C be a monotone circuit in n variables, let $\mathcal{A} \subseteq \{0,1\}^m$, and let $F_1, \ldots, F_n : \{0,1\}^s \longrightarrow \mathcal{A}$, $f_1, \ldots, f_m : \{0,1\}^s \longrightarrow [r]$ be r-surjective functions, where the f_i's are possibly partial and m, r, s are arbitrary integer parameters. Then the following bounds hold:*

(a) (*cf. Lemma* 3.3). *If $k(C) \le d_1(\mathcal{A})$, then $S_T(\tau(C, \mathcal{A}, \vec{F}, \vec{f})) \le O(|C| \cdot 2^{s(k(C)+1)})$.*

(b) $\widetilde{w}(\tau(C, \mathcal{A}, \vec{F}, \vec{f}) \vdash \emptyset) \ge \frac{r}{2} \cdot \min\{k(C), d_0(\mathcal{A})\}$.

(c) $S(\tau(C, \mathcal{A}, \vec{F}, \vec{f})) \ge \exp(\Omega(\frac{r^2}{s} \cdot \min\{k(C), d_0(\mathcal{A})\}))$.

Proof of Lemma 3.8.

Part (a). We show this part by formalizing the proof of Lemma 3.3. Let $k \stackrel{\mathrm{def}}{=} k(C)$ and $a_1, \ldots, a_n$ be such that $C(a_1, \ldots, a_n) = 1$ and $k(a) = k$. Assume for simplicity

that $a_1 = \cdots = a_k = 1,\ a_{k+1} = \cdots = a_n = 0$. Fix arbitrary admissible vectors

$$\vec{a}_1 \stackrel{\text{def}}{=} \begin{pmatrix} a_{i1} \\ \vdots \\ a_{m1} \end{pmatrix}, \ldots, \vec{a}_k \stackrel{\text{def}}{=} \begin{pmatrix} a_{ik} \\ \vdots \\ a_{mk} \end{pmatrix}$$

and, using the inequality $d_1(\mathcal{A}) \geq k$, pick up an arbitrary $i \in [m]$ (depending in general on $\vec{a}_1, \ldots, \vec{a}_k$) such that $a_{i1} = a_{i2} = \cdots = a_{ik} = 1$. We want to infer from $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ all clauses in the CNF expansion of

$$[\mathrm{Column}_1 \neq \vec{a}_1] \vee \cdots \vee [\mathrm{Column}_k \neq \vec{a}_k] \vee [\mathrm{Control}_i \neq c] \tag{6}$$

for all $c \in [r]$. It is fairly obvious how to do this efficiently in general resolution. Namely, let V be the set of all nodes of the circuit C that are evaluated to 1 by the assignment $(1^k, 0^{n-k})$. Then we may proceed by induction on the construction of C and subsequently infer

$$([\mathrm{Column}_1 = \vec{a}_1] \wedge \cdots \wedge [\mathrm{Column}_k = \vec{a}_k] \wedge [\mathrm{Control}_i = c]) \supset z^c_{iv}$$

for all $v \in V$ until we reach v_{fin}.

In order to get a *tree-like* proof, however, we should employ a dual (top-down) strategy. Namely, enumerate the set V in some order which is consistent with the topology of C: $V = \langle v_1 = p_1, v_2 = p_2, \ldots, v_k = p_k, v_{k+1}, v_{k+2}, \ldots, v_t = v_{\mathrm{fin}} \rangle$; all wires between vertices in V go from left to right. Then, by a reverse induction on $\mu = t, t-1, \ldots, k$ we infer (all clauses in the CNF expansion of) $[\mathrm{Control}_i = c] \supset (\bar{z}^c_{i,v_1} \vee \cdots \vee \bar{z}^c_{i,v_\mu})$. For $\mu = t$ this is (a weakening of) (5), and for the inductive step we resolve with the appropriate axiom in (4). When we descend to $[\mathrm{Control}_i = c] \supset \left(\bar{z}^c_{i,p_1} \vee \cdots \vee \bar{z}^c_{i,p_k}\right)$, we consecutively resolve with the corresponding axioms (3) to get rid of $\bar{z}^c_{i,p_j}$ and arrive at (6). Clearly, this resolution inference of every individual clause in (6) is tree-like and has size $O(|C|)$.

Finally, for every $i \in [m]$, every clause in the variables $\{\, y^\nu_i \mid 1 \leq \nu \leq s \,\}$ appears in one of the CNFs resulting from the predicate $\{\, [\mathrm{Control}_i \neq c] \mid c \in [r] \,\}$ or the predicates in (2), and every clause in the variables $\left\{\, x^\nu_j \mid 1 \leq \nu \leq s \,\right\}$ appears in one of $[\mathrm{Column}_j \neq \vec{a}_j]$. This gives us an obvious tree-like refutation of the set of clauses (2), (6) that has size $O(2^{s(k+1)})$. Combining this refutation with previously constructed inferences of (6) from $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$, we get the desired upper bound.

Part (b). We follow the general strategy proposed in [10]. Note that every one of the axioms (2)–(5) "belongs" to a uniquely defined row; let Row_i be the set of axioms in $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ that correspond to the row i. For a clause D, let $\mu(D)$ be the smallest cardinality of $I \subseteq [m]$ such that $\cup\{\, \mathrm{Row}_i \mid i \in I \,\}$ (semantically) implies D. $\mu(D)$ is subadditive, that is, $\mu(D) \leq \mu(D_1) + \mu(D_2)$ whenever D is obtained from D_1, D_2 via a single application of the resolution rule. It is also obvious that $\mu(A) = 1$ for any axiom $A \in \tau(C, \mathcal{A}, \vec{F}, \vec{f})$.

We claim that $\mu(\emptyset) > d_0(\mathcal{A})$. Indeed, fix any $I \subseteq [m]$ with $|I| \leq d_0(\mathcal{A})$. We need to construct an assignment that satisfies all axioms in $\cup\{\, \mathrm{Row}_i \mid i \in I \,\}$. Pick $\vec{a}$ accordingly to Definition 3.6 in such a way that for all $i \in I (a_i = 0)$. Assign every x^ν_j to α^ν_j, where $\alpha^1_j, \ldots, \alpha^s_j$ is an arbitrary vector such that $F_j(\alpha^1_j, \ldots, \alpha^s_j) = \vec{a}$; assign y^ν_i in an arbitrary way with the only requirement that they satisfy (2), and assign all z-variables to 0. This assignment will satisfy all axioms in $\cup\{\, \mathrm{Row}_i \mid i \in I \,\}$, which proves $\mu(\emptyset) > d_0(\mathcal{A})$.

Thus, any resolution refutation of $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ must contain a clause D with $\frac{1}{2}d_0(\mathcal{A}) \le \mu(D) \le d_0(\mathcal{A})$, and we need only show that this implies $\widetilde{w}(D) \ge \frac{r}{2} \cdot \min\{k(C), d_0(\mathcal{A})\}$. Fix $I \subseteq [m]$ such that $\cup\{\mathrm{Row}_i \mid i \in I\}$ semantically implies D and $|I|$ is minimal with this property; $\frac{1}{2}d_0(\mathcal{A}) \le |I| \le d_0(\mathcal{A})$.

If for every $i \in I$ at least one of the following two events is true, then we are done:

1. the clause D contains at least r variables among $\{y_i^\nu \mid \nu \in [s]\}$;
2. for every control $c \in [r]$ the clause D contains at least one variable among $\{z_{iv}^c \mid v \text{ is a node}\}$.

Indeed, if h is the number of indices i for which 1 is true, then $w_y(D) \ge rh$. For those $i \in I$ for which 1 does not hold, we apply 2 to conclude $\min_{c\in[r]} w_c(D) \ge (|I| - h)$, and altogether we have $\widetilde{w}(D) \ge w_y(D) + r \cdot \min_{c\in[r]} w_c(D) \ge r \cdot |I| \ge \frac{r}{2}d_0(\mathcal{A})$.

Thus, suppose that for some $i_0 \in I$ neither of these two is true. In particular, there exists a control $c_0 \in [r]$ such that no variable of the form $z_{i_0,v}^{c_0}$ appears in D. Fix an arbitrary assignment α that satisfies all axioms in $\{\mathrm{Row}_i \mid i \in I \setminus \{i_0\}\}$ and falsifies D (such an assignment exists due to the minimality of $|I|$).

Let J_0 consist of those $j \in [n]$ for which the clause D contains at least r variables from $\{x_j^\nu \mid \nu \in [s]\}$. If $|J_0| \ge k(C)$, we are also done. If this is not the case, we will show how to alter the assignment α so that it will satisfy all axioms in $\cup\{\mathrm{Row}_i \mid i \in I\}$ (including Row_{i_0}) but still will falsify D, and this will give us the contradiction.

According to Definition 3.6, there exists $\vec{a} \in \mathcal{A}$ such that $a_i = 0$ for all $i \in I$. We alter α as follows.

Step 1. Using that F_j is r-surjective, we change for every $j \notin J_0$ the values of the variables $\{x_j^\nu \mid \nu \in [s]\}$ not appearing in D in such a way that $F_j(x_j^1, \dots, x_j^s) = \vec{a}$.

Step 2. Using the fact that f_{i_0} is r-surjective, we change the values of variables $\{y_{i_0}^\nu \mid \nu \in [s]\}$ not appearing in D in such a way that $f_{i_0}(y_{i_0}^1, \dots, y_{i_0}^s) = c_0$. Finally, we reassign every $z_{i_0,v}^{c_0}$ to the value computed by the node v on the characteristic vector of the set J_0. Note that $z_{i_0,p_j}^{c_0}$ is set to 1 for $j \in J_0$, whereas $z_{i_0,v_{\mathrm{fin}}}^{c_0}$ is set to 0 since $|J_0| < k(C)$.

We claim that this altered assignment α' satisfies all axioms in $\cup\{\mathrm{Row}_i \mid i \in I\}$. Indeed, we made sure in our construction that it satisfies all axioms in Row_{i_0} of types (2), (4), (5), and for $i \in I \setminus \{i_0\}$ axioms of these types are satisfied since we have not touched any variable appearing in them. Thus, we have only to check the axiom (3). If $j \in J_0$ and $i \ne i_0$, this axiom has not been touched, and if $j \notin J_0$, it becomes satisfied because of the first step in our construction of α', and due to the condition $a_i = 0$ ($i \in I$). Finally, if $j \in J_0$ and $i = i_0$, the axiom (3) gets satisfied during the second step (in which we set $z_{i_0,p_j}^{c_0}$ to 1).

But α' also falsifies D since we have not touched variables appearing in it. This contradiction with the fact that $\{\mathrm{Row}_i \mid i \in I\}$ implies that D completes the proof of part (b).

Part (c). We apply the standard argument of width-reducing restrictions (cf. [8]). For doing this we observe that the CNFs of the form $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ behave well with respect to certain restrictions. Namely, let $d \le r$ and $R \subseteq [r]$ be an arbitrary set of controls. Denote by $\mathcal{R}_{d,R}$ the set of all restrictions that arbitrarily assign to a Boolean value d variables in every one of the groups $\{x_j^\nu \mid \nu \in [s]\}, \{y_i^\nu \mid \nu \in [s]\}$ with $j \in [n], i \in [m]$ as well as all the variables z_{iv}^c with $c \notin R$. Then it is easy to see that for $\rho \in \mathcal{R}_{d,R}$, every nontrivial clause in $\tau(C, \mathcal{A}, \vec{F}, \vec{f})|_\rho$, after a suitable re-enumeration of variables and controls, contains a subclause from $\tau(C, \mathcal{A}, \vec{F}|_\rho, (\vec{f}|_\rho)|_R)$ (the *partial* function $(f_i|_\rho)|_R$ is obtained from $f_i|_\rho$ by restricting its domain to $\{y_i \mid f_i|_\rho(y_i) \in R\}$ and range to R).

Pick now $\boldsymbol{\rho}$ uniformly at random from $\mathcal{R}_{r/2,\boldsymbol{R}}$, where $\boldsymbol{R}$ is picked at random from $[r]^{r/2}$. Then $F_j|_{\boldsymbol{\rho}}, (f_i|_{\boldsymbol{\rho}})|_{\boldsymbol{R}}$ will be $(r/2)$-surjective. Therefore, by the already proven part (b), for every refutation P of $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$, $\boldsymbol{\rho}(P)$ will contain a clause of controlled width

$$\Omega\left(r \cdot \min\{k(C), d_0(\mathcal{A})\}\right) \tag{7}$$

with probability 1.

It is easy to see, however, that every clause D whose controlled width $\widetilde{w}(D)$ is that large is killed (that is, set to 1 and hence removed from the proof) by $\boldsymbol{\rho}$ with probability

$$1 - \exp\left(-\Omega\left(\frac{r^2}{s} \cdot \min\{k(C), d_0(\mathcal{A})\}\right)\right).$$

Indeed, according to Definition 3.7, either $w_x(D)$ or $w_y(D)$ or $r \cdot \min_{c\in[r]} w_c(D)$ is bounded from below by a quantity of the form (7). Let $w_{x,j}(D)$ be the number of variables in the corresponding group that appear in D so that $w_x(D) = \sum_{j\in[n]} w_{x,j}(D)$. Then the probability that D is not killed by variables in the jth group is $\exp\big(-\Omega\big(\frac{r\cdot w_{x,j}(D)}{s}\big)\big)$, and these events are independent so the probabilities of survival multiply to $\exp\big(-\Omega\big(\frac{r\cdot w_{x,j}(D)}{s}\big)\big)$. The case when $w_y(D)$ is large is treated in exactly the same way, and the case when $r \cdot \min_{c\in[r]} w_c(D)$ is large is even simpler since for *every* choice of $\boldsymbol{R}$, the number of assigned z-variables is at least $\frac{r}{2} \cdot \min_{c\in[r]} w_c(D)$ (and $s \geq r$).

Therefore, the size of P must be at least $\exp\big(\Omega\big(\frac{r^2}{s} \cdot \min\{k(C), d_0(\mathcal{A})\}\big)\big)$ since otherwise a random restriction $\boldsymbol{\rho}$ would have killed all such clauses with nonzero probability, which is impossible.

Lemma 3.8 is completely proved. □

Proof of Lemma 3.1. Our construction of $\tau(C, m)$ proceeds in polynomial time as follows.

1. Let p be the smallest prime greater than or equal to m. Since $m \leq p \leq 2m$, both bounds in Lemma 3.1 remain unchanged if we replace m by p or vice versa. Therefore, w.l.o.g. we may assume from the beginning that m itself is a prime. Let P_m be the $(m \times m)$ 0-1 *Paley matrix* given by $a_{ij} = 1$ iff $j \neq i$ and $(j-i)$ is a quadratic residue mod m. Let $\mathcal{A} \subseteq \{0,1\}^m$ consist of all columns of P_m. Then $|\mathcal{A}| = m$ and $d_0(\mathcal{A}), d_1(\mathcal{A}) \geq \frac{1}{4}\log m$ (see, e.g., [5]).

2. Fix any $\mathbb{F}_2$-linear code $L \subseteq \{0,1\}^{h\lceil\log m\rceil}$ of dimension $\lceil\log m\rceil$ that is computable (as a language) in time $m^{O(1)}$ and has minimal distance $\geq \lceil\log m\rceil$ ($h > 0$ is an absolute constant). Consider the linear mapping $G : \{0,1\}^{h\lceil\log m\rceil} \longrightarrow \{0,1\}^{\lceil\log m\rceil}$ dual to the inclusion $L \to \{0,1\}^{h\lceil\log m\rceil}$ (that is, we fix in L an arbitrary basis $x_1, \ldots, x_{\lceil\log m\rceil}$ and let $G(y) \stackrel{\text{def}}{=} (\langle x_1, y\rangle, \ldots, \langle x_{\lceil\log m\rceil}, y\rangle)$). By linear duality, the fact that L has minimal distance $\geq \lceil\log m\rceil$ is equivalent to $\lceil\log m\rceil$-surjectivity of G. Set $r \stackrel{\text{def}}{=} \lceil\log m\rceil$ and $s \stackrel{\text{def}}{=} h\lceil\log m\rceil$. Consider arbitrary (polynomial-time computable) surjective mappings $\Pi : \{0,1\}^{\lceil\log m\rceil} \longrightarrow \mathcal{A}$, $\pi : \{0,1\}^{\lceil\log m\rceil} \longrightarrow [r]$ and let $F_j \stackrel{\text{def}}{=} \Pi \cdot G, f_i \stackrel{\text{def}}{=} \pi \cdot G$ for all i, j.

3. Construct $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$. Note that the size of this CNF is polynomial in $|C|, m, 2^s$, which is polynomial in $|C|, m$ due to our choice of parameters.

At this point, Lemma 3.8(c) already implies (1) for $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$. The only remaining problem is that a priori we do not have the condition $k(C) \leq d_1(\mathcal{A})$ needed for part (a) of Lemma 3.8. We circumvent this by a trick similar to one used in [3].

Namely, let τ_m be a fixed unsatisfiable (polynomial-time constructible) CNF with $S(\tau_m), S_T(\tau_m) = m^{\theta(\log m)}$ (for example, one can take a Tseitin tautology with $\theta((\log m)^2)$ variables) and such that its set of variables is disjoint from the set of variables of $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$. We finally set $\tau(C, m) \stackrel{\text{def}}{=} \tau(C, \mathcal{A}, \vec{F}, \vec{f}) \wedge \tau_m$.

Since both $\tau(C, \mathcal{A}, \vec{F}, \vec{f})$ and τ_m satisfy the lower bound (1), the weak feasible disjunction property (see, e.g., [21] and the literature cited therein) for resolution implies that $\tau(C, m)$ satisfies this bound, too. If $k(C) \le \frac{1}{4}\log m$ then, since $d_1(\mathcal{A}) \ge \frac{1}{4}\log m$, we can apply Lemma 3.8(a) to get the required upper bound $S_T(\tau(C, m)) \le |C| \cdot m^{O(k(C))}$. If, on the other hand, $k(C) \ge \frac{1}{4}\log m$, the required upper bound $S_T(\tau(C, m)) \le m^{O(\log m)}$ simply follows from the upper bound for τ_m. This completes the proof of Lemma 3.1. □

4. Self-improvement. In this section we complete the proof of Theorems 2.5 and 2.7 by combining Lemma 3.1 with a (nontrivial) self-improvement technique. First, we need to get rid of the dummy parameter m in the statement of Lemma 3.1.

LEMMA 4.1. *If either resolution or tree-like resolution is automatizable, then there exists an absolute constant $h > 1$ and an algorithm Φ working on pairs $\langle C, k\rangle$, where C is a monotone circuit and k is an integer such that*

1. *the running time of $\Phi(\langle C, k\rangle)$ is at most $\exp(O(k^2)) \cdot |C|^{O(1)}$;*
2. *if $k(C) \le k$, then $\Phi(\langle C, k\rangle) = 1$;*
3. *if $k(C) \ge hk$, then $\Phi(\langle C, k\rangle) = 0$.*

Proof. Combining the reduction in Lemma 3.1 with an automatizing algorithm for either resolution or tree-like resolution, we get an integer-valued function $S(C, m)$ computable in time $\left(|C| \cdot m^{\min\{k(C),\ \log m\}}\right)^{h_0}$ and such that

$$m^{\epsilon \cdot \min\{k(C),\ \log m\}} \le S(C, m) \le \left(|C| \cdot m^{\min\{k(C),\ \log m\}}\right)^{h_1}$$

for some absolute constants $\epsilon, h_0, h_1 > 0$. Set the constant h in the statement in such a way that

$$h^2 > \frac{h_1}{\epsilon}(h + 1). \tag{8}$$

Our algorithm Φ works as follows. We set

$$m \stackrel{\text{def}}{=} 2^{h \cdot \max\{k,\ \log |C|/k\}}.$$

Φ simulates $\left(|C| \cdot m^k\right)^{h_0}$ steps in the computation of $S(C, m)$, outputs 1 if the computation halts within this time, and its result $S(C, m)$ satisfies the inequality $S(C, m) \le \left(|C| \cdot m^k\right)^{h_1}$ and outputs 0 in all other cases.

Our choice of m ensures that $m^k \le \exp(O(k^2)) \cdot |C|^{O(1)}$, which implies property 1.

Since $\log m \ge hk \ge k$, under the assumption $k(C) \le k$ the limitations we have imposed on the running time and the output value of the algorithm Φ are less stringent than the bounds known of the underlying algorithm computing $S(C, m)$. This observation implies property 2.

Finally, using again the inequality $\log m \ge hk$, $k(C) \ge hk$ implies that $S(C, m) \ge m^{\epsilon hk}$, and elementary calculations show that, along with (8), this gives us $S(C, m) > \left(|C| \cdot m^k\right)^{h_1}$. Thus, if $k(C) \ge hk$, the algorithm Φ outputs the value 0.

Lemma 4.1 is proved. □

Proof of Theorem 2.5. First, we extract from Lemma 4.1 an algorithm which meets the bound on the running time and achieves the ratio of approximation h. For that we consecutively run the algorithm Φ from that lemma on the inputs $\langle C, 1\rangle, \ldots, \langle C, k\rangle, \ldots$ and output the first value k for which we get the answer 0.

Combining this algorithm with the self-improving reduction from Proposition 2.4 (for $d = \lceil \frac{1}{\epsilon} \ln h \rceil$), we get an approximating algorithm with the required properties. □

In the established terminology, what we have seen so far under the assumption of automatizability of (tree-like) resolution is a polynomial-time approximation scheme (PTAS) for MMCSA in the context of parameterized complexity (the latter referring to the term $\exp\left(k(C)^{O(1)}\right)$ in the bound on the running time). Unfortunately, our PTAS is not efficient (see the discussion in section 2.2), as the reduction from Proposition 2.4 blows up the size of the circuit. The task of converting an arbitrary PTAS into an EPTAS seems to be hopeless in general even in the context of parameterized complexity (where it appears to be easier). We nonetheless can perform it (in the latter context) for the specific problem MMCSA using a much trickier self-improvement construction. This construction (that completes the proof of our main theorem, Theorem 2.7) might be of independent interest, and its idea is roughly as follows.

We need to improve the approximation ratio of the algorithm Φ in Theorem 2.5 from (say) 2 to (say) $(1 + \frac{1}{\sqrt{k}})$, and the straightforward way of doing this is by iteratively applying Proposition 2.4 (say) $d = \sqrt{k}$ times. The corresponding reduction will map any circuit $C(x_1, \ldots, x_n)$ into an n-ary tree of C-gates, and of depth d, and the resulting increase in size is too costly to us. What we basically show is that we can circumvent this by replacing the tree with a *random* directed acyclic graph (DAG) of the same depth d and of width polynomial in n.

Proof of Theorem 2.7. Let C be a monotone circuit in n variables and k be an integer such that

$$10 \le k \le \epsilon(\log n / \log\log n)^2 \tag{9}$$

for a sufficiently small constant $\epsilon > 0$ (we will remark later how to get rid of this condition). Our goal is to construct in polynomial time a randomized monotone circuit $\boldsymbol{\pi}(C, k)$ and an integer $\alpha(k)$ (deterministically depending only on k) such that α is recursive and the following conditions hold:

$$k(C) \le k \implies \mathbf{P}[k(\boldsymbol{\pi}(C,k)) \le \alpha(k)] = 1; \tag{10}$$

$$k(C) \ge k+1 \implies \mathbf{P}[k(\boldsymbol{\pi}(C,k)) \ge 2\alpha(k)] \ge 1/2. \tag{11}$$

First, we apply to C the reduction from Proposition 2.4 with $d = 2$ that maps the range $[k, k+1]$ to $[k^2, k^2 + 2k + 1]$. Redenoting k^2 back to k, we may assume w.l.o.g. that in (11) we have the stronger premise

$$k(C) \ge k + 2\sqrt{k} \implies \mathbf{P}[k(\boldsymbol{\pi}(C,k)) \ge 2\alpha(k)] \ge 1/2. \tag{12}$$

Now comes our main reduction. Let N, d be two parameters (to be specified later). The randomized circuit $\boldsymbol{\pi}(C, N, d)$ in (nN) variables consists of d layers. Each layer consists of N independent copies of the circuit C (see Figure 1); thus, it has (nN) inputs and N outputs. We connect input nodes at the $(i+1)$st level to output nodes at the ith level at random. Finally, we pick up an arbitrary output node at the last dth level and declare it to be the output of the whole circuit $\boldsymbol{\pi}(C, N, d)$.

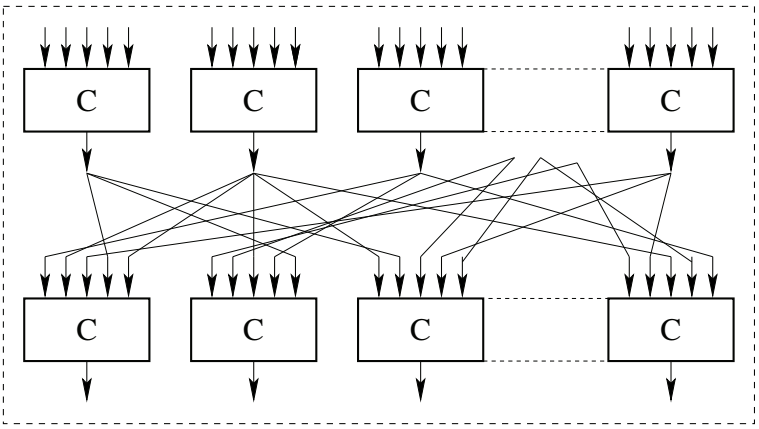


FIG. 1. *One layer of* $\boldsymbol{\pi}(C, N, d)$.

Clearly, this construction is polynomial in $|C|, N, d$. Also, an obvious induction on d shows that

$$k(\boldsymbol{\pi}(C, N, d)) \le k(C)^d \tag{13}$$

with probability 1. In order to get a lower bound on $k(\boldsymbol{\pi}(C, N, d))$, we need the following easy lemma. It is of course yet another version of the well-known fact that a random (bipartite) graph makes an extremely good expander.

LEMMA 4.2. *Let* $\boldsymbol{\chi} : [N] \times [n] \longrightarrow [N]$ *be a randomly chosen function, and let* k, a *be any parameters. Then* $\mathbf{P}\big[\exists V \in [N]^k (|\boldsymbol{\chi}(V \times [n])| \le kn - a)\big] \le N^k \cdot (\frac{4k^2n^2}{N})^a$.

Proof of Lemma 4.2. This event takes place iff there exist $V \in [N]^k$ and disjoint $D_1, \dots, D_r \subseteq V \times [n]$ such that $|D_1|, \dots, |D_r| \ge 2$, $\sum_{i=1}^r (|D_i| - 1) = a$, and $\boldsymbol{\chi}|_{D_i} =$ const for all $i \in [r]$. Since the two first properties imply $\sum_{i=1}^r |D_i| \le 2a$, the overall number of all choices of $\langle V, D_1, \dots, D_r \rangle$ does not exceed $N^k \cdot (2kn)^{2a}$. On the other hand, for every fixed choice, we have

$$\mathbf{P}[\boldsymbol{\chi}|_{D_1} = \text{const}, \dots, \boldsymbol{\chi}|_{D_r} = \text{const}] = \frac{N^r}{N^{(\sum_{i=1}^r |D_i|)}} = N^{-a}.$$

Lemma 4.2 follows. □

Now we can complete the description of our reduction. Namely, we set $N \stackrel{\text{def}}{=} n^3$, $d \stackrel{\text{def}}{=} \sqrt{k}$ and let $\boldsymbol{\pi}(C, k) = \boldsymbol{\pi}(C, n^3, \sqrt{k})$, $\alpha(k) \stackrel{\text{def}}{=} k^{\sqrt{k}}$.

Equation (10) follows from (13).

In order to check (12), denote by $\boldsymbol{\chi_i} : [N] \times [n] \longrightarrow [N]$ the function used for connecting input nodes at the $(i+1)$st level of $\boldsymbol{\pi}(C, n^3, \sqrt{k})$ to the output nodes at the ith level. Let $k_i \stackrel{\text{def}}{=} (k + \sqrt{k})^{d-i}$. Let us call $\boldsymbol{\pi}(C, N, d)$ *bad* if for at least one of these functions $\boldsymbol{\chi_i}$ there exists a set V of circuits at the $(i+1)$st level such that $|V| = k_{i+1}$ and $|\boldsymbol{\chi_i}(V \times [n])| \le k_{i+1}(n - \sqrt{k})$. Using Lemma 4.2 and (9), we get the

bound

$$\begin{aligned}\mathbf{P}[\boldsymbol{\pi}(C,N,d) \text{ is bad}] &\le \sum_{i=1}^{d-1} N^{k_{i+1}} \cdot \left(\frac{4k_{i+1}^2 n^2}{N}\right)^{\sqrt{k}\cdot k_{i+1}} \\ &= \sum_{i=1}^{d-1} \left(\frac{4k_{i+1}^2}{n^{1-3/\sqrt{k}}}\right)^{\sqrt{k}\cdot k_{i+1}} \\ &\le \sum_{i=1}^{d-1} \left(\frac{1}{3}\right)^{\sqrt{k}\cdot k_{i+1}} \le \frac{1}{2}.\end{aligned}$$

On the other hand, it is easy to see by induction on $i = d, \dots, 1$ that if $k(C) \ge k + 2\sqrt{k}$ and $\boldsymbol{\pi}(C,N,d)$ is good, then every satisfying assignment a should satisfy at least k_i output nodes at the ith level. Indeed, the base $i = d$ is obvious ($k_d = 1$). For the inductive step, assume that a satisfies the output nodes of a set V of circuits at the $(i+1)$st level, $|V| = k_{i+1}$. Then at least $(k + 2\sqrt{k}) \cdot k_{i+1}$ *input* nodes to these circuits should be satisfied. Since $\boldsymbol{\chi_i}$ is good, there are at most $\sqrt{k} \cdot k_{i+1}$ collisions between the $(k + 2\sqrt{k}) \cdot k_{i+1}$ wires leading to these input nodes from the ith level. Therefore, at least $(k + 2\sqrt{k}) \cdot k_{i+1} - \sqrt{k} \cdot k_{i+1} = k_i$ output nodes at the ith level should be satisfied.

In particular, at the first level we will have $\ge (k + \sqrt{k})^{d-1}$ satisfied circuits and $\ge (k + 2\sqrt{k}) \cdot (k + \sqrt{k})^{\sqrt{k}-1} > 2\alpha(k)$ satisfied input nodes. This completes the proof that our probabilistic reduction $\boldsymbol{\pi}(C,k)$ has the properties (10), (12) (and, as we already remarked, improving (12) to (11) takes one more easy step).

Now we finish the proof of Theorem 2.7. Suppose that either resolution or tree-like resolution is automatizable. Since WEIGHTED MONOTONE CIRCUIT SATISFIABILITY is **W[P]**-complete (see [17, Chapter 13]), we have only to show that the language $\{\, \langle C,k\rangle \mid k(C) \le k \,\}$ is in co-**FPR**. Given an input $\langle C,k\rangle$ we check condition (9). If it is violated, we apply a straightforward brute-force algorithm with running time $O(|C|\cdot n^k) \le |C|\cdot f(k)\cdot n^9$ for some recursive f. Otherwise we simply combine our probabilistic reduction $\langle \boldsymbol{\pi}, \alpha\rangle$ with the deterministic algorithm for deciding whether $k(\boldsymbol{\pi}(C,k)) \le \alpha(k)$ or $k(\boldsymbol{\pi}(C,k)) \ge 2\alpha(k)$ provided by Theorem 2.5. Theorem 2.7 is completely proved. □

5. Open problems. The main problem left open by this paper is whether general resolution is quasi-automatizable. Since the width algorithm by Ben-Sasson and Wigderson [10] finds a resolution refutation of any unsatisfiable CNF τ in time $n^{O(w(\tau\vdash\emptyset))}$, a negative solution to this problem must involve a construction of a broad and "tractable" family of CNF τ for which $S(\tau)$ is much smaller than $2^{w(\tau\vdash\emptyset)}$. Such families are not so easy to come by (e.g., our techniques involve showing the *opposite* in the proof of Lemma 3.8(c)), although some progress toward this goal was reported in [6].

As we already mentioned in section 1.1, the same paper [6] also proposed an interesting notion of weak automatizability. Namely, a proof system P is *weakly automatizable* if there exists any automatizable proof system that polynomially simulates P. Is resolution *weakly* automatizable (under any reasonable complexity assumptions in the case of a negative answer)? Paper [6] showed that this is equivalent to another important open question in proof complexity, namely, if the system $Res(2)$ has the feasible interpolation property (for definitions, see, e.g., [22]).

We were not able to derandomize the proof of Lemma 4.2. In the terminology of [1], we need explicit constructions of $(N \times N)$ 0-1 matrices that would be $(k, n, n - O(1))$-expanders for $n \geq N^{\Omega(1)}$ and an arbitrary function $k = k(N)$ tending to infinity. Explicit constructions based on Ramanujan graphs seem to give only $(k, n, n - k^{\epsilon})$-expanders for any *fixed* ϵ which is not sufficient for our purposes. Can we weaken the hardness assumption in Theorem 2.7 to $\mathbf{W[P]} \neq \mathbf{FPT}$ by an explicit construction of better expanders (or by using any other means)?

Acknowledgment. The second author is greatly indebted to all three anonymous referees of the journal version of this paper for their constructive criticism and many useful remarks and suggestions.

REFERENCES

[1] M. Alekhnovich, E. Ben-Sasson, A. A. Razborov, and A. Wigderson, *Pseudorandom generators in propositional proof complexity*, SIAM J. Comput., 34 (2004), pp. 67–88.

[2] M. Alekhnovich, M. Braverman, V. Feldman, A. R. Klivans, and T. Pitassi, *Learnability and automatizability*, in Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (Rome, Italy), 2004, pp. 621–630.

[3] M. Alekhnovich, S. Buss, S. Moran, and T. Pitassi, *Minimum propositional proof length is NP-hard to linearly approximate*, J. Symbolic Logic, 66 (2001), pp. 171–191.

[4] M. Alekhnovich and A. Razborov, *Satisfiability, branch-width and Tseitin tautologies*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (Vancouver, Canada), 2002, pp. 593–603.

[5] N. Alon, *Tools from higher algebra*, in Handbook of Combinatorics, Vol. II, Elsevier, Amsterdam, 1995, pp. 1749–1783.

[6] A. Atserias and M. Bonet, *On the automatizability of resolution and related propositional proof systems*, Inform. and Comput., 189 (2004), pp. 182–201.

[7] C. Bazgan, *Schémas d'approximation et complexité paramétrée*, Rapport de stage de DEA d'Informatique à Orsay, 1995.

[8] P. Beame and T. Pitassi, *Simplified and improved resolution lower bounds*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (Burlington, VT), 1996, pp. 274–282.

[9] P. Beame and T. Pitassi, *Propositional proof complexity: Past, present and future*, in Current Trends in Theoretical Computer Science: Entering the 21st Century, G. Paun, G. Rozenberg, and A. Salomaa, eds., World Scientific, River Edge, NJ, 2001, pp. 42–70.

[10] E. Ben-Sasson and A. Wigderson, *Short proofs are narrow-resolution made simple*, J. ACM, 48 (2001), pp. 149–169.

[11] M. Bonet, C. Domingo, R. Gavaldá, A. Maciel, and T. Pitassi, *Non-automatizability of bounded-depth Frege proofs*, Comput. Complexity, 13 (2004), pp. 47–68.

[12] M. Bonet and N. Galesi, *A study of proof search algorithms for resolution and polynomial calculus*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (New York, NY), 1999, pp. 422–431.

[13] M. L. Bonet, T. Pitassi, and R. Raz, *On interpolation and automatization for Frege systems*, SIAM J. Comput., 29 (2000), pp. 1939–1967.

[14] M. Cesati and L. Trevisan, *On the efficiency of polynomial time approximation schemes*, Inform. Process. Lett., 64 (1997), pp. 165–171.

[15] S. A. Cook and A. R. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979), pp. 36–50.

[16] I. Dinur and M. Safra, *On the hardness of approximating label-cover*, Inform. Process. Lett., 89 (2004), pp. 247–254.

[17] R. Downey and M. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[18] K. Iwama, *Complexity of finding short resolution proofs*, in Proceedings of the 22nd International Symposium on the Mathematical Foundations of Computer Science (Bratislava, 1997), Lecture Notes in Comput. Sci. 1295, P. Ružička and I. Prívara, eds., Springer-Verlag, New York, 1997, pp. 309–318.

[19] J. Krajíček, *Proof complexity*, in Proceedings of the European Congress of Mathematics (ECM), Stockholm, Sweden, June 27–July 2, 2004, A. Laptev, ed., European Mathematical Society, Zürich, 2005, pp. 221–231.

[20] J. Krajíček and P. Pudlák, *Some consequences of cryptographical conjectures for S_2^1 and EF*, Inform. and Comput., 142 (1998), pp. 82–94.
[21] P. Pudlák, *On reducibility and symmetry of disjoint NP-pairs*, Theoret. Comput. Sci., 295 (2003), pp. 323–339.
[22] N. Segerlind, *The complexity of propositional proofs*, Bull. Symbolic Logic, 13 (2007), pp. 417–481.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1364–1381

PRESERVATION UNDER EXTENSIONS ON WELL-BEHAVED FINITE STRUCTURES*

ALBERT ATSERIAS†, ANUJ DAWAR‡, AND MARTIN GROHE§

Abstract. A class of relational structures is said to have the extension preservation property if every first-order sentence that is preserved under extensions on the class is equivalent to an existential sentence. The class of all finite structures does not have the extension preservation property. We study the property on classes of finite structures that are better behaved. We show that the property holds for classes of acyclic structures, structures of bounded degree, and more generally structures that are *wide* in a sense that we will make precise. We also show that the preservation property holds for the class of structures of treewidth at most k, for any k. In contrast, we show that the property fails for the class of planar graphs.

Key words. finite model theory, first-order logic, bounded treewidth, planar graphs, Gaifman locality

AMS subject classifications. 03C13, 03C40, 03C52, 68Q19

DOI. 10.1137/060658709

1. Introduction.

The subject of model theory is concerned with the relationship between syntactic and semantic properties of logic. Among classical results in the subject are preservation theorems which relate syntactic restrictions on first-order logic with structural properties of the classes of structures defined. A key example is the Łoś–Tarski theorem, which asserts that a first-order formula is preserved under extensions on all structures if and only if it is logically equivalent to an existential formula (see [13]). One direction of this result is easy, namely, that any formula that is purely existential is preserved under extensions, and this holds for any class of structures. The other direction, going from the semantic restriction to the syntactic restriction, makes key use of the compactness of first-order logic and hence of infinite structures.

In the early development of finite model theory, when it was realized that finite structures are the ones that are interesting from the point of view of studying computation, it was observed that most classical preservation theorems from model theory fail when only finite structures are allowed. In particular, the Łoś–Tarski theorem fails on finite structures [16, 12]. These results suggest that the class of finite structures is not well behaved from the point of view of model theory. However, when one considers the computational structures that arise in practice and are used as interpretations for logical languages (for instance, program models interpreting specifications or databases interpreting queries), in many cases they are not only finite but also satisfy other structural restrictions as well. This motivates the study of not just the

*Received by the editors May 3, 2006; accepted for publication (in revised form) March 27, 2008; published electronically August 27, 2008. A preliminary short version of this paper appeared in *Proceedings of the* 32*nd International Conference on Automata, Languages, and Programming (ICALP)*, Lecture Notes in Comput. Sci. 3580, Springer-Verlag, New York, 2005, pp. 1437–1449.
http://www.siam.org/journals/sicomp/38-4/65870.html

†Departament de Llenguatges i Sistemes Informatics, Universitat Politècnica de Catalunya, Barcelona, Spain (atserias@lsi.upc.edu). This author was supported in part by CICYT (grant TIN2004-04343) and by the European Commission through the RTN COMBSTRU (grant HPRN-CT2002-00278).

‡Computing Laboratory, University of Cambridge, Cambridge, UK (anuj.dawar@cl.cam.ac.uk).

§Institut fur Informatik, Humboldt Universität zu Berlin, Berlin, Germany (grohe@informatik.hu-berlin.de).

class of finite structures, but that of well-behaved subclasses of this class as well. Note that classical model theory, in most of its more advanced parts, also considers restricted classes of structures such as stable, simple, and o-minimal structures, and specific structures that are of interest in other areas of mathematics.

There are certain restrictions on finite structures that have proved especially useful in modern graph structure theory and also from an algorithmic point of view. For instance, many intractable computational problems become tractable when restricted to planar graphs or structures of bounded treewidth [4]. This is also the case in relation to evaluation of logical formulas [9]. A common generalization of classes of bounded treewidth and planar graphs are classes of structures that exclude a minor, which have also been extensively studied.

A study of preservation properties for such restricted classes of finite structures was initiated in [1]. There, the focus was on the homomorphism preservation theorem, whose status on the class of finite structures was open. It was shown that this preservation property holds for any class of structures of bounded degree or treewidth or that excludes some minor (and has certain other closure properties). In the present paper, we investigate the Łoś–Tarski extension preservation property on these classes of finite structures. Note that the failure of the property on the class of all finite structures does not imply its failure on subclasses. If one considers the nontrivial direction of the preservation theorem on a class $\mathcal{C}$, it says that any sentence φ that is preserved under extensions *on* $\mathcal{C}$ is equivalent *on* $\mathcal{C}$ to an existential sentence. Thus, restricting to a subclass $\mathcal{C}'$ of $\mathcal{C}$ weakens both the hypothesis and the conclusion of the statement.

We show that the extension preservation theorem holds for any class of finite structures closed under substructures and disjoint unions that is also *wide* in the sense that any sufficiently large structure in the class contains a large number of elements that are far apart. This includes, for instance, any class of structures of bounded degree. While classes of structures of bounded treewidth are not wide, they are nearly so in that they can be made wide by removing a small number of elements. We use this property and show that it implies the extension preservation theorem for the class $\mathcal{T}_k$—the class of structures of treewidth k or less (note that this is not as general as saying that the property holds for all classes of bounded treewidth). Finally, although all classes defined by excluded minors are known to be *almost wide* in the same sense as $\mathcal{T}_k$ is, we show that the construction does not extend to them. We provide a counterexample to the extension preservation property for the class of planar graphs and, indeed, even for the class of planar graphs of treewidth at most four. This contrasts with the results obtained for the homomorphism preservation property in [1] as this property was shown to hold for all classes excluding a graph minor and closed under substructures and disjoint unions.

The main methodology in establishing the preservation property for a class of structures $\mathcal{C}$ is to show an upper bound on the size of a minimal model of a first-order sentence φ that is preserved under extensions on $\mathcal{C}$. The way we do this is to show that for any sufficiently large model $\mathbf{A}$ of φ, there is a proper substructure of $\mathbf{A}$ and an extension of $\mathbf{A}$ that cannot be distinguished by φ. In section 3 we establish this for the relatively simple case of acyclic structures by means of a Hanf locality argument. Section 4 contains the main combinatorial argument for wide structures which uses Gaifman locality and an iterated construction of the substructure of $\mathbf{A}$. In section 5, the combinatorial argument is adapted to the classes $\mathcal{T}_k$. Finally, in section 6 we discuss the existence of a counterexample in the case of planar graphs. We begin in section 2 with some background and definitions.

2. Preliminaries. We use standard notation and terminology from finite model theory (see [5]). Some particular definitions and notation are explained in this section.

2.1. Relational structures. A *relational vocabulary* σ is a finite set of *relation symbols*, each with a specified *arity*. A *σ-structure* $\mathbf{A}$ consists of a *universe* A, or *domain*, and an *interpretation* which associates to each relation symbol $R \in \sigma$ of some arity r a relation $R^{\mathbf{A}} \subseteq A^r$. A *graph* is a structure $\mathbf{G} = (V, E)$, where E is a binary relation that is symmetric and antireflexive. Thus, our graphs are undirected, loopless, and without parallel edges.

A σ-structure $\mathbf{B}$ is called a *substructure* of $\mathbf{A}$ if $B \subseteq A$ and $R^{\mathbf{B}} \subseteq R^{\mathbf{A}}$ for every $R \in \sigma$. It is called an *induced substructure* if $R^{\mathbf{B}} = R^{\mathbf{A}} \cap B^r$ for every $R \in \sigma$ of arity r. Notice the analogy with the graph-theoretical concept of *subgraph* and *induced subgraph*. A substructure $\mathbf{B}$ of $\mathbf{A}$ is proper if $\mathbf{A} \neq \mathbf{B}$. If $\mathbf{A}$ is an induced substructure of $\mathbf{B}$, we say that $\mathbf{B}$ is an *extension* of $\mathbf{A}$. If $\mathbf{A}$ is a proper induced substructure, then $\mathbf{B}$ is a proper extension. If $\mathbf{B}$ is the disjoint union of $\mathbf{A}$ with another σ-structure, we say that $\mathbf{B}$ is a *disjoint extension* of $\mathbf{A}$. If $S \subseteq A$ is a subset of the universe of $\mathbf{A}$, then $\mathbf{A} \cap S$ denotes the *induced substructure generated by* S; in other words, the universe of $\mathbf{A} \cap S$ is S, and the interpretation in $\mathbf{A} \cap S$ of the r-ary relation symbol R is $R^{\mathbf{A}} \cap S^r$.

The *Gaifman graph* of a σ-structure $\mathbf{A}$, denoted by $\mathcal{G}(\mathbf{A})$, is the (undirected) graph whose set of nodes is the universe of $\mathbf{A}$, and whose set of edges consists of all pairs (a, a') of distinct elements of A such that a and a' appear together in some tuple of a relation in $\mathbf{A}$. The *degree* of a structure is the degree of its Gaifman graph, that is, the maximum number of neighbors of nodes of the Gaifman graph.

2.2. Neighborhoods and treewidth. Let $\mathbf{G} = (V, E)$ be a graph. Moreover, let $u \in V$ be a node and let $d \geq 0$ be an integer. The *d-neighborhood* of u in $\mathbf{G}$, denoted by $N_d^{\mathbf{G}}(u)$, is defined inductively as follows:

1. $N_0^{\mathbf{G}}(u) = \{u\}$;
2. $N_{d+1}^{\mathbf{G}}(u) = N_d^{\mathbf{G}}(u) \cup \{v \in V : (v, w) \in E \text{ for some } w \in N_d^{\mathbf{G}}(u)\}$.

If $\mathbf{A}$ is a σ-structure, a is a point in $\mathbf{A}$, and $\mathbf{G}$ is the Gaifman graph of $\mathbf{A}$, we let $N_d^{\mathbf{A}}(a)$ denote the d-neighborhood of a in $\mathbf{G}$. Where it causes no confusion, we also write $N_d^{\mathbf{A}}(a)$ for the substructure of $\mathbf{A}$ generated by this set.

A *tree* is an acyclic connected graph. A *tree-decomposition* of $\mathbf{G} = (V, E)$ is a pair (T, L) where T is a tree and $L : T \to \wp(V)$ is a labeling of the nodes of T by sets of vertices of $\mathbf{G}$ such that

1. for every edge $\{u, v\} \in E$, there is a node t of T such that $\{u, v\} \subseteq L(t)$;
2. for every $u \in V$, the set $\{t \in T : u \in L(t)\}$ forms a connected subtree of T.

The *width* of a tree-decomposition (T, L) is $\max_{t \in T} |L(t)| - 1$. The *treewidth* of $\mathbf{G}$ is the smallest k for which $\mathbf{G}$ has a tree-decomposition of width k. The *treewidth* of a σ-structure is the treewidth of its Gaifman graph. Note that trees have treewidth one.

2.3. First-order logic, monadic second-order logic, and types. Let σ be a relational vocabulary. The *atomic formulas* of σ are those of the form $R(x_1, \ldots, x_r)$, where $R \in \sigma$ is a relation symbol of arity r, and $x_1, \ldots, x_r$ are first-order variables that are not necessarily distinct. Formulas of the form $x = y$ are also atomic.

The collection of *first-order formulas* is obtained by closing the atomic formulas under negation, conjunction, disjunction, and universal and existential first-order quantification. The collection of *existential* first-order formulas is obtained by closing the atomic formulas and the negated atomic formulas under conjunction, disjunction,

and existential quantification. The semantics of first-order logic is standard.

The collection of *monadic second-order formulas* is obtained by closing the atomic formulas under negation, conjunction, disjunction, universal and existential first-order quantification, and universal and existential second-order quantification over sets. The semantics of monadic second-order logic is also standard.

The quantifier rank of a formula, be it first-order or monadic second-order, is the depth of nesting of quantifiers in the formula.

Let $\mathbf{A}$ be a σ-structure, and let $a_1, \ldots, a_n$ be points in $\mathbf{A}$. If $\varphi(x_1, \ldots, x_n)$ is a formula with free variables $x_1, \ldots, x_n$, we use the notation $\mathbf{A} \models \varphi(a_1, \ldots, a_n)$ to denote the fact that φ is true in $\mathbf{A}$ when x_i is interpreted by a_i. If m is an integer, the first-order m-type of $a_1, \ldots, a_n$ in $\mathbf{A}$ is the collection of all first-order formulas $\varphi(x_1, \ldots, x_n)$ of quantifier rank at most m, up to logical equivalence, for which $\mathbf{A} \models \varphi(a_1, \ldots, a_n)$. The monadic second-order m-type of $a_1, \ldots, a_n$ in $\mathbf{A}$ is the collection of all monadic second-order formulas $\varphi(x_1, \ldots, x_n)$ of quantifier rank at most m, up to logical equivalence, for which $\mathbf{A} \models \varphi(a_1, \ldots, a_n)$. In this definition, by quantifier rank of a monadic second-order formula we mean the *total* quantifier rank, which means that we include both first-order and second-order quantifiers in the count. We note that some definitions of monadic second-order type in the literature distinguish between first-order and second-order quantifier rank [14], but we do not need this refinement.

2.4. Preservation under extensions and minimal models. Let $\mathcal{C}$ be a class of finite σ-structures that is closed under induced substructures. Let φ be a first-order sentence. We say that φ is *preserved under extensions* on $\mathcal{C}$ if whenever $\mathbf{A}$ and $\mathbf{B}$ are structures in $\mathcal{C}$ such that $\mathbf{B}$ is an extension of $\mathbf{A}$, then $\mathbf{A} \models \varphi$ implies $\mathbf{B} \models \varphi$. We say that $\mathbf{A}$ is a *minimal model* of φ if $\mathbf{A} \models \varphi$ and every proper induced substructure $\mathbf{A}'$ of $\mathbf{A}$ is such that $\mathbf{A}' \not\models \varphi$. The following lemma states that the existential sentences are precisely those that have finitely many minimal models. Its proof is part of folklore.

LEMMA 2.1. *Let $\mathcal{C}$ be a class of finite σ-structures that is closed under induced substructures. Let φ be a first-order sentence that is preserved under extensions on $\mathcal{C}$. Then the following are equivalent:*

1. *φ is equivalent on $\mathcal{C}$ to an existential sentence.*
2. *φ has finitely many minimal models in $\mathcal{C}$.*

In the rest of the paper, we use several times the implication from item 2 to item 1. Just for completeness, this is proved by taking the disjunction of the existential closure of the atomic types of each of the finitely many minimal models.

3. Acyclic structures. We begin with the simple case of acyclic structures, by which we mean structures whose Gaifman graph is acyclic. We show that any class of such structures satisfying certain closure properties admits the extension preservation property. Note that for structures whose Gaifman graphs are acyclic, there is no loss of generality in assuming that the vocabulary σ consists of unary and binary relations only.

The proof makes heavy use of a technique known as Hanf locality, for which we provide the necessary background first.

Let $\mathbf{A}$ and $\mathbf{B}$ be structures. If $\mathbf{a} \in A^m, \mathbf{b} \in B^m$ are m-tuples, we write $(\mathbf{A}, \mathbf{a}) \equiv^m (\mathbf{B}, \mathbf{b})$ to denote that the first-order m-type of $\mathbf{a}$ in $\mathbf{A}$ is the same as the first-order m-type of $\mathbf{b}$ in $\mathbf{B}$. In particular $\mathbf{A} \equiv^m \mathbf{B}$ denotes that the structures $\mathbf{A}$ and $\mathbf{B}$ are not distinguished by any first-order sentence of quantifier rank m or less. The equivalence relation $\equiv^m$ is characterized by Ehrenfeucht–Fraïssé games (see, for instance, [5]). These can be used to show that the relation is a congruence with respect to disjoint

union with a multiplicity threshold of m. A precise statement of this useful property is given in the following lemma. We write $\mathbf{A} \oplus \mathbf{B}$ to denote the disjoint union of the structures $\mathbf{A}$ and $\mathbf{B}$ and $n\mathbf{A}$ to denote the disjoint union of n copies of $\mathbf{A}$ (see [5, Prop. 2.3.10]).

LEMMA 3.1. *Let $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{B}_1$, and $\mathbf{B}_2$ be structures, and let m, n, and n' be integers.*

1. *If $\mathbf{A}_1 \equiv^m \mathbf{B}_1$ and $\mathbf{A}_2 \equiv^m \mathbf{B}_2$, then $\mathbf{A}_1 \oplus \mathbf{A}_2 \equiv^m \mathbf{B}_1 \oplus \mathbf{B}_2$.*
2. *If $n, n' \geq m$ and $\mathbf{A} \equiv^m \mathbf{B}$, then $n\mathbf{A} \equiv^m n'\mathbf{B}$.*

A useful sufficient condition for the $\equiv^m$ equivalence of structures is provided by Hanf locality. The *Hanf type* of radius r of a structure $\mathbf{A}$ is the multiset of isomorphism types of r-neighborhoods of elements in $\mathbf{A}$. We say that two structures $\mathbf{A}$ and $\mathbf{B}$ are *Hanf equivalent* with radius r and threshold q, written $\mathbf{A} \simeq_{r,q} \mathbf{B}$, if, for every $a \in A$, *either* the number of occurrences of the isomorphism type of $N_r^{\mathbf{A}}(a)$ in the Hanf type of $\mathbf{A}$ is the same as that in the Hanf type of $\mathbf{B}$ *or* it is at least q, and conversely for every element $b \in B$. This allows us to state the following (for a proof see, for instance, [14, Thm. 4.24]).

THEOREM 3.2 (Hanf locality). *For every vocabulary σ and every m there are r and q such that for any pair of σ-structures $\mathbf{A}$ and $\mathbf{B}$ if $\mathbf{A} \simeq_{r,q} \mathbf{B}$, then $\mathbf{A} \equiv^m \mathbf{B}$.*

As a first step towards the main result of this section, we establish a useful property of connected, acyclic structures with degree at most 2. These are structures whose Gaifman graph consists of a simple path. This is a very restricted class of structures. In particular, any class of such structures is wide, in the sense of Theorem 4.3 below. Thus, on any class of such structures, the extension preservation property holds by virtue of Theorem 4.3. However, the property in Lemma 3.3 provides a useful stepping stone in our proof for all acyclic structures and also serves as a useful warm-up for the proof in section 4.

LEMMA 3.3. *For every vocabulary σ and every $m > 0$ there is a p such that if $\mathbf{A}$ is a σ-structure whose Gaifman graph is connected, acyclic, and of degree at most 2 and $|A| > p$, then there is a disjoint extension $\mathbf{B}$ of $\mathbf{A}$ and a proper substructure $\mathbf{A}'$ of $\mathbf{A}$ such that $\mathbf{A}' \equiv^m \mathbf{B}$.*

Proof. Given m, let r and q be obtained from Theorem 3.2. We first consider the $2r$-neighborhoods of elements of $\mathbf{A}$, returning later to consider r-neighborhoods when we wish to establish the Hanf types of the structures we construct. Clearly, the $2r$-neighborhood type of an element determines its r-neighborhood type. Also note that among σ-structures whose degree is bounded (by 2) there are only finitely many isomorphism types of $2r$-neighborhoods. Let n be the number of such types, let $l = 2r(n+1)+1$, and let $p = nl(q+l)$.

For t the isomorphism type of a $2r$-neighborhood in $\mathbf{A}$, we say that t is *frequent* if there are at least $q+l$ elements in $\mathbf{A}$ whose type is t. Since there are at most n types, the number of occurrences of elements whose type is not frequent is less than $n(q+l)$. Thus, in a path of length p there must be a sequence of l consecutive elements of frequent type. Let $a_1, \ldots, a_l$ be such a sequence. Among the $2rn+1$ central elements of the sequence $a_{r+1}, \ldots, a_{(2n+1)r+1}$ there must be a pair a_i, a_j which have the same type and such that $j - i > 2r$. Let $\mathbf{C}$ be the substructure of $\mathbf{A}$ generated by the elements $a_{i+1}, \ldots, a_j$. We define $\mathbf{B}$ to be $\mathbf{A} \oplus \mathbf{C}$ and $\mathbf{A}'$ to be the substructure of $\mathbf{A}$ generated by $A \setminus C$.

Our aim is to prove $\mathbf{A}' \equiv^m \mathbf{B}$ by showing that $\mathbf{A}' \simeq_{r,q} \mathbf{B}$. We do this by considering how the Hanf type changes in going from $\mathbf{A}$ to $\mathbf{A}'$ and also how it changes in going from $\mathbf{A}$ to $\mathbf{B}$. So, for t the isomorphism type of an r-neighborhood in $\mathbf{A}$, we

say that t is *rare* if there are fewer than q elements in $\mathbf{A}$ whose type is t. Write D for the set of elements $\{a_{i-r+1}, \ldots, a_i, a_{j+1}, \ldots, a_{j+r}\}$. That is, D consists of the r elements that occur immediately before C and the r elements that occur immediately after C in the sequence $a_1, \ldots, a_l$. For any element $a \in \mathbf{A}$ that is not in $C \cup D$, $N_r^{\mathbf{A}}(a) = N_r^{\mathbf{A}'}(a)$. For any element a of $C \cup D$, the multiplicity of the type t of $N_r^{\mathbf{A}}(a)$ may decrease in going from $\mathbf{A}$ to $\mathbf{A}'$. However, t occurs at least $q + l$ times in $\mathbf{A}$ and this multiplicity cannot decrease by more than l as $|C \cup D| \leq l$. Thus, t is not rare in $\mathbf{A}'$. Clearly the elements of D may have types in $\mathbf{A}'$ that are different from their types in $\mathbf{A}$, and therefore the multiplicities of these types may increase.

Similarly, for any element $a \in A$, $N_r^{\mathbf{A}}(a) = N_r^{\mathbf{B}}(a)$, thus any type t that occurs in $\mathbf{A}$ has at least the same multiplicity in $\mathbf{B}$. Let $C' = \{a'_{i+1}, \ldots, a'_j\}$ denote the elements in the new disjoint copy of $\mathbf{C}$. If $a'_k \in C'$ is such that $i + r < k \leq j - r$, then the r-neighborhood of a'_k is isomorphic to $N_r^{\mathbf{A}}(a_k)$. Since the type of a_k is frequent, adding to its multiplicity is not significant. Thus, we only need to consider the types of the elements in $D' = \{a'_{i+1}, \ldots, a'_{i+r+1}, a'_{j-r+1}, \ldots, a'_j\}$. For these elements, the types of their r-neighborhoods in $\mathbf{B}$ may be new and result in an increase of the multiplicities of these types over their occurrences in $\mathbf{A}$. Thus, to establish our result that $\mathbf{A}' \simeq_{r,q} \mathbf{B}$ it suffices to show that there is a bijection $f : D \to D'$ such that for all $a \in D$, $N_r^{\mathbf{A}'}(a) \cong N_r^{\mathbf{B}}(f(a))$. By construction, there is an isomorphism $h : N_{2r}^{\mathbf{A}}(a_i) \to N_{2r}^{\mathbf{A}}(a_j)$ and therefore in particular, for $-r \leq k \leq r$, $N_r^{\mathbf{A}}(a_{i+k}) \cong N_r^{\mathbf{A}}(a_{j+k})$. We can now define the desired bijection f as follows: for $1 \leq k \leq r$, $f(a_{i-k+1}) = a'_{j-k+1}$ and $f(a_{j+k}) = a'_{i+k}$. □

We now use the above lemma to obtain a similar result for connected acyclic structures without a bound on the degree. This is done by reducing the case of general degree to those with degree at most 2 by means of an appropriate translation. For the vocabulary σ, there are only finitely many first-order m-types of σ-structures. Let $\tau_1, \ldots, \tau_n$ be an enumeration of the possible types of a in $\mathbf{A}$, where $\mathbf{A}$ is a connected, acyclic structure and $a \in A$. We refer to a as *the distinguished element* of $(\mathbf{A}, a)$. We define a new vocabulary σ' which has the same binary relations as σ and a unary relation T_i for each τ_i.

Let $\mathbf{A}$ be a σ'-structure that is connected, acyclic, and of degree at most 2 with the property that for each $a \in A$ there is a unique i such that $T_i(a)$. We construct from $\mathbf{A}$ a σ-structure $\tilde{\mathbf{A}}$ as follows: each element $a \in A$ with $T_i(a)$ is replaced by a structure $\mathbf{T}_a$ of type τ_i. Moreover, for any binary relation R, $(b, c) \in R^{\tilde{\mathbf{A}}}$ if and only if *either* b and c are in the same structure $\mathbf{T}_a$ and $(b, c) \in R^{\mathbf{T}_a}$ *or* b is the distinguished element of $\mathbf{T}_a$, c is the distinguished element of $\mathbf{T}_{a'}$, and $(a, a') \in R^{\mathbf{A}}$. The structure $\tilde{\mathbf{A}}$ is not uniquely determined by $\mathbf{A}$ as there are, in general, many structures of type τ_i. However, the following lemma is easily established along the lines of Lemma 3.1.

LEMMA 3.4. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be connected, acyclic structures of degree at most* 2 *with the property that for each element there is a unique* i *such that* T_i *holds, and let* m *be an integer. If* $\mathbf{A} \equiv^m \mathbf{B}$, *then* $\tilde{\mathbf{A}} \equiv^m \tilde{\mathbf{B}}$.

We will call a structure of the form $\tilde{\mathbf{A}}$ a σ-companion of $\mathbf{A}$.

LEMMA 3.5. *For every vocabulary* σ *and every* $m > 0$ *there is a* p *such that if* $\mathbf{A}$ *is a structure whose Gaifman graph is connected and acyclic and which contains a path with more than* p *elements, then there is a disjoint extension* $\mathbf{B}$ *of* $\mathbf{A}$ *and a proper substructure* $\mathbf{A}'$ *of* $\mathbf{A}$ *such that* $\mathbf{A}' \equiv^m \mathbf{B}$.

Proof. Let σ' be the vocabulary, as above, with a unary relation for each m-type of σ-structures, and let p be as in Lemma 3.3 for the vocabulary σ'. Let $a_1, \ldots, a_p$ be the path of length p in $\mathbf{A}$. For each i, let S_i be the set of elements that are reachable

(in the Gaifman graph of $\mathbf{A}$) from a_i without going through a_j for any $j \neq i$, and let $\mathbf{S}_i$ be the substructure generated by S_i. We define the σ'-structure $s\mathbf{A}$ as follows. The universe of $s\mathbf{A}$ is $\{a_1, \ldots, a_p\}$; $T_k(a_i)$ holds if and only if a_i has type τ_k in $\mathbf{S}_i$; and $(a_i, a_j) \in R^{s\mathbf{A}}$ if and only if $(a_i, a_j) \in R^{\mathbf{A}}$. Then it is easily seen that $\mathbf{A}$ is a σ-companion of $s\mathbf{A}$ (which is defined, since the Gaifman graph of $\mathbf{A}$ is acyclic).

Let $s\mathbf{A}'$ and $s\mathbf{B}$ be the structures obtained from $s\mathbf{A}$ by Lemma 3.3. We obtain $\mathbf{A}'$ as a σ-companion of $s\mathbf{A}'$ by replacing each element a_i by the structure $(\mathbf{S}_i, a_i)$. This ensures that $\mathbf{A}'$ is a substructure of $\mathbf{A}$. Similarly, we obtain $\mathbf{B}$ as a σ-companion of $s\mathbf{B}$, ensuring that $\mathbf{B}$ is a disjoint extension of $\mathbf{A}$. Since $s\mathbf{A}' \equiv^m s\mathbf{B}$ by Lemma 3.3, we also have $\mathbf{A}' \equiv^m \mathbf{B}$ by Lemma 3.4. □

Note that in both Lemmas 3.3 and 3.5 $\mathbf{B}$ is not only a disjoint extension of $\mathbf{A}$, it is in fact also the disjoint union of $\mathbf{A}$ with a substructure of $\mathbf{A}$.

In order to prove the main theorem of this section, we need one further composition property of acyclic structures along the lines of the properties in Lemma 3.1. In order to define it, we introduce some further notation. Given an acyclic structure $\mathbf{A}$ and an element $a \in A$, for every neighbor b of a let S_b be the set of elements in A which are reachable from b (in the Gaifman graph) without going through a and let $\mathrm{tp}_a(b)$ denote the first-order m-type of b in $\mathbf{S}_b$. We define the *child-type* of b with respect to a to be the pair $(\mathrm{at}(a,b), \mathrm{tp}_a(b))$, where $\mathrm{at}(a,b)$ is the atomic type of the pair (a,b). Finally, we define the *child-type* of an element a, written $\mathrm{ct}^{\mathbf{A}}(a)$, to be the multiset of the child-types of its neighbors with respect to a. Write $(\mathbf{A}, a) \sim_m (\mathbf{B}, b)$ to denote that every type *either* occurs the same number of times in $\mathrm{ct}^{\mathbf{A}}(a)$ as it does in $\mathrm{ct}^{\mathbf{B}}(b)$ *or* occurs at least m times in both. The following lemma is now a straightforward application of games.

LEMMA 3.6. *If* $(\mathbf{A}, a) \sim_m (\mathbf{B}, b)$*, then* $(\mathbf{A}, a) \equiv_m (\mathbf{B}, b)$.

We are now ready for the main theorem of this section.

THEOREM 3.7. *Let* $\mathcal{C}$ *be a class of acyclic finite structures, closed under substructures and disjoint unions. Then, on* $\mathcal{C}$*, every first-order sentence that is preserved under extensions is equivalent to an existential sentence.*

Proof. Let φ be such a sentence of quantifier rank m. We aim to show that there is an N such that if $\mathbf{A}$ in $\mathcal{C}$ is a model of φ with more than N elements, then $\mathbf{A}$ is not minimal. Let p be as in Lemma 3.5, let n be the number of distinct first-order m-types of connected structures in $\mathcal{C}$, and let q be the number of distinct types of the form $(\mathrm{at}(a,b), \mathrm{tp}_a(b))$, where a and b are neighbors in a structure in $\mathcal{C}$. Let $N = mn(qm)^p$.

Now, suppose $\mathbf{A}$ is a minimal model of φ in $\mathcal{C}$ with more than N elements. We consider three cases.

Case 1. $\mathbf{A}$ has more than mn distinct connected components. Then there must be some collection of more than m such components that have the same first-order m-type. Consider the structure $\mathbf{A}'$ obtained by removing one of these components. By Lemma 3.1 $\mathbf{A}' \equiv^m \mathbf{A}$, contradicting the minimality of $\mathbf{A}$.

If $\mathbf{A}$ has mn or fewer connected components, one of these components must have at least $(qm)^p$ elements. Call this component $\mathbf{C}$ the large component.

Case 2. The large component of $\mathbf{A}$ has a node of degree greater than qm. Call this node a. The type $\mathrm{ct}^{\mathbf{A}}(a)$ must contain a type with more than m occurrences. Let b be a neighbor of a that has this child-type with respect to a. Let $\mathbf{A}'$ be the substructure of $\mathbf{A}$ obtained by removing all elements in S_b. By Lemma 3.6, we have $\mathbf{A}' \equiv^m \mathbf{A}$, again contradicting the minimality of $\mathbf{A}$.

Case 3. If $\mathbf{C}$ does not contain a node of degree greater than qm, it must contain a path of length p. Thus, by Lemma 3.5, there is a proper substructure $\mathbf{C}'$ of $\mathbf{C}$ and a

disjoint extension $\mathbf{D}$ of $\mathbf{C}$ such that $\mathbf{C}' \equiv^m \mathbf{D}$. Let $\mathbf{A}'$ be the structure obtained from $\mathbf{A}$ by replacing $\mathbf{C}$ by $\mathbf{C}'$ and let $\mathbf{B}$ be the structure obtained from $\mathbf{A}$ by replacing $\mathbf{C}$ by $\mathbf{D}$. Then, by Lemma 3.1, $\mathbf{A}' \equiv^m \mathbf{B}$. Note also that $\mathbf{A}'$ and $\mathbf{B}$ are in $\mathcal{C}$ since it is closed under substructures and disjoint unions. Since φ is preserved under extensions on $\mathcal{C}$, $\mathbf{B} \models \varphi$, and hence $\mathbf{A}' \models \varphi$, again contradicting the minimality of $\mathbf{A}$. □

4. Wide structures. This section will focus on classes of structures that are *wide*, meaning that large enough structures contain many points that are pairwise far apart from each other. It was shown in [1] that the homomorphism preservation theorem holds for any wide class of structures. Here we aim to establish the analogous result for the extension preservation property.

DEFINITION 4.1. *A set of elements B in a σ-structure $\mathbf{A}$ is* d-scattered *if for every pair of distinct $a, b \in B$ we have $N_d^{\mathbf{A}}(a) \cap N_d^{\mathbf{A}}(b) = \emptyset$.*

We say that a class of finite σ-structures $\mathcal{C}$ is wide *if for every d and m there exists an N such that every structure in $\mathcal{C}$ of size at least N contains a d-scattered set of size m.*

The canonical example of a wide class of structures is the collection of all structures of degree bounded by a constant. More generally, any class of structures whose maximum degree is bounded by $n^{o(1)}$, where n is the number of elements of the structure, is wide.

Unfortunately, the techniques and arguments of section 3 based on Hanf locality will not be enough for our current purpose. Instead, we will have to resort to Gaifman locality, for which we provide the necessary background first.

For every integer $r \geq 0$, let $\delta(x, y) \leq r$ denote the first-order formula expressing that the distance between x and y in the Gaifman graph is at most r. Let $\delta(x, y) > r$ denote the negation of this formula. Note that the quantifier rank of $\delta(x, y) \leq r$ is bounded by r. A *basic local sentence* is a sentence of the form

$$(\exists x_1) \cdots (\exists x_n) \left(\bigwedge_{i \neq j} \delta(x_i, x_j) > 2r \wedge \bigwedge_i \psi^{N_r(x_i)}(x_i) \right), \tag{4.1}$$

where ψ is a first-order formula with one free variable. Here, $\psi^{N_r(x_i)}(x_i)$ stands for the relativization of ψ to $N_r(x_i)$; that is, the subformulas of ψ of the form $(\exists x)(\theta)$ are replaced by $(\exists x)(\delta(x, x_i) \leq r \wedge \theta)$, and the subformulas of the form $(\forall x)(\theta)$ are replaced by $(\forall x)(\delta(x, x_i) \leq r \rightarrow \theta)$. The *locality radius* of a basic local sentence is r. Its *width* is n. Its *local quantifier rank* is the quantifier rank of ψ. We will use the fact that basic local sentences are preserved under disjoint extensions. Note, however, that they may not be preserved under plain extensions since in that case the neighborhoods can grow.

The main result about basic local sentences is that they form a building block for first-order logic. This is known as Gaifman's theorem (for a proof, see, for example, [5, Thm. 2.5.1]).

THEOREM 4.2 (Gaifman locality). *Every first-order sentence is equivalent to a Boolean combination of basic local sentences.*

The following theorem contains the main technical construction of the paper.

THEOREM 4.3. *Let $\mathcal{C}$ be a class of finite σ-structures that is wide and closed under substructures and disjoint unions. Then, on $\mathcal{C}$, every first-order sentence that is preserved under extensions is equivalent to an existential sentence.*

Proof. Let φ be a first-order sentence that is preserved under extensions on $\mathcal{C}$.

By Gaifman's theorem we may assume that $\varphi = \bigvee_{i\in I} \tau_i$, with

$$\tau_i = \bigwedge_{j\in J_i} \theta^i_j \wedge \bigwedge_{k\in K_i} \neg\theta^i_k, \tag{4.2}$$

where each θ^i_h is a basic local sentence. Now we define a list of parameters that we need in the proof (the reader may skip this list now and use it to look up the values when they are needed):

- r is the maximum of the locality radii of all θ^i_h;
- s is the sum of all widths of all θ^i_h;
- m is the maximum of the local quantifier ranks of all θ^i_h;
- ℓ is the number of disjuncts in φ, so $\ell = |I|$;
- $n = (\ell + 2)s$;
- $M = m + 3r + 3$;
- $d = 2(r + 1)(\ell + 1)s + 6r + 2$;
- q is the number of monadic second-order M-types with one free variable;
- N is such that every structure in $\mathcal{C}$ of size at least N contains a $(4dq+2r+1)$-scattered set of size $(n-1)q + s + \ell s + 1$.

Our goal is to show that the minimal models of φ have size less than N. Suppose on the contrary that $\mathbf{A}$ is a minimal model of φ of size at least N. We define the *type* of a point $a \in A$ to be its monadic second-order M-type in $\mathbf{A} \cap N^{\mathbf{A}}_d(a)$. In other words, the type of a is the collection of all monadic second-order formulas $\psi(x)$ of quantifier rank at most M, up to logical equivalence, for which $\mathbf{A} \cap N^{\mathbf{A}}_d(a) \models \psi(a)$. We say that a realizes its type. The reason we consider monadic second-order types, instead of first-order types, will become clear later in the proof. Let $t_1, \ldots, t_q$ be all possible types. We need a couple of definitions. Let C be a subset of A and t a type. We say that t is *covered by* C if for all realizations a of t we have $N^{\mathbf{A}}_d(a) \subseteq C$. We say that t is *free over* C if there are at least n realizations $a_1, \ldots, a_n$ of t such that $N^{\mathbf{A}}_d(a_i)$ and $N^{\mathbf{A}}_d(a_j)$ are pairwise disjoint and do not intersect C.

Claim 4.4. *There exist a radius $e \leq 2dq$ and a set D of at most $(n-1)q$ points in A such that each type is either covered by $N^{\mathbf{A}}_e(D)$ or free over $N^{\mathbf{A}}_e(D)$.*

Proof. We define D and e inductively. Let $D_0 = \emptyset$ and $e_0 = 0$. Suppose now that D_i and e_i are already defined. Let $C = N^{\mathbf{A}}_{e_i}(D_i)$. If all types are either covered by C or free over C, then let $D = D_i$ and $e = e_i$. Otherwise, let j be minimal such that type t_j is neither covered by C nor free over C. We define a set E inductively as follows. Let $E_0 = \emptyset$. Suppose now that E_t is already defined. If there is no realization of t_j outside $N^{\mathbf{A}}_{2d}(C \cup E_t)$, then let $E = E_t$ and we are done with the construction of E. Otherwise, let a_{t+1} be a realization of t_j outside $N^{\mathbf{A}}_{2d}(C \cup E_t)$ and let $E_{t+1} = E_t \cup \{a_{t+1}\}$. Note that this iteration cannot continue beyond $n-1$ steps since otherwise t_j would be free over C. This means that the iteration stops, and when it does $|E| \leq n-1$ and t_j is covered by any set that contains $N^{\mathbf{A}}_{2d}(C \cup E)$, and in particular by $N^{\mathbf{A}}_{e_i+2d}(D_i \cup E)$. Let $D_{i+1} = D_i \cup E$ and $e_{i+1} = e_i + 2d$. The construction stops after at most q steps because at each step one new type is covered and remains covered for the rest of the construction. This shows that $|D| \leq (n-1)q$ and $e \leq 2dq$, which proves the claim. ∎

In the following, we fix e and D according to Claim 4.4. We say that a type t is *frequent* if it is not covered by $N^{\mathbf{A}}_e(D)$. Otherwise we say that t is *rare*.

We shall build a finite sequence of sets $S_0 \subseteq S_1 \subseteq \cdots \subseteq S_p \subseteq A$, with $p \leq \ell$, so that the last set S_p in the sequence will be such that the substructure of $\mathbf{A}$ induced by S_p is a proper substructure of $\mathbf{A}$ that satisfies φ. This will contradict the minimality

of $\mathbf{A}$ and will prove the theorem. The sequence S_i is constructed inductively together with a second sequence of sets $C_0 \subseteq C_1 \subseteq \cdots \subseteq C_p \subseteq A$ called the *centers*, and a sequence of sets of indices $I_0 \subseteq I_1 \subseteq \cdots \subseteq I_p \subseteq I$ (recall that φ is the disjunction of the formulas τ_i from (4.2) for $i \in I$). Moreover, the following conditions will be preserved by the inductive construction for every $i < p$.

(a) $S_i \subseteq N_r^{\mathbf{A}}(C_i)$.
(b) $|C_i| \leq is$.
(c) No disjoint extension of $\mathbf{A} \cap S_i$ satisfies $\bigvee_{j \in I_i} \tau_j$.
(d) $N_e^{\mathbf{A}}(D)$ and $N_d^{\mathbf{A}}(C_i)$ are disjoint.
(e) $|I_i| = i$.

Observe that it is a direct consequence of property (d) that the type of each $a \in C_i$ is frequent.

Let $S_0 = C_0 = I_0 = \emptyset$, and let us assume that S_i, C_i, and I_i have already been defined with the properties above. We construct S_{i+1}, C_{i+1}, and I_{i+1}. Let $\mathbf{B}$ be the disjoint union of $\mathbf{A}$ with a copy of $\mathbf{A} \cap S_i$.

(4.3) Since $\mathbf{B}$ is an extension of $\mathbf{A}$, it satisfies φ.

Therefore, there exists an $i' \in I$ such that $\mathbf{B}$ satisfies $\tau_{i'}$. By property (c), since the extension is disjoint, we know that $i' \notin I_i$. Let $I_{i+1} = I_i \cup \{i'\}$. For the rest of the proof, the index i' will be fixed so we drop any reference to it. For example, we will write τ instead of $\tau_{i'}$ and θ_h instead of $\theta_h^{i'}$. Recall that

$$\tau = \bigwedge_{j \in J} \theta_j \wedge \bigwedge_{k \in K} \neg \theta_k.$$

Since $\mathbf{B}$ satisfies τ, in particular it satisfies the positive requirements: $\mathbf{B} \models \bigwedge_{j \in J} \theta_j$. Let W_j be a minimal set of witnesses in $\mathbf{B}$ for the outermost existential quantifiers in θ_j, and let $W = \bigcup_{j \in J} W_j$. We have $|W| \leq s$. Some of these witnesses may be in $\mathbf{A}$ and some may be in the new copy of $\mathbf{A} \cap S_i$ in $\mathbf{B}$. Let $W_A \cup W_B = W$ be such a partition, with W_A being the witnesses in A. The following claim shows that W_A can be chosen far from C_i. This will be needed later.

CLAIM 4.5. *There is a set W of witnesses such that $N_{r+1}^{\mathbf{A}}(C_i) \cap N_r^{\mathbf{A}}(W_A) = \emptyset$.*

Proof. Fix a set W of witnesses so that the number of points b in W_A for which $N_{r+1}^{\mathbf{A}}(C_i)$ and $N_r^{\mathbf{A}}(b)$ are not disjoint is minimal. Suppose that this number is not zero, and let $b \in W_A$ with $N_{r+1}^{\mathbf{A}}(C_i) \cap N_r^{\mathbf{A}}(b) \neq \emptyset$. Let $a \in C_i$ be such that $N_{r+1}^{\mathbf{A}}(a) \cap N_r^{\mathbf{A}}(b) \neq \emptyset$. Then $N_r^{\mathbf{A}}(b) \subseteq N_{3r+1}^{\mathbf{A}}(a) \subseteq N_d^{\mathbf{A}}(a)$. By property (d), the type t of a is frequent. So let a' be a realization of t such that $N_{r+1}^{\mathbf{A}}(W \cup C_i)$ and $N_{3r+1}^{\mathbf{A}}(a')$ are disjoint. Such an a' exists because t is frequent and thus, by Claim 4.4, is free over $N_e^{\mathbf{A}}(D)$ and thus has

$$n > (\ell + 1)s \geq |W \cup C_i|$$

realizations whose d-neighborhoods are pairwise disjoint and disjoint from $N_e^{\mathbf{A}}(D)$.

The goal now is to find a b' such that $N_r^{\mathbf{A}}(b') \subseteq N_{3r+1}^{\mathbf{A}}(a') \subseteq N_d^{\mathbf{A}}(a')$ and such that b and b' have the same first-order m-type on $\mathbf{A} \cap N_r^{\mathbf{A}}(b)$ and $\mathbf{A} \cap N_r^{\mathbf{A}}(b')$, respectively. If we achieve this, then b' can replace b as a witness in W_A, and since $N_{r+1}^{\mathbf{A}}(W \cup C_i)$ and $N_{3r+1}^{\mathbf{A}}(a')$ are disjoint, so are $N_{r+1}^{\mathbf{A}}(C_i)$ and $N_r^{\mathbf{A}}(b')$. This will contradict the minimality of W.

In order to find b' as above, let T be the first-order m-type of b on $\mathbf{A} \cap N_r^{\mathbf{A}}(b)$,

and let $\xi(x)$ be the following first-order formula:

$$(\exists y)\left((\forall z)(\delta(y,z)\leq r\rightarrow\delta(x,z)\leq 3r+1)\wedge\bigwedge_{\chi\in T}\chi^{N_r(y)}(y)\right).$$

Note that the conjunction is finite because the first-order m-type T contains finitely many formulas up to logical equivalence, and that the quantifier rank of this formula is bounded by $3r+3+m\leq M$. Also $N_d^{\mathbf{A}}(a)\models\xi(a)$ because b can serve as a witness for y. Therefore, since a and a' have the same monadic second-order M-type and hence the same first-order M-type in $N_d^{\mathbf{A}}(a)$ and $N_d^{\mathbf{A}}(a')$, also $N_d^{\mathbf{A}}(a')\models\xi(a')$. Note here that we are not yet using the full power of monadic second-order type, only the fact that it contains the first-order type as a subset. Let b' be the witness to y in $N_d^{\mathbf{A}}(a')\models\xi(a')$, completing the proof. □

In the following, we fix a set W of witnesses such that $N_{r+1}^{\mathbf{A}}(C_i)\cap N_r^{\mathbf{A}}(W_A)=\emptyset$. We let $\mathbf{C}$ be the substructure of $\mathbf{A}$ induced by $N_e^{\mathbf{A}}(D)\cup N_r^{\mathbf{A}}(W_A)\cup S_i$. We claim that $\mathbf{C}$ satisfies the positive requirements of τ.

Claim 4.6. *$\mathbf{C}$ is a substructure of $\mathbf{A}$ such that $\mathbf{C}\models\bigwedge_{j\in J}\theta_j$.*

Proof. It is obvious that $\mathbf{C}$ is a substructure of $\mathbf{A}$. The point, however, is that $\mathbf{C}$ is in fact the disjoint union of the substructure induced by $N_e^{\mathbf{A}}(D)\cup N_r^{\mathbf{A}}(W_A)$ and the substructure induced by S_i. This is because $S_i\subseteq N_r^{\mathbf{A}}(C_i)$ and $N_{r+1}^{\mathbf{A}}(C_i)$ is disjoint from $N_e^{\mathbf{A}}(D)$ by property (d) and also disjoint from $N_r^{\mathbf{A}}(W_A)$ by Claim 4.5. It follows that the witnesses from $\mathbf{B}$ in W_B can also be found in $\mathbf{C}$. Obviously, also the witnesses from $\mathbf{B}$ in W_A can be found in $\mathbf{C}$. This proves that $\mathbf{C}$ satisfies the positive requirements of τ. □

Consider φ on $\mathbf{C}$. If $\mathbf{C}$ is a model of φ, let $S_p=N_e^{\mathbf{A}}(D)\cup N_r^{\mathbf{A}}(W_A)\cup S_i$ and we are done. Notice that $\mathbf{C}$ is a proper substructure of $\mathbf{A}$ because $\mathbf{A}$ contains $(n-1)q+s+\ell s+1$ points that are $(4dq+2r+1)$-scattered, but $S_p\subseteq N_{2dq+r}^{\mathbf{A}}(D\cup W_A\cup C_i)$ and

$$|D\cup W_A\cup C_i|\leq(n-1)q+s+\ell s.$$

If $\mathbf{C}$ is not a model of φ, it cannot satisfy τ. However, by Claim 4.6, $\mathbf{C}$ satisfies the positive requirements $\bigwedge_{j\in J}\theta_j$. Therefore, $\mathbf{C}$ does not satisfy $\bigwedge_{k\in K}\neg\theta_k$. Let $k\in K$ such that $\mathbf{C}\models\theta_k$. In the next claim we find a substructure of $\mathbf{A}$ that extends $\mathbf{A}\cap S_i$ and forces all its disjoint extensions to satisfy θ_k.

Claim 4.7. *There exist $C_{i+1}\supseteq C_i$ and $S_{i+1}\supseteq S_i$ as required by conditions* (a)–(d).

Proof. Suppose that

$$\theta_k=(\exists x_1)\dots(\exists x_{s'})\left(\bigwedge_{i\neq j}\delta(x_i,x_j)>2r'\wedge\bigwedge_i\psi^{N_{r'}(x_i)}(x_i)\right)$$

for some $r'\leq r, s'\leq s$, and some formula ψ of quantifier rank $m'\leq m$. Without loss of generality we may assume that $m'=m$, and in order to simplify the notation, we will assume that $r'=r$ and $s'=s$. It will suffice to replace r by r' and s by s' in the appropriate places.

We have $\mathbf{C}\models\theta_k$. Let $V=\{a_1,\dots,a_s\}$ be a set of witnesses for the outermost existential quantifiers in θ_k. Then $N_r^{\mathbf{C}}(a_i)\cap N_r^{\mathbf{C}}(a_j)=\emptyset$ for all $i\neq j$ and $\mathbf{C}\cap N_r^{\mathbf{C}}(a_i)\models\psi^{N_r(x_i)}(a_i)$ for all i. Necessarily, the type t of some $a\in V$ is frequent. Otherwise

$N_r^{\mathbf{A}}(V) \subseteq N_e^{\mathbf{A}}(D) \subseteq A$, so $\mathbf{A} \models \theta_k$, and thus $\mathbf{B} \models \theta_k$, because $\mathbf{B}$ is a disjoint extension of $\mathbf{A}$. However, this is impossible because $\mathbf{B} \models \tau$.

So let $a \in V$ have frequent type t. Let Z be a set of s realizations of t such that

(i) $N_d^{\mathbf{A}}(b) \cap N_d^{\mathbf{A}}(b') = \emptyset$ for every pair of distinct $b, b' \in Z$,
(ii) $N_e^{\mathbf{A}}(D) \cap N_d^{\mathbf{A}}(Z) = \emptyset$,
(iii) $N_{r+1}^{\mathbf{A}}(C_i) \cap N_r^{\mathbf{A}}(Z) = \emptyset$.

Such a set Z exists because t is frequent, $n = (\ell+2)s$, and $|C_i| \leq \ell s$ by property (b).

Now, let $F = N_r^{\mathbf{C}}(a)$. Remember that $\mathbf{C} \cap F \models \psi^{N_r(x)}(a)$. As $F \subseteq N_r^{\mathbf{A}}(a)$, it follows that $\mathbf{A} \cap F \models \psi^{N_r(x)}(a)$. Let X be a set variable, and let $\psi^{N_r(x)\cap X}(X, x)$ denote the simultaneous relativization of $\psi(x)$ to $N_r(x)$ and X, that is, the formula obtained from ψ by replacing each subformula of the form $(\exists z)\xi$ by $(\exists z)(\delta(x, z) \leq r \wedge X(z) \wedge \xi)$, and similarly for universally quantified subformulas. Observe that the quantifier rank of $\psi^{N_r(x)\cap X}(X, x)$ is at most $m+r \leq M-1$, where we take r as an upper bound for the quantifier rank of the formula expressing $\delta(x, z) \leq r$. Moreover, $\mathbf{A} \models \psi^{N_r(x)\cap X}(F, a)$ and hence $\mathbf{A} \models \exists X \psi^{N_r(x)\cap X}(a)$.

Next comes the place where we use the full power of monadic second-order types. Since every $b \in Z$ has the same monadic second-order M-type as a, we have $\mathbf{A} \models \exists X \psi^{N_r(x_i)\cap X}(b)$. Thus there is a set $F_b \subseteq N_r^{\mathbf{A}}(b)$ such that $\mathbf{A} \models \psi^{N_r(x_i)\cap X}(F_b, b)$. It follows that

$$\mathbf{A} \cap F_b \models \psi^{N_r(x)}(b).$$

Define $C_{i+1} = C_i \cup Z$ and

$$S_{i+1} = S_i \cup \bigcup_{b \in Z} F_b.$$

Let us prove that C_{i+1} and S_{i+1} satisfy the properties (a), (b), (c), and (d). Property (a) is clear since $F_b \subseteq N_r^{\mathbf{A}}(b)$. For property (b) we have $|C_{i+1}| = |C_i| + s \leq (i+1)s$. Property (d) is satisfied by (ii) in our choice of Z.

Finally, for property (c) we argue as follows. First note that $\mathbf{A} \cap S_{i+1}$ is a disjoint extension of $\mathbf{A} \cap S_i$ because $N_{r+1}^{\mathbf{A}}(C_i) \cap N_r^{\mathbf{A}}(Z) = \emptyset$ by (iii) and $S_i \subseteq N_r^{\mathbf{A}}(C_i)$ by (a). Therefore, no disjoint extension of $\mathbf{A} \cap S_{i+1}$ satisfies τ_j for any $j \in I_i$. It remains to show that no disjoint extension of $\mathbf{A} \cap S_{i+1}$ satisfies τ. However, this is clear from the construction because every disjoint extension of $\mathbf{A} \cap S_{i+1}$ contains witnesses for the outermost existential quantifiers in θ_k, namely, the elements of the set Z. Suppose that $Z = \{b_1, \ldots, b_s\}$. Note that b_i have pairwise distance $> 2r$ by (i), and we have $\mathbf{A} \cap S_{i+1} \models \psi^{N_r(x_i)}(b_i)$, because $N^{\mathbf{A}\cap S_{i+1}}(b_i) = F_{b_i}$ and $\mathbf{A} \cap F_{b_i} \models \psi^{N_r(x_i)}(b_i)$. □

Note that I_{i+1} is constructed to satisfy property (e) as well. This completes the definition of the inductive construction. All that remains to be shown is that the construction stops in at most ℓ steps. Because suppose for contradiction that we have constructed S_ℓ, C_ℓ, and I_ℓ satisfying (a)–(e). Then $I_\ell = I$ by (e), and by (c) no disjoint extension of $\mathbf{A} \cap S_\ell$ satisfies $\varphi = \bigvee_{i \in I} \tau_i$. However,

(4.4) the disjoint union $\mathbf{B}$ of $\mathbf{A} \cap S_\ell$ with $\mathbf{A}$ is an extension of $\mathbf{A}$ and hence does satisfy φ.

This is a contradiction. □

As a direct application of Theorem 4.3, let us consider the class $\mathcal{D}_r$ of all finite σ-structures of degree bounded by r. This class is both wide and closed under substructures and disjoint unions. To see the wideness, note that when the degree of

every node is at most r, for any element a, $N_d(a)$ contains at most r^d elements. Thus, if a structure has size greater than $m(r^d)$, it must contain a d-scattered set of m elements.

THEOREM 4.8. *Let r be an integer. Then, on $\mathcal{D}_r$, every first-order sentence that is preserved under extensions is equivalent to an existential sentence.*

In the following section we show how the argument of Theorem 4.3 can be extended, in some cases, to classes of structures that are *almost wide*.

5. Bounded treewidth structures. The class of structures of bounded degree provide a canonical example of a wide class. On the other hand, acyclic structures (which we considered in section 3) are not wide. Indeed, in an arbitrarily large tree of height 1 all pairs of nodes are at distance at most 2 from each other and there is therefore no large d-scattered set for any $d > 2$, yet the tree may be arbitrarily large. However, in such a structure, the removal of just one element, the root, creates a large scattered set. This motivates the definition below.

DEFINITION 5.1. *A class of finite σ-structures $\mathcal{C}$ is* almost wide *if there is a k such that for every d and m there exists an N such that every structure $\mathbf{A}$ of size at least N in $\mathcal{C}$ contains a set B with at most k elements such that $\mathbf{A} - B$ contains a d-scattered set of size m.*

It was shown in [1] that the homomorphism preservation property holds for almost wide classes of structures which are closed under substructures and disjoint unions. It was also established that any class of graphs that excludes a minor is almost wide.

It is not the case that the extension preservation property holds for all almost wide classes. This can be seen in the next section, where we show, in particular, that it fails for the class of planar graphs. It turns out that the requirement that an almost wide class be closed under substructures and disjoint unions is not sufficient to guarantee the extension preservation property. Nevertheless, closure under unions *over a set of bottlenecks* suffices, a notion we make more precise later. In this section we show that this yields the preservation under extensions property for some particularly interesting almost wide classes. To be precise, we show that the property holds for the class $\mathcal{T}_k$ of all finite σ-structures of treewidth less than k. In other words, we aim to prove the following result.

THEOREM 5.2. *Let k be an integer. Then, on $\mathcal{T}_k$, every first-order sentence that is preserved under extensions is equivalent to an existential sentence.*

The proof of this result requires three ingredients. The first ingredient is a generalization of the disjoint union operation on structures by allowing some nonempty intersection. Let $\mathbf{A}$ and $\mathbf{B}$ be σ-structures, and let $C \subseteq A \cap B$ be such that $\mathbf{A}\cap C = \mathbf{B}\cap C$. The *union of* $\mathbf{A}$ *and* $\mathbf{B}$ *through* C, denoted by $\mathbf{A} \oplus_C \mathbf{B}$, is a new σ-structure defined as follows. The universe of $\mathbf{D} = \mathbf{A} \oplus_C \mathbf{B}$ is $A' \cup B' \cup C$, where A' is a disjoint copy of $A - C$ and B' is a disjoint copy of $B - C$. The relations of $\mathbf{D}$ are defined in the obvious way: If $a_1, \ldots, a_r$ are points in A and $a'_1, \ldots, a'_r$ are the corresponding points in $A' \cup C$, then $(a'_1, \ldots, a'_r) \in R^{\mathbf{D}}$ if and only if $(a_1, \ldots, a_r) \in R^{\mathbf{A}}$. Similarly, if $b_1, \ldots, b_r$ are points in B and $b'_1, \ldots, b'_r$ are the corresponding points in $B' \cup C$, then $(b'_1, \ldots, b'_r) \in R^{\mathbf{D}}$ if and only if $(b_1, \ldots, b_r) \in R^{\mathbf{B}}$. Observe that this construction is precisely the disjoint union of $\mathbf{A}$ and $\mathbf{B}$ when $C = \emptyset$.

The next lemma is a straightforward generalization of the obvious fact that $\mathcal{T}_k$ is closed under disjoint unions. The lemma states, roughly, that $\mathcal{T}_k$ is closed under unions through *subsets of bags of tree-decompositions*.

LEMMA 5.3. *Let k be an integer. Let $\mathbf{A}$ and $\mathbf{B}$ be two σ-structures, let $C \subseteq A \cap B$ be such that $\mathbf{A}\cap C = \mathbf{B}\cap C$, and let (T, L) and (T', L') be tree-decompositions of width k*

of $\mathbf{A}$ *and* $\mathbf{B}$*, respectively. Then, if there exists nodes* $u \in T$ *and* $u' \in T'$ *such that* $C \subseteq L(u) \cap L'(u')$*, then the union of* $\mathbf{A}$ *and* $\mathbf{B}$ *through* C *has treewidth at most* k*.*

Proof. The tree-decomposition of the union is $(T'', L \cup L')$, where $T'' = T \cup T'$ with a new tree edge joining u and u'. □

The second ingredient is the fact that the class of structures of treewidth less than k is almost wide, in the sense of Definition 5.1 that there exists a small set of vertices whose removal produces a large scattered set. Such a set is henceforth called a *bottleneck*. This was proved in [1], but here we state the stronger claim that the bottleneck can be found in a single bag of a tree-decomposition. The proof is the same as in [1] and is sketched here for completeness.

LEMMA 5.4. *For every* k*, and for every* d *and* m*, there exists an* N *such that if* $\mathbf{A}$ *is a* σ*-structure of size at least* N *and* (T, L) *is a tree-decomposition of* $\mathbf{A}$ *of width* k*, then there exist* $u \in T$ *and* $K \subseteq L(u)$ *such that* $\mathbf{A} - K$ *contains a* d*-scattered set of size* m*.*

Proof sketch. Let $p = (m-1)(2d+1)+1$, $M = k!(p-1)^k$, and $N = k(m-1)^M$ and suppose that $\mathbf{A}$ is a structure with more than N elements. Let (T, L) be a tree decomposition of $\mathbf{A}$ such that $L(u)$ has size at most k for all $u \in T$. Note that T has size at least $N/k+1$. Furthermore, suppose T has a node u of degree at least m. But then it is easy to see that taking $K = L(u)$ gives a graph with at least m distinct connected components and therefore a scattered set of size m. On the other hand, if every node of T has degree less than m, then T must have a path with length greater than M. By the sunflower lemma of Erdös and Rado [7], it follows that we can find p distinct nodes $u_1, \ldots, u_p \in T$ and a set $K \subseteq A$ such that for $i \neq j$, $L(u_i) \cap L(u_j) = K$. It can then be shown that $\mathbf{A} - K$ must contain a d-scattered set of size m. □

The third ingredient in the proof is a first-order bi-interpretation between an almost wide structure and a wide structure. From now on we focus on graphs; the construction extends easily to the general case. Let $P_1, \ldots, P_k, Q_1, \ldots, Q_k$ be unary relation symbols and $\sigma = \{E, P_1, \ldots, P_k, Q_1, \ldots, Q_k\}$. For every graph $\mathbf{G} = (V, E^{\mathbf{G}})$ and every tuple $\mathbf{a} = (a_1, \ldots, a_k) \in V^k$ we define a σ-structure $\mathbf{A} = \mathbf{A}(\mathbf{G}, \mathbf{a})$ as follows:

1. $A = V$.
2. $E^{\mathbf{A}} = E^{\mathbf{G}} - \{(a,b) \in A^2 : \{a,b\} \cap \{a_1, \ldots, a_k\} \neq \emptyset\}$.
3. $P_i^{\mathbf{A}} = \{a_i\}$.
4. $Q_i^{\mathbf{A}} = \{b \in A : (a_i, b) \in E^{\mathbf{G}}\}$.

Let us call a σ-structure $\mathbf{A}$ *derived* if $E^{\mathbf{A}}$ is a symmetric and antireflexive binary relation, and there are elements $a_1, \ldots, a_k \in A$ such that $P_i^{\mathbf{A}} = \{a_i\}$ for $1 \leq i \leq k$ and a_i is isolated in the graph underlying $\mathbf{A}$; that is, for $1 \leq i \leq k$ there is no b such that $(a_i, b) \in E^{\mathbf{A}}$. Note that for every derived structure $\mathbf{A}$ there is a unique graph $\mathbf{G}(\mathbf{A})$ and a unique k-tuple $\mathbf{a}(\mathbf{A})$ of vertices of $\mathbf{G}(\mathbf{A})$ such that

$$\mathbf{A} = \mathbf{A}(\mathbf{G}(\mathbf{A}), \mathbf{a}(\mathbf{A})).$$

The point behind the construction of $\mathbf{A} = \mathbf{A}(\mathbf{G}, \mathbf{a})$ is that if $K = \{a_1, \ldots, a_k\}$ is a bottleneck of $\mathbf{G}$ in the sense that $\mathbf{G} - K$ contains a large scattered set, then $\mathbf{A}$ itself has a large scattered set and maintains all the information needed to reconstruct $\mathbf{G}$. Indeed, $\mathbf{G}(\mathbf{A})$ is first-order interpretable in $\mathbf{A}$, and thus we get the following lemma.

LEMMA 5.5. *For every first-order sentence* φ *of vocabulary* $\{E\}$ *there is a sentence* $\tilde{\varphi}$ *of vocabulary* σ *such that for all* σ*-structures* $\mathbf{A}$ *we have the following:*

1. *If* $\mathbf{A} \models \tilde{\varphi}$*, then* $\mathbf{A}$ *is derived.*
2. *If* $\mathbf{A}$ *is derived, then* $\mathbf{A} \models \tilde{\varphi}$ *if and only if* $\mathbf{G}(\mathbf{A}) \models \varphi$*.*

This follows at once from a standard result on syntactical interpretations (cf., for example, Theorem VIII.2.2 of [6]).

Equipped with these three ingredients, we are ready for the main argument.

Proof of Theorem 5.2. Let φ be a first-order sentence that is preserved under extensions in $\mathcal{T}_k$. It suffices to show that φ has finitely many minimal models. Let $\mathbf{G} = (V, E^{\mathbf{G}})$ be a graph in $\mathcal{T}_k$ that is a minimal model of φ. Suppose for contradiction that $\mathbf{G}$ is very large. Let (T, L) be a tree-decomposition of width k of $\mathbf{G}$, and let $K = \{b_1, \ldots, b_k\} \subseteq V$ be a bottleneck; that is, a set such that $\mathbf{G} - K$ contains a large scattered set. By Lemma 5.4 we may assume that $K \subseteq L(u)$ for some $u \in T$. Let $\mathbf{A} = \mathbf{A}(\mathbf{G}, \mathbf{b})$, where $\mathbf{b} = (b_1, \ldots, b_k)$. The idea is to work with $\mathbf{A}$ and $\tilde{\varphi}$ instead of $\mathbf{G}$ and φ and proceed as in the proof of Theorem 4.3. The difference is that $\tilde{\varphi}$ is *not* preserved under extensions. However, preservation under extensions is used only twice in the proof of section 4 (in (4.3) and (4.4)), both times to prove that the disjoint union $\mathbf{B}$ of the structure $\mathbf{A}$ with $\mathbf{A} \cap S_i$ is a model of φ. Claim 5.6 shows that in both cases, $\mathbf{B}$ is a model of $\tilde{\varphi}$.

Claim 5.6. *Let $C \subseteq A$ such that the type of each $a \in C$ is frequent. Let $S \subseteq N_r(C)$ and let $\mathbf{B}$ be the disjoint union of $\mathbf{A}$ with a disjoint copy of $\mathbf{A} \cap S$. Then $\mathbf{B}$ is derived, $\mathbf{G}$ is an induced subgraph of $\mathbf{G}(\mathbf{B})$, and $\mathbf{G}(\mathbf{B})$ belongs to $\mathcal{T}_k$.*

Proof. The bottleneck points are not in C as their type is not frequent and therefore not in $N_r(C)$ as they are isolated in $\mathbf{A}$. Thus, note that $\mathbf{B}$ is derived because the bottleneck points are not in S. Let $\mathbf{H} = \mathbf{G}(\mathbf{B})$. Clearly, $\mathbf{G}$ is an induced subgraph of $\mathbf{H}$. Thus all we have to prove is that $\mathbf{H}$ belongs to $\mathcal{T}_k$. Let $\mathbf{A}' = \mathbf{A} \cap (S \cup K)$, where K is the bottleneck of $\mathbf{G}$. Again, $\mathbf{A}'$ is derived. Let $\mathbf{G}' = \mathbf{G}(\mathbf{A}')$. Clearly, $\mathbf{G}'$ is an induced subgraph of $\mathbf{G}$. In particular, $\mathbf{G}'$ is in $\mathcal{T}_k$ so it has a tree-decomposition of width k. More importantly, since $K \subseteq L(u)$, we can assume as well that K is a subset of some bag of the tree-decomposition of $\mathbf{G}'$. These two facts together imply that the union of $\mathbf{G}$ and $\mathbf{G}'$ through K, which is precisely $\mathbf{H}$, is in $\mathcal{T}_k$ by Lemma 5.3. ∎

This then shows that the $\mathbf{B}$ in (4.3) and (4.4) is a model of $\tilde{\varphi}$. The proof proceeds until we construct a structure $\mathbf{C}$ that satisfies $\tilde{\varphi}$ and is a proper substructure of $\mathbf{A}$. We claim that $\mathbf{C}$ is derived. This is because all bottleneck points have rare type, so they belong to D. Let $\mathbf{H} = \mathbf{G}(\mathbf{C})$. Note now that $\mathbf{H}$ is the union of two subgraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ of $\mathbf{G}$ through the bottleneck K. Again K is a subset of a bag of the tree-decompositions of $\mathbf{G}_1$ and $\mathbf{G}_2$, so $\mathbf{H}$ belongs to $\mathcal{T}_k$ by Lemma 5.3. Moreover $\mathbf{H}$ is a proper induced subgraph of $\mathbf{G}$ and $\mathbf{H} \models \varphi$ by Lemma 5.5. This contradicts the minimality of $\mathbf{G}$, which concludes the proof. ∎

This completes the proof of Theorem 5.2. ∎

Note that this does not imply that the existential preservation theorem holds on all classes of bounded treewidth. Indeed, we show in the next section that it fails, in particular, for the class of planar graphs of treewidth 4.

6. Counterexample for planar graphs. The aim of this section is to show that the preservation-under-extensions property fails on the class of planar graphs. Let us focus first on the class of planar graphs whose vertices are colored either black or white. Later we show how to remove the colors. The vocabulary contains a binary relation symbol E for the edge relation, and a unary relation symbol P for the color. Let φ be the following first-order sentence:

$$\begin{aligned}
\varphi &= (\exists x)(\exists y)\big(x \neq y \wedge P(x) \wedge P(y) \wedge (\varphi_1(x,y) \rightarrow \varphi_2(x,y))\big),\\
\varphi_1(x,y) &= (\forall z)\big(z \neq x \wedge z \neq y \rightarrow \neg P(z) \wedge E(x,z) \wedge E(y,z)\big),\\
\varphi_2(x,y) &= (\forall u)\big(u \neq x \wedge u \neq y\\
&\qquad \rightarrow (\exists v)(\exists w)\big(v \neq w \wedge \neg P(v) \wedge \neg P(w) \wedge E(u,v) \wedge E(u,w)\big)\big).
\end{aligned}$$

We claim that φ is preserved under extensions on the class of black/white-colored

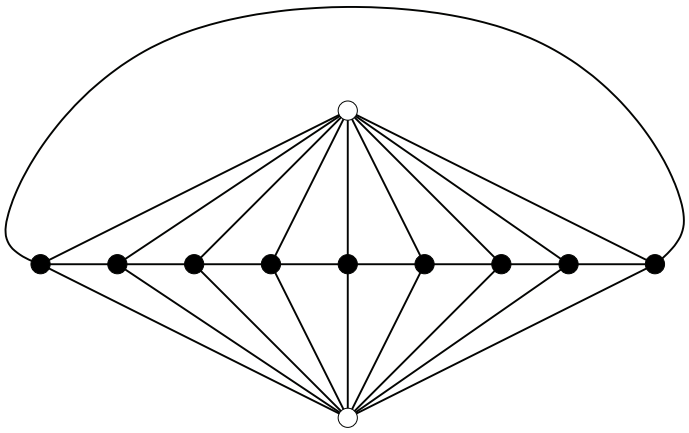


FIG. 1. $\mathbf{G}_9$.

planar graphs. Before we prove this we need a technical gadget. For every $n \geq 3$, let $\mathbf{G}_n$ be the black/white-colored planar graph displayed in Figure 1, where the number of black vertices is exactly n.

It is not hard to see that $\mathbf{G}_n$ does not have any planar proper extension in which all other vertices are adjacent to both white vertices. Let us state this as follows.

LEMMA 6.1. *Let $n \geq 3$, and let $\mathbf{H}$ be a black/white-colored planar graph that is a proper extension of $\mathbf{G}_n$. Then no vertex in $\mathbf{H} - \mathbf{G}_n$ is adjacent to both white vertices in $\mathbf{G}_n$.*

Proof. Let u be a vertex in $\mathbf{H} - \mathbf{G}_n$. Suppose that u is adjacent to both white vertices in $\mathbf{G}_n$. Then $\mathbf{H}$ contains a $\mathbf{K}_5$ minor by contracting one of the edges connecting u to a white vertex in $\mathbf{G}_n$, and by contracting all but two of the edges in $\mathbf{G}_n$ that do not have a white endpoint. This contradicts the planarity of $\mathbf{H}$. □

Now we are ready to show that φ is preserved under extensions on the class of black/white-colored planar graphs.

LEMMA 6.2. *Let $\mathbf{G}$ and $\mathbf{H}$ be black/white-colored planar graphs such that $\mathbf{H}$ is a proper extension of $\mathbf{G}$. If $\mathbf{G}$ is a model of φ, so is $\mathbf{H}$.*

Proof. Suppose that $\mathbf{G}$ is a model of φ, so let a and b be two different white vertices in $\mathbf{G}$. If $\mathbf{G} \not\models \varphi_1(a, b)$, then clearly $\mathbf{H} \not\models \varphi_1(a, b)$ because $\mathbf{G}$ is an induced substructure of $\mathbf{H}$. In this case, $\mathbf{H}$ is also a model of φ and we are done. Otherwise, since $\mathbf{G} \models \varphi$ and $\mathbf{G} \models \varphi_1(a, b)$, we have $\mathbf{G} \models \varphi_2(a, b)$. This means that every vertex in $\mathbf{G} - \{a, b\}$ is adjacent to at least two other black vertices. It follows that $\mathbf{G}$ contains some $\mathbf{G}_n$ as a (not necessarily induced) subgraph with a and b as white vertices. Here $n \geq 3$. It follows then by Lemma 6.1 that some vertex in $\mathbf{H} - \mathbf{G}_n$ fails to be connected to both a and b. But then $\mathbf{H} \not\models \varphi_1(a, b)$ so $\mathbf{H}$ is a model of φ again. □

To complete the argument we need to show that φ is not equivalent to an existential sentence on the class of black/white-colored graphs.

LEMMA 6.3. *There is no existential sentence equivalent to φ on all black/white-colored planar graphs.*

Proof. By virtue of Lemma 2.1, we only need to show that φ has infinitely many minimal models among planar graphs. It is easily seen that for all n, $\mathbf{G}_n$ is a minimal model of φ. Indeed, if we remove at least one of the white vertices from $\mathbf{G}_n$, we would not have witnesses for the two outermost existential quantifiers in φ, and if we remove at least one of the black vertices, then φ_1 remains true while φ_2 fails. □

This shows that the preservation-under-extensions property fails for the class of black/white-colored planar graphs. Removing the colors is easy. It suffices to replace each occurrence of $P(x)$ by a formula $\varphi_3(x)$ stating that x is attached to a 4×4-grid that is otherwise disconnected from the rest of the graph. One point to note is that a node without such a grid attached in a graph $\mathbf{G}$ may have a grid in an extension of $\mathbf{G}$. However, this would mean that φ_1 would fail in the extension and thus φ would necessarily be true. Thus, the formula is still preserved under extensions. This shows then that the preservation-under-extensions property fails for the class of planar graphs.

Note further that for any n, the treewidth of $\mathbf{G}_n$ is at most 4. This implies that the existential preservation theorem fails, even for the class $\mathcal{P}$ of planar graphs of treewidth at most 4. Indeed, the sentence φ is preserved under extensions on $\mathcal{P}$ since it is preserved under extensions on all planar graphs. However, φ still has infinitely many minimal models in this class as each $\mathbf{G}_n$ is in $\mathcal{P}$.

7. Conclusions. We have established the extension preservation theorem for a number of interesting classes of finite structures. These include all wide classes—such as any class of structures of bounded degree—and some almost wide classes, such as $\mathcal{T}_k$, the class of all structures of treewidth less than k. The situation for the extension preservation theorem is quite different from that established for the homomorphism preservation theorem in [1]. In particular, the former fails on the class of planar, while the latter holds on all classes that exclude a graph minor. Indeed, the methods of proof used here to establish the extension preservation property are rather different from those used in [1]. It should also be noted that Rossman [15] recently established that the homomorphism preservation theorem holds for the class of all finite structures; compare this with the known failure of the extension preservation theorem for the same class.

A number of recent results in finite model theory [1, 2, 3, 8, 10, 11] indicate that classes of structures such as trees or structures of bounded treewidth, planar graphs, and graphs of bounded genus, graphs with excluded minors, and graphs of bounded degree are well behaved in various ways related to their first-order model theory (in a broad sense). So far, no serious attempt has been made to identify general criteria connecting the different results. The locality of first-order logic always appears to play a crucial role, and the notion of *wideness* formally introduced here seems to be a good structural counterpart. But there is more to it than this simple observation; for example, the result of this paper holds on graphs of bounded degree, but not on planar graphs, whereas for the algorithmic results of [3] it is the other way round. The order invariance result of [2] has so far eluded all efforts to extend it beyond acyclic structures.

REFERENCES

[1] A. Atserias, A. Dawar, and P. G. Kolaitis, *On preservation under homomorphisms and unions of conjunctive queries*, J. ACM, 53 (2006), pp. 208–237.

[2] M. Benedikt and L. Segoufin, *Towards a characterization of order-invariant queries over tame structures*, in Proceedings of the 19th International Workshop on Computer Science Logic, C.-H. L. Ong, ed., Lecture Notes in Comput. Sci. 3634, Springer, New York, 2005, pp. 276–291.

[3] A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt, *Approximation schemes for first-order definable optimization problems*, in Proceedings of the 21st IEEE Symposium on Logic in Computer Science, 2006, pp. 411–420.

[4] R. G. Downey and M. R. Fellows, *Parametrized Complexity*, Springer, New York, 1999.

[5] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, 2nd ed., Springer, Berlin, 1999.
[6] H.-D. Ebbinghaus, J. Flum, and W. Thomas, *Mathematical Logic*, 2nd ed., Springer, New York, 1994.
[7] P. Erdös and R. Rado, *Intersection theorems for systems of sets*, J. London Math. Soc., 35 (1960), pp. 85–90.
[8] J. Flum and M. Grohe, *Fixed-parameter tractability, definability, and model-checking*, SIAM J. Comput., 31 (2001), pp. 113–145.
[9] J. Flum, M. Frick, and M. Grohe, *Query evaluation via tree-decompositions*, J. ACM, 49 (2002), pp. 716–752.
[10] M. Frick and M. Grohe, *Deciding first-order properties of locally tree-decomposable structures*, J. ACM, 48 (2001), pp. 1184–1206.
[11] M. Grohe and G. Turán, *Learnability and definability in trees and similar structures*, Theory Comput. Syst., 37 (2004), pp. 193–220.
[12] Y. Gurevich, *Toward logic tailored for computational complexity*, in Computation and Proof Theory, M. Richter et al., eds., Lecture Notes in Math. 1104, Springer, Berlin, 1984, pp. 175–216.
[13] W. Hodges, *Model Theory*, Cambridge University Press, Cambridge, UK, 1993.
[14] L. Libkin, *Elements of Finite Model Theory*, Springer, Berlin, 2004.
[15] B. Rossman, *Existential positive types and preservation under homomorphisms*, in Proceedings of the 20th IEEE Symposium on Logic in Computer Science, 2005, pp. 467–476.
[16] W. W. Tait, *A counterexample to a conjecture of Scott and Suppes*, J. Symbolic Logic, 24 (1959), pp. 15–16.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1382–1410

CLOSEST SUBSTRING PROBLEMS WITH SMALL DISTANCES*

DÁNIEL MARX†

Abstract. We study two pattern matching problems that are motivated by applications in computational biology. In the CLOSEST SUBSTRING problem k strings $s_1, \ldots, s_k$ are given, and the task is to find a string s of length L such that each string s_i has a consecutive substring of length L whose distance is at most d from s. We present two algorithms that aim to be efficient for small fixed values of d and k: for some functions f and g, the algorithms have running time $f(d) \cdot n^{O(\log d)}$ and $g(d,k) \cdot n^{O(\log \log k)}$, respectively. The second algorithm is based on connections with the extremal combinatorics of hypergraphs. The CLOSEST SUBSTRING problem is also investigated from the parameterized complexity point of view. Answering an open question from [P. A. Evans, A. D. Smith, and H. T. Wareham, *Theoret. Comput. Sci.*, 306 (2003), pp. 407–430, M. R. Fellows, J. Gramm, and R. Niedermeier, *Combinatorica*, 26 (2006), pp. 141–167, J. Gramm, J. Guo, and R. Niedermeier, *Lecture Notes in Comput. Sci.* 2751, Springer, Berlin, 2003, pp. 195–209, J. Gramm, R. Niedermeier, and P. Rossmanith, *Algorithmica*, 37 (2003), pp. 25–42], we show that the problem is W[1]-hard even if both d and k are parameters. It follows as a consequence of this hardness result that our algorithms are optimal in the sense that the exponent of n in the running time cannot be improved to $o(\log d)$ or to $o(\log \log k)$ (modulo some complexity-theoretic assumptions). CONSENSUS PATTERNS is the variant of the problem where, instead of the requirement that each s_i has a substring that is of distance at most d from s, we have to select the substrings in such a way that the average of these k distances is at most δ. By giving an $f(\delta) \cdot n^9$ time algorithm, we show that the problem is fixed-parameter tractable. This answers an open question from [M. R. Fellows, J. Gramm, and R. Niedermeier, *Combinatorica*, 26 (2006), pp. 141–167].

Key words. closest substring, consensus pattern, parameterized complexity, fixed-parameter tractability, computational complexity

AMS subject classifications. 68W01, 68Q17

DOI. 10.1137/060673898

1. Introduction.

Computational biology applications provide a steady source of interesting stringology problems. In this paper we investigate two pattern matching problems that received considerable attention lately. Finding similar regions in multiple DNA, RNA, or protein sequences plays an important role in many applications, for example, in locating binding sites [27] and in finding conserved regions in unaligned sequences [24, 28, 34]. This task can be formalized the following way. Given k strings $s_1, \ldots, s_k$ over an alphabet Σ and an integer L, the task is to find a pattern that appears (possibly with some errors) in each string s_i. More precisely, we have to find a length L string s and a length L substring s_i' of each s_i such that s is "close" to every s_i'. We investigate two variants of the problem that differ in how closeness is defined. In the CLOSEST SUBSTRING problem the goal is to find a string s such that the Hamming-distance of s is at most d from every s_i'. An equally natural optimization goal is to minimize the sum of the distances of s from the substrings s_i': in the CONSENSUS PATTERNS problem we have to find a string s such that this sum is at most D. An equivalent way of formulating this problem is to require that the average distance is at most $\delta := D/k$.

*Received by the editors November 1, 2006; accepted for publication (in revised form) April 24, 2008; published electronically August 27, 2008. A preliminary version of the paper was presented at FOCS 2005. This research was partially supported by the Magyary Zoltán Felsőoktatási Közalapítvány and the Hungarian National Research Fund (grant OTKA 67651).

http://www.siam.org/journals/sicomp/38-4/67389.html

†Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Budapest H-1521, Hungary (dmarx@cs.bme.hu).

The CLOSEST SUBSTRING problem is NP-hard even in the special case when $\Sigma = \{0, 1\}$ and every string s_i has length L [18]. This means that most likely there are only exponential-time algorithms for the problem. However, an exponential-time algorithm can still be efficient if the exponential dependence is restricted to parameters that are typically small in practice (for example, the size of the alphabet or the maximum number of mismatches that we allow) and the running time depends polynomially on all the other parameters (such as the lengths of the strings and length of the pattern). Parameterized complexity is the systematic study of problem parameters with the goal of restricting the exponential increase of the running time to as few parameters of the instance as possible.

1.1. Parameterized complexity. In classical complexity theory, the running time of an algorithm is usually expressed as a function of the input size. Parameterized complexity provides a more refined, two-dimensional analysis of the running time: the goal is to study how the different parameters of the input instance affect the running time. We assume that every input instance has an integer number k associated to it, which will be called the *parameter.* For example, in the case of (the decision version of) MAXIMUM CLIQUE, we can associate to each instance the size of the clique that has to be found. When evaluating an algorithm for a parameterized problem we take into account both the input size n and the parameter k, and we try to express the running time as a function of n and k. The goal is to develop algorithms that run in *uniformly polynomial time*: the running time is $f(k) \cdot n^c$, where c is a constant and f is a (possibly exponential) function depending only on k. We call a parameterized problem *fixed-parameter tractable (FPT)* if such an algorithm exists. This means that the exponential increase of the running time can be restricted to the parameter k. It turns out that several NP-hard problems are fixed-parameter tractable, for example, MINIMUM VERTEX COVER, LONGEST PATH, and DISJOINT TRIANGLES. Therefore, for small values of k, the $f(k)$ term is just a constant factor in the running time, and the algorithms for these problems can be efficient even for large values of n. This has to be contrasted with algorithms that have a running time such as n^k: in this case the algorithm becomes practically useless for large values of n even if k is as small as 10. Analogously to NP-completeness in classical complexity, the theory of W[1]-hardness can be used to show that a problem is unlikely to be fixed-parameter tractable, which means that for every algorithm the parameter has to appear in the exponent of n. For example, for MAXIMUM CLIQUE and MINIMUM DOMINATING SET the running time of the best known algorithms is $n^{\Omega(k)}$, and the W[1]-hardness of these problems tells us that it is unlikely that an algorithm with a running time, say, $O(2^k \cdot n)$ can be found.

For a particular problem, there are many possible parameters that can be defined. For example, in the case of the MAXIMUM CLIQUE problem, the maximum degree of the graph, the genus of the graph, or the treewidth of the graph are also natural choices for the parameter. Different applications might suggest different parameters: whether a particular choice of parameter is relevant to an application depends on whether it can be assumed that this parameter is typically "small" in practice. The theory can be extended in a straightforward way to the case when there is more than one parameter: if there are two parameters k_1 and k_2, then the goal is to develop algorithms with a running time $f(k_1, k_2) \cdot n^c$. For more details, see section 2 and [12, 16].

1.2. Previous work on CLOSEST SUBSTRING. The NP-completeness of CLOSEST SUBSTRING was first shown by Frances and Litman [18] by considering an equiv-

alent problem in coding theory. Li, Ma, and Wang [30] presented a polynomial-time approximation scheme, but the running time of their approximation algorithm is prohibitive. Heuristic approaches for the problem are discussed in [6, 31, 32, 26]; see also the references therein.

Under the standard complexity-theoretic assumptions, the NP-completeness of Closest Substring means that any exact algorithm has to run in exponential time. However, there can be great qualitative differences between exponential-time algorithms: for example, it can be a crucial difference whether the running time is exponential in the length of the strings or in the number of the strings. This question was investigated in the framework of parameterized complexity by several papers. Formally, the following problem is studied:

Closest Substring

Input:
k strings $s_1, \ldots, s_k$ over an alphabet Σ and integers d and L.

Parameters:
k, $|\Sigma|$, d, L

Task:
Find a string s of length L such that for every $1 \leq i \leq k$, the string s_i has a length L consecutive substring s'_i with $d(s, s'_i) \leq d$.

The Hamming-distance of two strings w_1 and w_2 (i.e., the number of positions where they differ) is denoted by $d(w_1, w_2)$. The string s in the solution is called the *center string*. Observe that for a given center string s, it is easy to check in polynomial time whether the substrings s'_i exist: we have to try every length L consecutive substring of the strings s_i. Therefore, the real difficulty of the problem lies in finding the best center string s. We will denote by n the size of the input, which is an upper bound on the total length of the strings. In the following, "substring" will always mean consecutive substring (and not an arbitrary subsequence of the symbols).

The problem can be solved in polynomial time if k, d, or L is fixed to a constant. For every fixed value of L, the problem can be solved in polynomial time by enumerating all the $|\Sigma|^L = O(n^L)$ possible center strings. If d is a fixed constant, then the problem can be solved in polynomial time by making a guess at s'_1 (at most n possibilities) and then trying every center string s that is of distance at most d from s'_1 (at most $(|\Sigma|L)^d = O(n^{2d})$ possibilities). For fixed values of k, the problem can be solved in polynomial time as follows. First, we guess the k substrings s'_k (at most n^k possibilities). Now we have to find a center string s that is of distance at most d from each s'_i. This can be done by dynamic programming in $O(n^k)$ time or by applying the linear-time algorithm of Gramm, Niedermeier, and Rossmanith [21] for Closest String that is based on integer linear programming. Therefore, for fixed values of L, d, or k, the problem can be solved in polynomial time. However, the algorithms described above are not uniformly polynomial: the exponent of n increases as we consider greater fixed values. The parameterized complexity analysis of the problem can reveal whether it is possible to remove these parameters from the exponent of n and obtain algorithms with a running time such as $f(k) \cdot n^c$.

In [14] and [13] it is shown that the problem is W[1]-hard even if all three of k, d, and L are parameters. Therefore, if the size of the alphabet Σ is not bounded in the input, then we cannot hope for an efficient exact algorithm for the problem. Fortunately, in the computational biology applications the strings are typically DNA

TABLE 1.1
Complexity of CLOSEST SUBSTRING *with different parameterizations. Asterisk denotes the new results of this paper.*

Parameters	$\|\Sigma\|$ is constant	$\|\Sigma\|$ is parameter	$\|\Sigma\|$ is unbounded
d	W[1]-hard (*)	W[1]-hard (*)	W[1]-hard
d, k	W[1]-hard (*)	W[1]-hard (*)	W[1]-hard
k	W[1]-hard	W[1]-hard	W[1]-hard
L	FPT	FPT	W[1]-hard
d, k, L	FPT	FPT	W[1]-hard

or protein sequences, hence the number of different symbols is a small constant (4 or 20). Therefore, we will focus on the case when the size of Σ is a parameter. Restricting $|\Sigma|$ only does not make the problem tractable, since CLOSEST SUBSTRING is NP-hard even if the alphabet is binary. On the other hand, if $|\Sigma|$ and L are both parameters, then the problem becomes fixed-parameter tractable: we can enumerate and check all the $|\Sigma|^L$ possible center strings. However, if the strings are long (which is often the case in practical applications), then it makes much more sense to assume that the number of strings k or the distance constraint d are parameters. In [14] it is shown that CLOSEST SUBSTRING is W[1]-hard with parameter k, even if the alphabet is binary. However, the complexity of the problem with parameter d or with combined parameters d, k remained an open question.

1.3. New results for CLOSEST SUBSTRING**.** We show that the problem is W[1]-hard with combined parameters k and d, even if the alphabet is binary. This resolves an open question asked in [13, 14, 20, 21]. Therefore, even in the binary case, there is no $f(k,d) \cdot n^c$ algorithm for CLOSEST SUBSTRING (unless FPT = W[1]); the exponential increase cannot be restricted to the parameters k and d. This completes the parameterized complexity analysis of CLOSEST SUBSTRING (see Table 1.1; the results of this paper are marked with an asterisk.)

As a first step of the reduction, we introduce a technical problem called SET BALANCING, and prove W[1]-hardness for this problem. This part of the proof contains most of the new combinatorial ideas. The SET BALANCING problem is reduced to CLOSEST SUBSTRING by a reduction very similar to the one presented in [14].

We present two exact algorithms for the CLOSEST SUBSTRING problem. These algorithms can be efficient if d, or both d and k, are small (say, $o(\log n)$). The first algorithm runs in $|\Sigma|^{d(\log d+2)} n^{O(\log d)}$ time. Notice that this algorithm is not uniformly polynomial, but only the logarithm of the parameter appears in the exponent of n. Therefore, the algorithm might be efficient for small values of d. The second algorithm has running time $|\Sigma|^d \cdot 2^{kd} \cdot d^{O(d \log\log k)} \cdot n^{O(\log\log k)}$. Here the parameter k appears in the exponent of n, but $\log\log k$ is a very slowly growing function. This algorithm is based on defining certain hypergraphs and enumerating all the places where one hypergraph appears in the other. Using some results from extremal combinatorics, we develop techniques that can speed up the search for hypergraphs. It turns out that if hypergraph H has bounded fractional edge cover number, then we can enumerate in uniformly polynomial time all the places where H appears in some larger hypergraph G. This result might be of independent interest.

Notice that the running times of our two algorithms are incomparable. Assume

that $|\Sigma| = 2$. If $d = \log n$ and $k = \sqrt{n}$, then the running time of the first algorithm is $n^{O(\log\log n)} \cdot n^{O(\log\log n)} = n^{O(\log\log n)}$, while the second algorithm needs at least $2^{kd} = 2^{\sqrt{n}\log n} = n^{\sqrt{n}}$ steps, which can be much larger. On the other hand, if $d = k = \log\log n$, then the first algorithm runs in something like $n^{O(\log\log\log n)}$ time, while the running time of the second algorithm is dominated by the $n^{O(\log\log k)}$ factor, which is only $n^{O(\log\log\log\log n)}$.

Our W[1]-hardness proof combined with some recent results on subexponential algorithms shows that the two exact algorithms are in some sense best possible. The exponents are optimal: we show that if there is an $f_1(k, d, |\Sigma|) \cdot n^{o(\log d)}$ or an $f_2(k, d, |\Sigma|) \cdot n^{o(\log\log k)}$ algorithm for Closest Substring, then n-variable 3-Sat can be solved in $2^{o(n)}$ time. It is widely believed that 3-Sat does not have subexponential-time algorithms; this conjecture is called the Exponential Time Hypothesis (cf. [25, 35]).

1.4. Relation to approximability. Li, Ma, and Wang [30] studied the optimization version of Closest Substring, where the task is to find the smallest d that makes the problem feasible. They presented a *polynomial-time approximation scheme (PTAS)* for the problem: for every $\epsilon > 0$, there is an $n^{O(1/\epsilon^4)}$ time algorithm that produces a solution that is at most $(1+\epsilon)$-times worse than the optimum. This PTAS was improved to $n^{O(\log(1/\epsilon)/\epsilon^2)}$ time by Andoni, Indyk, and Pătraşcu [2] using an idea from an earlier version of this paper. However, such a PTAS becomes practically useless for large n, even if we ask for an error bound of, say, 20%. As pointed out in [11], there are numerous approximation schemes in the literature where the degree of the algorithm increases very rapidly as we decrease ϵ: having $O(n^{1,000,000})$ or worse for 20% error is not uncommon. Clearly, such approximation schemes do not yield efficient approximation algorithms. Nevertheless, these results show that there are no theoretical limitations on the approximation ratio that can be achieved.

An *efficient PTAS (EPTAS)* is an approximation scheme that produces a $(1+\epsilon)$-approximation in $f(\epsilon) \cdot n^c$ time for some constant c. If $f(\epsilon)$ is, e.g., $2^{1/\epsilon}$, then such an approximation scheme can be practical even for $\epsilon = 0.1$ and large n. A standard consequence of W[1]-hardness is that there is no EPTAS for the optimization version of the problem [7, 4]. Hence our hardness result shows that the approximation schemes of [30] and [2] for Closest Substring cannot be improved to an EPTAS.

1.5. Previous work on Consensus Patterns. The Consensus Patterns problem is the same as Closest Substring, but instead of minimizing the maximum distance between the center string and the substrings s'_i, now the goal is to minimize the sum of the distances. Similarly to Closest Substring, the problem is NP-complete and admits a PTAS [29]. Heuristic algorithms for Consensus Patterns and some generalizations are given in, e.g., [31, 23, 17, 33, 5].

We will study the decision version of the problem:

Consensus Patterns

Input:
k strings $s_1, \ldots, s_k$ over an alphabet Σ and integers D and L.

Parameters:
k, $|\Sigma|$, D, L

Task:
Find a string s of length L, and a length L consecutive substring s'_i of s_i for every $1 \leq i \leq k$ such that $\sum_{i=1}^{k} d(s, s'_i) \leq D$ holds.

TABLE 1.2
Complexity of CONSENSUS PATTERNS *with different parameterizations. Asterisk denotes the new results of this paper.*

Parameters	$\|\Sigma\|$ is constant	$\|\Sigma\|$ is parameter	$\|\Sigma\|$ is unbounded
δ	FPT (*)	FPT (*)	W[1]-hard
D	FPT (*)	FPT (*)	W[1]-hard
k	W[1]-hard	W[1]-hard	W[1]-hard
L	FPT	FPT	W[1]-hard
k, L	FPT	FPT	W[1]-hard
D, k, L	FPT	FPT	W[1]-hard

The string s in the solution is called the *median string*. Similarly to CLOSEST SUBSTRING, the problem is fixed-parameter tractable if both $|\Sigma|$ and L are parameters: we can enumerate and test every possible median string. Fellows, Gramm, and Niedermeier [14] showed that their hardness results for CLOSEST SUBSTRING can be adapted for CONSENSUS PATTERNS. Thus the problem is W[1]-hard with combined parameters L, k, D in the unbounded alphabet case, and W[1]-hard with parameter k in the binary alphabet case. The complexity of the problem in the binary alphabet case with parameter D or combined parameters k and D remained open.

Notice that if $D < k$, then the problem can be solved in polynomial time. To see this, observe that $\sum_{i=1}^{k} d(s, s'_i) \leq D < k$ is only possible if $d(s, s'_i) = 0$ for at least one i. This means that the median string is a substring of some s_i, thus a solution can be found by trying every length L substring of the input strings. Therefore, we can assume that $D \geq k$ holds in the problem instance. It follows that the complexity of CONSENSUS PATTERNS is the same with parameter D and with combined parameters k, D.

1.6. New results for CONSENSUS PATTERNS. We define and investigate the new parameter $\delta := D/k$, which is the average error that is allowed between the median string and the substrings s'_i. Parameterization by δ (and not by k) is relevant for applications where we want to find a solution with small average error, but the number of strings is allowed to be large.

By presenting an algorithm with running time $\delta^{O(\delta)} \cdot |\Sigma|^{\delta} \cdot n^9$, we show that CONSENSUS PATTERNS is fixed-parameter tractable if both $|\Sigma|$ and δ are parameters. The algorithm uses similar hypergraph techniques as the $f(k, d, |\Sigma|) \cdot n^{O(\log \log k)}$ time algorithm for CLOSEST SUBSTRING. However, a subtle difference in the combinatorics of the two problems allows us to replace the $O(\log \log k)$ term in the exponent of n with a constant.

Since parameter δ is not greater than parameter D, it follows trivially that the problem is fixed-parameter tractable with combined parameters $|\Sigma|$ and D. This settles another open question from [14]. The results for CONSENSUS PATTERNS are summarized in Table 1.2, with an asterisk marking the results of this paper.

1.7. Organization. The paper is organized as follows. Section 2 briefly reviews the most important notions of parameterized complexity. The first algorithm for CLOSEST SUBSTRING is presented in section 3. In section 4 we discuss techniques for finding one hypergraph in another. In section 5 we present the second algorithm

for Closest Substring. This section introduces a new hypergraph property called *half-covering,* which plays an important role in the algorithm. The algorithm for Consensus Patterns is presented in section 6. We define the Set Balancing problem in section 7 and prove that it is W[1]-hard. In section 8 the Set Balancing problem is used to show that Closest Substring is W[1]-hard with combined parameters d and k. We conclude the paper with a summary in section 9.

Algorithm 1 (section 3) and Algorithm 2 (sections 4 and 5) for the Closest Substring problem are independent from each other. The algorithm for Consensus Patterns (section 6) is very similar to the algorithm in section 5, but is presented in a self-contained way. The algorithm of section 6 is also based on the hypergraph techniques developed in section 4.

The hardness results in sections 7 and 8 are independent from the algorithms; the reductions can be understood without the preceding sections. However, the combinatorics of the reduction in section 7 has subtle connections with the half-covering property discussed in section 5. In some sense, section 5 explains why the reduction in section 7 has to be done that way.

2. Parameterized complexity. We follow [16] for the standard definitions of parameterized complexity. Let Σ be a finite alphabet. A decision problem is represented by a set $Q \subseteq \Sigma^*$ of strings over Σ. A *parameterization* of a problem is a polynomial-time computable function $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized decision problem* is a pair (Q, κ), where $Q \subseteq \Sigma^*$ is an arbitrary decision problem and κ is a parameterization. Intuitively, we can imagine a parameterized problem as a decision problem where each input instance $x \in \Sigma^*$ has a positive integer $\kappa(x)$ associated with it. A parameterized problem (Q, κ) is FPT if there is an algorithm that decides whether $x \in Q$ in time $f(\kappa(x)) \cdot |x|^c$ for some constant c and computable function f. An algorithm with such a running time is called an *fpt-time algorithm* or simply *fpt-algorithm.*

Many NP-hard problems were investigated in the parameterized complexity literature, with the goal of identifying fixed-parameter tractable problems. There is a powerful toolbox of techniques for designing fpt-algorithms: kernelization, bounded search trees, color coding, well-quasi ordering—just to name some of the more important ones. On the other hand, certain problems resisted every attempt at obtaining fpt-algorithms. Analogously to NP-completeness in classical complexity, the theory of W[1]-hardness can be used to give strong evidence that certain problems are unlikely to be fixed-parameter tractable. We omit the somewhat technical definition of the complexity class W[1]; see [12, 16] for details. Here it will be sufficient to know that there are several problems, including Maximum Clique, that were proved to be W[1]-hard. Furthermore, we also expect that there is no $n^{o(k)}$ (or even $f(k) \cdot n^{o(k)}$) algorithm for Maximum Clique: recently it was shown that if there exists an $f(k) \cdot n^{o(k)}$ algorithm for n-vertex Maximum Clique, then n-variable 3-Sat can be solved in time $2^{o(n)}$ (see [8] and [15]).

To prove that a parameterized problem (Q', κ') is W[1]-hard, we have to present a parameterized reduction from a known W[1]-hard problem (Q, κ) to (Q', κ'). A *parameterized reduction* from problem (Q, κ) to problem (Q', κ') is a function that transforms a problem instance x of Q into a problem instance x' of Q' in such a way that

1. $x' \in Q'$ if and only if $x \in Q$,
2. $\kappa'(x)$ can be bounded by a function of $\kappa(x)$, and
3. the transformation can be computed in time $f(\kappa(k)) \cdot |x|^c$ for some constant c and function $f(k)$.

It is easy to see that if there is a parameterized reduction from (Q, κ) to (Q', κ'), and (Q', κ') is fixed-parameter tractable, then it follows that (Q, κ) is fixed-parameter tractable as well. The most important difference between parameterized reductions and classical polynomial-time many-to-one reductions is the second requirement: in most NP-completeness proofs the new parameter is not a function of the old parameter. Therefore, finding parameterized reductions is usually more difficult and the constructions have a somewhat different flavor than classical reductions.

There are many possible parameters that can be defined for a particular problem; different parameters can be relevant in different applications. Usually, the parameter is either some property of the solution we seek (number of vertices, quality of the solution, etc.) or describes some aspect of the input structure (degree/genus/treewidth of the input graph, number of variables/clauses in the input formula, etc.). The complexity of the problem can be different with different parameters. Observe that if parameter k_1 is never greater than parameter k_2, then the problem cannot be easier with parameter k_1 than with k_2: an $f(k_1) \cdot n^c$ time algorithm implies the existence of an $f(k_2) \cdot n^c$ time algorithm.

In some cases we want to investigate the complexity of the problem by considering two or more parameters at the same time; i.e., we assume that both parameter k_1 and parameter k_2 are typically small in applications. The problem is fixed-parameter tractable with combined parameters k_1 and k_2 if there is an algorithm with running time $f(k_1, k_2) \cdot n^{O(1)}$. For a particular problem, we can investigate several different combinations of parameters. In general, if we increase the set of parameters, then we cannot make the problem harder: for example, if the problem is fixed-parameter tractable with parameter k_1, then clearly it is fixed-parameter tractable with combined parameters k_1 and k_2.

3. Finding generators. In this section we present an algorithm for Closest Substring that has running time proportional to roughly $n^{\log d}$. The algorithm is based on the following observation: if all the strings $s'_1, \ldots, s'_k$ agree at some position p in the solution, then we can safely assume that the same symbol appears at the pth position of the center string s. However, if we look at only a subset of the strings s'_1, $\ldots$, s'_k, then it is possible that they all agree at some position, but the center string contains a different symbol at this position. We will be interested in sets of strings that do not have this problem.

Definition 3.1. *Let $G = \{g_1, g_2, \ldots, g_\ell\}$ be a set of length L strings. We say that G is a* generator *of the length L string s if whenever every g_i has the same character at some position p, then string s has this character at position p. The* size *of the generator is ℓ, the number of strings in G. The* conflict size *of the generator is the number of those positions where not all of the strings g_i have the same character.*

As we have argued above, it can be assumed that the strings $s'_1, \ldots, s'_k$ of a solution form a generator of the center string s. Furthermore, these strings have a subset of size at most $\log d + 2$ that is also a generator.

Lemma 3.2. *If an instance of* Closest Substring *is solvable, then there is a solution s that has a generator G having the following properties:*

1. *each string in G is a substring of some s_i,*
2. *G has size at most $\log d + 2$,*
3. *the conflict size of G is at most $d(\log d + 2)$.*

Proof. Let $s, s'_1, \ldots, s'_k$ be a solution such that $\sum_{i=1}^{k} d(s, s'_i)$ is minimal. We prove by induction that for every j we can select a subset G_j of j strings from $\{s'_1, \ldots, s'_k\}$ such that there are less than $(d+1)/2^{j-1}$ *bad positions* where the strings in G_j all

CLOSEST SUBSTRING-1$(k, L, d, (s_1, \ldots, s_k))$
1. Construct S, the set of all length L substrings of the input strings.
2. **for** every $G \subseteq S$ with $|G| = \log d + 2$ **do**
3. **if** the strings in G agree on all but at most $d(\log d + 2)$ positions
4. **for** every string s that is generated by G **do**
5. **if** $\max_{i=1}^{k} \min_{\{s'_i \text{ is a substring of } s_i\}} d(s, s'_i) \leq d$ **then**
6. s is a solution, STOP.
7. There is no solution, STOP.

FIG. 3.1. *Algorithm* 1 *for* CLOSEST SUBSTRING.

agree, but this common character is different from the character in s at this position. The lemma follows from $j = \lceil \log(d+1) \rceil + 1 \leq \log d + 2$: the set G_j has no bad positions, hence it is a generator of s. Furthermore, each string in G_j is at distance at most d from s, thus the conflict size of G_j can be at most $d(\log d + 2)$.

For the case $j = 1$ we can set $G_1 = \{s'_1\}$, since s'_1 differs from s at not more than d positions. Now assume that the statement is true for some j. Let P be the set of bad positions, where the j strings in G_j agree, but they differ from s. We claim that there is some string s'_t in the solution and a subset $P' \subseteq P$ with $|P'| > |P|/2$ such that s'_t differs from all the strings in G_j at every position of P'. If this is true, then we add s'_t to the set G_j to obtain G_{j+1}. Only the positions in $P \setminus P'$ are bad for the set G_{j+1}: for every position p in P', the strings cannot all agree at p, since s'_t do not agree with the other strings at this position. Thus there are at most $|P \setminus P'| < |P|/2 < (d+1)/2^j$ bad positions, completing the induction.

Assume that there is no such string s'_t. In this case we modify the center string s the following way: for every position $p \in P$, let the character at position p be the same as in string s'_1. Denote by s^* the new center string. We show that $d(s^*, s'_i) \leq d(s, s'_i) \leq d$ for every $1 \leq i \leq k$, hence s^* is also a solution. By assumption, every string s'_i in the solution agrees with s'_1 on at least $|P|/2$ positions of P. Therefore, if we replace s with s^*, the distance of s'_i from the center string decreases on at least $|P|/2$ positions, and the distance can increase only on the remaining at most $|P|/2$ positions. Therefore, $d(s^*, s'_i) \leq d(s, s'_i)$ follows. Furthermore, $d(s^*, s'_1) = d(s, s'_1) - |P|$ implies $\sum_{i=1}^{k} d(s^*, s'_i) < \sum_{i=1}^{k} d(s, s'_i)$, which contradicts the minimality of s. □

We note that Lemma 3.2 (appearing in an earlier version of this paper) was used by Andoni, Indyk, and Pătraşcu [2] to improve the running time of the PTAS of Li, Ma, and Wang [30] to $n^{O(\log(1/\epsilon)/\epsilon^2)}$ time.

Our algorithm first creates a set S containing all the length L substrings of s_1, $\ldots$, s_k. For every subset $G \subseteq S$ of $\log d + 2$ strings, we check whether G generates a center string s that solves the problem. Since $|S| \leq n$, there are at most $n^{\log d + 2}$ possibilities to try. By Lemma 3.2 we have to consider only those generators whose conflict size is at most $d(\log d + 2)$, hence at most $|\Sigma|^{d(\log d+2)}$ possible center strings have to be tested for each G.

THEOREM 3.3. CLOSEST SUBSTRING *can be solved in time* $|\Sigma|^{d(\log d+2)} n^{\log d + O(1)}$.

Proof. The algorithm is presented in pseudocode in Figure 3.1. Let S be the set of all length L substrings in s_1, $\ldots$, s_k, clearly $|S| \leq n$ (recall that n is the total length of the input). If there is a solution s, then Lemma 3.2 ensures that there is a subset $G \subseteq S$ of size at most $\log d + 2$ that generates s. We test every size $\log d + 2$ subset of S to see whether it can generate a solution. First, by Lemma 3.2 we can restrict our attention to those G where the strings in G agree on all but at

most $d(\log d + 2)$ positions. If such a G generates a string s, then the characters of s are determined everywhere except on the conflicting positions of G. Therefore, G can be the generator of at most $|\Sigma|^{d(\log d+2)}$ different strings. We try all the possible combinations of assigning characters on the conflicting positions of G and check for each resulting string s whether it is true for every $1 \le i \le k$ that there is a substring s_i' of s_i such that $d(s, s_i') \le d$. This method will eventually find a solution, if one exists.

We try $O(n^{\log d+2})$ different subsets G (line 2), and each G can generate at most $|\Sigma|^{d(\log d+2)}$ different center strings s (line 4). It can be checked in polynomial time whether a center string s is a solution (line 5), hence the total running time is $|\Sigma|^{d(\log d+2)} n^{\log d+O(1)}$. □

We remark here that the algorithm can be made slightly more efficient: it is sufficient to check those generators where the $\log d + 2$ strings come from different strings s_i. However, this observation does not improve the asymptotics of the running time, and we did not want to complicate the notation to accommodate this improvement.

4. Finding hypergraphs. Let us recall some standard definitions concerning hypergraphs. A *hypergraph* $H(V_H, E_H)$ consists of a set of *vertices* V_H and a collection of *edges* E_H, where each edge is a subset of V_H. Let $H(V_H, E_H)$ and $G(V_G, E_G)$ be two hypergraphs. We say that H *appears at* $V' \subseteq V_G$ *as partial hypergraph* if there is a bijection π between the elements of V_H and V' such that for every edge $E \in E_H$ we have that $\pi(E)$ is an edge of G (where the mapping π is extended to the edges the obvious way). For example, if H has the edges $\{1, 2\}$, $\{2, 3\}$, and G has the edges $\{a, b\}$, $\{b, c\}$, $\{c, d\}$, then H appears as a partial hypergraph at $\{a, b, c\}$ and at $\{b, c, d\}$. We say that H *appears at* $V' \subseteq V_G$ *as subhypergraph* if there is such a bijection π where for every $E \in E_H$, there is an edge $E' \in E_G$ with $\pi(E) = E' \cap V'$. For example, let the edges of H be $\{1, 2\}$, $\{2, 3\}$, and let the edges of G be $\{a, c, d\}$, $\{b, c, d\}$. Now H does not appear in G as partial hypergraph, but H appears as subhypergraph at $\{a, b, c\}$ and at $\{a, b, d\}$. If H appears at some $V' \subseteq V_G$ as partial hypergraph, then it appears there as subhypergraph as well.

A *stable set* in $H(V_H, E_H)$ is a subset $S \subseteq V_H$ such that every edge of H contains at most one element from S. The *stable number* $\alpha(H)$ is the size of the largest stable set in H. A *fractional stable set* is an assignment ϕ: $V_H \to [0, 1]$ such that $\sum_{v \in E} \phi(v) \le 1$ for every edge E of H. The *fractional stable number* $\alpha^*(H)$ is the maximum of $\sum_{v \in V_H} \phi(v)$ taken over all fractional stable sets ϕ. The incidence vector of a stable set is a fractional stable set, hence $\alpha^*(H) \ge \alpha(H)$.

An *edge cover* of H is a subset $E' \subseteq E_H$ such that each vertex of V_H is contained in at least one edge of E'. The *edge cover number* $\rho(H)$ is the size of the smallest edge cover in H. (The hypergraphs considered in this paper do not have isolated vertices, hence every hypergraph has an edge cover.) A *fractional edge cover* is an assignment ψ: $E_H \to [0, 1]$ such that $\sum_{E: v \in E} \psi(E) \ge 1$ for every vertex v. The *fractional cover number* $\rho^*(H)$ is the minimum of $\sum_{E \in E_H} \psi(E)$ taken over all fractional edge covers ψ, clearly $\rho^*(H) \le \rho(H)$. It follows from the duality theorem of linear programming that $\alpha^*(H) = \rho^*(H)$ for every hypergraph H with no isolated vertices.

Friedgut and Kahn [19] determined the maximum number of times a hypergraph $H(V_H, E_H)$ can appear as partial hypergraph in a hypergraph G with m edges. That is, we are interested in the maximum number of different subsets $V' \subseteq V_G$ where H can appear in G. A trivial upper bound is $m^{|E_H|}$: if we fix $\pi(E) \in E_G$ for each edge $E \in E_H$, then this uniquely determines $\pi(V_H)$. This trivial bound can be improved to $m^{\rho(H)}$: if edges $E_1, E_2, \ldots, E_{\rho(H)}$ cover every vertex of V_H, then by fixing $\pi(E_1)$,

$\pi(E_2), \ldots, \pi(E_{\rho(H)})$ the set $\pi(V_H)$ is determined. The result of Friedgut and Kahn says that ρ can be replaced with the (possibly smaller) ρ^*.

THEOREM 4.1 (see [19]). *Let H be a hypergraph with fractional cover number $\rho^*(H)$, and let G be a hypergraph with m edges. There are at most $|V_H|^{|V_H|} \cdot m^{\rho^*(H)}$ different subsets $V' \subseteq V_G$ such that H appears in G at V' as partial hypergraph. Furthermore, for every H and sufficiently large m, there is a hypergraph with m edges where H appears $m^{\rho^*(H)}$ times.*

We remark here that Theorem 4.1 was proved for the special case of graphs in the first published paper of Alon [1].

To appreciate the strength of Theorem 4.1, it is worth pointing out that $\rho^*(H)$ can be much smaller than $\rho(H)$, hence the upper bound can be much stronger than $m^{\rho(H)}$. For example, consider the hypergraph where the vertices correspond to the k-element subsets of $\{1, 2, \ldots, n\}$, and edge E_i $(1 \leq i \leq n)$ contains those vertices that correspond to sets containing i. Now $\rho = n - k + 1$: if we select less than $n - k + 1$ edges, then there is a k-element set that is not covered by the less than $n - k + 1$ elements corresponding to the edges. On the other hand, we can construct a fractional edge cover of total weight n/k by assigning weight $1/k$ to each edge. This is a fractional edge cover, since each vertex is contained in exactly k edges. Therefore, the ratio $\rho/\rho^* = (n - k + 1)/(n/k)$ can be arbitrarily large.

Theorem 4.1 does not remain valid if we replace "partial hypergraph" with "subhypergraph." For example, let H contain only one edge $\{1, 2\}$, and let G have one edge E of size ℓ. Now H appears at each of the $\binom{\ell}{2}$ two element subsets of E as subhypergraph. However, if we bound the size of the edges in G, then we can state a subhypergraph analog of Theorem 4.1.

COROLLARY 4.2. *Let H be a hypergraph with fractional cover number $\rho^*(H)$, and let G be a hypergraph with m edges, each of size at most ℓ. Hypergraph H can appear in G as subhypergraph at most $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot m^{\rho^*(H)}$ times.*

Proof. Let $G'(V_G, E_{G'})$ be a hypergraph over V_G where $E' \in E_{G'}$ if and only if $|E'| \leq |V_H|$ and E' is a subset of some edge $E \in E_G$. An edge of G contributes at most $\ell^{|V_H|}$ edges to G', hence G' has at most $\ell^{|V_H|} \cdot m$ edges. If H appears as subhypergraph at $V' \subseteq V_G$ in G, then H appears as partial hypergraph at V' in G'. By Theorem 4.1, hypergraph H can appear at most $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot m^{\rho^*(H)}$ times in G' as partial hypergraph, proving the lemma. □

Given hypergraphs $H(V_H, E_H)$ and $G(V_G, E_G)$, we would like to find all the places $V' \subseteq V_G$ in G where H appears as subhypergraph. If there are t such places, then obviously we cannot enumerate all of them in less than t steps. Therefore, our aim is to find an algorithm with running time polynomial in the upper bound $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot m^{\rho^*(H)}$ on t given by Corollary 4.2. The proof of Theorem 4.1 is not algorithmic (it is based on Shearer's Lemma [10], which is proved by entropy arguments), hence it does not directly imply an efficient way of enumerating all the places where H appears. However, in Theorem 4.3, we show that there is a very simple algorithm for enumerating all these places. Corollary 4.2 is used to bound the running time of the algorithm. This result might be useful in other applications as well.

THEOREM 4.3. *Let $H(V_H, E_H)$ be a hypergraph with fractional cover number $\rho^*(H)$, and let $G(V_H, E_H)$ be a hypergraph where each edge has size at most ℓ. There is an algorithm that enumerates in time $|V_H|^{O(V_H)} \cdot \ell^{|V_H|\rho^*(H)+1} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$ every subset $V' \subseteq V_G$ where H appears in G as subhypergraph.*

Proof. Let $V_H = \{1, 2, \ldots, r\}$. For each $1 \leq i \leq r$, let $H_i(V_i, E_i)$ be the subhy-

```
FIND-SUBHYPERGRAPH(H,G)
 1. L_1 := all the places where H_1 appears in G
 2. for i := 1 to r − 1 do
 3.   for every X ∈ L_i do
 4.     for every x ∈ V_G \ X do
 5.       X' := X ∪ {x}
 6.       if X' ∉ L_{i+1} then
 7.         for every bijection π : V_{i+1} → X' do
 8.           for every E ∈ E_{i+1} do
 9.             for every E' ∈ E_G do
10.               if π(E) = E' ∩ X' then
11.                 Go to step 8, select next E
12.             Go to step 7, select next π
13.         Add X' to L_{i+1}
14. return L_r
```

FIG. 4.1. *Algorithm for enumerating all the places where hypergraph H appears in G as subhypergraph.*

pergraph of H induced by $V_i = \{1, 2, \ldots, i\}$; that is, if E is an edge of H, then $E \cap V_i$ is an edge of H_i. For each $i = 1, 2, \ldots, r$, we find all the places where H_i appears in G as subhypergraph. Since $H = H_r$ this method will solve the problem.

For $i = 1$ the problem is trivial, since V_i has only one vertex. Assume now that we have a list L_i of all the i-element subsets of V_G where H_i appears as subhypergraph. The important observation is that if H_{i+1} appears as subhypergraph at some $(i+1)$-element subset $V' \subseteq V_G$, then V' has an i-element subset $V'' \in L_i$ where H_i appears as subhypergraph. Thus for each set $X \in L_i$, we try all the $|V_G \setminus X|$ different ways of extending X to an $(i+1)$-element set X', and check whether H_{i+1} appears at X' as subhypergraph. This can be checked by trying all the $(i+1)!$ possible bijections π between V_{i+1} and X', and by checking for each edge E of H_{i+1} whether there is an edge E' in G with $\pi(E) = E' \cap X'$.

The structure of the algorithm is presented in Figure 4.1. Let us make a rough estimate of the running time. The loop in step 2 consists of $|V_H| - 1$ iterations. Notice first that $\rho^*(H_i) \leq \rho^*(H)$, since a fractional edge cover of H can be used to obtain a fractional edge cover of H_i. Therefore, by Corollary 4.2, each list L_i has size at most $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot |E_G|^{\rho^*(H)}$, which bounds the maximum number of times the loop in step 3 is iterated. When we determine the list L_{i+1}, we have to check for at most $|L_i| \cdot |V_G|$ different sets X' of size $i+1$ whether H_{i+1} appears at X' as subhypergraph (step 4). Adding duplicate entries into the list L_{i+1} should be avoided; otherwise we would not have the bound on the size of L_i claimed above. Therefore, in step 6, we check whether X' is already in L_i. If the list L_i is implemented as a trie structure, then the test in step 6 can be performed in time $O(|V_H| \cdot |V_G|)$. The trie structure can increase the time required to enumerate the list L_i by a factor of $|V_H|$. Checking one X' requires us to test $(i+1)!$ different bijections π (step 7). Testing a bijection π means that for each $E \in E_{i+1}$ (step 8), it has to be checked whether there is a corresponding $E' \in E_G$ (step 9) such that $E' \cap X' = E$ (step 10). Hypergraph H has at most $2^{|V_H|}$ edges, hence the loop of step 8 is iterated at most $2^{|V_H|}$ times. If the edges of G are represented as lists of vertices, then the check in step 10 can be implemented in $O(\ell)$ time. Adding a new element into the trie structure (step 13)

can be done in $O(|V_H| \cdot |V_G|)$ time.

The dominating part of the running time comes from steps 7–13, which are repeated $|V_H|^{O(V_H)} \cdot \ell^{|V_H|\rho^*(H)} \cdot |E_G|^{\rho^*(H)} \cdot |V_G|$ times. The loop in steps 7–12 takes $O(|V_H|! \cdot 2^{|V_H|} \cdot |E_G| \cdot \ell) = |V_H|^{O(V_H)} \cdot |E_G| \cdot \ell$ time, while step 13 takes $O(|V_H| \cdot |V_G|)$ time. Therefore, the total running time can be bounded by $|V_H|^{O(V_H)} \cdot \ell^{|V_H|\rho^*(H)+1} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$. □

We can use a similar technique to find all the places where H appears in G as partial hypergraph. This result is not used in this paper, but might be useful in some other applications.

COROLLARY 4.4. *Let $H(V_H, E_H)$ be a hypergraph with fractional cover number $\rho^*(H)$, and let $G(V_G, E_G)$ be an arbitrary hypergraph. There is an algorithm that enumerates in time $|V_H|^{O(|V_H|\rho^*(H))} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$ all the subsets $V' \subseteq V_G$ where H appears in G as partial hypergraph.*

Proof. We can throw away from G every edge larger than $|V_H|$ without changing the problem. Now Theorem 4.3 can be used to find in time $|V_H|^{O(|V_H|\rho^*(H))} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$ the list L of all the subsets $V' \subseteq V_G$ where H appears in G as subhypergraph. If H appears at V' as partial hypergraph, then this is only possible if H appears at V' as subhypergraph. Therefore, the algorithm returns a list that is a superset of the expected result. Let us modify step 10 of the algorithm of Theorem 4.3 such that in iteration $i = r - 1$ it tests $\pi(E) = E'$ instead of $\pi(E) = E' \cap X'$. This ensures that L_r contains only those positions where H appears as partial hypergraph. □

5. Half-covering and the CLOSEST SUBSTRING problem. The following hypergraph property plays a crucial role in our second algorithm for the CLOSEST SUBSTRING problem.

DEFINITION 5.1. *We say that a hypergraph $H(V, E)$ has the* half-covering *property if for every nonempty subset $Y \subseteq V$ there is an edge $X \in E$ with $|X \cap Y| > |Y|/2$.*

Theorem 4.3 says that finding a hypergraph H is easy if H has small fractional cover number. In our algorithm for the CLOSEST SUBSTRING problem (described later in this section), we have to find hypergraphs satisfying the half-covering property. The following combinatorial lemma shows that such hypergraphs have small fractional cover number, hence they are easy to find.

LEMMA 5.2. *If $H(V, E)$ is a hypergraph with m edges satisfying the half-covering property, then the fractional cover number ρ^* of H is $O(\log \log m)$.*

Proof. The fractional cover number equals the fractional stable number, thus there is a function ϕ: $V \to [0, 1]$ such that $\sum_{v \in X} \phi(v) \leq 1$ holds for every edge $X \in E$, and $\sum_{v \in V} \phi(v) = \rho^*$. The lemma is proved by a probabilistic argument: we show that if a random subset $Y \subseteq V$ is selected such that the probability of selecting a vertex v is proportional to $\phi(v)$, then with nonzero probability no edge covers more than half of Y, unless the number of edges is double exponential in ρ^*. The idea is to show that for each edge X, the expected size of Y is ρ^* times the expected size of $Y \cap X$, hence the Chernoff Bound can be used to show that there is only a small probability that X covers more than half of Y. However, the straightforward application of this idea gives only an exponential lower bound on the number of edges. To improve the bound to double exponential, we have to restrict our attention to a suitable subset T of vertices, and scale the probabilities appropriately.

Let $v_1, v_2, \ldots, v_{|V|}$ be an ordering of the vertices by nonincreasing value of $\phi(v_i)$. First, we give a bound on the sum of the largest $\phi(v_i)$'s.

PROPOSITION 5.3. $\sum_{j=1}^{i} \phi(v_j) \leq -4 \log_2 \phi(v_i) + 4$ *holds for every* $1 \leq i \leq |V|$.

Proof. The proof is by induction on i. Since $\phi(v_1) \le 1$, the claim is trivial for $i = 1$. For an arbitrary $i > 1$, let $i' \le i$ be the smallest value such that $\phi(v_{i'}) \le 2\phi(v_i)$. By assumption, there is an edge X of H that covers more than half of the set $S = \{v_{i'}, \dots, v_i\}$. Every weight in S is at least $\phi(v_i)$, hence X can cover at most $1/\phi(v_i)$ elements of S. Thus $|S| \le 2/\phi(v_i)$, and $\sum_{j=i'}^{i} \phi(v_j) \le 4$ follows from the fact that $\phi(v_j) \le 2\phi(v_i)$ for $i' \le j \le i$. If $i' = 1$, then we are done. Otherwise $\sum_{j=1}^{i'-1} \phi(v_j) \le -4\log_2 \phi(v_{i'-1}) + 4 < -4(\log_2 \phi(v_i) + 1) + 4$ follows from the induction hypothesis and from $\phi(v_{i'-1}) > 2\phi(v_i)$. Therefore, $\sum_{j=1}^{i} \phi(v_j) = \sum_{j=1}^{i'-1} \phi(v_j) + \sum_{j=i'}^{i} \phi(v_j) \le -4\log_2 \phi(v_i) + 4$, what we had to show. □

In the rest of the proof, we assume that ρ^* is sufficiently large, say $\rho^* \ge 100$. Let i be the largest value such that $\sum_{j=i}^{|V|} \phi(v_j) \ge \rho^*/2$. By the definition of i, $\sum_{j=i+1}^{|V|} \phi(v_j) < \rho^*/2$, hence $\sum_{j=1}^{i} \phi(v_j) \ge \rho^*/2$. Thus by Proposition 5.3, the weight of v_i (and every v_j with $j \ge i$) is at most $2^{-(\rho^*/2-4)/4} \le 2^{-\rho^*/10}$ (assuming that ρ^* is sufficiently large). Define $T := \{v_i, \dots, v_{|V|}\}$, and let us select a random subset $Y \subseteq T$: independently each vertex $v_j \in T$ is selected into Y with probability $p(v_j) := 2^{\rho^*/10} \cdot \phi(v_j) \le 1$. We show that if H does not have $2^{2^{\Omega(\rho^*)}}$ edges, then with nonzero probability every edge of H covers at most half of Y, contradicting the assumption that H satisfies the half-covering property.

The size of Y is the sum of $|T|$ independent 0-1 random variables. The expected value of this sum is $\mu = \sum_{j=i}^{|V|} p(v_j) = 2^{\rho^*/10} \cdot \sum_{j=i}^{|V|} \phi(v_j) \ge 2^{\rho^*/10} \cdot \rho^*/2$. We show that with nonzero probability $|Y| \ge \mu/2$, but $|X \cap Y| \le \mu/4$ for every edge X. To bound the probability of the bad events, we use the following form of the Chernoff Bound.

THEOREM 5.4 (see [3]). *Let $X_1, X_2, \dots, X_n$ be independent 0-1 random variables with $\Pr[X_i = 1] = p_i$. Denote $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then*

$$\Pr[X \le (1-\beta)\mu] \le \exp(-\beta^2\mu/2) \quad \text{for } 0 < \beta \le 1,$$
$$\Pr[X \ge (1+\beta)\mu] \le \begin{cases} \exp(-\beta^2\mu/3) \text{ for } 0 < \beta \le 1, \\ \exp(-\beta^2\mu/(2+\beta)) \text{ for } \beta > 1. \end{cases}$$

Thus by setting $\beta = \frac{1}{2}$, the probability that Y is too small can be bounded as

$$\Pr[|Y| \le \mu/2] \le \exp(-\mu/8).$$

For each edge X, the random variable $|X \cap Y|$ is the sum of $|X \cap T|$ independent 0-1 random variables. The expected value of this sum is $\mu_X = \sum_{v \in X \cap T} p(v) = 2^{\rho^*/10} \cdot \sum_{v \in X \cap T} \phi(v) \le 2^{\rho^*/10} \le \mu/(\rho^*/2)$, where the first inequality follows from the fact that ϕ is a fractional stable set, hence the total weight X can cover is at most 1. Notice that if ρ^* is sufficiently large, then the expected size of $X \cap Y$ is much smaller than the expected size of Y. We want to bound the probability that $|X \cap Y|$ is at least $\mu/4$. Setting $\beta = (\mu/4)/\mu_X - 1 \ge \rho^*/8 - 1$, the Chernoff bound gives

$$\begin{aligned} \Pr\big[|X \cap Y| \ge \mu/4\big] &= \Pr\big[|X \cap Y| \ge (1+\beta)\mu_X\big] \le \exp(-\beta^2\mu_X/(2+\beta)) \\ &\le \exp(-\beta^2\mu_X/(2\beta)) = \exp(-\mu/8 + \mu_X/2) \le \exp(-\mu/16). \end{aligned}$$

Here we assumed that ρ^* is sufficiently large such that $\beta \ge 2$ (second inequality) and $\mu_X/2 \le \mu/16$ (third inequality) hold. If H has m edges, then the probability that $|Y| \le \mu/2$ holds or an edge X covers at least $\mu/4$ vertices of Y is at most

$$\exp(-\mu/8) + m \cdot \exp(-\mu/16) \le (m+1)\exp(-2^{\rho^*/10} \cdot \rho^*/32) \le m \cdot 2^{-2^{\Omega(\rho^*)}}. \tag{5.1}$$

If H satisfies the half-covering property, then for every Y there has to be at least one edge that covers more than half of Y. Therefore, the upper bound (5.1) cannot be smaller than 1. This is only possible if m is $2^{2^{\Omega(\rho^*)}}$, and it follows that $\rho^* = O(\log\log m)$, what we had to show. □

The following example shows that the bound $O(\log\log m)$ is tight in Lemma 5.2. Fix an integer r, and consider the 2^r-1 vertices $V := \{1,2,\ldots,2^r-1\}$. We construct a hypergraph that has not more than 2^{2^r} edges and its fractional cover number is at least $r/2$. Given a finite nonempty set F of positive integers, define $\mathrm{up}(F)$ to be the largest $\lceil(|F|+1)/2\rceil$ elements of this set. For every nonempty subset X of V, add the edge $\mathrm{up}(X)$ to the set system. This results in not more than $2^{2^r-1}-1$ edges. (There will be lots of parallel edges, but let us not worry about that.) Clearly, the set system satisfies the half-covering property: for every set Y, the set $\mathrm{up}(Y)$ covers more than half of Y.

We claim that the fractional cover number of the hypergraph is at least $r/2$. This can be proved by presenting a fractional stable set of weight $r/2$. Let the weight of v_1 be $1/2$, the weight of v_2 and v_3 be $1/4$, the weight of v_4, v_5, v_6, v_7 be $1/8$, and so on. It is easy to see that the total weight assigned is exactly $r/2$. Furthermore, observe that the weight of v_t is at most $1/(t+1)$ (there is equality if t is of the form 2^k-1; otherwise v_t is strictly smaller). To show that this weight assignment is indeed a fractional stable set, suppose that the vertices covered by some edge have total weight of more than 1. Let this edge be $\mathrm{up}(X)$ for some subset X of V. Let t be the smallest element in $\mathrm{up}(X)$. Vertex v_t has weight of at most $1/(t+1)$, and if t is the smallest element in $\mathrm{up}(X)$, then $\mathrm{up}(X)$ contains at most $t+1$ elements. Therefore, the total weight of the vertices covered by this edge is at most $(t+1)/(t+1)=1$. We remark that the W[1]-hardness proof in section 7 is essentially based on this example (see the construction of the enforcer systems in the proof of Proposition 7.2).

Now we are ready to prove the main result of this section.

THEOREM 5.5. CLOSEST SUBSTRING *can be solved in time* $|\Sigma|^d \cdot 2^{kd} \cdot d^{O(d\log\log k)} \cdot n^{O(\log\log k)}$.

Proof. Let us fix the first substring $s_1' \in s_1$ of the solution. We will repeat the following algorithm for each possible choice of s_1'. Since there are at most n possibilities for choosing s_1', the running time of the algorithm presented below has to be multiplied by a factor of n, which is dominated by the $n^{O(\log\log k)}$ term.

The center string s can differ on at most d positions from s_1'. Therefore, if we can find the set P of these positions, then the problem can be solved by trying all the $|\Sigma|^{|P|} \le |\Sigma|^d$ possible assignments on the positions in P. We show how to enumerate efficiently all the possible sets P.

We construct a hypergraph G over the vertex set $\{1,\ldots,L\}$. The edges of the hypergraph describe the possible substrings in the solution. If w is a length L substring of some string s_i and the distance of w is at most $2d$ from s_1', then we add an edge E to G such that $p \in E$ if and only if the pth character of w differs from the pth character of s_1'. Clearly, G has at most n edges, each of size at most $2d$. If $(s, s_1', \ldots, s_k')$ is a solution, then let H be the partial hypergraph of G that contains only the $k-1$ edges corresponding to the $k-1$ substrings $s_2', \ldots, s_k'$. (Note that the distance of s_1' and s_i' is at most $2d$, hence G indeed contains the corresponding edges.) Denote by P the set of at most d positions where s and s_1' differ. Let H_0 be the subhypergraph of H induced by P: the vertex set of H_0 is P, and for each edge E of H there is an edge $E \cap P$ in H_0. Hypergraph H_0 is subhypergraph of H and H is partial hypergraph of G, thus H_0 appears in G at P as subhypergraph.

CLOSEST SUBSTRING-2$(k, L, d, (s_1, \ldots, s_k))$
1. **for** each substring s'_1 of s_1 having length L **do**
2. Construct the hypergraph G on $\{1, \ldots, L\}$
3. **for** every hypergraph H_0 having $\leq d$ vertices and $\leq k$ edges **do**
4. **if** H_0 has the half-covering property **then**
5. **for** every place P where H_0 appears in G as subhypergraph **do** (Algorithm FIND-SUBHYPERGRAPH of Theorem 4.3)
6. **for** every string s that differs from s'_1 only at P **do**
7. **if** $\max_{i=1}^{k} \min_{\{s'_i \text{ is a substring of } s_i\}} d(s, s'_i) \leq d$ **then**
8. s is a solution, STOP.
9. There is no solution, STOP.

FIG. 5.1. *Algorithm* 2 *for* CLOSEST SUBSTRING.

We say that a solution is *minimal* if $\sum_{i=1}^{k} d(s, s'_i)$ is minimal. In Proposition 5.6, we show that if the solution $(s, s'_1, \ldots, s'_k)$ is minimal, then H_0 has the half-covering property. Therefore, we can enumerate all the possible P's by considering every hypergraph H_0 on at most d vertices that has the half-covering property (there are only a constant number of them), and for each such H_0, we enumerate all the places in G where H_0 appears as subhypergraph. Lemma 5.2 ensures that every H_0 considered has small fractional cover number. By Lemma 4.3, this means that we can enumerate efficiently all the places P where H_0 appears in G as subhypergraph. As discussed above, for each such P we can check whether there is a solution where the center string s differs from s'_1 only on P. By repeating this method for every hypergraph H_0 having the half-covering property, we eventually find a solution, if one exists.

PROPOSITION 5.6. *For every minimal solution* $(s, s'_1, \ldots, s'_k)$*, the corresponding hypergraph* H_0 *has the half-covering property.*

Proof. To see that H_0 has the half-covering property, assume that for some $Y \subseteq P$, every edge of H_0 covers at most half of Y. We show that in this case the solution is not minimal. Modify s such that it is the same as s'_1 on every position of Y; let s^* be the new center string. Clearly, $d(s^*, s'_1) = d(s, s'_1) - |Y|$. Furthermore, we show that this modification does not increase the distance for any i, that is, $d(s^*, s'_i) \leq d(s, s'_i)$ for every i. It follows that s^* is also a good center string, contradicting the minimality of the solution.

Let E_i be the edge of H_0 corresponding to the substring s'_i. This means that s'_1 and s'_i differ on $Y \cap E_i$, and they are the same on $Y \setminus E_i$. Therefore, $d(s^*, s'_i) \leq d(s, s'_i) + |Y \cap E_i| - |Y \setminus E_i|$. By assumption, E_i can cover at most half of Y, hence $d(s^*, s'_i) \leq d(s, s'_i)$, as required. □

The overall algorithm is presented in Figure 5.1. There are at most n different possibilities for the string s'_1 in step 1. The construction of the hypergraph G in step 2 takes polynomial time. There are not more than 2^{kd} different hypergraphs on at most d vertices having at most k edges, since there are at most 2^d possibilities for each edge. Therefore, the loop in step 3 is iterated at most 2^{kd} times. In step 4 the half-covering property can be tested by complete enumeration: we have to test for at most 2^d different subsets whether there is an edge that covers more than half of it. If H_0 satisfies the half-covering property, then by Lemma 5.2 its fractional cover number is at most $O(\log \log k)$. Therefore, by Theorem 4.3, step 5 takes $d^{O(d \log \log k)} \cdot n^{O(\log \log k)}$ time. If H_0 appears at P in G as subhypergraph, then in step 6 we have to try at most

$|\Sigma|^d$ possible center strings. Testing each center string can be done in polynomial time (step 7). Therefore, the total running time is $n \cdot 2^{kd} \cdot d^{O(d \log\log k)} \cdot n^{O(\log\log k)} \cdot |\Sigma|^d$. □

6. Algorithm for Consensus Patterns. The aim of this section is to show that the problem Consensus Patterns is fixed-parameter tractable in the bounded alphabet case if the parameter is $\delta := D/k$, the average distance. The algorithm is very similar to the algorithm of Theorem 5.5. The crucial difference is that here we can obtain a constant bound on the fractional cover number of the small hypergraphs H_0, instead of the weaker $O(\log\log k)$ bound coming from the half-covering property. This means that the exponent of n in the running time of the Find-Subhypergraph algorithm of Theorem 4.3 is a constant and we obtain a uniformly polynomial algorithm.

Theorem 6.1. Consensus Patterns *can be solved in time* $\delta^{O(\delta)} \cdot |\Sigma|^\delta \cdot n^9$.

Proof. If $\{s, s'_1, \ldots, s'_k\}$ is a solution for an instance of Consensus Patterns, then $d(s, s'_i) \leq \delta$ for at least one i. Therefore, if there is a solution for the instance, then it can be found by considering every string s_0 that is a length L substring of some s_i, and by checking for each such s_0 whether there is a solution with $d(s, s_0) \leq \delta$. Below we describe how to perform this check for a particular s_0. There are at most n possibilities for s_0, hence the total running time is at most n times greater than for a single s_0. We will assume that $\delta \geq 2$; otherwise it is easy to check every possible s with $d(s, s_0) \leq \delta$.

We construct a hypergraph G over the vertex set $\{1, \ldots, L\}$. If w is a length L substring of some string s_i, then we add an edge E to G such that $p \in E$ if and only if the pth character of w differs from the pth character of s_0. (Note that, unlike in the proof of Theorem 5.5, the hypergraph G can have edges larger than $2d$.) If $(s, s'_1, \ldots, s'_k)$ is a solution, then let H be the partial hypergraph of G that contains only the k edges corresponding to the k substrings $s'_1, \ldots, s'_k$.

Let $(s, s_1, \ldots, s_k)$ be a minimal solution; that is, $\sum_{i=1}^k d(s, s'_i)$ is as small as possible. Denote by P the set of positions where s and s_0 differ. Let H_0 be the subhypergraph of H induced by P: the vertex set of H_0 is P, and for each edge E of H there is an edge $E \cap P$ in H_0. Hypergraph H_0 is subhypergraph of H and H is partial hypergraph of G, thus H_0 appears in G at P as subhypergraph.

We follow the same path as in the proof of Theorem 5.5. It can be shown that the fractional cover number of H_0 is at most 2 (see the proof of Proposition 6.2 below). Therefore, we can find all the possible places P by enumerating every suitable hypergraph H_0, and by using Theorem 4.3 to enumerate all the places where H_0 appears in G as subhypergraph. The problem is that there can be large edges in G, and the algorithm of Theorem 4.3 can be used only if the size of the edges is bounded by the parameter. However, we argue that the same technique works even if the large edges are thrown away from G.

Remove every edge of size greater than 20δ from G (resp., H), let G^* (resp., H^*) be the resulting hypergraph, and let H_0^* be the subhypergraph of H^* induced by P. It is clear that H_0^* is a subhypergraph of G^*. Furthermore, the fractional edge cover number of H_0^* can be bounded by a constant.

Proposition 6.2. *For every minimal solution* $(s, s'_1, \ldots, s'_k)$*, the corresponding hypergraph* H_0^* *has fractional cover number at most* $5/2$.

Proof. We claim that every element of P is covered by at least $k/2$ edges of H_0. Assume that only $k' < k/2$ edges of H_0 cover some $p \in P$. This means that only k' of the strings $s'_1, \ldots, s'_k$ differ from s_0 at position p. Let us change position p of the median string s: let this character be the same as the character at position p of

CONSENSUS PATTERNS$(k, L, \delta, (s_1, \ldots, s_k))$
1. **for** each substring s_0 of $s_1, \ldots, s_k$ having length L **do**
2. Construct the hypergraph G^* on $\{1, \ldots, L\}$
3. **for** every hypergraph H_0^{**} having $\leq \delta$ vertices and $\leq 200 \ln \delta$ edges **do**
4. **if** every vertex of H_0^{**} is covered by at least $1/5$ part of the edges **then**
5. **for** every place P where H_0^{**} appears in G^* as subhypergraph **do** (Algorithm FIND-SUBHYPERGRAPH of Theorem 4.3)
6. **for** every string s that differs from s_0 only at P **do**
7. **if** $\sum_{i=1}^{k} \min_{\{s_i' \text{ is a substring of } s_i\}} d(s, s_i') \leq \delta k$ **then**
8. s is solution, STOP
9. There is no solution, STOP

FIG. 6.1. *Algorithm for* CONSENSUS PATTERNS.

s_0. Now $d(s, s_i')$ decreases for $k - k' > k/2$ values of i, and it increases for at most $k' < k/2$ values of i. Therefore, $\sum_{i=1}^{k} d(s, s_i')$ strictly decreases, contradicting the minimality of s. This shows that every vertex of H_0 is covered by at least $k/2$ of the k edges, hence the fractional cover number of H_0 is at most 2.

From $d(s, s_0) \leq \delta$ and $\sum_{i=1}^{k} d(s, s_i') \leq D = k\delta$, it follows that $\sum_{i=1}^{k} d(s_0, s_i') \leq 2k\delta$. Therefore, the total size of the edges in H is at most $2k\delta$, which means that there are at most $2k/20 \leq k/10$ edges in H that have size greater than 20δ. Each element of P is covered by at least $k/2$ edges of H_0, hence even if the edges greater than 20δ are thrown away, there remain at least $k/2 - k/10 = 2k/5$ edges in H_0^* to cover each element. Therefore, if we set the weight of each edge to $(5/2) \cdot (1/k)$, then we obtain a fractional edge cover with total weight $5/2$. □

Proposition 6.2 shows that we can find all the possible places P by enumerating every hypergraph H_0^* on δ vertices having fractional cover number at most $5/2$, and then enumerating every place in G^* where H_0^* appears. To reduce the number of hypergraphs H_0^* that has to be considered, we show that it is sufficient to restrict our attention to hypergraphs having $O(\log \delta)$ edges.

PROPOSITION 6.3. *Assume $\delta \geq 2$. If $(s, s_1', \ldots, s_k')$ is a minimal solution and H_0^* is the corresponding hypergraph, then it is possible to select $200 \ln \delta$ edges of H_0^* in such a way that if we delete all the other edges, then the resulting hypergraph H_0^{**} has fractional cover number at most 5.*

Proof. Let $k' \leq k$ be the number of edges of H_0^*. Let us select each edge of H_0^* independently with probability $(150 \ln \delta)/k'$. The expected number of selected edges is $150 \ln \delta$; from Theorem 5.4 ($\beta = 1/3$) it follows that the probability of selecting more than $200 \ln \delta$ edges is at most $\exp((-150 \ln \delta)/27) < 1/\delta^2$. We have seen in Proposition 6.2 that each vertex of H_0^* is covered by at least $2k/5$ edges, thus the expected number of edges that cover a given vertex of H_0^{**} is at least $60(k/k') \ln \delta \geq 60 \ln \delta$. Furthermore, by Theorem 5.4 ($\beta = 1/3$) the probability that a given vertex of H_0^{**} is covered by less than $40 \ln \delta$ edges is at most $\exp(-60 \ln \delta/18) \leq 1/\delta^3$. Therefore, with probability at least $1 - 1/\delta^2 - \delta \cdot 1/\delta^3 > 0$, we select a maximum of $200 \ln \delta$ edges and each vertex is covered by at least $40 \ln \delta$ edges. This means that the fractional cover number of H_0^{**} is at most 5: setting the weight of each edge to $1/(40 \ln \delta)$ gives a fractional edge cover. □

The overall algorithm is presented in Figure 6.1. There are at most n different possibilities for the string s_0 in step 1. The rest of the algorithm checks whether there is a solution where the median string differs from s_0 on at most δ positions. The

construction of the hypergraph G^* can be done in $O(Ln)$ time in step 2. Since we try to find solutions with $d(s_0, s) \leq \delta$, it can be assumed that H_0^{**} has at most δ vertices. There are not more than $2^{O(\delta \ln \delta)} = \delta^{O(\delta)}$ different hypergraphs on at most δ vertices having at most $200 \ln \delta$ edges, since there are at most 2^δ possibilities for each edge. Therefore, the loop in step 3 is iterated at most $2^{O(\delta \ln \delta)}$ times. The test in step 4 is trivial. Since the fractional cover number of H_0^{**} is at most 5 and every edge of G^* has size at most 20δ, step 5 takes $\delta^{O(\delta)} \cdot n^6 L^2$ time. If H_0^{**} appears at P in G^* as subhypergraph, then in step 6 we have to try at most $|\Sigma|^\delta$ possible median strings. Testing each median string can be done in $O(Ln)$ time (step 7). Therefore, the total running time is $\delta^{O(\delta)} \cdot |\Sigma|^\delta \cdot n^9$. □

7. Set balancing. In this section we introduce a new problem called Set Balancing. The problem is somewhat technical and it is not motivated by practical applications. However, as we will see it in section 8, the problem is useful in proving the W[1]-hardness of Closest Substring.

Set Balancing

Input:
A collection of m set systems $\mathscr{S}_i = \{S_{i,1}, \ldots, S_{i,|\mathscr{S}_i|}\}$ $(1 \leq i \leq m)$ over the same ground set A and a positive integer d. The size of each set $S_{i,j}$ is at most ℓ, and there is an integer weight $w_{i,j}$ associated to each set $S_{i,j}$.

Parameters:
m, d, ℓ

Task:
Find a set $X \subseteq A$ of size at most d and select a set $S_{i,a_i} \in \mathscr{S}_i$ for every $1 \leq i \leq m$ in such a way that

$$|X \triangle S_{i,a_i}| \leq w_{i,a_i} \tag{7.1}$$

holds for every $1 \leq i \leq m$.

Here $X \triangle S_{i,a_i}$ denotes the symmetric difference $(X \setminus S_{i,a_i}) \cup (S_{i,a_i} \setminus X)$. We have to select a set X and a set from each set system in such a way that the balancing requirement (7.1) is satisfied: every selected set is close to X. The weight $w_{i,j}$ of each set $S_{i,j}$ prescribes the maximum distance of X from this set. The smaller the weight, the more restrictive the requirement. The distance is measured by symmetric difference; therefore, adding an element outside $S_{i,j}$ to X can be compensated by adding an element from $S_{i,j}$ to X. If (7.1) holds for some set S_{i,a_i}, then we say that S_{i,a_i} is *balanced*, or X *balances* S_{i,a_i}.

It can be assumed that the weight of each set is at most $\ell + d$; otherwise the requirement would be automatically satisfied for every possible X. If a set appears in multiple set systems, then it can have different weights in the different systems.

In this section we show that Set Balancing is W[1]-hard even when all of m, d, and ℓ are parameters. It is not very difficult to show that the problem is W[1]-hard if we consider the variant of the problem where the size of X has to be *exactly* d. However, the proof becomes significantly more complicated if we have only the requirement $|X| \leq d$. Intuitively, now the problem is that we have to ensure that the reduction does not construct instances that can be solved by a "small" X, since such an X could be found with an exhaustive search. An easy way to ensure that X is

large would be to have a set that can be balanced only by selecting d elements from this set. However, this would reduce the search space to the d-element subsets of this set, and the problem would be easy, since there are at most $\binom{\ell}{d}$ such sets. The main combinatorial challenge in the proof is to ensure that there are no small solutions, but there are lots of possible sets that could form a solution. It should be the combined effect of several set systems that prevent $|X|$ from being small. Furthermore, each set in a set system should be useful for many possible solutions, since the set systems cannot be too large.

THEOREM 7.1. SET BALANCING *is* W[1]*-hard with combined parameters* m*,* d*, and* ℓ*.*

Proof. The proof is by reduction from the MAXIMUM CLIQUE problem. Assume that a graph $G(V, E)$ is given with n vertices and e edges; the task is to find a clique of size t. It can be assumed that $n = 2^{2^C}$ for some integer C: we can ensure that the number of vertices has this form by adding at most $|V|^2$ isolated vertices. Furthermore, we can assume that $C \geq t$ (i.e., $n \geq 2^{2^t}$): if $n < 2^{2^t}$, then MAXIMUM CLIQUE can be solved directly in time $O((2^{2^t})^t \cdot n)$ by enumerating every set of size t.

The ground set A of the constructed instance of SET BALANCING is partitioned into t groups $A_0, \ldots, A_{t-1}$. The group A_i is further partitioned into 2^i blocks $A_{i,1}$, $\ldots$, $A_{i,2^i}$; the total number of blocks is $2^t - 1$. The block $A_{i,j}$ contains $n^{1/2^i} = 2^{2^{C-i}}$ elements. Set $d := 2^t - 1$. Later we will argue that it is sufficient to restrict our attention to solutions where X contains exactly one element from each block $A_{i,j}$. Let us call such a solution a *standard solution*. We construct the set systems in such a way that there is one-to-one correspondence between the standard solutions and the size t cliques of G. In a standard solution X contains exactly 2^i elements from group A_i, and there are $(n^{1/2^i})^{2^i} = n$ different possibilities for selecting these 2^i elements from the blocks of A_i. Let $X_{i,1}, \ldots, X_{i,n}$ be these n different 2^i-element sets. These n possibilities will correspond to the choice of the ith vertex of the clique.

The set systems are of two types: the verifier systems and the enforcer systems. The role of the verifier systems is to ensure that every standard solution corresponds to a clique of size t, while the enforcer systems ensure that there are only standard solutions.

For each $0 \leq i_1 < i_2 \leq t-1$ the verifier system $\mathscr{S}_{i_1,i_2}$ ensures that the i_1th and the i_2th vertices of the clique are adjacent. The set system $\mathscr{S}_{i_1,i_2}$ contains $2e$ sets of size $2^{i_1} + 2^{i_2}$ each. If vertices u and v are adjacent in G, then $X_{i_1,u} \cup X_{i_2,v}$ is in $\mathscr{S}_{i_1,i_2}$. The weight of every set in $\mathscr{S}_{i_1,i_2}$ is $(2^t - 1) - (2^{i_1} + 2^{i_2})$.

PROPOSITION 7.2. *There is a standard solution if and only if* G *has a size* t *clique.*

Proof. Assume that $v_0, \ldots, v_{t-1}$ is a clique in G. Let

$$X = \bigcup_{i=0}^{t-1} X_{i,v_i}.$$

The size of X is $\sum_{i=0}^{t-1} 2^i = 2^t - 1$. Select the set $X_{i_1,v_{i_1}} \cup X_{i_2,v_{i_2}}$ from the verifier system $\mathscr{S}_{i_1,i_2}$. This set is balanced by X: it is a size $2^{i_1} + 2^{i_2}$ subset of X having weight $(2^t - 1) - (2^{i_1} + 2^{i_2})$.

To prove the other direction, assume now that there is a standard solution X. In a standard solution, $X \cap A_i$ is a 2^i-element set X_{i,v_i} for some v_i. We claim that these v_i's form a size t clique in G.

Suppose that for some $i_1 < i_2$ vertices v_{i_1} and v_{i_2} are not connected by an edge (including the possibility $v_{i_1} = v_{i_2}$). Consider the set $S \in \mathscr{S}_{i_1,i_2}$ selected in the solution. The size of X is $2^t - 1$ in a standard solution, thus the set X contains at least $2^t - 1 - (2^{i_1} + 2^{i_2})$ elements outside the set S. Therefore, S can be balanced only if all the $2^{i_1} + 2^{i_2}$ elements of S are in X. Assume that the set S selected from $\mathscr{S}_{i_1,i_2}$ is $X_{i_1,u} \cup X_{i_2,v}$. Now $X_{i_1,u} \cup X_{i_2,v} \subseteq X$, which means that $u = v_{i_1}$ and $v = v_{i_2}$. By construction, if $X_{i_1,u} \cup X_{i_2,v}$ is in $\mathscr{S}_{i_1,i_2}$, then u and v are adjacent, hence v_{i_1} and v_{i_2} are indeed neighbors. □

The job of the enforcer systems is to ensure that every solution of weight at most $d = 2^t - 1$ is standard. The $2^t - 1$ blocks $A_{i,j}$ are indexed by two indices i and j. In the following, it will be more convenient to index the blocks by a single variable. Let $B_1, \ldots, B_{2^t-1}$ be an ordering of the blocks such that B_1 is the only block of group A_0, the blocks B_2, B_3 are the blocks of A_1, the next four blocks after that are the blocks of A_2, etc.

A naive way of constructing the enforcer set systems would be to have for each block B_i a corresponding set system $\mathscr{S}_i$ that contains $|B_i|$ one-element sets: there is one set of weight $2^t - 2$ for each element of B_i. This ensures that if a solution contains at least one element from every block other than B_i (i.e., it contains at least $2^t - 2$ elements outside B_i), then it has to contain an element of B_i as well (otherwise the symmetric difference is at least $2^t - 1$). The problem with this construction is that every set of $\mathscr{S}_i$ is balanced by the solution $X = \emptyset$, hence such systems cannot ensure that every solution is standard.

There are $2^{2^t-1} - 1$ enforcer set systems: there is a set system $\mathscr{S}_F$ corresponding to each nonempty subset F of $\{1, 2, \ldots, 2^t - 1\}$. The job of $\mathscr{S}_F$ is to rule out the possibility that a solution X contains no elements from the blocks indexed by F, but X contains at least one element from every other block. Clearly, these systems will ensure that no block is empty in a solution, hence every solution of weight $2^t - 1$ is standard. One possible way of constructing the system $\mathscr{S}_F$ is to have one set of size $|F|$ and weight $2^t - 1 - |F|$ for each possible way of selecting one element from each block indexed by F. Clearly, this makes it impossible to have at least one element in each of the $2^t - 1 - |F|$ blocks outside F, but none in F. Now the problem is that the size of $\mathscr{S}_F$ can be too large, in particular when $F = \{1, 2, \ldots, 2^t - 1\}$. We use a somewhat more complicated construction to keep the size of the systems small.

Recall the definition of $\mathrm{up}(F)$ from section 5: given a finite set F of positive integers, $\mathrm{up}(F)$ is defined to be the largest $\lceil (|F| + 1)/2 \rceil$ elements of this set. The enforcer system corresponding to F is defined as

$$\mathscr{S}_F = \prod_{p \in \mathrm{up}(F)} B_p. \tag{7.2}$$

That is, we consider the blocks indexed by the upper half of F, and put into $\mathscr{S}_F$ all the possible combinations of selecting one element from each block. Let the weight of each set in $\mathscr{S}_F$ be $2^t - 1 - |\mathrm{up}(F)|$. Notice that it is possible that $\mathrm{up}(F_1) = \mathrm{up}(F_2)$ for some $F_1 \neq F_2$, which means that for such F_1 and F_2 the systems $\mathscr{S}_{F_1}$ and $\mathscr{S}_{F_2}$ are in fact the same. However, we do not care about that.

We have to verify that these set systems are not too large; i.e., they can be constructed in uniformly polynomial time.

PROPOSITION 7.3. *For every nonempty $F \subseteq \{1, 2, \ldots, 2^t - 1\}$, the enforcer system $\mathscr{S}_F$ contains at most n^2 sets.*

Proof. Let x be the smallest element of $\mathrm{up}(F)$, and assume that $2^p \leq x < 2^{p+1}$

for some integer p. There is one block of size n, there are 2 blocks of size $n^{1/2}$, ..., there are 2^i blocks of size $n^{1/2^i}$, hence the size of B_{2^p} is $n^{1/2^p}$. The size of the blocks is decreasing, thus all the sets in the product (7.2) are of size at most $n^{1/2^p}$. If the smallest element of $\mathrm{up}(F)$ is x, then it can contain at most $x+1$ elements. This means that we take the direct product of at most $x+1$ sets of size at most $n^{1/2^p}$ each. Therefore, the total number of sets in $\mathscr{S}_F$ is at most $(n^{1/2^p})^{x+1} \leq (n^{1/2^p})^{2^{p+1}} = n^2$. □

The following proposition completes the proof of the first direction: if the solution is standard, then we can select a set from each enforcer system. Together with Proposition 7.2, it follows that if there is a clique of size t, then there is a (standard) solution for the constructed instance of Set Balancing.

Proposition 7.4. *If X is a standard solution, then each $\mathscr{S}_F$ contains a set that is balanced by X.*

Proof. For the enforcer system $\mathscr{S}_F$, let us select the set

$$S_F = X \cap \bigcup_{p \in \mathrm{up}(F)} B_p.$$

That is, S_F contains those elements of X that belong to the blocks indexed by $\mathrm{up}(F)$. The set S_F is a size $|\mathrm{up}(F)|$ subset of X. Therefore, $|X \triangle S_F| = 2^t - 1 - |\mathrm{up}(F)|$, which is exactly the weight of the selected set. Thus S_F is balanced. □

On the other hand, if there is a solution for the constructed instance of Set Balancing with $|X| \leq d = 2^t - 1$, then this solution has to be standard, and by Proposition 7.2 there is a clique of size t in G. This completes the proof of the second direction.

Proposition 7.5. *If $|X| \leq 2^t - 1$, then X contains exactly one element from each block.*

Proof. Assume first that X does not contain elements from some of the blocks. Let F contain the indices of those blocks that are disjoint from X. This means that X contains at least one element from each block not in F, hence $|X| \geq 2^t - 1 - |F|$. Assume that some set S is selected from $\mathscr{S}_F$ in the solution. This set contains elements only from blocks indexed by $\mathrm{up}(F) \subseteq F$, hence S is disjoint from X. Thus $|X \triangle S| = |X| + |S| \geq 2^t - 1 - |F| + |\mathrm{up}(F)| > 2^t - 1 - |\mathrm{up}(F)|$, which means that S is not balanced (here we used $|F| - |\mathrm{up}(F)| < |\mathrm{up}(F)|$). Therefore, each block contains at least one element of X. Since there are $2^t - 1$ blocks, this is only possible if each block contains exactly one element of X. □

The distance $d = 2^t - 1$ is a function of the original parameter t. The number m of the constructed set systems is $\binom{t}{2} + 2^{2^t - 1} - 1$, which is also a function of t. Each set in the constructed systems has size at most $\ell := 2^t - 1$. We have seen that the size of each set system is polynomial in n, hence the reduction is a correct parameterized reduction. □

8. Hardness of Closest Substring. In this section we show that Closest Substring is W[1]-hard with combined parameters k and d. The reduction is very similar to the reduction presented in [14], where it is proved that problem is W[1]-hard with parameter k only. As in that reduction, the main technical trick is that each string s_i is divided into blocks and we ensure that the string s'_i is one of these blocks in every solution. However, here the reduction is from Set Balancing, and not from Maximum Clique. This allows us to construct an instance of Closest Substring where the distance parameter d is bounded by a constant.

THEOREM 8.1. CLOSEST SUBSTRING *is* W[1]*-hard with parameters* d *and* k*, even if* $\Sigma = \{0, 1\}$.

Proof. The reduction is from the SET BALANCING problem, whose W[1]-hardness was shown in section 7. Assume that m set systems $\mathscr{S}_i = \{S_{i,1}, \ldots, S_{i,|\mathscr{S}_i|}\}$ and an integer d are given. Let $0 \leq w_{i,j} \leq d + \ell$ be the weight of $S_{i,j}$ in $\mathscr{S}_i$, and assume that each set has size at most ℓ. We construct an instance of CLOSEST SUBSTRING with distance parameter $d' := d + \ell$, where $d' + 1$ strings $s_{i,1}, s_{i,2}, \ldots, s_{i,d'+1}$ correspond to each set system $\mathscr{S}_i$, and there is one additional string s_0 called the *template string*. Thus there are $k := (d' + 1)m + 1$ strings in total.

Set $L := 6d' + 3d'(3d' + 1) + |A| + d' - d + 2d'm(d' + 1)$, where A is the common ground set of the set systems. The template string s_0 has length L, hence $s'_0 = s_0$ in every solution. The string $s_{i,j}$ is the concatenation of *blocks* $B_{i,j,1}, \ldots, B_{i,j,|\mathscr{S}_i|}$ of the same length L, and each block corresponds to a set in $\mathscr{S}_i$. We will ensure that in a solution the substring $s'_{i,j}$ is one complete block from $s_{i,j}$. Therefore, selecting $s'_{i,j}$ from $s_{i,j}$ in the constructed CLOSEST SUBSTRING instance plays the same role as selecting a set S_i from $\mathscr{S}_i$ in SET BALANCING.

Each block $B_{i,j,k}$ of the string $S_{i,j}$ is the concatenation of four parts: the front tag, the core, the complete tag, and the back tag. The *front tag* is the same in every block: $1^{3d'}(10^{3d'})^{3d'}1^{3d'}$. The *core* corresponds to the ground set A in the SET BALANCING problem. The length of the core is $|A|$, and the pth character of the core in block $B_{i,j,k}$ is 1 if and only if the set $S_{i,k} \in \mathscr{S}_i$ contains the pth element of A. The *complete tag* is $1^{d'-d}$ in every block. The *back tag* is the concatenation of $m(d' + 1)$ segments $C_{i,j}$ $(1 \leq i \leq m, 1 \leq j \leq d' + 1)$ (the order in which these segments are concatenated will not be important). The length of each segment is $2d'$. In block $B_{i,j,k}$ of string $s_{i,j}$ the back tag contains 1's only in segment $C_{i,j}$: there is 1 on the first $d' - w_{i,k} \geq 0$ positions of $C_{i,j}$; the rest of $C_{i,j}$ is 0. This completes the description of the strings $s_{i,j}$. Notice that the blocks $B_{i,j_1,k}$ and $B_{i,j_2,k}$ differ only in the back tag. The length L template string s_0 is similar to the blocks defined above: it has the same front tag as all the other blocks, but its core, complete tag, and back tag contain only 0's.

The first direction of the proof is shown in the following proposition.

PROPOSITION 8.2. *If the* SET BALANCING *instance has a solution, then the constructed instance of* CLOSEST SUBSTRING *also has a solution.*

Proof. Let $X \subseteq A$ and $S_{1,a_1} \in \mathscr{S}_1, \ldots, S_{m,a_m} \in \mathscr{S}_m$ be a solution of SET BALANCING. Let the center string s be the concatenation of the front tag, the incidence vector of X, the string $1^{d'-d}$, and the string $0^{2d'm(d'+1)}$. The distance of s and s_0 is $|X| + d' - d \leq d'$: the distance is $|X|$ on the core and $d' - d$ on the complete tag. Furthermore, we claim that the block B_{i,j,a_i} in string $s_{i,j}$ is at distance at most d' from s. If we can show this, then it follows that CLOSEST SUBSTRING has a solution.

The front tag of B_{i,j,a_i} is the same as the front tag of s. In the core the distance is the symmetric difference of X and S_{i,a_i}. The complete tag is the same in s and B_{i,j,a_i}. The back tag of s is all 0, while the back tag of B_{i,j,a_i} contains $d' - w_{i,k}$ characters 1 (in the segment $C_{i,j}$). Therefore,

$$d(s, B_{i,j,a_i}) = |X \triangle S_{i,a_i}| + d' - w_{i,k} \leq d',$$

where the inequality follows from the fact that X balances the set S_{i,a_i}, that is, $|X \triangle S_{i,a_i}| \leq w_{i,k}$. ∎

To prove the reverse direction, first we show that each substring $s'_{i,j}$ has to be a complete block of the string $s_{i,j}$. By the triangle inequality, $d(s_0, s'_{i,j}) = d(s'_0, s'_{i,j}) \leq$

$d(s'_0, s) + d(s, s'_{i,j}) \leq 2d'$ has to hold in every solution. We show that $d(s_0, s'_{i,j}) \leq 2d'$ implies that $s'_{i,j}$ is a complete block.

PROPOSITION 8.3. *If $d(s_0, s'_{i,j}) \leq 2d'$ for some substring $s'_{i,j}$ of $s_{i,j}$, then $s'_{i,j}$ is the block $B_{i,j,b}$ for some b.*

Proof. Assume that $s'_{i,j}$ starts on the pth character of some block $B_{i,j,b}$. This means that $s'_{i,j}$ contains the last $L - p + 1$ characters from $B_{i,j,b}$ and the first $p - 1$ characters from the next block $B_{i,j,b+1}$. We show that if $p \neq 1$, then $d(s_0, s'_{i,j}) > 2d'$. Denote by $f = 6d' + 3d'(3d' + 1)$ the length of the front tag. Assume first that $3d' < p \leq L - f$. In this case the first $3d'$ characters of $B_{i,j,b+1}$ (all of them are 1's) are aligned with characters $L+1-p, \ldots, L+3d'-p$ of s_0 (all of them are 0's), hence $d(s_0, s'_{i,j}) > 2d'$ follows. Assume now that $L - f < p \leq L - 3d'$; a similar argument shows that the last $3d'$ characters in the front tag of $B_{i,j,b+1}$ cause $3d'$ mismatches. If $1 < p \leq 3d'$, then the 1's in the $(10^{3d'})^{3d'}$ part of the front tag are not aligned, which increases the difference to more than $2d'$. The same thing happens if $L - 3d' < p \leq L$ holds. This concludes the proof that $p = 1$; that is, the string $s'_{i,j}$ is one complete block $B_{i,j,b}$. □

Since the template string and each block begins with the front tag, it cannot hurt if the center string also begins with the front tag.

PROPOSITION 8.4. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING*, then there is such a solution where the front tag of the center string s is the same as the front tag of s_0.*

It can be assumed that the back tag of the center string s contains only 0's.

PROPOSITION 8.5. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING*, then there is such a solution where the back tag of the center string s contains only 0's.*

Proof. Let s be the center string of a solution. Since the back tag of s_0 contains only 0's, in the back tag of s at most d' characters can be 1. This means that with the exception of at most d' segments, the segments of the back tag contain only 0's. Thus for every $1 \leq i \leq m$, there is a $1 \leq x_i \leq d' + 1$ such that segment C_{i,x_i} of the back tag of s contains only 0's. Let s^* be the same as s but with the back tag set to 0's. It is clear that $d(s^*, s_0) \leq d(s, s_0) \leq d'$: the back tag of s_0 is empty, hence setting the back tag to 0 cannot increase the distance.

We claim that a block can be selected from each string $s_{i,j}$ in such a way that the distance of each selected block is at most d' from s^*. For the string s_{i,x_i} we can select the same s'_{i,x_i} as before: the back tag of s'_{i,x_i} contains 1's only in segment C_{i,x_i}, but s does not contain any 1's in C_{i,x_i}. This means that setting to 0 the back tag of s does not increase the distance between s and s'_{i,x_i}. Assume that s'_{i,x_i} is block $B_{i,x_i,t}$ for some t. For every $j \neq x_i$, select block $B_{i,j,t}$ from the string $s_{i,j}$. The only difference between blocks $B_{i,x_i,t}$ and $B_{i,j,t}$ is in the back tag: they have the same number of 1's in the back tag, but in different segments. However, s^* has only 0's in the back tag, hence $d(B_{i,j,t}, s^*) = d(B_{i,x_i,t}, s^*) \leq d'$. Therefore, s^* and the selected blocks form a solution where the back tag of the center string s^* contains only 0's. □

We can assume that the complete tag is $1^{d'-d}$ in s.

PROPOSITION 8.6. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING*, then there is such a solution where the complete tag of the center string s contains only 1's.*

Proof. Let s be a solution where the number of 0's in the complete tag of the center string is minimal. Assume first that there is a 1 in the core of s. In this case replace this 1 with a 0, and set one of the 0's in the complete tag to 1. This modification

does not change the difference of s from s_0. Furthermore, it does not increase the distance of s from $s'_{i,j}$: replacing the 0 with a 1 in the complete tag decreases the distance, while replacing the 1 with 0 in the core may or may not increase the distance. Therefore, the new center string contradicts the minimality of s.

Assume now that the core of s contains only 0's. We have already seen that the front tag of s is the same as the front tag of s_0 (Proposition 8.4), and the back tag contains only 0's (Proposition 8.5). Therefore, s differs from s_0 only in the complete tag. This means that in the complete tag of s we can replace every 0 with 1: the distance between s and s_0 increases only to $d'-d$, while the distance decreases between s and every string $s'_{i,j}$. □

The proofs of Propositions 8.4–8.6 are independent in the sense that we can assume that there is a solution where the center string s satisfies all three requirements at the same time. Assuming that s is of this form, it is not difficult to prove the converse of Proposition 8.2.

PROPOSITION 8.7. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING*, then there is a solution for the* SET BALANCING *problem.*

Proof. Consider a solution where the complete tag of s contains only 1's, and the back tag of s contains only 0's. Define the set $X \subseteq A$ based on the core of s: let an element of A be in X if and only if the corresponding character is 1 in the core of s. The string s differs from the template string s_0 at $|X|$ positions in the core and at $d'-d$ positions in the complete tag. Since $d(s,s_0) \leq d'$, it follows that $|X| \leq d$.

We claim that for every $1 \leq i \leq s$, a set can be selected from $\mathscr{S}_i$ that is balanced by X. Assume that $s'_{i,1}$ is the block $B_{i,1,t}$ of $s_{i,1}$ for some t. We show that $S_{i,t} \in \mathscr{S}_i$ is balanced by X. Let us determine the distance $d(B_{i,1,t}, s)$, which is by assumption at most d'. In the core, the two strings differ on the symmetric difference of $S_{i,t}$ and X. The strings do not differ on the complete tag, but they differ on every position of the back tag where $B_{i,1,t}$ is 1. There are exactly $d'-w_{i,t}$ such positions, hence

$$d(s, s'_{i,j}) = |X \triangle S_{i,k}| + d' - w_{i,t} \leq d',$$

which means that $|X \triangle S_{i,t}| \leq w_{i,t}$ and the set $S_{i,t}$ is balanced. □

Propositions 8.2 and 8.7 together prove the correctness of the reduction. □

Putting together Theorems 7.1 and 8.1 gives a two-step reduction from MAXIMUM CLIQUE to CLOSEST SUBSTRING. Let us follow how the parameters depend on each other during this reduction. If an instance of MAXIMUM CLIQUE is given with parameter t, then Theorem 7.1 constructs an instance of SET BALANCING with the following parameters:

$$\begin{aligned} d &:= 2^t - 1, \\ m &:= \binom{t}{2} + 2^{2^{t-1}} - 1 = 2^{2^{O(t)}}, \\ \ell &:= 2^{t-1}. \end{aligned}$$

Theorem 8.1 transforms this instance into an instance of CLOSEST SUBSTRING with the following parameters:

$$\begin{aligned} k &:= (d+\ell+1)m + 1 = 2^{2^{O(t)}}, \\ d' &:= d + \ell = 2^{O(t)}. \end{aligned}$$

Theorem 3.3 gives an $|\Sigma|^{d(\log d+2)}n^{\log d+O(1)}$ time algorithm for CLOSEST SUBSTRING and Theorem 5.5 gives an algorithm with running time $2^{kd} \cdot d^{O(d \log\log k)}$.

$n^{O(\log\log k)} \cdot |\Sigma|^d$. We argue that in some sense these algorithms are best possible: the exponent of n cannot be improved to $o(\log d)$ or to $o(\log\log k)$ (modulo some complexity-theoretic assumptions).

Assume that there is an $f_1(k,d)\cdot n^{o(\log d)}$ time algorithm for Closest Substring. We can construct an algorithm for Maximum Clique by reducing it to Closest Substring and using our assumed algorithm for the latter problem. The running time of this algorithm for finding a size t clique is $f_1(k,d) \cdot n^{o(\log d)} = f_1(2^{2^{O(t)}}, 2^t) \cdot n^{o(\log 2^t)} = f_1'(t) \cdot n^{o(t)}$ (it can be assumed that the running time of the reduction to Closest Substring is dominated by the time required to solve the Closest Substring instance.) By a result of [9], the existence of an $f_1'(t) \cdot n^{o(t)}$ algorithm for Maximum Clique would imply that n-variable 3-Sat can be solved in $2^{o(n)}$ time; i.e., the Exponential Time Hypothesis would be violated. Therefore, it is highly unlikely that there is an algorithm for Closest Substring with $o(\log d)$ in the exponent.

Corollary 8.8. *There is no $f_1(k,d) \cdot n^{o(\log d)}$ time algorithm for* Closest Substring*, unless n-variable* 3-Sat *can be solved in time $2^{o(n)}$.*

Similarly, an $f_2(k,d) \cdot n^{o(\log\log k)}$ time algorithm for Closest Substring would imply that there is an $f_2(2^{2^{O(t)}}, 2^t) \cdot n^{o(\log\log 2^{2^{O(t)}})} = f_2'(t) \cdot n^{o(t)}$ algorithm for Maximum Clique.

Corollary 8.9. *There is no $f_2(k,d) \cdot n^{o(\log\log k)}$ time algorithm for* Closest Substring*, unless n-variable* 3-Sat *can be solved in time $2^{o(n)}$.*

In our reduction from Maximum Clique to Closest Substring, the blow-up of the parameters is unusually large: double exponential. It might seem that with some more careful construction we could give a simpler reduction, where the parameters of the constructed instance are smaller. However, the connection with subexponential algorithms shows that the double exponential increase cannot be avoided, i.e., it is a necessary part of any reduction from Maximum Clique to Closest Substring. Assume that there is an $f(t) \cdot n^c$ time parameterized reduction where $d = g_1(t)$ and $k = g_2(t) = 2^{2^{o(t)}}$. This reduction and the $h(k,d) \cdot n^{O(\log\log k)}$ time algorithm of Theorem 5.5 would give an algorithm for Maximum Clique with running time

$$\begin{aligned} &f(t)n^c + h(g_1(t), g_2(t)) \cdot (f(t)n^c)^{O(\log\log g_2(t))} \\ &\qquad = h'(t) \cdot n^{O(\log\log 2^{2^{o(t)}})} = h'(t) \cdot n^{o(t)}, \end{aligned}$$

which is not possible, unless 3-Sat has subexponential algorithms.

Cesati and Trevisan [7] and Bazgan [4] have shown that if a problem is W[1]-hard, then the corresponding optimization problem cannot have an EPTAS (i.e., a PTAS with running time $f(\epsilon) \cdot n^c$), unless FPT = W[1]. Let us recall the argument here. Assume that there is an approximation scheme with running time $f(\epsilon) \cdot n^c$ for Closest Substring. Running the algorithm with $\epsilon = 1/2k$ decides whether there is a solution with $d \le k$: if there is such a solution, then the approximation scheme always produces a solution with d at most $(1+\epsilon)k < k+1$. This would imply an $f(1/2k)\cdot n^c$ algorithm for Closest Substring, and it would follow that the problem is fixed-parameter tractable, which is not possible, unless FPT = W[1].

We can push this argument a bit further: it can be shown that there is no PTAS with running time $f(\epsilon) \cdot n^{o(\log 1/\epsilon)}$. The same reasoning as in the previous paragraph shows that such a PTAS would imply an $f(1/2k) \cdot n^{o(\log 2k)}$ algorithm for Closest Substring. In Corollary 8.8, we have seen that this is not possible, unless there are subexponential algorithms for 3-Sat.

COROLLARY 8.10. *There is no* $f(\epsilon) \cdot n^{o(\log 1/\epsilon)}$ *time PTAS for* CLOSEST SUBSTRING*, unless* n*-variable* 3-SAT *can be solved in time* $2^{o(n)}$.

The lower bound of Corollary 8.10 does not match the known approximation schemes for the problem. Using a different approach, Andoni, Indyk, and Pătraşcu [2] proved an essentially tight lower bound. However, in a strict technical sense, their lower bound is not directly comparable with Corollary 8.10.

9. Conclusions. We have presented algorithms and complexity results for two string matching problems, CLOSEST SUBSTRING and CONSENSUS PATTERNS. We have proved that CLOSEST SUBSTRING parameterized by the distance parameter d and by the number of strings k is W[1]-hard, even if the alphabet is binary. This improves the previous result of [14], where it is proved that the problem is W[1]-hard with parameter k only (and binary alphabet). Our hardness result also improves [20], where it is proved that DISTINGUISHING SUBSTRING SELECTION (a generalization of CLOSEST SUBSTRING) is W[1]-hard with parameters k and d (again with binary alphabet). In our reduction we used some of the techniques from these results, but new ideas were also required. In particular, we had to ensure that in the constructed instance of CLOSEST SUBSTRING there is no solution where the center string is very close to some substring. This is easy to ensure if d is unbounded, or if we can use the additional features of DISTINGUISHING SUBSTRING SELECTION. However, if d is a parameter, then we have to develop new combinatorial machinery to make sure that no solution can be close to some substring.

The W[1]-hardness of a problem is usually interpreted as evidence that the problem is unlikely to be fixed-parameter tractable; that is, the parameter has to appear in the exponent of n. Furthermore, using recent connections with subexponential algorithms, we can even give a lower bound on the exponent of n. Our reduction for CLOSEST SUBSTRING is "weak" in the sense that the parameters are significantly increased (exponentially and double exponentially). Therefore, we obtain only weak lower bounds on the exponent of n: all we can show is that the exponent cannot be $o(\log d)$ or $o(\log \log k)$. However, it turned out that these bounds are tight: we presented two algorithms where the exponent of n is $O(\log d)$ and $O(\log \log k)$, respectively. The second algorithm is based on some surprising connections with the extremal combinatorics of hypergraphs. We have introduced and investigated the half-covering property, which played an important role in the algorithm. Furthermore, we have shown that all the copies of hypergraph H in hypergraph G can be efficiently found if H has small fractional cover number. This result might be useful in some other applications as well. More generally, the fractional cover number and Shearer's lemma (which is the main combinatorial idea behind Theorem 4.1 and hence behind Theorem 4.3) can be useful algorithmic tools in other contexts; see [22].

The same hypergraph techniques can be used in the case of the CONSENSUS PATTERNS problem. However, the combinatorial structure of this problem is slightly different, and this slight difference allows us to obtain a uniformly polynomial time algorithm with running time $f(|\Sigma|, \delta) \cdot n^9$. Therefore, in the constant alphabet case the problem is fixed-parameter tractable with parameter δ (and also with the larger parameter D). This resolves another open question from [14].

The algorithms of Theorems 5.5 and 6.1 are based on the same idea: we want to enumerate all the "small" places P in a large hypergraph G that are "well-covered" in a certain sense. Our algorithms do this in a somewhat cumbersome way: first every small well-covered hypergraph is enumerated, and then for each such H, the algorithm enumerates all the places where H appears in G. It might be possible to do this in

a more direct and elegant way. What we need is an algorithm that enumerates every maximal subset of vertices having the property that it can be fractionally covered by weight k with the running time being something like $n^{O(k)}$. Developing such an algorithm could improve the running time of our algorithms, and, more importantly, would give us more insight into the nature of fractional edge covers.

Our results present an example where parameterized complexity and subexponential algorithms are closely connected. First, a weak parameterized reduction might be the sign that some kind of subexponential algorithm is possible for the problem. On the other hand, a parameterized reduction can be used to show the optimality of a subexponential algorithm. It is possible that this interplay between parameterized complexity and subexponential algorithms appears in the case of some other problems as well.

Acknowledgments. I'm grateful to Mike Fellows for directing my attention to this problem and to Ildi Schlotter for reading the manuscript.

REFERENCES

[1] N. Alon, *On the number of subgraphs of prescribed type of graphs with a given number of edges*, Israel J. Math., 38 (1981), pp. 116–130.

[2] A. Andoni, P. Indyk, and M. Pătraşcu, *On the optimality of the dimensionality reduction method*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 449–458.

[3] D. Angluin and L. G. Valiant, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[4] C. Bazgan, *Schémas d'approximation et complexité paramétrée*, Technical report, Université Paris Sud, Paris, France, 1995.

[5] M. Blanchette, B. Schwikowski, and M. Tompa, *Algorithms for phylogenetic footprinting*, J. Comput. Biol., 9 (2002), pp. 211–223.

[6] J. Buhler and M. Tompa, *Finding motifs using random projections*, in Proceedings of the Fifth Annual International Conference on Computational Biology (RECOMB'01), 2001, pp. 69–76.

[7] M. Cesati and L. Trevisan, *On the efficiency of polynomial time approximation schemes*, Inform. Process. Lett., 64 (1997), pp. 165–171.

[8] J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanj, and G. Xia, *Tight lower bounds for certain parameterized NP-hard problems*, in Proceedings of 19th Annual IEEE Conference on Computational Complexity, 2004, pp. 150–160.

[9] J. Chen, X. Huang, I. A. Kanj, and G. Xia, *Linear FPT reductions and computational lower bounds*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, ACM, New York, pp. 212–221.

[10] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer, *Some intersection theorems for ordered sets and graphs*, J. Combin. Theory Ser. A, 43 (1986), pp. 23–37.

[11] R. G. Downey, *Parameterized complexity for the skeptic*, in Proceedings of the 18th IEEE Annual Conference on Computational Complexity, 2003, pp. 147–169.

[12] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Monogr. Comput. Sci., Springer-Verlag, New York, 1999.

[13] P. A. Evans, A. D. Smith, and H. T. Wareham, *On the complexity of finding common approximate substrings*, Theoret. Comput. Sci., 306 (2003), pp. 407–430.

[14] M. R. Fellows, J. Gramm, and R. Niedermeier, *On the parameterized intractability of motif search problems*, Combinatorica, 26 (2006), pp. 141–167.

[15] J. Flum and M. Grohe, *Parameterized complexity and subexponential time*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 84 (2004), pp. 71–100.

[16] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer-Verlag, Berlin, 2006.

[17] Y. M. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit, *Identification of common motifs in unaligned DNA sequences: Application to Escherichia coli Lrp regulon*, Computer Applications in the Biosciences, 11 (1995), pp. 379–387.

[18] M. Frances and A. Litman, *On covering problems of codes*, Theory Comput. Syst., 30 (1997), pp. 113–119.

[19] E. Friedgut and J. Kahn, *On the number of copies of one hypergraph in another*, Israel J. Math., 105 (1998), pp. 251–256.
[20] J. Gramm, J. Guo, and R. Niedermeier, *On exact and approximation algorithms for distinguishing substring selection*, in Fundamentals of Computation Theory, Lecture Notes in Comput. Sci. 2751, Springer, Berlin, 2003, pp. 195–209.
[21] J. Gramm, R. Niedermeier, and P. Rossmanith, *Fixed-parameter algorithms for closest string and related problems*, Algorithmica, 37 (2003), pp. 25–42.
[22] M. Grohe and D. Marx, *Constraint solving via fractional edge covers*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06), ACM, New York, SIAM, Philadelphia, 2006, pp. 289–298.
[23] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, UK, 1997.
[24] G. Hertz and G. Stormo, *Identification of consensus patterns in unaligned DNA and protein sequences: A large-deviation statistical basis for penalizing gaps*, in Proceedings of the 3rd International Conference on Bioinformatics and Genome Research, 1995, pp. 201–216.
[25] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity?*, J. Comput. System Sci., 63 (2001), pp. 512–530.
[26] U. Keich and P. A. Pevzner, *Finding motifs in the twilight zone*, in Proceedings of the Sixth Annual International Conference on Computational Biology (RECOMB'02), 2002, pp. 195–204.
[27] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, *Distinguishing string selection problems*, Inform. and Comput., 185 (2003), pp. 41–55.
[28] C. Lawrence and A. Reilly, *An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences.*, Proteins, 7 (1990), pp. 41–51.
[29] M. Li, B. Ma, and L. Wang, *Finding similar regions in many sequences*, J. Comput. System Sci., 65 (2002), pp. 73–96.
[30] M. Li, B. Ma, and L. Wang, *On the closest string and substring problems*, J. ACM, 49 (2002), pp. 157–171.
[31] P. A. Pevzner and S.-H. Sze, *Combinatorial approaches to finding subtle signals in DNA sequences*, in Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, AAAI Press, Menlo Park, CA, 2000, pp. 269–278.
[32] A. Price, S. Ramabhadran, and P. Pevzner, *Finding subtle motifs by branching from sample strings*, Bioinformatics, 19 (2003), pp. II149–II155.
[33] I. Rigoutsos and A. Floratos, *Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm*, Bioinformatics, 14 (1998), pp. 55–67.
[34] G. Stormo, *Consensus patterns in DNA*, Methods Enzymol., 183 (1990), pp. 211–221.
[35] G. J. Woeginger, *Exact algorithms for NP-hard problems: A survey*, in Combinatorial Optimization—Eureka, You Shrink!, Lecture Notes in Comput. Sci. 2570, Springer, Berlin, 2003, pp. 185–207.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1411–1429

APPROXIMATION ALGORITHMS FOR DATA PLACEMENT PROBLEMS*

IVAN BAEV†, RAJMOHAN RAJARAMAN‡, AND CHAITANYA SWAMY§

Abstract. We develop approximation algorithms for the problem of placing replicated data in arbitrary networks, where the nodes may both issue requests for data objects and have capacity for storing data objects so as to minimize the average data-access cost. We introduce the *data placement problem* to model this problem. We have a set of caches $\mathcal{F}$, a set of clients $\mathcal{D}$, and a set of data objects $\mathcal{O}$. Each cache i can store at most u_i data objects. Each client $j \in \mathcal{D}$ has demand d_j for a specific data object $o(j) \in \mathcal{O}$ and has to be assigned to a cache that stores that object. Storing an object o in cache i incurs a storage cost of f_i^o, and assigning client j to cache i incurs an access cost of $d_j c_{ij}$. The goal is to find a placement of the data objects to caches respecting the capacity constraints, and an assignment of clients to caches so as to minimize the total storage and client access costs. We present a 10-approximation algorithm for this problem. Our algorithm is based on rounding an optimal solution to a natural linear-programming relaxation of the problem. One of the main technical challenges encountered during rounding is to preserve the cache capacities while incurring only a constant-factor increase in the solution cost. We also introduce the *connected data placement problem* to capture settings where write-requests are also issued for data objects, so that one requires a mechanism to maintain consistency of data. We model this by requiring that all caches containing a given object be connected by a Steiner tree to a root for that object, which issues a multicast message upon a write to (any copy of) that object. The total cost now includes the cost of these Steiner trees. We devise a 14-approximation algorithm for this problem. We show that our algorithms can be adapted to handle two variants of the problem: (a) a k-median variant, where there is a specified bound on the number of caches that may contain a given object, and (b) a generalization where objects have lengths and the total length of the objects stored in any cache must not exceed its capacity.

Key words. approximation algorithms, data placement, facility location, linear programming

AMS subject classifications. 68W40, 68W25, 90B80, 90C59

DOI. 10.1137/080715421

1. Introduction.

Consider a distributed network, some of whose nodes need to periodically access certain data objects, and some of whose nodes have storage capacity and may serve as caches to store data objects thereby reducing the cost of accessing data objects. For example, one could have a network of distributed caches and/or processors in a large-scale information system or computing environment. A powerful paradigm for improving performance, which has been explored in several studies [10, 5, 2, 28], is cooperative caching, wherein the caches cooperate in making

*Received by the editors February 11, 2008; accepted for publication (in revised form) May 7, 2008; published electronically August 27, 2008. This work is a combined version of two papers: an extended abstract by I. D. Baev and R. Rajaraman, *Approximation algorithms for data placement in arbitrary networks*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 661–670, and an unpublished manuscript by C. Swamy, *Algorithms for Data Placement Problems*, 2004.

http://www.siam.org/journals/sicomp/38-4/71542.html

†Java, Compilers, and Tools Laboratory, Hewlett-Packard Company, 11000 Wolfe Road, Cupertino, CA 95014 (ivan.baev@hp.com).

‡College of Computer Science, Northeastern University, Boston, MA 02115 (rraj@ccs.neu.edu). This author's research was supported by NSF CAREER award NSF CCR-9983901.

§Combinatorics and Optimization, University of Waterloo, Waterloo N2L 3G1, ON, Canada (cswamy@math.uwaterloo.ca). This author's research was supported in part by NSERC grant 32760-06. Part of this work was done while the author was a student at Cornell University, Ithaca, NY 14853.

storage decisions and in serving each other's requests. (Cooperation is of course likely to be the default mode under centralized control, where all the network nodes are under the control of a single entity, e.g., as in an organization's local area network.) Clearly, cooperation has the potential to improve system performance by reducing average access cost and improving storage-space utilization. A basic problem that arises in such a cooperative setup is the following: given a cost function specifying the cost of accessing an object stored at one location from another, and the access pattern of each node for each object, determine a placement or mapping of the data objects to caches so as to minimize the average cost of accessing the data objects.

We abstract this problem via the following mathematical formulation, which we call the *data placement problem*. We are given a set of caches $\mathcal{F}$, a set of data objects $\mathcal{O}$, and a set of clients $\mathcal{D}$. Each cache $i \in \mathcal{F}$ has a capacity u_i that limits the total number of data objects that may be stored in the cache. Each client $j \in \mathcal{D}$ has demand d_j for a specific data object $o(j) \in \mathcal{O}$ and has to be assigned to a cache that stores that object. Storing an object o in cache i incurs a *storage cost* of f_i^o, and assigning client j to cache i incurs an *access cost* of $d_j c_{ij}$ proportional to the "distance" c_{ij} between i and j. The storage costs could be used to model the cost of realizing a placement; e.g., the cost f_i^o might represent the cost of expunging some items from the cache in order to free up storage space. The data placement problem seeks a placement of the data objects to caches that respects the cache capacities, and an assignment of clients to caches so as to minimize the total storage and client access costs. More precisely, we want to determine a set of objects $\mathcal{O}(i) \subseteq \mathcal{O}$ to place in each cache $i \in \mathcal{F}$ satisfying $|\mathcal{O}(i)| \leq u_i$ and assign each client j to a cache $i(j)$ that stores object $o(j)$ (i.e., $o(j) \in \mathcal{O}(i(j))$) so as to minimize $\sum_{i \in \mathcal{F}} \sum_{o \in \mathcal{O}(i)} f_i^o + \sum_{j \in \mathcal{D}} d_j c_{i(j)j}$. As in several previous studies, especially on facility location [5, 2, 37, 34, 7, 6], we assume that the caches and clients are located in a common metric space, so the distances c_{ij} form a metric.

More generally, each object $o \in \mathcal{O}$ may have a *length* l_o, and the capacity u_i of cache $i \in \mathcal{F}$ now bounds the total length of data objects that may be stored in the cache. The access cost of an object is weighted by its length, so if client j is assigned to cache i, it incurs an access cost of $d_j l_{o(j)} c_{ij}$. Unless otherwise stated, we will use the data placement problem to denote the problem with unit (or, equivalently, uniform) object lengths.

The data placement problem can also be motivated from a facility-location perspective. In a typical facility-location setting, we are given a set of facilities with facility-opening costs and a set of clients with demands, and we want to open facilities and assign clients to open facilities so as to minimize the sum of the facility-opening costs and client-assignment costs. In various such applications, the clients are differentiated according to the kind of service they require, and in order to satisfy a client we need to assign (the demand of) the client to a facility where the service required by it has been "installed" (so that the facility can provide this service). The data placement problem can be used to abstract such settings, wherein the caches represent facilities and the objects correspond to the different services required by the clients; the storage cost models the cost of installing service at a given facility, and the cache capacity imposes a restriction on the *number of services* that may be installed at a facility. Shmoys, Swamy, and Levi [35] and Ravi and Sinha [33] introduced problems closely related to the data placement problem, motivated by such facility-location applications.

The data placement problem is a generalization of the metric *uncapacitated facility location* (UFL) problem and, hence, is *APX*-hard. Moreover, as we show in section 6

by a reduction from metric UFL, the problem (with uniform object lengths) remains *APX*-hard even when there are no storage costs.

Our results and techniques. Our main result (section 3) is a 10-approximation algorithm for the data placement problem. The algorithm we present here is an improvement over the approximation algorithm described in [3]. For the benefit of the reader, in section 1.1 we briefly sketch the differences between this algorithm and the one in [3].

Our algorithm is based on rounding an optimal solution to a natural linear-programming (LP) relaxation of the problem. Observe that the placement problem for each individual object is a UFL instance; however, these instances are coupled due to the cache-capacity constraints, which is what makes the problem hard. One of the main technical challenges faced in the rounding is to preserve the cache capacities while losing only a constant factor in the approximation ratio. Despite the similarity with UFL, hard capacities make it quite difficult to apply the standard rounding ideas underlying the design of approximation algorithms for UFL. All LP-based algorithms for UFL either employ filtering [26, 37] or use the dual to bound the solution cost [8, 18, 17, 6]. Filtering typically involves blowing up the LP variables, thereby violating the cache capacities, and the dual of the LP relaxation of the data placement problem contains negative variables (corresponding to the primal capacity constraints), which presents a serious obstacle to using the dual to bound the solution cost. Instead, we use the techniques developed by Charikar et al. [7] for the k-median problem.

Our algorithm proceeds in two phases. In the first phase, we build upon a clustering method introduced by Charikar et al. (Step 1 in [7]) and round the LP solution to a *half-integral* solution. In the second phase of our algorithm, we use the Shmoys–Tardos–Aardal clustering method [37] *without any filtering* to cluster the demand-nodes for each object and obtain a solution with the property that for every object o and cache i, there is at most one demand-node for o that is served by i. The key observation that allows us to do away with the problematic filtering step is that, in a half-integral solution, the distance between a client and any cache serving it fractionally is already bounded relative to its access cost in the half-integral solution. Once we have the aforementioned property, we can view the fractional solution as a feasible flow to a minimum-cost flow problem with integral capacities. By the integrality property of flows one can now extract an integer solution of no greater cost. This algorithm and its analysis are described in section 3.

The formulation of the data placement problem appears most suitable for applications where objects are rarely written. In a setting where write-requests are issued for data objects, one needs to have a separate mechanism to maintain consistency among the replicas of an object. In section 4, we formulate the *connected data placement problem*, which incorporates this aspect of data management (which is not captured by the data placement problem). As proposed by Krick, Räcke, and Westermann [22] in the context of another caching problem, we model this by requiring that, for every object o, all caches containing o be connected via a Steiner tree T_o to a root r_o. When a write-request is issued for object o, the root initiates an update of all the copies of object o using the tree T_o as a multicast tree. The objective is to minimize the total cost incurred in storing and accessing objects and building the Steiner trees, that is, to minimize $\sum_{i\in\mathcal{F}}\sum_{o\in\mathcal{O}(i)} f_i^o + \sum_{j\in\mathcal{D}} d_j c_{i(j)j} + \sum_{o\in\mathcal{O}} M_o \sum_{e\in T_o} c_e$, where the M_os are input scaling parameters. This generalizes the *connected facility location* problem [14, 39, 15] for which the best known guarantee is 8.55 [39]. We present a 14-approximation algorithm for this problem. One noteworthy feature here is the ease

with which one can interface the algorithm developed in section 3 (which handles the data placement part of the problem) with the rounding ideas proposed in [32, 14] to handle the connectivity aspect of the problem.

In section 5, we consider a couple of extensions. First, we consider the k-median variant, where, for every object o, there is a bound of k_o on the number of caches that may store object o. Our rounding algorithm is versatile and extends easily with minimal changes to this variant, yielding the same approximation guarantee. Second, we consider the data placement problem with arbitrary object lengths. It is easy to show (see section 6) via a reduction from the PARTITION problem that with arbitrary object lengths, it is *NP*-complete to even decide if there exists a feasible solution; hence, no approximation ratio is achievable in polynomial time unless $P = NP$. We can modify our algorithm to obtain a bicriteria approximation guarantee in this setting: we return a placement of cost at most 10 times the optimal where the total length of objects stored in a cache may exceed its capacity by the maximum object length. We conclude in section 6 with a couple of hardness results about the data placement problem with (i) no storage costs, and (ii) nonuniform object lengths.

Related work. The problem of data management in a distributed network has been extensively studied. Dowdy and Foster [10] initiated the study of cooperative caching in the context of allocating files in a distributed network, and this problem has since received much attention. We limit ourselves to an overview of the work in models that most closely resemble our model; the reader is referred to the surveys [10, 12] for a more detailed discussion.

Various works [5, 1, 2] have considered an *online* version of our problem, both with and without cache capacities, where read- and write-requests arrive online and have to be taken care of on the fly. The competitive ratios achievable in the online setting are, not surprisingly, weaker than the approximation ratios achievable in the offline setting. Awerbuch, Bartal, and Fiat [1] gave a randomized algorithm with competitive ratio $\text{polylog}(\sum_{i\in\mathcal{F}} u_i)$ for the uniform metric, whereas in [2] they give a $\text{polylog}(\max_{ij} c_{ij})$-competitive algorithm for arbitrary metrics, but require a $\text{polylog}(\max_{ij} c_{ij})$-factor blow-up in the cache capacities. Various studies have incorporated routing information into the caching problem, for instance, by having intermediate nodes cache copies of an object when the object is being routed [16, 31, 41], or by considering the problem of minimizing network congestion due to routing of requests [27, 28]. In contrast, we abstract away routing concerns by assuming that the c_{ij}-values, which determine the access costs, are given to us as input.

The *offline* data placement problem that we consider was first studied for hierarchical networks, or, equivalently, when the access costs form an ultrametric (a more restricted class of metrics). Leff, Wolf, and Yu [23] considered ultrametrics derived from a star, Korupolu, Plaxton, and Rajaraman [20] gave exact and approximation algorithms for general ultrametrics, and Korupolu and Dahlin [19] evaluated the practical performance of several placement algorithms for ultrametrics. Independent of [3], Meyerson, Munagala, and Plotkin [29] considered a generalization of our problem (called the page placement problem), where a cache also has a client capacity limiting the number of clients that may be assigned to it. They gave a constant approximation, but with a logarithmic violation of both the client capacities and the object capacities. Subsequently, Guha and Munagala [13] obtained a constant-factor approximation where the capacities are violated by only a constant factor. Fleischer et al. [11] considered a maximization version of the data placement problem with similar client-capacity constraints that limit the total demand that may be assigned to a cache. They give a $(1 - \frac{1}{e} - \epsilon)$-approximation algorithm, for any $\epsilon > 0$, and show

that no better guarantee is achievable unless $NP \subseteq \mathrm{DTIME}\left[n^{O(\log \log n)}\right]$.

As mentioned earlier, the data placement problem can also be motivated from a facility-location perspective, where caches correspond to facilities and the objects correspond to the different *services* required by clients. Shmoys, Swamy, and Levi [35] formulated a closely related problem in this context called *facility location with service installation costs (FLSIC)*. Using the terminology of the data placement problem, in FLSIC the caches (facilities) are uncapacitated, but one has to pay a location-dependent cache-setup (facility-opening) cost to "build" a cache at a location before storing any data object at that cache. Independently, Ravi and Sinha [33] proposed the multicommodity facility-location problem giving a similar motivation. Shmoys, Swamy, and Levi [35] give a 6-approximation algorithm for FLSIC under a certain assumption on the service installation costs.

The data placement problem (without storage costs) and FLSIC have also been studied for the special case of the *directed* line-metric under the names of *broadcast scheduling* [4] and the *joint replenishment problem* [24], respectively. In both problems, both the clients and the caches correspond to points on the timeline. In broadcast scheduling, the objects correspond to pages. A client corresponds to a request for a page, a cache corresponds to a page-broadcast, and a request at time t must be assigned to a broadcast of that page at some time $t' > t$. At most c pages may be broadcast at any time; the goal is to minimize the average response time of the requests. The best-known approximation factor for this problem is $O\left(\frac{\log^2 |\mathcal{O}|}{\log \log |\mathcal{O}|}\right)$ due to Bansal, Coppersmith, and Sviridenko [4]. In the joint replenishment problem, the objects are items. Demands for items occur at various points of time, and one has to determine which items to order at which times, so that all demand can be met by orders that are placed at earlier points of time. Placing an order for a subset of items incurs a joint ordering cost to start the order and an item-dependent cost, and each demand gets charged the cost incurred to hold the inventory for that demand. Levi, Roundy, and Shmoys [24] gave a 2-approximation algorithm for this problem.

The data placement problem is a generalization of UFL, which corresponds to the special case with only one object. There is a large body of literature that deals with designing approximation algorithms for metric UFL; see [34] for a survey of this and earlier work. The first constant approximation guarantee for UFL was obtained by Shmoys, Tardos, and Aardal [37] via an LP-rounding algorithm, and the current state-of-the-art is a 1.5-approximation algorithm due to Byrka [6]. For the closely related *k-median problem*, the first constant-factor approximation algorithm was given by Charikar et al. [7] using LP rounding. As mentioned earlier, the clustering method developed by them plays a key role in our algorithm.

Finally, we remark that the presence of cache capacities might suggest a similarity to the capacitated facility location (CFL) problem, but this resemblance is only superficial. The capacities in our problem limit the number of objects that may be assigned to a cache, but there is *no bound* on the demand (or number) of clients requesting a given object, or total demand, that may be assigned to a cache. (The data placement problem may be viewed as a collection of *UFL instances*, one for each object, that are coupled due to the cache-capacity constraints.) In particular, as our algorithm shows, the integrality gap of the natural LP relaxation for our problem is at most a constant, whereas there is no known LP relaxation of CFL with constant integrality gap. Moreover, the local search algorithms in [21, 9, 30, 42] do not directly apply, and it is not clear if they can be adapted to our problem.

1.1. Relationship with the work of [3] and [38]. This work is a merger of two earlier papers: an extended abstract of Baev and Rajaraman [3] and an unpublished manuscript of Swamy [38]. In order to place our work in proper bibliographic context, and for the benefit of the reader, we include a brief comparison of our work with [3] and [38].

The data placement problem that we consider was introduced by Baev and Rajaraman [3]. (In their model, each client j has demand d_{jo} for every object $o \in \mathcal{O}$; this easily reduces to the model considered here since one can create a colocated copy $j^{(o)}$ of client j with demand d_{jo} for every object o.) They gave a 20.5-approximation algorithm for this problem, which is based on rounding an optimal solution to the same LP relaxation of the problem that we consider. The 10-approximation algorithm described in this paper is from [38] and is based on an improved rounding procedure for the same LP. We briefly describe the main differences between the two algorithms.

As described earlier, our algorithm proceeds in two phases. The first phase of our algorithm, where we round the LP solution to a *half-integral* solution, is identical to the first half (steps 1–3) of the algorithm in [3]. From here on the two algorithms proceed along different tracks. In both algorithms, the goal is to modify the previously obtained half-integral solution into one that has the property that for every object o and cache i, there is at most one demand-node for o that is served by i, so that one can then set up a minimum-cost flow problem to round the half-integral solution to an integral one. In the second phase of our algorithm, we use the Shmoys–Tardos–Aardal clustering method [37] (without filtering) to obtain a solution with the above property. In contrast, the Baev–Rajaraman algorithm dovetails the rounding procedure of [7] (creating 1-level trees that are used to cluster the clients) to obtain a solution with the aforementioned property. By adopting a different clustering approach that better exploits half-integrality, we obtain a simpler algorithm that also yields a much better approximation guarantee.

The connected data placement problem was introduced by Swamy [38], and the 14-approximation algorithm that we present for this problem was described therein.

2. An LP relaxation. We can express the data placement problem as an integer program and relax the integrality constraints to get a linear program. Throughout we will use i to index the caches in $\mathcal{F}$, j to index the clients in $\mathcal{D}$, and o to index the objects in $\mathcal{O}$.

$$\text{(P)} \qquad \min \quad \sum_i \sum_o f_i^o y_i^o + \sum_j \sum_i d_j c_{ij} x_{ij}$$

$$\text{subject to} \qquad \sum_i x_{ij} \geq 1 \qquad \forall j,$$

$$x_{ij} \leq y_i^{o(j)} \qquad \forall i, j,$$

$$\text{(1)} \qquad \sum_o y_i^o \leq u_i \qquad \forall i,$$

$$x_{ij}, y_i^o \geq 0 \qquad \forall i, j, o.$$

Variable y_i^o indicates if object o is stored in cache i, and x_{ij} indicates if client j is assigned to cache i. The first and second constraints say that each client must be assigned to a cache and if client j is assigned to cache i, then object $o(j)$ must be

stored in cache i. The third constraint states that the total length of items stored in any cache i is at most its capacity u_i. An integer solution corresponds exactly to a solution to our problem. We let G_o denote the set of clients that demand object o, i.e., $G_o = \{j : o(j) = o\}$.

3. The rounding procedure. Let (x, y) denote the optimal solution to (P) and OPT be its value. We will round this to an integer solution losing a factor of at most 10. We use the terms access cost and assignment cost interchangeably.

3.1. Overview of the algorithm. We first give a high level description of the algorithm. Suppose for a moment that the optimal solution (x, y) satisfies the following property: for any cache i and object o, there is *at most one* client $j \in G_o$ such that $x_{ij} > 0$ $(*)$. We can then set up the following min-cost flow problem: create a bipartite graph with vertex set $\mathcal{D} \cup \mathcal{F}$ and edges (i, j) for every i, j such that $x_{ij} > 0$ with cost $f_i^{o(j)} + d_j c_{ij}$ and capacity 1; client j has a demand of 1, and cache i has capacity u_i. The LP solution translates to a feasible fractional flow in this graph of cost at most OPT. *Note that property* $(*)$ *is crucial for this.* Conversely an integer flow yields an integer solution to (P) of cost equal to the flow cost. Therefore by the integrality property of flows (given integer capacities) we can round (x, y) to an integer solution of no greater cost. Of course, the LP solution need not have property $(*)$, so our goal will be (loosely speaking) to transform (x, y) to a solution that has property $(*)$ without increasing the cost by much. One of the major challenges encountered is to do this transformation *without violating the cache capacities*, while increasing the cost by only a constant factor.

Roughly speaking we want to do the following: for each object o, cluster the clients in G_o around certain "centers" (also clients in G_o) such that (a) every client k is mapped to a "nearby" cluster center j whose LP assignment cost is less than that of k, and (b) the facilities serving the cluster centers in the fractional solution (x, y) are disjoint. Thus, the modified instance where the demand of a client is moved to the center of its cluster has a fractional solution, namely, the solution induced by (x, y), that satisfies $(*)$ and has cost at most OPT. Furthermore, given a solution to the modified instance we can obtain a solution to the original instance losing a small additive factor. This clustering idea lies at the core of most algorithms for facility location; however, the necessity of preserving cache capacities renders many of the known clustering methods [37, 8, 25] unsuitable for our purposes. For example, one option is to use the decomposition method of Shmoys, Tardos, and Aardal [37] that produces precisely such a clustering. The problem, however, is that [37] uses filtering which involves blowing up the x_{ij} and y_i^o values and thus violating the cache capacities. Chudak and Shmoys [8] and Levi, Shmoys, and Swamy [25] use similar clustering ideas but without filtering, using the dual solution to bound (portions of) the cost. The difficulty here in bounding the cost using the dual solution is that there are terms with negative coefficients in the dual objective function that correspond to the primal capacity constraints (1). Although [40, 25] showed that it is possible to overcome this difficulty in certain cases, the situation here looks more complicated, and it is not clear how to use their techniques.

Instead, we use the clustering technique of Charikar et al. [7] developed for the k-median problem. Our algorithm proceeds in two phases. In the first phase (section 3.2), we extract a modified instance and a fractional solution to this instance from the LP solution, and round this to a *half-integral* solution $(\hat{x}, \hat{y})$, that is, each $\hat{x}_{ij}, \hat{y}_i \in \left\{0, \frac{1}{2}, 1\right\}$, losing a factor of 3. Further, any solution here will give a solution

to the original instance while increasing the cost by at most $4 \cdot OPT$. We do this by first transferring demands to certain well-separated centers (Step I) exactly as in the demand-consolidation step of [7], so as to ensure that each center has its own private set of caches that serve it to an extent of at least half. This allows us to set up a minimum-cost flow problem (Step II) with half-integral capacities with a one-one correspondence between solutions and flows, and thereby round the fractional solution on the centers to a half-integral solution.

In phase two (section 3.3), we observe that we can now use the clustering method in [37] on the half-integral solution $(\hat{x}, \hat{y})$ *without any filtering* (Step III) since such a solution is essentially already filtered: if client j is assigned to i and i' in $\hat{x}$, then $c_{ij}, c_{i'j} \leq 2(c_{ij}\hat{x}_{ij} + c_{i'j}\hat{x}_{i'j})$. This clustering satisfies the requirements (a) and (b) mentioned above. Thus, one can obtain an integer solution for the new cluster centers by solving a suitable min-cost flow problem. This is essentially what we do, but we set up the min-cost flow problem more carefully (Step IV) so as to lose only a factor of 2 in converting $(\hat{x}, \hat{y})$ to an integer solution (for the modified instance extracted in phase 1). So overall we get an approximation ratio of $4 + 2 \times 3 = 10$ (Theorem 3.5).

We now describe each of these steps in detail. Let $\bar{C}_j = \sum_i c_{ij} x_{ij}$ denote the cost incurred by the LP solution to assign one unit of demand of client j.

3.2. Obtaining a half-integral solution $(\hat{x}, \hat{y})$.

Step I: *Consolidating demands around centers.* We first consider every object o separately and consolidate (or cluster) the demand of clients in G_o at certain clients, which we call *cluster centers*. We do not modify the fractional solution (x, y) but modify only the demands so that for some clients j, the demand d_j is "moved" to a "nearby" center k. We assume every client has a nonzero demand.

Set $d'_j \leftarrow 0$ for every j. Consider the clients in G_o in increasing order of $\bar{C}_j$. For each client j, if there exists a client k (including j) such that $d'_k > 0$ and $c_{jk} < 4\max(\bar{C}_j, \bar{C}_k) = 4\bar{C}_j$, set $d'_k \leftarrow d'_k + d_j$; otherwise set $d'_j \leftarrow d_j$. We do this for every object o. Let $D_o = \{j \in G_o : d'_j > 0\}$ and $D = \bigcup_o D_o$. Each client in D is a cluster center. Let $OPT' = \sum_{i,s} f_i^o y_i^o + \sum_{j \in D, i} d'_j c_{ij} x_{ij}$ denote the cost of (x, y) for the modified instance consisting of the cluster centers. Since the demand of each client $k \notin D$ moves a distance of at most $4\bar{C}_k$, it is clear that any solution to the modified instance yields a solution for client-set $\mathcal{D}$ incurring an additive factor of at most $4\sum_{k \notin D} d_k \bar{C}_k \leq 4 \cdot OPT$. We obtain the following lemma.

LEMMA 3.1. *The following hold:* (i) *if* $j, k \in D_o$, *then* $c_{jk} \geq 4\max(\bar{C}_j, \bar{C}_k)$; (ii) $OPT' \leq OPT$; *and* (iii) *any solution* (x', y') *to the modified instance can be converted to a solution to the original instance incurring an additional cost of at most* $4 \cdot OPT$.

From now on we will focus on the modified instance with client-set D and modified demands d'_j. At the very end we will use the above lemma to translate an integer solution to the modified instance to an integer solution to the original instance.

Step II: *Transforming to a half-integral solution.* We define the cluster of a client $j \in D_o$ to consist of all clients $k \in G_o$ whose demand d_k was moved to j, and a set of facilities F_j. F_j consists of all facilities i to which j is fractionally assigned such that j is the center in D_o closest to i; that is, $F_j = \{i : x_{ij} > 0 \text{ and } c_{ij} = \min_{k \in D_o} c_{ik}\}$, with ties broken arbitrarily. Let $F'_j \subseteq F_j = \{i \in F_j : c_{ij} \leq 2\bar{C}_j\}$. Define γ_j to be $\min_{i \notin F_j : x_{ij} > 0} c_{ij}$. Clearly the sets F_j for $j \in D_o$ are disjoint. By property (i) of Lemma 3.1, we have that F_j contains all the facilities i such that $x_{ij} > 0$ and $c_{ij} \leq 2\bar{C}_j$. So $\sum_{i \in F'_j} x_{ij} = \sum_{i : c_{ij} \leq 2\bar{C}_j} x_{ij} \geq \frac{1}{2}$, where the last inequality follows from Markov's inequality.

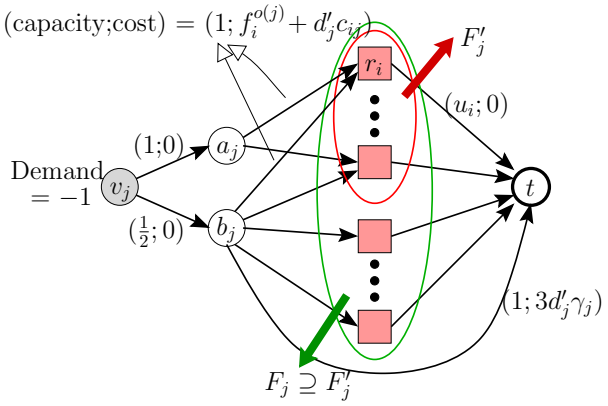


FIG. 3.1. *The min-cost flow network constructed in Step* II. *The tuple labeling an edge gives the (capacity;cost) for the edge.*

In the half-integral solution $(\hat{x}, \hat{y})$, we will store object o only at caches that lie in some set F_j for $j \in D_o$. To obtain $(\hat{x}, \hat{y})$, we set up a min-cost flow problem. We create a sink t and a node r_i for every cache i in $\bigcup_{j \in D} F_j$ with an outgoing edge (r_i, t) of capacity u_i and cost 0 (see Figure 3.1). For each client $j \in D$ we create three nodes v_j, a_j, and b_j. Node v_j has demand -1 (i.e., the net outgoing flow should be 1) to denote the requirement that j must be assigned to a cache. We add edges (v_j, a_j) with capacity 1 and cost 0, and (v_j, b_j) with capacity $\frac{1}{2}$ and cost 0. Node a_j represents the option that j is assigned to a facility in F'_j, so we add edges (a_j, r_i) to every $i \in F'_j$ with capacity 1 and cost $f_i^{o(j)} + d'_j c_{ij}$. Notice that setting the capacity of (v_j, b_j) to $\frac{1}{2}$ forces j to be assigned to an extent of at least $\frac{1}{2}$ to facilities in F'_j. Node b_j signifies that j is assigned either to a facility in F_j or to some other facility. To encode this, we add edges (b_j, r_i) to every $i \in F_j$ with capacity 1 and cost $f_i^{o(j)} + d'_j c_{ij}$, and an edge (b_j, t) with capacity 1 and cost $3d'_j\gamma_j$ (since, as we show later, there is always a facility at distance at most $3\gamma_j$ from j that is at least half-open). Figure 3.1 shows the portion of the min-cost flow instance corresponding to client j.

Since all edge capacities are $\frac{1}{2}$ or 1, the network has a half-integral min-cost flow. Given such a flow, we obtain $(\hat{x}, \hat{y})$ as follows. We initialize all $\hat{x}_{ij}$, $\hat{y}_i^o$ to 0. Consider object o. For every $j \in D_o$ and cache $i \in F'_j$, we set $\hat{y}_i^o = \hat{x}_{ij} =$ flow along (a_j, r_i) + flow along (b_j, r_i). For every $i \in F_j \setminus F'_j$, we set $\hat{y}_i^o$ and $\hat{x}_{ij}$ equal to the flow along edge (b_j, r_i). Observe that there is at least one cache $i \in F'_j$ such that $\hat{x}_{ij} > 0$; we call the cache in F'_j closest to j with $\hat{x}_{ij} > 0$ the *primary cache* of j. Note that since the sets F_j (and hence F'_j) for $j \in D_o$ are disjoint, every client in D_o has a unique primary cache i. Let i' be the cache nearest to j, other than its primary cache, with $\hat{y}_{i'}^o > 0$. If edge (b_j, t) carries positive flow (so no edge (b_j, r_i) carries any flow implying that $\hat{y}_i^o = 0$ for every $i \in F_j \setminus F'_j$), we set $\hat{x}_{i'j} =$ flow on $(b_j, t) = \frac{1}{2}$. We do this for every object o. If a client j is assigned to a cache other than its primary cache, we call the other cache the *secondary cache* of j. It is easy to verify that $(\hat{x}, \hat{y})$ is a feasible solution to (P), where the client-set is D. The following lemma shows that the cost of $(\hat{x}, \hat{y})$ is at most $3 \cdot OPT$.

LEMMA 3.2. *The cost of* $(\hat{x}, \hat{y})$, *that is,* $\sum_{i,o} f_i^o \hat{y}_i^o + \sum_{j \in D, i} d'_j c_{ij} \hat{x}_{ij}$, *is at most* $3 \cdot OPT' \leq 3 \cdot OPT$.

Proof. First we show that (x, y) induces a flow of cost at most $3 \cdot OPT'$, so the cost

of the min-cost flow is no greater. Then we show that the cost of $(\hat{x}, \hat{y})$ is bounded by the cost of the min-cost flow.

Consider the following flow: each edge (v_j, a_j) has flow $\sum_{i \in F'_j} x_{ij}$, (v_j, b_j) has flow $1 - \sum_{i \in F'_j} x_{ij}$, and (b_j, t) has flow $1 - \sum_{i \in F_j} x_{ij}$; every edge (a_j, r_i) or (b_j, r_i) has flow x_{ij}; the flow on (r_i, t) is $\sum_o \sum_{j \in D_o : i \in F_j} x_{ij}$. It is easy to see that this is a feasible flow. The cost of this flow is

$$\sum_{o, j \in D_o} \left(\sum_{i \in F_j} (d'_j c_{ij} + f_i^o) x_{ij} + 3 d'_j \gamma_j \left(1 - \sum_{i \in F_j} x_{ij} \right) \right)$$

$$\leq \sum_{i,o} f_i^o y_i^o + \sum_j d'_j \left(\sum_{i \in F_j} c_{ij} x_{ij} + 3\gamma_j \left(1 - \sum_{i \in F_j} x_{ij} \right) \right).$$

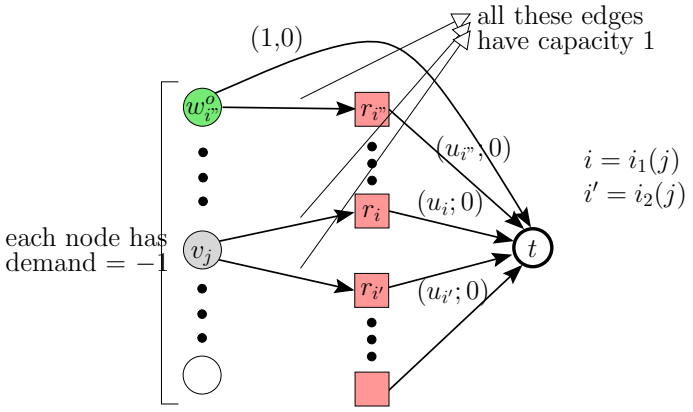
We have $OPT' = \sum_{i,o} f_i^o y_i^o + \sum_j d'_j \bar{C}_j$. For any $j \in D$, $\bar{C}_j = \sum_{i \in F_j} c_{ij} x_{ij} + \sum_{i \notin F_j} c_{ij} x_{ij} \geq \sum_{i \in F_j} c_{ij} x_{ij} + \gamma_j (1 - \sum_{i \in F_j} x_{ij})$ since γ_j was defined as $\min_{i \notin F_j : x_{ij} > 0} c_{ij}$. This shows that the cost of the constructed flow, and hence of the min-cost flow, is at most $3 \cdot OPT'$.

Now consider the solution $(\hat{x}, \hat{y})$ induced by the half-integral min-cost flow. By construction, the quantity $\sum_{i,o} f_i^o \hat{y}_i^o + \sum_{j \in D, i \in F_j} d'_j c_{ij} \hat{x}_{ij}$ is exactly equal to the total cost of the flow on edges (a_j, r_i) and (b_j, r_i). For any $j \in D$ the remaining cost $\sum_{i \notin F_j} d'_j c_{ij} \hat{x}_{ij}$ is equal to $d'_j c_{i'j} \cdot (\text{flow on } (b_j, t))$, where i' is the secondary cache of j. So it suffices to show that $c_{i'j} \leq 3\gamma_j$. Let $\gamma_j = c_{i''j}$, where $i'' \notin F_j$ and $x_{i''j} > 0$. Let k be the center in D_o nearest to i'' and let ℓ be the primary cache of k. Then, $c_{i'j} \leq c_{\ell j}$ and $4 \max(\bar{C}_j, \bar{C}_k) \leq c_{jk} \leq c_{i''j} + c_{i''k} \leq 2\gamma_j$. Also $c_{\ell k} \leq 2\bar{C}_k$ since $\ell \in F'_k$. Combining the inequalities we get that $c_{i'j} \leq 3\gamma_j$ which completes the proof of the lemma. ∎

3.3. Converting $(\hat{x}, \hat{y})$ to an integer solution $(\tilde{x}, \tilde{y})$. Define $\hat{C}_j = \sum_i c_{ij} \hat{x}_{ij}$ for $j \in D$. Let $i_1(j)$ denote the primary cache of j. For convenience, we will say that every client $j \in D$ has both a primary cache $i_1(j)$ and a secondary cache i' with $\hat{x}_{i_1(j)j} = \hat{x}_{i'j} = \frac{1}{2}$, with the understanding that if j does not have a secondary cache, then i' is a copy of $i_1(j)$, so effectively $\hat{x}_{i_1(j)j} = 1$. We denote the secondary cache by $i_2(j)$. Then we have $\hat{C}_j = \frac{1}{2}(c_{i_1(j)j} + c_{i_2(j)j})$, $c_{i_1(j)j} \leq \hat{C}_j$, and $c_{i_1(j)j} \leq c_{i_2(j)j} \leq 2\hat{C}_j$. Notice that $i_1(j)$ and $i_2(j)$ are the (one or) two caches with $\hat{y}_i^{o(j)} > 0$ that are nearest to j. Let $L_o = \{i : \hat{y}_i^o > 0\}$ and $L = \bigcup_o L_o$.

Step III: *Clustering.* First for every object o we cluster the clients in D_o as follows: pick $j \in D_o$ with smallest $\hat{C}_j$. Remove every client $k \in D_o$ such that both j and k are (fractionally) assigned to a cache $i \in L_o$, and recurse on the remaining set of clients until no client in D_o is left. Let D'_o be the set of clients picked for object o and let $D' = \bigcup_o D'_o$; D' denotes the *new cluster centers*. It is clear that for any cache in L_o at most one client in D'_o is assigned to it. Observe that for every client $k \in D_o \setminus D'_o$ there is some $j \in D'_o$ such that $\hat{C}_j \leq \hat{C}_k$ and $\hat{x}_{ij}, \hat{x}_{ik} > 0$ for some $i \in L_o$, implying that $c_{jk} \leq 4\hat{C}_k$. We call j the *center* of k and denote it by $\mathsf{ctr}(k)$.

Now for every client $k \in D \setminus D'$ we can move its demand d'_k to $j = \mathsf{ctr}(k)$. The resulting instance with client-set D' (and the new demands) satisfies the property $(*)$ mentioned in section 3.1. Hence, one can set up a min-cost flow problem as mentioned in section 3.1 to get an integer solution to the instance with client-set D', which translates to a solution with client-set D (and the original demands d'_j).



cost of (v_j, r_i) is $f_i^{o(j)} + d'_j c_{ij} + \sum_{k \in A_j} d'_k c_{ik}$
cost of $(v_j, r_{i'})$ is $f_{i'}^{o(j)} + d'_j c_{i'j} + \sum_{k \in A_j} d'_k c_{i'k} + \sum_{k \in B_j} d'_k (c_{i'k} - c_{i_2(k)k})$
cost of $(w^o_{i''}, r_{i''})$ is $f^o_{i''} + \sum_{k \in B^o_{i''}} d'_k (c_{i''k} - c_{i_2(k)k})$

FIG. 3.2. *The min-cost flow network constructed in Step* IV. *The tuple labeling an edge gives the (capacity;cost) for the edge.*

Doing this naively, we lose an additive factor of (at most) $4 \sum_{k \in D} d'_k \hat{C}_k$ in translating the demands back from D' to D. We will set up the min-cost flow network more carefully so that we lose only a multiplicative factor of 2 in rounding $(\hat{x}, \hat{y})$ to an integer solution for the client-set D. We want to capture the following observation: suppose the demand of a client $k \in D_o$ is moved to $j = \mathsf{ctr}(k)$. Let $\hat{x}_{ik} = \hat{x}_{i'k} = \frac{1}{2}$ and $\hat{x}_{ij} = \hat{x}_{i''j} = \frac{1}{2}$. The per-unit-demand assignment cost of k is at most $\frac{1}{2}(c_{ik} + c_{i''k}) \leq c_{ik} + \hat{C}_j \leq 3\hat{C}_k$ (since $c_{i''k} \leq c_{i''j} + c_{ij} + c_{ik}$), which is much less than the naive bound of $4\hat{C}_k + \hat{C}_j$.

Step IV: *The min-cost flow network.* Fix $o \in \mathcal{O}$ and consider a client $j \in D'_o$. We will maintain two sets A_j and B_j for j. Let $i = i_1(j)$ and $i' = i_2(j)$ be the primary and secondary caches of j. We define $A_j = \{k \in D_o : \mathsf{ctr}(k) = j\}$ and $B_j = \{k \in D_o : \mathsf{ctr}(k) \neq j \text{ and } i' = i_1(k)\}$. Also, for every cache $i \in L_o$ such that $\hat{x}_{ij} = 0$ *for every* $j \in D'_o$, we define $B^o_i = \{k \in D_o : i = i_1(k)\}$ (which is either empty or a singleton). Note that all the sets A_j, B_j, and B^o_i are subsets of $D_o \setminus D'_o$.

We create a sink t, and a node r_i for every $i \in L$ for which $\hat{x}_{ij} > 0$ for some $j \in D'$, or $B^o_i \neq \phi$ for some o (see Figure 3.2). We have an edge (r_i, t) of capacity u_i and cost 0. For every client $j \in D'$ we create a node v_j. Further, for every $i \in L, o \in \mathcal{O}$ with $B^o_i \neq \phi$ we create a node w^o_i. The nodes v_j and w^o_i all have demand -1. For every node v_j we have edges (v_j, r_i) to every i with $\hat{x}_{ij} > 0$, and we have edges (w^o_i, r_i), (w^o_i, t) for every node w^o_i. All these edges have capacity 1. The cost of these edges is set as follows. Consider a node v_j and let $i = i_1(j), i' = i_2(j)$. We set the cost of (v_j, r_i) to $f_i^{o(j)} + d'_j c_{ij} + \sum_{k \in A_j} d'_k c_{ik}$ and the cost of $(v_j, r_{i'})$ to $f_{i'}^{o(j)} + d'_j c_{i'j} + \sum_{k \in A_j} d'_k c_{i'k} + \sum_{k \in B_j} d'_k (c_{i'k} - c_{i_2(k)k})$. We set the cost of (w^o_i, r_i) to $f^o_i + \sum_{k \in B^o_i} d'_k (c_{ik} - c_{i_2(k)k})$ and the cost of (w^o_i, t) to 0; see Figure 3.2.

Since all capacities are integer, there is an integer min-cost flow. We map this to an integer solution $(\tilde{x}, \tilde{y})$ to the instance with client-set D. Set $\tilde{x}_{ij}, \tilde{y}_i \leftarrow 0$ for all i, j. Consider object o. First, for every $j \in D'_o$ and $i \in \{i_1(j), i_2(j)\}$, we set

$\tilde{x}_{ij}$ = flow on edge (v_j, r_i). For every client $k \in B_j$ we set $\tilde{x}_{i_2(j)k} = \tilde{x}_{i_2(j)j}$, and for every $k \in B_i^o$ we set $\tilde{x}_{ik}$ = flow on (w_i^o, r_i). Next, for every $j \in D'_o$ and every $k \in A_j$ that has not yet been assigned (i.e., $\sum_i \tilde{x}_{ik} = 0$), we set $\tilde{x}_{ik} = \tilde{x}_{ij}$ for $i \in \{i_1(j), i_2(j)\}$. Finally, set $\tilde{y}_i^o = \max_{j \in D_o} \tilde{x}_{ij}$. We do this for every o. Observe that $\tilde{y}_i^o = 1$ for at most one facility from F'_j for every client $j \in D_o$. This will be useful in section 4. It is easy to see that $(\tilde{x}, \tilde{y})$ is a feasible integer solution. We now bound its cost.

LEMMA 3.3. *The cost of the min-cost flow in the network is at most twice the cost of* $(\hat{x}, \hat{y})$.

Proof. We exhibit a fractional flow of cost at most the claimed cost. The fractional flow is obtained by setting the flow on every edge (v_j, r_i) to $\hat{x}_{ij}$, and the flow on (w_i^o, t) and (w_i^o, r_i) to $\max_{k \in B_i^o} \hat{x}_{ik} = \frac{1}{2}$, where the equality follows since every $k \in B_i^o$ is assigned to an extent of $\frac{1}{2}$ to $i_2(k) \neq i$. The flow on the edges (r_i, t) is set accordingly to $\sum_o \big(\sum_{j \in D'_o} \hat{x}_{ij} + \max_{k \in B_i^o} \hat{x}_{ik}\big)$. This is a feasible flow since, for every i, o, either $B_i^o = \phi$ and there is exactly one $j \in D'_o$ such that $\hat{x}_{ij} > 0$, or $B_i^o \neq \phi$ and $\hat{x}_{ij} = 0$ for every $j \in D'_o$. So $\sum_{j \in D'_o} \hat{x}_{ij} + \max_{k \in B_i^o} \hat{x}_{ik}$ is at most y_i^o.

The cost of an edge (v_j, r_i) or (w_i^o, r_i) consists of a storage component ($f_i^{o(j)}$ or f_i^o) and an assignment component that can be attributed to various clients. We call the contribution of the storage components to the flow cost the *flow storage cost*, and the contribution of the assignment components the *flow assignment cost*. The flow storage cost is $\sum_{i,o} f_i^o \big(\sum_{j \in D'_o} \hat{x}_{ij} + \max_{k \in B_i^o} \hat{x}_{ik}\big) \leq \sum_{i,o} f_i^o y_i^o$ by the above reasoning. To evaluate the flow assignment cost we consider the contribution of each client to the assignment components separately. Fix an object o. First consider $j \in D'_o$ with $i = i_1(j)$, $i' = i_2(j)$. Client j figures only in the assignment component of (v_j, r_i) and $(v_j, r_{i'})$, and its contribution is $d'_j(c_{ij}\hat{x}_{ij} + c_{i'j}\hat{x}_{i'j}) = d'_j \hat{C}_j$. A client $k \in D_o \setminus D'_o$ is in exactly one set A_j, where $j = \mathsf{ctr}(k)$, and may possibly also lie in one of the sets $B_{j'}$ or $B_{i''}^o$. Let $i = i_1(j)$ and $i' = i_2(j)$.

1. If k does not lie in any set $B_{j'}$ or $B_{i''}^o$, then it must be that $\hat{x}_{i_1(k)j} > 0$. Client k contributes only to the assignment component of edges (v_j, r_i) and $(v_j, r_{i'})$, and this contribution is $d'_k(c_{ik}\hat{x}_{ij} + c_{i'k}\hat{x}_{i'j}) \leq d'_k(c_{i_1(k)k} + \hat{C}_j) \leq 2d'_k \hat{C}_k$ since $\hat{x}_{ij} = \hat{x}_{i'j} = \frac{1}{2}$ and $c_{ik} + c_{i'k} \leq 2c_{i_1(k)k} + c_{ij} + c_{i'j}$.

2. Now suppose k is also in one of the sets $B_{j'}$ or $B_{i''}^o$, so that it also contributes to the assignment component of an edge $(v_{j'}, r_{i_1(k)})$ or an edge $(w_{i''}^o, r_{i''})$. The contribution in both cases is $\frac{d'_k}{2}(c_{i_1(k)k} - c_{i_2(k)k})$ since we must have $x_{i_1(k)j'} = \frac{1}{2} = x_{i''k}$. Adding the contributions to edges (v_j, r_i) and $(v_j, r_{i'})$, the total contribution is $\frac{d'_k}{2}(c_{ik} + c_{i'k} + c_{i_1(k)k} - c_{i_2(k)k}) \leq d'_k(\hat{C}_k + \hat{C}_j) \leq 2d'_k \hat{C}_k$ since $c_{ik} + c_{i'k} \leq 2c_{i_2(k)k} + c_{ij} + c_{i'j}$.

So the flow assignment cost is at most $2\sum_{j \in D} d'_j \hat{C}_j$. Thus the total flow cost is at most $\sum_{i,o} f_i^o \hat{y}_i^o + 2\sum_{j \in D} d'_j \hat{C}_j$, which is at most twice the cost of $(\hat{x}, \hat{y})$. ∎

LEMMA 3.4. *The cost of the integer solution* $(\tilde{x}, \tilde{y})$ *is at most the cost of the min-cost integer flow.*

Proof. Observe that, for any o, $\sum_i f_i^o \tilde{y}_i^o = \sum_{i, j \in D'_o} f_i^o \tilde{x}_{ij} + \sum_{\text{nodes } w_i^o} f_i^o$ (flow on (w_i^o, r_i)). So the total storage cost is $\sum_{e=(v_j, r_i)} f_i^{o(j)}$ (flow on e) $+ \sum_{e=(w_i^o, r_i)} f_i^o$ (flow on e) which is just the flow storage cost.

We will bound the assignment cost of a client by the contribution it makes to the flow assignment cost. Fix object o. Consider $j \in D'_o$. Let $i = i_1(j)$ and $i' = i_2(j)$. At most one of the edges (v_j, r_i), $(v_j, r_{i'})$ carries nonzero flow, and we set $\tilde{x}_{ij}, \tilde{x}_{i'j}$ equal to the flow on the corresponding edge. So the assignment cost of j is

$d'_j(c_{ij}(\text{flow on } (v_j, r_i)) + c_{i'j}(\text{flow on } (v_j, r_{i'})))$, which is also the contribution of j to the assignment flow cost. The same argument holds for $k \in A_j$ if k is assigned to one of i or i'. The remaining case is when $k \in A_j$, and k is not assigned to i or i', but is assigned to $i'' = i_1(k)$ either because $k \in B_{j'}$ where $i'' = i_2(j')$ and $(v_{j'}, r_{i''})$ has nonzero flow, or because $k \in B^o_{i''}$ and $(w^o_{i''}, r_{i''})$ carries nonzero flow. The assignment cost of k is $d'_k c_{i''k}$. The contribution of k to the assignment flow cost is at least $d'_k(c_{i''k} - c_{i_2(k)k}) + d'_k \min(c_{ik}, c_{i'k})$ since $k \in A_j$. This is at least $d'_k c_{i''k}$ since both $c_{ik}, c_{i'k}$ are at least $c_{i_2(k)}$. So the assignment cost of $(\tilde{x}, \tilde{y})$ is bounded by the flow assignment cost. This completes the proof. ∎

Combining Lemmas 3.1–3.4, we obtain that $(\tilde{x}, \tilde{y})$ yields an integer solution to the original instance of cost at most $10 \cdot OPT$. Thus, we obtain the following theorem.

THEOREM 3.5. *There is a* 10*-approximation algorithm for the data placement problem.*

4. The connected data placement problem. The formulation of the data placement problem seems most suitable for applications where objects are rarely written. In the presence of write-requests, one needs to have a mechanism that ensures that all the copies of a data object replicated in the various caches are consistent, and this requires that a write-request updates all the replicas of the data object. One way of modeling this, as proposed by Krick, Räcke, and Westermann [22], is to insist that all caches containing the same data object be interconnected via a Steiner tree, which would serve as a multicast tree that is used to update all copies of an object when a write-request is issued for it.

This gives rise to the *connected data placement problem*. We assume that there is a root $r_o \in \mathcal{D} \cup \mathcal{F}$ for each object o that issues the multicast message when a write-request is issued for o, and require that all caches containing object o be connected to r_o. Thus, our goal is to find a placement $\{\mathcal{O}(i)\}_{i \in \mathcal{F}}$ of objects to caches respecting the cache-capacity constraints, assign each client j to a cache $i(j)$ containing the object $o(j)$, and, for each object o, connect the caches storing object o to r_o via a Steiner tree T_o, so as to minimize

$$\sum_{i \in \mathcal{F}} \sum_{o \in \mathcal{O}(i)} f^o_i + \sum_{j \in \mathcal{D}} d_j c_{i(j)j} + \sum_{o \in \mathcal{O}} M_o \sum_{e \in T_o} c_e.$$

Here $M_o \geq 1$ is an input scaling parameter; e.g., it might denote the total number of write-requests for object o.

The LP relaxation (P) is modified as follows. We introduce variables $z^o_e \geq 0$ for each object o, and each edge e (of the complete graph on $\mathcal{D} \cup \mathcal{F}$) that indicates (in the integer program) if edge e is part of the tree T_o. The objective function includes the additional term $\sum_o M_o \sum_e c_e z^o_e$. For each object o, set $S \subseteq \mathcal{D} \cup \mathcal{F}$ such that $r_o \notin S$, and client $j \in G_o$, we add the constraint $\sum_{e \in \delta(S)} z^o_e \geq \sum_{i \in S} x_{ij}$, where $\delta(S) = \{e = (u, v) : |S \cap \{u, v\}| = 1\}$. Although this LP has an exponential number of constraints, it can be solved efficiently via the ellipsoid method.

Observe that the connected data placement is a generalization of the *connected facility location* problem [14, 39, 15] (which is the special case with only one object) for which the best-known approximation guarantee is 8.55 [39]. However, due to the presence of cache capacities, it is not clear how to apply the primal-dual technique in [39] or the random-sampling idea in [15]. We show that the LP-rounding technique proposed in [32, 14] to handle such connectivity requirements can be overlaid almost directly on top of our rounding procedure from section 3, to round an optimal solution to the above LP losing a factor of at most 14.

We briefly sketch the main steps. Let (x, y, z) be an optimal fractional solution, and let $\bar{C}_j = \sum_i c_{ij} x_{ij}$. We slightly modify the demand-consolidation step (Step I) of our rounding procedure: we now move the demand of client k to client j (where $d'_j > 0,\ \bar{C}_j \le \bar{C}_k$) if $c_{jk} < 8\max(\bar{C}_j, \bar{C}_k)$. Recall that $F'_j = \{i : x_{ij} > 0,\ c_{ij} \le 2\bar{C}_j\}$ and that the sets F'_j are disjoint for clients in D_o. Due to the above change, we lose an additive factor of $8\sum_j d_j \bar{C}_j$ in translating a solution for client-set $D = \bigcup_o D_o$ to a solution for $\mathcal{D}$. More importantly, for any two facilities $i \in F'_j$ and $i' \in F'_k$, where $j, k \in D_o,\ j \ne k$, we now have that $c_{ii'} \ge 4\max(\bar{C}_j, \bar{C}_k)$. The rest of the rounding process in section 3 is unchanged. Thus, the sum of the storage costs and access costs is at most $14\big(\sum_{i,o} f_i^o y_i^o + \sum_{j,i} d_j c_{ij} x_{ij}\big)$.

For each object o, we build the tree T_o as follows. We contract the sets F'_j for $j \in D_o$ into supernodes and build a minimum spanning tree (MST) T'_o connecting these to r_o, and then connect the caches storing object o to T'_o. To bound the cost of T'_o, notice that $2z^o$ yields a fractional Steiner tree on the supernodes and r_o, since for any set S containing a supernode F'_j and not containing r_o, we have $\sum_{e\in\delta(S)} z_e^o \ge \sum_{i\in F'_j} x_{ij} \ge \frac{1}{2}$. Thus, we get $c(T'_o) \le 4\sum_e z_e^o$ since it is well known that the cost of the MST is at most twice the cost of a fractional solution for the Steiner tree LP. Observe that an edge e of T'_o joining F'_j and F'_k has $c_e \ge 4\max(\bar{C}_j, \bar{C}_k)$. Let i be a facility on which object o is stored. Notice there is a *unique* client $j \in D_o$ such that $i \in F'_j$. To connect i to T'_o, we add the edge (i, j) and add edges joining j to every cache in F'_j that has an edge incident to it in T'_o. We do this for every cache on which o is stored. Let δ_j denote the degree of the supernode F'_j in the tree T'_o. The cost of adding these extra edges is at most $\sum_{j\in D_o}(1+\delta_j)2\bar{C}_j \le 2\sum_{j\in D_o}\delta_j \cdot 2\bar{C}_j \le 2c(T'_o)$. Thus, $c(T_o) \le 3c(T'_o) \le 12\sum_e z_e^o$, and the total cost incurred is at most $14\sum_{i,o} f_i^o y_i^o + 14\sum_{j,i} d_j c_{ij} x_{ij} + 12\sum_o M_o \sum_e z_e^o$, yielding a 14-approximation algorithm.

THEOREM 4.1. *There is a* 14*-approximation algorithm for the connected data placement problem.*

5. Extensions.

The k-median variant. We can easily adapt our techniques to handle an extension of the data placement problem where additionally, for every object o, there is a bound of k_o on the number of caches that can store object o. This adds the constraints $\sum_i y_i^o \le k_o$ for all o to (P). We need to modify the min-cost flow network construction slightly in Steps II and IV of section 3. In Step II, we remove the edges (b_j, t). Instead, for every object o, we add a node p_o with demand $|D_o| - k_o$ and edges (b_j, p_o) for $j \in D_o$ of capacity $\frac{1}{2}$ and cost $3\gamma_j$. We also add an edge (p_o, t) with capacity k_o and cost 0. The effect of these changes is to limit the total flow on edges (a_j, r_i) and (b_j, r_i), where $j \in D_o$, to at most k_o so that at most k_o caches store object o (half-integrally). The half-integral solution $(\hat{x}, \hat{y})$ is obtained as before with p_o now playing the role of t. It is easy to see that $(\hat{x}, \hat{y})$ is feasible and Lemma 3.2 still holds. Similarly, in Step IV, we remove the edges (w_i^o, t). For every o, we add a node p_o with demand $|\{i : B_i^o \ne \phi\}| - (k_o - |D'_o|)$, add edges (w_i^o, p_o) with capacity 1 and cost 0, and add edge (p_o, t) with capacity $k_o - |D'_o|$ and cost 0. This limits the total flow on edges (v_j, r_i), where $j \in D'_o$, and (w_i^o, r_i) to at most k_o. The integer solution $(\tilde{x}, \tilde{y})$ is obtained as before, and Lemmas 3.3 and 3.4 still hold. So we get the following theorem.

THEOREM 5.1. *There is a* 10*-approximation algorithm for the data placement problem with a priori bounds on the number of caches that may store an object.*

Nonuniform object lengths. We can obtain a bicriteria approximation algorithm for the setting where each object o has a nonuniform length l_o and the total length of the objects stored in any cache must not exceed its capacity. Constraint (1) of (P) now reads $\sum_o l_o y_o^i \leq u_i$. As mentioned in the introduction, no approximation ratio is achievable in polynomial time in this case, unless $P = NP$ (see Theorem 6.2). We show the following theorem.

THEOREM 5.2. *For the data placement problem with arbitrary object lengths, one can compute in polynomial time a placement of cost at most* $10 \cdot OPT$ *where the cache capacities are violated by an additive amount of at most* $\max_o l_o$.

Proof. We need only modify Steps II and IV above. Instead of formulating a min-cost flow problem to take care of cache capacities, we will now construct an instance of the *generalized assignment problem* (GAP) [36]. In Step II, each node a_j, b_j of the min-cost flow network represents a job, and each node r_i and the sink t represent a machine. Each machine r_i has processing-time capacity $2u_i$, and the sink t has 0 capacity. An edge (a_j, r_i) or (b_j, r_i) denotes that job a_j or b_j has processing time $l_{o(j)}$ on machine r_i. Its assignment cost for machine r_i is $f_i^{o(j)} + d'_j l_{o(j)} c_{ij}$, which is simply a modification of the cost of the corresponding edge in the flow network that takes into account the length $l_{o(j)}$. Job b_j also has processing time 0 and assignment cost $3d'_j l_{o(j)} \gamma_j$ on machine t, corresponding to the edge (b_j, t). All other processing times (corresponding to nonedges) are infinity. It is not hard to see that $(2x, 2y)$ induces a feasible solution to this GAP instance of cost at most $6 \cdot OPT'$. Hence, by [36], there exists an integer solution $(2\hat{x}, 2\hat{y})$ of no greater cost, where $\sum_o l_o \cdot 2\hat{y}_i^o \leq 2u_i + \max_o l_o$ for every $i \in \mathcal{F}$. Thus, $(\hat{x}, \hat{y})$ yields a half-integral solution of cost at most $3 \cdot OPT'$, where the cache capacities are violated by at most $\frac{1}{2} \max_o l_o$.

Similarly, in Step IV, we have a job for each node v_j and each node w_i^o, and a machine for each node r_i and the sink t. Each machine r_i has capacity $u_i + \frac{1}{2} \max_o l_o$, and machine t has 0 capacity. As before, an edge (v_j, r_i) or (w_i^o, r_i) represents that the corresponding job has processing time $l_{o(j)}$ or l_o, respectively, on machine r_i. The assignment cost is the cost of the corresponding edge in the flow network modified (as above) to incorporate object lengths by multiplying the terms not involving the storage cost by the object length ($l_{o(j)}$ in case of job v_j, and l_o in case of job w_i^o). For example, corresponding to the edge $(v_j, r_{i'})$, where $i' = i_2(j)$, we set the assignment cost of job v_j on a machine $r_{i'}$ to be $f_{i'}^{o(j)} + l_{o(j)} \big(d'_j c_{i'j} + \sum_{k \in A_j} d'_k c_{i'k} + \sum_{k \in B_j} d'_k (c_{i'k} - c_{i_2(k)k}) \big)$ (note that $o(k) = o(j)$ for all $k \in A_j \cup B_j$). Edge (w_i^o, t) denotes that job w_i^o has 0 processing time and 0 assignment cost on machine t. All job-machine processing times corresponding to nonedges are infinity. As in Lemma 3.3, $(\hat{x}, \hat{y})$ induces a half-integral feasible solution of cost at most twice the cost of $(\hat{x}, \hat{y})$. Using the algorithm in [36] directly, one can obtain an integer solution of no greater cost where the load of every machine r_i is at most $u_i + \frac{3}{2} \max_o l_o$. A more careful analysis that exploits the half-integrality of the solution shows that the violation in the capacity of r_i is in fact at most $\frac{1}{2} \max_o l_o$, and the load of r_i is at most $u_i + \max_o l_o$. As in Lemma 3.4 and Theorem 3.5, this yields an integer solution of cost at most $10 \cdot OPT$. ∎

We observe that for the connected versions of the above extensions, one obtains the same guarantees as for the connected data placement problem. We simply use the algorithms described above (with the modification to Step I specified in section 4) to handle the data-placement part of the problem; then we apply the rounding method of section 4 to build the Steiner trees. The analysis from section 4 still applies, since it is still true that for any cache i on which an object o is stored, there is a unique client $j \in D_o$ such that $i \in F'_j$.

THEOREM 5.3. *There is a* 14*-approximation algorithm for the connected version of the following data placement problems:*

(i) *the placement problem with a priori bounds on the number of caches that may store a data object;*

(ii) *the placement problem with arbitrary object lengths; here we obtain a bicriteria guarantee where the cache capacities may be violated by an additive amount of at most* $\max_o l_o$.

6. Hardness results. In this section, we establish two hardness results. It is clear that the data placement problem with storage costs is *APX*-hard, since it is a generalization of metric UFL. We show that the data placement problem is *APX*-hard even when there are no storage costs. Our second result is that for the data placement problem with arbitrary object lengths, it is *NP*-complete to even decide if there exists a feasible solution; hence, one cannot achieve any approximation ratio in polynomial time unless $P = NP$.

THEOREM 6.1. *The data placement problem is* APX*-hard even when there are no storage costs.*

Proof. We give a reduction from metric UFL. In the unit-demand version of metric UFL (which is still *APX*-hard), we are given a set of n facilities F with facility-opening costs $\{f_i\}_{i\in\mathcal{F}}$, a client-set D, and distances/assignment costs $\{C_{ij}\}$ that form a metric. The goal is to open a subset of the facilities and assign each client to an open facility, so as to minimize the sum of the facility-opening and client-assignment costs.

Given such a UFL instance, we construct the following instance of the data placement problem. We let $\mathcal{F} = F \cup \{\Gamma\}$ be the set of caches, and $\mathcal{D} = D \cup F'$ be the set of clients, where F' is a copy of F; i.e., for every $i \in F$, we create a unique client $\sigma(i) \in F'$. There are $|F| + 1$ data objects $o_0, o_1, \ldots, o_n$. Each client $j \in D$ has unit demand for object o_0. Each client $\sigma(i) \in F'$ has demand f_i/M for object o_i, where M is some large number such that $M \gg \max_{i,j} C_{ij}$. Each cache $i \in F$ has unit capacity, and cache Γ has capacity $n + 1 = |F| + 1$. We define the distances c_{ij} for $i \in \mathcal{F}$ and $j \in \mathcal{D}$; all other distances are equal to the shortest-path distances in the bipartite graph $(\mathcal{F} \cup \mathcal{D}, \{(i, j) : i \in \mathcal{F},\ j \in D \cup \{\sigma(i)\}\})$ with these c_{ij}'s as the edge weights. For every $i \in F$, we set $c_{ij} = C_{ij}$ if $j \in D \subseteq \mathcal{D}'$, and 0 if $j = \sigma(i)$; for $i = \Gamma$ and every $j \in \mathcal{D}$, we set $c_{ij} = M$. It is easy to see that the c_{ij}'s form a metric.

We show that this is an approximation-preserving reduction by arguing that any UFL solution translates to a data placement solution of no greater cost and vice versa. Consider a UFL solution that opens the facilities in $S \subseteq F$ (and assigns each client to the nearest facility in S). We map this to the data placement solution, where each cache in S stores object o_0, each cache $i \in F \setminus S$ stores object o_i, and cache Γ stores the objects o_i for $i \in S$. Clearly, the total access cost incurred for object o_0 is equal to the client-assignment cost of the UFL solution, the total access cost incurred for the objects o_i, where $i \in S$, is $\sum_i f_i$, and the access cost for all other objects is 0. So the cost of this data placement solution is exactly the cost of the UFL solution.

Conversely, suppose we have a data placement solution. We may assume that object o_0 is stored in some cache in F; otherwise we can improve the solution cost by storing o_0 in some cache $i \in F$ (and moving the object stored in i to Γ if necessary). Let $S \subseteq F$ be the set of caches that store o_0. We open the facilities corresponding to S (and assign each client to the nearest facility in S). Since $M \gg \max_{i,j} C_{ij}$, the client-assignment cost in the UFL solution is at most the total access cost for o_0. For each cache $i \in S$, the access cost for object o_i is at least $f_i/M \cdot M = f_i$ (since the distance from i to any other cache is at least M), so the facility-opening cost of the

UFL solution is at most the access cost for the objects o_i, where $i \in S$. Thus, the cost of the UFL solution is at most that of the data placement solution. □

THEOREM 6.2. *It is NP-complete to decide if there exists a feasible solution to an instance of the data placement problem with arbitrary object lengths. Consequently, there is no polynomial-time approximation algorithm for this problem unless* $P = NP$.

Proof. Membership in *NP* is immediate. The *NP*-hardness proof follows from an easy reduction from the PARTITION problem. Let $a_1, \ldots, a_m$ be an instance of the PARTITION problem with $A = \sum_i a_i/2$. In the data placement instance, we have two caches with capacity A, m objects with lengths $a_1, \ldots, a_m$, and m clients, each of which has unit demand for a unique object. (The distances and the locations of the clients and the caches are not important.) Clearly, any feasible solution to the data placement problem yields a solution to the PARTITION problem, and vice versa. The *NP*-completeness result follows. □

REFERENCES

[1] B. AWERBUCH, Y. BARTAL, AND A. FIAT, *Heat & dump: Competitive distributed paging*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 22–31.

[2] B. AWERBUCH, Y. BARTAL, AND A. FIAT, *Distributed paging for general networks*, J. Algorithms, 28 (1998), pp. 67–104.

[3] I. D. BAEV AND R. RAJARAMAN, *Approximation algorithms for data placement in arbitrary networks*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 661–670.

[4] N. BANSAL, D. COPPERSMITH, AND M. SVIRIDENKO, *Improved approximation algorithms for broadcast scheduling*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 344–353.

[5] Y. BARTAL, A. FIAT, AND Y. RABANI, *Competitive algorithms for distributed data management*, J. Comput. System Sci., 51 (1995), pp. 341–358.

[6] J. BYRKA, *An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem*, in Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization, Springer, Berlin, 2007, pp. 29–43.

[7] M. CHARIKAR, S. GUHA, É. TARDOS, AND D. B. SHMOYS, *A constant-factor approximation algorithm for the k-median problem*, J. Comput. System Sci., 65 (2002), pp. 129–149.

[8] F. A. CHUDAK AND D. B. SHMOYS, *Improved approximation algorithms for the uncapacitated facility location problem*, SIAM J. Comput., 33 (2003), pp. 1–25.

[9] F. A. CHUDAK AND D. P. WILLIAMSON, *Improved approximation algorithms for capacitated facility location problems*, Math. Program., 102 (2005), pp. 207–222.

[10] L. DOWDY AND D. FOSTER, *Comparative models of the file assignment problem*, ACM Comput. Surv., 14 (1982), pp. 287–313.

[11] L. FLEISCHER, M. X. GOEMANS, V. S. MIRROKNI, AND M. SVIRIDENKO, *Tight approximation algorithms for maximum general assignment problems*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 611–620.

[12] B. GAVISH AND O. SHENG, *Dynamic file migration in distributed computer systems*, Comm. ACM, 33 (1990), pp. 177–189.

[13] S. GUHA AND K. MUNAGALA, *Improved algorithms for the data placement problem*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 106–107.

[14] A. GUPTA, J. KLEINBERG, A. KUMAR, R. RASTOGI, AND B. YENER, *Provisioning a virtual private network: A network design problem for multicommodity flow*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 389–398.

[15] A. GUPTA, A. KUMAR, AND T. ROUGHGARDEN, *Simple and better approximation algorithms for network design*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, 2003, pp. 365–372.

[16] A. Heddaya and S. Mirdad, *WebWave: Globally load balanced fully distributed caching of hot published documents*, in Proceedings of the 17th International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, 1997, pp. 160–168.

[17] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, *Greedy facility location algorithms analyzed using dual-fitting with factor-revealing LP*, J. ACM, 50 (2003), pp. 795–824.

[18] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[19] M. Korupolu and M. Dahlin, *Coordinated placement and replacement for large-scale distributed caches*, in Proceedings of the IEEE Workshop on Internet Applications, 1999, pp. 62–71.

[20] M. Korupolu, C. Plaxton, and R. Rajaraman, *Placement algorithms for hierarchical cooperative caching*, J. Algorithms, 38 (2001), pp. 260–302.

[21] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, J. Algorithms, 37 (2000), pp. 146–188.

[22] C. Krick, H. Räcke, and M. Westermann, *Approximation algorithms for data management in networks*, in Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, 2001, pp. 237–246.

[23] A. Leff, J. Wolf, and P. Yu, *Replication algorithms in a remote caching architecture*, IEEE Trans. Parallel Distrib. Syst., 4 (1993), pp. 1185–1204.

[24] R. Levi, R. Roundy, and D. Shmoys, *Primal-dual algorithms for deterministic inventory problems*, Math. Oper. Res., 31 (2006), pp. 267–284.

[25] R. Levi, D. Shmoys, and C. Swamy, *LP-based approximation algorithms for capacitated facility location*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 3064, Springer, Berlin, 2004, pp. 206–218.

[26] J. H. Lin and J. S. Vitter, *ϵ-approximations with minimum packing constraint violation*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 771–782.

[27] B. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann, *Exploiting locality for data management in systems of limited bandwidth*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 284–293.

[28] F. Meyer auf der Heide, B. Vöcking, and M. Westermann, *Caching in networks*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 430–439.

[29] A. Meyerson, K. Munagala, and S. Plotkin, *Web caching using access statistics*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 354–363.

[30] M. Pál, É. Tardos, and T. Wexler, *Facility location with nonuniform hard capacities*, in Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 329–338.

[31] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, *A dynamic object replication and migration protocol for an Internet hosting service*, in Proceedings of the IEEE International Conference on Distributed Computing Systems, 1999, pp. 101–113.

[32] R. Ravi and F. S. Selman, *Approximation algorithms for the traveling purchaser problem and its variants in network design*, in Algorithms—ESA '99, Lecture Notes in Comput. Sci. 1643, Springer, Berlin, 1999, pp. 29–40.

[33] R. Ravi and A. Sinha, *Multicommodity facility location*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 335–342.

[34] D. B. Shmoys, *Approximation algorithms for facility location problems*, in Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Comput. Sci. 1913, Springer, Berlin, 2000, pp. 27–33.

[35] D. B. Shmoys, C. Swamy, and R. Levi, *Facility location with service installation costs*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 1081–1090.

[36] D. B. Shmoys and É. Tardos, *An approximation algorithm for the generalized assignment problem*, Math. Programming, 62 (1993), pp. 461–474.

[37] D. B. Shmoys, É. Tardos, and K. I. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 265–274.

[38] C. Swamy, *Algorithms for Data Placement Problems*, manuscript, 2004.

[39] C. Swamy and A. Kumar, *Primal-dual algorithms for connected facility location problems*, Algorithmica, 40 (2004), pp. 245–269.

[40] C. Swamy and D. B. Shmoys, *Fault-tolerant facility location*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 735–736.
[41] O. Wolfson, S. Jajodia, and Y. Huang, *An adaptive data replication algorithm*, ACM Trans. Database Syst., 22 (1997), pp. 255–314.
[42] J. Zhang, B. Chen, and Y. Ye, *A multi-exchange local search algorithm for the capacitated facility location problem*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 3064, Springer, Berlin, 2004, pp. 219–233.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1430–1447

FASTER ALGORITHMS FOR MINIMUM CYCLE BASIS IN DIRECTED GRAPHS*

RAMESH HARIHARAN†, TELIKEPALLI KAVITHA‡, AND KURT MEHLHORN§

Abstract. We consider the problem of computing a minimum cycle basis in a directed graph. The input to this problem is a directed graph G whose edges have nonnegative weights. A cycle in this graph is actually a cycle in the underlying undirected graph with edges traversable in both directions. A $\{-1, 0, 1\}$ edge incidence vector is associated with each cycle: edges traversed by the cycle in the right direction get 1 and edges traversed in the opposite direction get -1. The vector space over $\mathbb{Q}$ generated by these vectors is the cycle space of G. A set of cycles is called a cycle basis of G if it forms a basis for this vector space. We seek a cycle basis where the sum of weights of the cycles is minimum. The current fastest algorithm for computing a minimum cycle basis in a directed graph with m edges and n vertices runs in $\tilde{O}(m^{\omega+1}n)$ time, where $\omega < 2.376$ is the exponent of matrix multiplication. We present an $O(m^3n + m^2n^2 \log n)$ algorithm. We obtain our algorithm by using fast matrix multiplication over rings and an efficient extension of Dijkstra's algorithm to compute a shortest cycle in G whose dot product with a function on its edge set is nonzero. We also present a simple $O(m^2n + mn^2 \log n)$ Monte Carlo algorithm. The problem of computing a minimum cycle basis in an *undirected* graph has been well studied. In this problem a $\{0, 1\}$ edge incidence vector is associated with each cycle and the vector space over $\mathbb{Z}_2$ generated by these vectors is the cycle space of the graph. The fastest known algorithm for computing a minimum cycle basis in an undirected graph runs in $O(m^2n + mn^2 \log n)$ time and our randomized algorithm for directed graphs matches this running time.

Key words. cycle basis, fast matrix multiplication, randomization, shortest paths

AMS subject classifications. 68W20, 68W40

DOI. 10.1137/060670730

1. Introduction.

Let $G = (V, E)$ be a directed graph with m edges and n vertices. A *cycle* in G is actually a cycle in the underlying undirected graph, i.e., edges are traversable in both directions. Associated with each cycle is a $\{-1, 0, 1\}$ edge incidence vector: edges traversed by the cycle in the right direction get 1, edges traversed in the opposite direction get -1, and edges not in the cycle at all get 0.[1] The vector space over $\mathbb{Q}$ generated by these vectors is the *cycle space* of G. A set of cycles is called a *cycle basis* if it forms a basis for this vector space. When G is connected, the cycle space has dimension $d = m - n + 1$.

We assume that there is a weight function $w : E \to \mathbb{R}^{\geq 0}$, i.e., the edges of G have nonnegative weights assigned to them. The weight of a cycle basis is the sum of the weights of its cycles. A *minimum cycle basis* of G is a cycle basis of minimum weight. We consider the problem of computing a minimum cycle basis in a given digraph.

*Received by the editors September 26, 2006; accepted for publication (in revised form) April 24, 2008; published electronically September 8, 2008. Preliminary versions of this work appeared in [14, 16].

http://www.siam.org/journals/sicomp/38-4/67073.html

†Strand Life Sciences, 237, Sir C. V. Raman Avenue, Sadashivnagar, Bangalore 560080, India (ramesh@strandls.com).

‡Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (kavitha@csa.iisc.ernet.in).

§Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, Saarbrücken 66123, Germany (mehlhorn@mpi-sb.mpg.de).

[1]Formally, the incidence vector is determined only up to a factor ± 1 as either direction (clockwise or counterclockwise) could be chosen to traverse a cycle.

A related problem pertains to undirected graphs, where we associate a $\{0,1\}$ edge incidence vector with each cycle; edges in the cycle get 1 and others get 0. Unlike directed graphs where the cycle space is defined over $\mathbb{Q}$, cycle spaces in undirected graphs are defined as vector spaces over $\mathbb{Z}_2$. The minimum cycle basis problem in an undirected graph G asks for the cycle basis of minimum weight in G.

The different possible settings for the minimum cycle basis problem are (i) the minimum cycle basis problem in undirected graphs, (ii) the minimum cycle basis problem in directed graphs (where the directions on the cycles are ignored), and (iii) the minimum *directed cycle* basis problem in directed graphs, where the cycle basis has to consist of cycles that traverse edges only along the direction of the edge.

We first compare problems (i) and (ii) and then discuss problem (iii). Problems (i) and (ii) are essentially different, since the first problem deals with computing a minimum weight spanning set of cycles that is linearly independent over $\mathbb{Z}_2$ while the second problem needs to compute a minimum weight spanning set of cycles that is linearly independent over $\mathbb{Q}$. Transforming cycles in a cycle basis of a directed graph by replacing both -1 and 1 by 1 does not necessarily yield a basis for the underlying undirected graph, since the given cycle basis could be linearly independent over $\mathbb{Q}$ but linearly dependent over $\mathbb{Z}_2$. In addition, lifting a minimum cycle basis of the underlying undirected graph by putting back directions does not necessarily yield a *minimum* cycle basis for the directed graph. Examples of both phenomena were presented in [22], which we include in section 2. Thus, one cannot find a minimum cycle basis for a directed graph by simply working with the underlying undirected graph.

A *directed cycle* basis is a spanning set of cycles where the incidence vector of each cycle in this basis is a vector in $\{0,1\}^m$, that is, each edge in a cycle here is traversed in the right direction. Note that a directed graph need not admit a directed cycle basis. Berge [2] studied the question of when a directed graph G admits such a cycle basis. He showed that if G is strongly connected, then G admits a directed cycle basis. Conversely, he showed that if G admits a directed cycle basis, then each maximal weakly connected induced subgraph of G with no cut vertex has to be strongly connected or a single arc.

Efficient algorithms for computing minimum cycle bases in the above settings have several applications. The minimum cycle basis problem is a special case of the *null space problem*. The null space problem is defined as follows: given a field $\mathbb{F}$ and an $n \times m$ matrix A with $n \leq m$, rank r, and entries in $\mathbb{F}$, find a matrix with the fewest nonzeros, whose columns span the null space of A. The null space problem was studied by Coleman and Pothen [6, 7], and this problem is NP-hard in general. The minimum cycle basis problem in undirected graphs with unit weights on the edges arises when the underlying field is $\mathbb{Z}_2$ and A is the $\{0,1\}$ vertex-edge incidence matrix of an undirected graph $G(V,E)$. The null vectors of this A are the vectors $(x_e)_{e\in E} \in \mathbb{Z}_2^{|E|}$ such that for each $v \in V$ we have $\sum_{e\in\delta(v)} x_e = 0 \pmod 2$, where $\delta(v)$ is the set of edges incident on v. Note that the set of such vectors $(x_e)_{e\in E}$ is the cycle space of G, and a solution to the null space problem is a minimum cycle basis of G.

Similarly, the minimum cycle basis problem in directed graphs with unit weights on the edges is an instance of the null space problem when the underlying field is $\mathbb{Q}$ and A is the $\{-1,0,1\}$ vertex-edge incidence matrix of a directed graph $G(V,E)$. The null vectors of such an A are the vectors $(x_e)_{e\in E} \in \mathbb{Q}^{|E|}$ such that for each $v \in V$ we have $\sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e = 0$, where $\delta^+(v)$ and $\delta^-(v)$ are the set of edges leaving v and entering v, respectively. It is easy to see that the set of such vectors $(x_e)_{e\in E}$ is the cycle space of G. The minimum cycle basis of G is a solution to this

null space problem. Indeed, assume that B is a minimum solution to the null space problem. We may assume that the entries of B are integral, as multiplication by a suitable constant makes all entries integral and does not change the number of nonzero entries. So assume that B contains a column C whose entries are not in $\{0, \pm 1\}$. Since C belongs to the null space of the vertex-edge incidence matrix A, C decomposes into a set of simple cycles C_1 to C_k. Each C_i uses a subset (not necessarily proper) of the edges of C and $B \setminus C \cup C_i$ is a basis for some i. We conclude that there exists a solution with $0, \pm 1$ entries to this null space problem, which implies that a minimum cycle basis of the directed graph G is a solution to this null space problem.

Some of the applications of minimum cycle bases arise from the above characterization of minimum cycle bases as the solutions of the null space problem in graphs. The cycle analysis of electrical networks [9] corresponding to Kirchoff's law is such an example. Applications of minimum cycle bases have also been shown in structural engineering [5], chemistry and biochemistry [11, 19], and surface reconstruction from point clouds [23]. The minimum directed cycle basis has applications in metabolic flux analysis [12]. A cycle basis of minimum weight in a directed graph whose $d \times m$ cycle-edge incidence matrix satisfies the constraint that all its regular $d \times d$ submatrices have determinant ± 1 has been found to be very useful in cyclic timetabling [20, 21]. Books by Deo [10] and Bollobás [4] have in-depth coverage of cycle bases.

Horton [15] designed the first polynomial time algorithm to compute a minimum cycle basis in an undirected graph and there are now several polynomial time algorithms for this problem [3, 9, 13, 18], the fastest running in $O(m^2n + mn^2 \log n)$ time [18]. Gleiss, Leydold, and Stadler [12] used Berge's characterization of directed cycle bases and showed that a generalization of Horton's minimum cycle basis algorithm in undirected graphs computes a minimum directed cycle basis in strongly connected directed graphs. The first polynomial time algorithm for computing a minimum cycle basis in a directed graph had a running time of $\tilde{O}(m^4n)$ [17]. Liebchen and Rizzi [22] gave an $\tilde{O}(m^{\omega+1}n)$ algorithm for this problem, where $\omega < 2.376$ [8] is the exponent of matrix multiplication; this was the fastest deterministic algorithm so far for this problem in directed graphs.

In this paper we present an $O(m^3n + m^2n^2 \log n)$ deterministic algorithm and an $O(m^2n + mn^2 \log n)$ Monte Carlo algorithm to compute a minimum cycle basis in a directed graph G with m edges, n vertices, and nonnegative edge weights. The running time of our deterministic algorithm is m times the running time of the fastest algorithm for computing minimum cycle bases in undirected graphs, we leave it as a challenge to close the gap. The increased complexity seems to stem from the larger base field. Arithmetic in $\mathbb{Z}_2$ suffices for undirected graphs. For directed graphs, the base field is $\mathbb{Q}$, which seems to necessitate the handling of large numbers. Also, the computation of a shortest cycle that has a nonzero dot product with a given vector seems more difficult in directed graphs than in undirected graphs.

The framework used in our algorithms was introduced by de Pina [9] and was also used in [3, 18, 17]: we compute cycles C_i and supporting vectors N_i so that each C_i is a shortest cycle *not* orthogonal to its corresponding N_i, and each N_i is orthogonal to all previous C_j, $j < i$. This collection of cycles C_i is known to be a minimum cycle basis. Our algorithms for computing the C_i's and N_i's rest on two ideas.

First, we show how to compute the vectors N_i efficiently using fast matrix multiplication and inversion. Our basic algorithm updates all vectors N_j with $j > i$ in iteration i, which results in an $\tilde{O}(m^4)$ algorithm. The improvement rests on an idea already used in [18] to delay the update of vectors with higher index and to perform these updates in bulk using matrix multiplication and inversion. However, this creates

a problem since the numbers involved in the arithmetic get very large. We show two approaches to solving this problem, leading to an $O(m^2n + mn^2 \log n)$ randomized algorithm and an $O(m^3n + m^2n^2 \log n)$ deterministic algorithm. In the randomized algorithm we work over a finite field $\mathbb{Z}_p$ for a small prime p, which ensures that the numbers involved here do not get large. We show that we compute a minimum cycle basis with a probability of at least $3/4$ when p is a prime chosen uniformly at random from a set of d^2 small primes. In the deterministic algorithm, we could run into intermediate numbers whose bit size is $\tilde{\Theta}(m^2)$ when we use fast matrix inversion algorithms. This makes the running time of our algorithm $\tilde{\Theta}(m^{\omega+2})$, which is worse than the original iterative algorithm. We circumvent the problem by working over a suitable ring $\mathbb{Z}_R$. The important point is that there is no fixed R over which we work during the entire algorithm: whenever we perform the matrix multiplication and inversion, we determine a suitable R so that the inverse of the matrix that we seek to invert exists in $\mathbb{Z}_R$. This leads us to the vectors $N_i \bmod R$, and the number R will be large enough so that we can easily recover the original vectors N_i from $N_i \bmod R$. The total time needed now is $\tilde{O}(m^{\omega+1})$.

The second key step in our algorithm is a subroutine to compute a shortest cycle whose dot product with a given vector N_i is nonzero modulo a small number p. We present an $O(mn + n^2 \log n)$ algorithm to compute such a cycle C_p. This algorithm is obtained by computing two types of paths between each adjacent pair of vertices. The first path is a shortest path between these two vertices and the second is a shortest path whose residue class is different from the residue class of the first path. Thus this yields an $O(m^2n + mn^2 \log n)$ algorithm over $\mathbb{Z}_p$ for computing the d cycles. For the deterministic algorithm, the computation of each cycle can be reduced via the Chinese remainder theorem to computing a shortest cycle C_p whose inner product with N_i is nonzero modulo p for some $p \in \{p_1, \ldots, p_d\}$, which is a collection of small primes. This procedure is repeated for each $p \in \{p_1, \ldots, p_d\}$ yielding $O(m^2n+mn^2 \log n)$ time for each C_i and thus $O(m^3n + m^2n^2 \log n)$ time overall for all of the d cycles.

Organization of this paper. In section 2 we discuss some preliminaries and describe the examples from [22] that were mentioned in section 1. Our framework is given in section 3, and we present a simple deterministic algorithm from [17] that follows from this framework. Section 4 lays the approach for a faster scheme and shows the problem of large numbers that such an approach runs into. Section 5 describes the deterministic algorithm that overcomes this problem, and section 6 gives a randomized algorithm. Section 7 describes the subroutine for computing the required cycles.

2. Preliminaries. We are given a digraph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Without loss of generality, the underlying undirected graph of G is connected. Then $d = m - n + 1$ is the dimension[2] of the cycle space of G. So a minimum cycle basis of G consists of d cycles $C_1, \ldots, C_d$. We describe cycles by their incidence vectors in $\{-1, 0, +1\}^m$.

A cycle basis of a directed graph need not project onto an undirected cycle

[2]Fix any spanning tree of G. For a nontree edge e, the fundamental cycle F_e induced by e consists of e plus the tree path connecting its endpoints. This set of cycles is clearly independent as every nontree edge is contained in a single cycle. We need to show that it spans all cycles. Consider any cycle $C = (x_e)_{e \in E}$ and define the sum $S = \sum_{e \text{ is a nontree edge}} x_e F_e$, in which every fundamental cycle is multiplied with the multiplicity of its defining edge in C. The vector S is in the cycle space and so is $Z = S - C$. The entries of Z corresponding to nontree edges are zero (by the definition of S) and Z satisfies the flow conservation constraints. Hence the entries of Z corresponding to tree edges must also be zero. Thus $Z = 0$ and the fundamental cycles form a basis. The number of fundamental cycles is exactly $m - n + 1$.

basis. Consider the following three 4-cycles in the directed graph in Figure 2.1: $C_1 = (e_1, e_2, e_3, e_4)$, $C_2 = (e_1, e_6, e_3, e_5)$, and $C_3 = (e_2, e_5, e_4, e_6)$ given by the vectors $(1, 1, 1, 1, 0, 0), (1, 0, -1, 0, -1, -1)$, and $(0, 1, 0, -1, -1, 1)$. It is easy to see that these vectors are linearly independent over $\mathbb{Q}$. Hence they form a cycle basis for the directed K_4. But in the underlying undirected graph, each of these cycles is equal to the sum modulo 2 of the other two, so C_1, C_2, C_3 do not form a cycle basis for the undirected K_4.

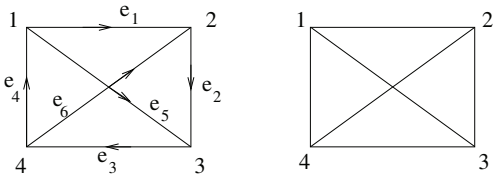


FIG. 2.1. *Directed K_4 and the underlying undirected graph.*

Further, there are directed weighted graphs in which the minimum cycle basis has lower weight than any cycle basis of the underlying undirected graph; such an example was given in [22]. Consider the generalized Petersen graph $P_{7,2}$ in Figure 2.2.

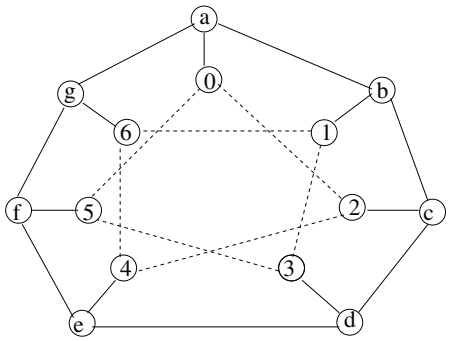


FIG. 2.2. *The generalized Petersen graph $P_{7,2}$.*

Call an edge (u, v) an *inner edge* if $\{u, v\} \subset \{0, 1, \ldots, 6\}$. Similarly call an edge (u, v) an *outer edge* if $\{u, v\} \subset \{a, \ldots, g\}$. The seven edges that remain are called *spokes*. Assign weight two to the seven inner edges and weight three to the outer edges and spokes. The shortest cycle in this graph has a weight of 14 and there are precisely eight cycles having a weight of 14, namely the cycle C_I consisting of only inner edges, and the seven cycles using one inner edge, two spokes, and two outer edges. Use C_i, $0 \leq i \leq 6$, to denote the cycle using the inner edge connecting i and $i + 2 \bmod 7$. Every other cycle has a length of at least 15.

Every edge of $P_{7,2}$ belongs to precisely two of the eight cycles with a weight of 14. Therefore in the undirected case, these $8 = m - n + 1$ cycles are not independent over $\mathbb{Z}_2$. Thus in the undirected case, every cycle basis has a weight of at least 113. In the directed case, these 8 cycles under any orientation of edges are linearly independent.[3] So there is a directed cycle basis of weight 112.

[3]Direct the inner edges clockwise, the spokes outward, and the outer edges counterclockwise. Then all eight cycles use their inner and outer edges in the forward direction. Assume $\alpha_I C_I + \sum_{0 \leq i \leq 6} \alpha_i C_i = 0$. The edge $(i, i + 2 \bmod 7)$ is used only by C_I and C_i, and hence we must have $\alpha_i = -\alpha_I$ for all i. But then the outer edges do not cancel out.

We will assume for the rest of this paper that the edges in $E = \{e_1, \ldots, e_m\}$ are ordered so that edges e_{d+1} to e_m form the edges of a spanning tree T of the underlying undirected graph. This means that the first d coordinates, of each of $C_1, \ldots, C_d$, correspond to edges outside the tree T, and the last $n-1$ coordinates are the edges of T. This will be important in our proofs in section 3. We can also assume that there are no multiple edges in G. It is easy to see that whenever there are two edges from u to v, the heavier edge (call it a) can be deleted from E, and the least weight cycle (call it $C(a)$) that contains the edge a can be added to the minimum cycle basis computed on $(V, E \setminus \{a\})$. The cycle $C(a)$ consists of the edge a and the shortest path between u and v in the underlying undirected graph. All such cycles can be computed by an all-pairs-shortest-paths computation in the underlying undirected graph of G, which takes $\tilde{O}(mn)$ time. So we will assume from now on that $m \leq n^2$.

3. Framework and a simple algorithm. We begin with a structural characterization of a minimum cycle basis. This characterization uses auxiliary rational vectors $N_1, \ldots, N_d$ which serve as a scaffold for proving properties of $C_1, \ldots, C_d$, as described below. We use $\langle v_1, v_2 \rangle$ to denote the standard inner product or dot product of the vectors v_1 and v_2.

THEOREM 3.1. *Cycles $C_1, \ldots, C_d$ form a minimum cycle basis if there are vectors $N_1, \ldots, N_d$ in $\mathbb{Q}^m$ such that for all i, $1 \leq i \leq d$:*

1. *Prefix orthogonality: $\langle N_i, C_j \rangle = 0$ for all j, $1 \leq j < i$.*
2. *Nonorthogonality: $\langle N_i, C_i \rangle \neq 0$.*
3. *Shortness: C_i is a shortest cycle with $\langle N_i, C_i \rangle \neq 0$.*

Proof. First, we show that $C_1, \ldots, C_d$ is a cycle basis of G by showing that these are linearly independent over $\mathbb{Q}$ (recall that any set of d linearly independent cycles is a cycle basis). Suppose it is not. Then a rational linear combination of a subset of these cycles yields 0. Let the cycle with the largest index in this subset be C_i. By properties 1 and 2 of the theorem, taking the dot product of this linear combination with N_i yields a nonzero value on one side of this linear combination and a 0 on the other side, a contradiction.

Second, we show that $C_1, \ldots, C_d$ is a minimum cycle basis of G. Suppose it is not. Then consider the smallest $i \geq 1$ such that $C_1, \ldots, C_i$ are not in any minimum cycle basis. Then $C_1, \ldots, C_{i-1}$ belong to some minimum cycle basis; call this basis $\mathcal{K}$ (in the event that $i = 1$, let $\mathcal{K}$ be any minimum cycle basis). We will exhibit a cycle $K \in \mathcal{K}$ such that (i) $\langle N_i, K \rangle \neq 0$ and (ii) K can be written as a rational linear combination of C_i along with cycles in $\mathcal{K}/\{K\}$. Demonstrating such a cycle $K \in \mathcal{K}$ is easy: since $\mathcal{K}$ is a basis not containing C_i, the cycle C_i must be a rational linear combination of cycles in $\mathcal{K}$. At least one of these cycles $K \in \mathcal{K}$ satisfies $\langle N_i, K \rangle \neq 0$, because $\langle N_i, C_i \rangle \neq 0$ by property 2 in the theorem. Therefore (i) holds. Further, (ii) follows by rewriting the above linear combination to switch the sides of C_i and K.

Property 1 of the theorem and condition (i) ensure that the cycle K is not one of $C_1, \ldots, C_{i-1}$. Condition (ii) implies that $\mathcal{K}/\{K\} \cup \{C_i\}$ is a cycle basis. Property 3 of the theorem and condition (i) imply that C_i has weight at most that of K and therefore $\mathcal{K}/\{K\} \cup \{C_i\}$ is also a minimum cycle basis. $C_1, \ldots, C_i$ belong to this minimum cycle basis, a contradiction. ∎

We present a simple deterministic algorithm from [17] that computes N_i's and C_i's satisfying the criteria in Theorem 3.1.

The algorithm deterministic-MCB.

1. Initialize the vectors $N_1, \ldots, N_d$ of $\mathbb{Q}^m$ to the first d vectors $e_1, \ldots, e_d$ of the standard basis of $\mathbb{Q}^m$.
   (The vector e_i has 1 in the ith position and 0's elsewhere.)

2. For $i = 1$ to d do
   - compute C_i to be a shortest cycle such that $\langle C_i, N_i \rangle \neq 0$,
   - for $j = i + 1$ to d do

$$\text{update } N_j \text{ as: } N_j = N_j - N_i \frac{\langle C_i, N_j \rangle}{\langle C_i, N_i \rangle},$$

$$\text{normalize } N_j \text{ as: } N_j = N_j \frac{\langle C_i, N_i \rangle}{\langle C_{i-1}, N_{i-1} \rangle}.$$

(We take $\langle C_0, N_0 \rangle = 1$.)

The above algorithm needs the vector N_i in the ith iteration to compute the cycle C_i. Instead of computing N_i from scratch in the ith iteration, it obtains N_i by update and normalization steps through iterations 1 to $i-1$. We describe how to compute a shortest cycle C_i such that $\langle C_i, N_i \rangle \neq 0$ in section 7. Let us now show that the N_i's obey the prefix orthogonality property. Lemma 3.2, proved in [17], shows this and more.

LEMMA 3.2. *For any i, at the end of iteration $i-1$, the vectors $N_i, \ldots, N_d$ are orthogonal to $C_1, \ldots, C_{i-1}$, and, moreover, for any j with $i \leq j \leq d$, $N_j = \langle N_{i-1}, C_{i-1} \rangle (x_{j,1}, \ldots, x_{j,i-1}, 0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 occurs in the jth coordinate and the vector $\mathbf{x} = (x_{j,1}, \ldots, x_{j,i-1})$ is the unique solution to the set of equations:*

$$\begin{pmatrix} \tilde{C}_1^T \\ \vdots \\ \tilde{C}_{i-1}^T \end{pmatrix} \mathbf{x} = \begin{pmatrix} -c_{1j} \\ \vdots \\ -c_{(i-1)j} \end{pmatrix}. \tag{3.1}$$

Here $\tilde{C}_k$, $1 \leq k < i$, is the restriction of C_k to its first $i-1$ coordinates and c_{kj} is the jth coordinate of C_k.

Proof. The claim is certainly true after the 0th iteration, that is, at the beginning of the algorithm. So consider the ith iteration and assume that the claim is true at the end of iteration $i-1$. In iteration i, we determine C_i with $\langle C_i, N_i \rangle \neq 0$ and then update the N_j's for all j with $i+1 \leq j \leq n$. Consider any j with $i+1 \leq j \leq n$ and use N_j' to denote the updated value of N_j.

N_j is updated by subtracting a scalar multiple of N_i from it. Since N_j and N_i are orthogonal to C_l for $l < i$ by induction hypothesis, N_j' is orthogonal to C_l. The update step also guarantees orthogonality to C_i. Indeed,

$$\langle C_i, N_j' \rangle = \langle C_i, N_j \rangle - \langle C_i, N_i \rangle \frac{\langle C_i, N_j \rangle}{\langle C_i, N_i \rangle} = 0.$$

By induction hypothesis, N_j is of the form $(t_{j,1}, \ldots, t_{j,i-1}, 0, \ldots, t_{j,j}, 0, \ldots)$, where $t_{j,j} = \langle C_{i-1}, N_{i-1} \rangle$ and N_i has nonzero entries only in its first i coordinates. So N_j' has the form $(t'_{j,1}, \ldots, t'_{j,i}, 0, \ldots, t_{j,j}, 0, \ldots)$. After normalization, the jth coordinate of N_j' is

$$t_{j,j} \frac{\langle N_i, C_i \rangle}{\langle N_{i-1}, C_{i-1} \rangle} = \langle N_{i-1}, C_{i-1} \rangle \frac{\langle N_i, C_i \rangle}{\langle N_{i-1}, C_{i-1} \rangle} = \langle N_i, C_i \rangle.$$

Hence N_j' has the form

$$N_j' = \langle N_i, C_i \rangle (u_{j,1}, \ldots, u_{j,i}, 0, \ldots, 1, 0, \ldots, 0).$$

Since N'_j is orthogonal to $C_1, \ldots, C_i$ and $\langle N_i, C_i \rangle \neq 0$, $(u_{j,1}, \ldots, u_{j,i})$ is a solution to the following set of equations:

$$\begin{pmatrix} \tilde{C}_1^T \\ \vdots \\ \tilde{C}_i^T \end{pmatrix} \mathbf{x} = \begin{pmatrix} -c_{1j} \\ \vdots \\ -c_{ij} \end{pmatrix}, \tag{3.2}$$

where $\tilde{C}_k$, for $k = 1, \ldots, i$, is the restriction of the vector C_k to its first i coordinates and c_{kj} is the jth coordinate of the vector C_k. We show that the matrix of $\tilde{C}_k$'s is nonsingular, hence $(u_{j,1}, \ldots, u_{j,i})$ is the unique solution of (3.2). The proof of the nonsingularity of this matrix mimics the argument in Theorem 3.1. Consider any linear combination of the rows adding to the zero vector:

$$\sum_{k=1}^{i} \alpha_k \tilde{C}_k = \mathbf{0}. \tag{3.3}$$

Assume that one of the α_k's is nonzero and consider the largest ℓ such that $\alpha_\ell \neq 0$. We take the inner product of both sides of (3.3) with $\tilde{N}_\ell$, where $\tilde{N}_\ell$ is the restriction of the vector N_ℓ to its first i coordinates. Note that $\tilde{N}_\ell$ has all of the nonzero entries of N_ℓ since $\ell \leq i$ and only the first ℓ entries of N_ℓ may be nonzero. So $\langle \tilde{C}_k, \tilde{N}_\ell \rangle = \langle C_k, N_\ell \rangle$ for all $k \leq \ell$. Hence the left-hand side is $\sum_{k=1}^{\ell} \alpha_k \langle C_k, N_\ell \rangle = \alpha_\ell \langle C_\ell, N_\ell \rangle$ since $\langle C_k, N_\ell \rangle = 0$ for each k with $k < \ell$. Since α_ℓ and $\langle C_\ell, N_\ell \rangle$ are nonzero while the right-hand side is zero, we get a contradiction. Thus all of the α_k's in (3.3) are zero and so the matrix with $\tilde{C}_k$'s as its rows is nonsingular and the proof is complete. □

Remark. Note that the ith coordinate of N_i is nonzero. This readily implies that there is at least one cycle that has nonzero dot product with N_i, namely the fundamental cycle F_{e_i} formed by the edge e_i and the path in the spanning tree T connecting its endpoints. The dot product $\langle F_{e_i}, N_i \rangle$ is equal to the ith coordinate of N_i, which is nonzero.

We next give an alternative characterization of these N_j's. This characterization helps us in bounding the running time of the algorithm deterministic-MCB. Let M denote the $(i-1) \times (i-1)$ matrix of $\tilde{C}_k$'s in (3.1), and let b_j denote the column vector of $-c_{kj}$'s on the right. We claim that solving $M\mathbf{x} = \det(M) \cdot b_j$ leads to the same vectors N_j for all j with $i \leq j \leq d$. We first show the following claim.

LEMMA 3.3. $\langle N_{i-1}, C_{i-1} \rangle = \det(M)$.

Proof. Let X be the $(i-1) \times (i-1)$ matrix with its kth column equal to N_k truncated to its first $i-1$ coordinates. We know from Lemma 3.2 that X is an upper triangular matrix with $X[k,k] = \langle N_{k-1}, C_{k-1} \rangle$. So $\det(X) = \prod_{k=1}^{i-1} \langle N_{k-1}, C_{k-1} \rangle$. The product MX has $\langle C_j, N_k \rangle$ as its (j,k)th element, so it is a lower triangular matrix by prefix orthogonality. Hence $\det(MX)$ is the product of its diagonal values, i.e, $\det(MX) = \prod_{k=1}^{i-1} \langle N_k, C_k \rangle$. Since $\det(M) \cdot \det(X) = \det(MX)$, $\langle N_0, C_0 \rangle = 1$, and $\langle N_k, C_k \rangle \neq 0$ for all k, the lemma follows. □

We know by Lemma 3.2 that $(x_{j,1}, \ldots, x_{j,i-1})$ is the unique solution to $M\mathbf{x} = b_j$. In the ith iteration of the algorithm, we could have directly computed the vector $N_i = \langle N_{i-1}, C_{i-1} \rangle (x_{i,1}, \ldots, x_{i,i-1}, 1, 0, \ldots)$, which is $\det(M)(x_{i,1}, \ldots, x_{i,i-1}, 1, 0, \ldots)$ (by Lemma 3.3), by solving the set of equations $M\mathbf{x} = \det(M) b_i$ and appending $(\det(M), 0, \ldots, 0)$ to $\mathbf{x}$. However, such an algorithm would be slower—it would take time $\tilde{\Theta}(m^{\omega+2})$, where $\omega < 2.376$ is the exponent of matrix multiplication. The updates and normalizations in the algorithm deterministic-MCB achieve the same result in a more efficient manner.

Let us now bound the running time of the ith iteration of deterministic-MCB. We will show in section 7 that a shortest cycle C_i such that $\langle C_i, N_i\rangle \neq 0$ can be computed in $O(m^2n+mn^2\log n)$ time. Let us look at bounding the time taken for the update and normalization steps. We take $O(m)$ arithmetic steps for updating and scaling each N_j since each N_j has m coordinates. Thus the total number of arithmetic operations in the ith iteration is $O((d-i)m) = O(md)$ over all j, $i+1 \leq j \leq d$. We next estimate the cost of the arithmetic. The coordinates of N_j are determined by the system $M\mathbf{x} = \det(M)b_j$ and hence are given by Cramer's rule. Fact 1 below shows that each entry in N_j is bounded by $d^{d/2}$. Thus we pay $\tilde{O}(d)$ time per arithmetic operation. Thus the running time of the ith iteration is $\tilde{O}(m^3)$, and hence the running time of deterministic-MCB is $\tilde{O}(m^4)$.

FACT 1. *Since M is a $\pm 1, 0$ matrix of size $(i-1)\times(i-1)$ and b_j is a $\pm 1, 0$ vector, all determinants used in Cramer's rule in solving $M\mathbf{x} = \det(M)b_j$ are bounded by $i^{i/2}$ using Hadamard's inequality. Therefore, the absolute value of each entry in N_j in the ith iteration, where $j \geq i$, is bounded by $i^{i/2}$.*

4. A faster scheme. The update and normalization steps form the bottleneck in the algorithm deterministic-MCB. We will reduce their cost from $\tilde{O}(m^4)$ to $\tilde{O}(m^{\omega+1})$.

- First, we delay updates until after several new cycles have been computed. For instance, we update $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ not after each new cycle but in bulk after *all* of $C_1, C_2, \ldots, C_{\lfloor d/2\rfloor}$ are computed.
- Second, we use a fast matrix multiplication method to do the updates for all of $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ together, and not individually as before.

The scheme. The faster deterministic algorithm starts with the same configuration for the N_i's as before, i.e., N_i is initialized to the ith unit vector, $1 \leq i \leq d$. It then executes three steps. First, it computes $C_1, \ldots, C_{\lfloor d/2\rfloor}$ and $N_1, \ldots, N_{\lfloor d/2\rfloor}$ recursively, leaving $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ at their initial values. Second, it runs a bulk update step in which $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ are modified so that they become orthogonal to $C_1, \ldots, C_{\lfloor d/2\rfloor}$. And third, $C_{\lfloor d/2\rfloor+1}, \ldots, C_d$ are computed recursively modifying $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ in the process.

A crucial point to note about the second recursive call is that it modifies $N_{\lfloor d/2\rfloor+1}$, $\ldots, N_d$ while ignoring $C_1, \ldots, C_{\lfloor d/2\rfloor}$ and $N_1, \ldots, N_{\lfloor d/2\rfloor}$; how then does it retain the orthogonality of $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ with $C_1, \ldots, C_{\lfloor d/2\rfloor}$ that we achieved in the bulk update step? The trick lies in the fact that whenever we update any N_j in $\{N_{\lfloor d/2\rfloor+1}, \ldots, N_d\}$ in the second recursive call, we do it as $N_j = \sum_{k=\lfloor d/2\rfloor+1}^{d} \alpha_k N_k$, where $\alpha_k \in \mathbb{Q}$. That is, the updated N_j is obtained as a rational linear combination of $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$. Since the bulk update step prior to the second recursive call ensures that $N_{\lfloor d/2\rfloor+1}, \ldots, N_d$ are all orthogonal to $C_1, \ldots, C_{\lfloor d/2\rfloor}$ at the beginning of this step, the updated N_j's remain orthogonal to $C_1, \ldots, C_{\lfloor d/2\rfloor}$. This property allows the second recursive call to work strictly in the bottom half of the data without looking at the top half.

The base case for the recursion is a subproblem of size 1 (let this subproblem involve C_ℓ, N_ℓ) in which case the algorithm simply retains N_ℓ as it is and computes C_ℓ using the algorithm in section 7. With regards to time complexity, the bulk update step will be shown to take $O(md^{\omega-1})$ arithmetic operations.

We describe the bulk update procedure in the recursive call that computes the cycles $C_\ell, \ldots, C_h$ for some h and ℓ with $h > \ell$. This recursive call works with the vectors $N_\ell, \ldots, N_h$: all of these vectors are already orthogonal to $C_1, \ldots, C_{\ell-1}$. The recursive call runs as follows:

1. Compute the cycles $C_\ell, \ldots, C_{mid}$, where $mid = \lceil (\ell + h)/2 \rceil - 1$, using the vectors $N_\ell, \ldots, N_{mid}$ recursively.
2. Modify $N_{mid+1}, \ldots, N_h$, which are untouched by the first step, to make them orthogonal to $C_\ell, \ldots, C_{mid}$.
3. Compute $C_{mid+1}, \ldots, C_h$ using these $N_{mid+1}, \ldots, N_h$ recursively.

Step 2 is the bulk update step. We wish to update each N_j, $mid + 1 \leq j \leq h$, to a rational linear combination of $N_\ell, \ldots, N_{mid}$ and N_j as follows:[4]

$$N_j = \frac{\langle N_{mid}, C_{mid} \rangle}{\langle N_{\ell-1}, C_{\ell-1} \rangle} N_j + \sum_{t=\ell}^{mid} \alpha_{tj} N_t,$$

where the α_{tj}'s are to be determined in a way which ensures that N_j becomes orthogonal to $C_\ell, \ldots, C_{mid}$. That is, for all i, j, where $\ell \leq i \leq mid$ and $mid + 1 \leq j \leq h$, we want

$$\frac{\langle N_{mid}, C_{mid} \rangle}{\langle N_{\ell-1}, C_{\ell-1} \rangle} \langle C_i, N_j \rangle + \sum_{t=\ell}^{mid} \alpha_{tj} \langle C_i, N_t \rangle = 0. \tag{4.1}$$

Rewriting the above relations in matrix form, we get

$$A \cdot \mathcal{N}_d \cdot D = -A \cdot \mathcal{N}_u \cdot X, \tag{4.2}$$

where (let $k = mid - \ell + 1$)

- A is a $k * m$ matrix, the ith row of which is $C_{\ell+i-1}$,
- $\mathcal{N}_d$ is an $m * (h - k)$ matrix, the jth column of which is N_{mid+j},
- D is an $(h - k) * (h - k)$ scalar matrix with $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ in the diagonal,
- $\mathcal{N}_u$ is an $m * k$ matrix, the tth column of which is $N_{\ell+t-1}$, and
- X is the $k * (h - k)$ matrix of variables α_{tj}, with t indexing the rows and j indexing the columns.

To compute the α_{tj}'s, we solve for $X = -(A \cdot \mathcal{N}_u)^{-1} \cdot A \cdot \mathcal{N}_d \cdot D$. Using fast matrix multiplication, we can compute $A \cdot \mathcal{N}_u$ and $A \cdot \mathcal{N}_d$ in $O(mk^{\omega-1})$ time by splitting the matrices into d/k square blocks and using fast matrix multiplication to multiply the blocks. Multiplying each element of $A \cdot \mathcal{N}_d$ with the scalar $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ gives us $A \cdot \mathcal{N}_d \cdot D$. Thus we compute the matrix $A \cdot \mathcal{N}_d \cdot D$ with $O(mk^{\omega-1})$ arithmetic operations. Next, we find the inverse of $A \cdot \mathcal{N}_u$ with $O(k^\omega)$ arithmetic operations (this inverse exists because $A \cdot \mathcal{N}_u$ is a lower triangular matrix whose diagonal entries are $\langle C_i, N_i \rangle \neq 0$). Then we multiply $(A \cdot \mathcal{N}_u)^{-1}$ with $A \cdot \mathcal{N}_d \cdot D$ using $O(k^\omega)$ arithmetic operations. Thus we obtain X. Finally, we obtain $N_{mid+1}, \ldots, N_d$ from X using the product $\mathcal{N}_u \cdot X$, which we can compute in $O(mk^{\omega-1})$ arithmetic operations, and adding $\mathcal{N}_d \cdot D$ to $\mathcal{N}_u \cdot X$. The total number of arithmetic operations required for the bulk update step is thus $O(mk^{\omega-1})$.

The total number of arithmetic operations required for the bulk update step is $O(mk^{\omega-1})$; however, each arithmetic operation is quite expensive since we deal with large numbers here. In this algorithm, the entries in $(A \cdot \mathcal{N}_u)^{-1}$ could be very large. The elements in $A \cdot \mathcal{N}_u$ have values up to $d^{\Theta(d)}$, which would result in the entries in $(A \cdot \mathcal{N}_u)^{-1}$ being as large as $d^{\Theta(d^2)}$. So each arithmetic operation then costs us

[4] Note that the coefficient $\langle N_{mid}, C_{mid} \rangle / \langle N_{\ell-1}, C_{\ell-1} \rangle$ for N_j is chosen so that the updated vector N_j here is exactly the same vector N_j that we would have obtained at this stage using the algorithm deterministic-MCB (section 3).

up to $\tilde{\Theta}(d^2)$ time, and the overall time for the outermost bulk update step would be $\tilde{\Theta}(m^{\omega+2})$ time, which makes this approach slower than the algorithm deterministic-MCB.

5. A fast deterministic algorithm. In the approach described in the previous section, we saw that entries in the matrix $(A \cdot \mathcal{N}_u)^{-1}$ (refer to (4.2)) could be as large as $d^{\Theta(d^2)}$. Thus we were not able to use the faster scheme for updating the vectors N_j, since each arithmetic operation could cost us up to $\tilde{\Theta}(d^2)$ time. But observe that the α_{tj}'s are just *intermediate* numbers in our computation. That is, they are the coefficients in

$$\sum_{t=\ell}^{mid} \alpha_{tj} N_t + \frac{\langle N_{mid}, C_{mid}\rangle}{\langle N_{\ell-1}, C_{\ell-1}\rangle} N_j.$$

Our final aim is to determine the updated coordinates of N_j which are at most $d^{d/2}$ (see Fact 1), since we know $N_j = (y_1, \ldots, y_{mid}, 0, \ldots, 0, \langle N_{mid}, C_{mid}\rangle, 0, \ldots, 0)$, where $\mathbf{y} = (y_1, \ldots, y_{mid})$ is the solution to the linear system: $M\mathbf{y} = \det(M) b_j$; M is the $mid \times mid$ matrix of $C_1, \ldots, C_{mid}$ truncated to the their first mid coordinates, and b_j is the column vector of negated j coordinates of $C_1, \ldots, C_{mid}$.

Since the final coordinates are bounded by $d^{d/2}$ while the intermediate values could be much larger, this suggests the use of modular arithmetic here. We could work over the finite fields $\mathbb{Z}_{p_1}, \mathbb{Z}_{p_2}, \ldots, \mathbb{Z}_{p_s}$ where $p_1, \ldots, p_s$ are small primes (say, in the range d to d^2) and try to retrieve N_j from $N_j \bmod p_1, \ldots, N_j \bmod p_s$, which is possible (by the Chinese remainder theorem) if $s \approx d/2$. Arithmetic in $\mathbb{Z}_p$ takes $O(1)$ time and we thus spend $O(s \cdot mk^{\omega-1})$ time for the update step now. However, if it is the case that some p is a divisor of some $\langle N_i, C_i\rangle$ where $\ell \le i \le mid$, then we cannot invert $A \cdot \mathcal{N}_u$ in the field $\mathbb{Z}_p$. Since each number $\langle N_i, C_i\rangle$ could be as large as $d^{d/2}$, it could be a multiple of up to $\Theta(d)$ primes which are in the range $d, \ldots, d^2$. So in order to be able to determine d primes which are relatively prime to each of $\langle N_\ell, C_\ell\rangle, \ldots, \langle N_{mid}, C_{mid}\rangle$, we might in the worst case have to test about $(mid - \ell + 1) \cdot d = kd$ primes. Testing kd primes for divisibility with respect to k d-bit numbers costs us $k^2 d^2$ time. We cannot afford so much time per update step.

Another idea is to work over just one finite field $\mathbb{Z}_q$ where q is a large prime. If $q > d^{d/2}$, then q can never be a divisor of any $\langle N_i, C_i\rangle$, so we can carry out all of our arithmetic in $\mathbb{Z}_q$ since the matrix $A \cdot \mathcal{N}_u$ will be nonsingular in $\mathbb{Z}_q$. Arithmetic in $\mathbb{Z}_q$ costs us $\tilde{\Theta}(d)$ time if $q \approx d^d$. Then our update step takes $\tilde{O}(m^2 k^{\omega-1})$ time, which will result in a total time of $\tilde{O}(m^{\omega+1})$ for all of the update steps, which is our goal. But computing such a large prime q is a difficult problem.

The solution is to work over a suitable ring instead of over a field; recall that fast matrix multiplication algorithms work over rings. Let us do the above computation modulo a large integer R, say, $R \approx d^d$. Then intermediate numbers do not grow more than R and we can retrieve N_j directly from $N_j \bmod R$, because R is much larger than any coordinate of N_j.

What properties of R do we need? The integer R must be relatively prime to the numbers: $\langle N_\ell, C_\ell\rangle$, $\langle N_{\ell+1}, C_{\ell+1}\rangle, \ldots, \langle N_{mid}, C_{mid}\rangle$ so that the triangular matrix $A \cdot \mathcal{N}_u$ which has these elements along the diagonal is invertible in $\mathbb{Z}_R$. And R must also be relatively prime to $\langle N_{\ell-1}, C_{\ell-1}\rangle$ so that $\langle N_{mid}, C_{mid}\rangle / \langle N_{\ell-1}, C_{\ell-1}\rangle$ is defined in $\mathbb{Z}_R$. Once we determine such an R, we will work in $\mathbb{Z}_R$. We stress the point that such an R is a number used only in this particular bulk update step—in another bulk update step of another recursive call, we need to compute another such large integer.

It is easy to see that the number R determined below is a large number that is relatively prime to $\langle N_{\ell-1}, C_{\ell-1}\rangle$, $\langle N_\ell, C_\ell\rangle, \langle N_{\ell+1}, C_{\ell+1}\rangle, \ldots,$ and $\langle N_{mid}, C_{mid}\rangle$ by doing the following.

1. Right at the beginning of the algorithm, compute d^2 primes $p_1, \ldots, p_{d^2}$, where each of these primes is at least d. Then form the d products: $P_1 = p_1 \cdots p_d$, $P_2 = p_1 \cdots p_{2d}$, $P_3 = p_1 \cdots p_{3d}$, $\ldots$, $P_d = p_1 \cdots p_{d^2}$.
2. Then during our current update step, compute the product
$$\mathcal{L} = \langle N_{\ell-1}, C_{\ell-1}\rangle\langle N_\ell, C_\ell\rangle \cdots \langle N_{mid}, C_{mid}\rangle.$$
3. By doing a binary search on $P_1, \ldots, P_d$, determine the smallest $s \geq 0$ such that P_{s+1} does not divide $\mathcal{L}$.
4. Determine a $p \in \{p_{sd+1}, \ldots, p_{sd+d}\}$ that does not divide $\mathcal{L}$. Let $R = p^d$.

Cost of computing R. The value of $\pi(r)$, the number of primes less than r, is given by $r/6\log r \leq \pi(r) \leq 8r/\log r$ [1]. So all the primes $p_1, \ldots, p_{d^2}$ are $\tilde{O}(d^2)$, and computing them takes $\tilde{O}(d^2)$ time using a sieving algorithm. The products $P_1, \ldots, P_d$ are computed just once in a preprocessing step. We will always perform arithmetic on large integers using Schönhage–Strassen multiplication, so that it takes $\tilde{O}(d)$ time to multiply two d-bit numbers. Whenever we perform a sequence of multiplications, we will use a tree so that d numbers (each of bit size $\tilde{O}(d)$) can be multiplied in $\tilde{O}(d^2)$ time. So computing $P_1, \ldots, P_d$ takes $\tilde{O}(d^3)$ preprocessing time.

In the update step, we compute $\mathcal{L}$, which takes $\tilde{O}(d^2)$ time. The product $P_{s+1} = p_{sd+1} \cdots p_{sd+d}$ is found in $\tilde{O}(d^2)$ time by binary search in the set $\{P_1, \ldots, P_d\}$. Determine a p in the set $\{p_{sd+1}, \ldots, p_{sd+d}\}$ that does not divide $\mathcal{L}$ by testing which of the two products $p_{sd+1} \cdots p_{sd+\lfloor d/2\rfloor}$ or $p_{sd+\lfloor d/2\rfloor+1} \cdots p_{sd+d}$ does not divide $\mathcal{L}$, and recurse on the product that does not divide $\mathcal{L}$. Thus R can be computed in $\tilde{O}(d^2)$ time.

Computation in $\mathbb{Z}_R$. We need to invert the matrix $A \cdot \mathcal{N}_u$ in the ring $\mathbb{Z}_R$. Recall that this matrix is lower triangular. Computing the inverse of a lower triangular matrix is easy. If

$$A \cdot \mathcal{N}_u = \begin{pmatrix} W & 0 \\ Y & Z \end{pmatrix}, \text{ then we have } (A \cdot \mathcal{N}_u)^{-1} = \begin{pmatrix} W^{-1} & 0 \\ -Z^{-1}YW^{-1} & Z^{-1} \end{pmatrix}.$$

Hence to invert $A \cdot \mathcal{N}_u$ in $\mathbb{Z}_R$ we need the multiplicative inverses of only its diagonal elements: $\langle C_\ell, N_\ell\rangle, \ldots, \langle C_{mid}, N_{mid}\rangle$ in $\mathbb{Z}_R$. Using Euclid's greatest common divisor (gcd) algorithm, each inverse can be computed in $\tilde{O}(d^2)$ time, since each of the numbers involved here and R have bit size $\tilde{O}(d)$. The matrix $A \cdot \mathcal{N}_u$ is inverted via fast matrix multiplication, and once we compute $(A \cdot \mathcal{N}_u)^{-1}$, the matrix X, that consists of all the coordinates α_{tj} that we need (see (4.1)), can be easily computed in $\mathbb{Z}_R$ as $-(A \cdot \mathcal{N}_u)^{-1} \cdot A \cdot \mathcal{N}_d \cdot D$ by fast matrix multiplication. Then we determine all N_j mod R for $mid + 1 \leq j \leq h$ from $\mathcal{N}_u \cdot X + \mathcal{N}_d \cdot D$. It follows from the analysis presented in section 4 that the time required for all of these operations is $\tilde{O}(m^2k^{\omega-1})$ since each number is now bounded by d^d.

Retrieving the actual N_j. Each entry of N_j can have absolute value at most $d^{d/2}$ (from Fact 1). The number R is much larger than this, $R > d^d$. So if any coordinate, say, n_l in N_j mod R, is larger than $d^{d/2}$, then we can retrieve the original n_l as $n_l - R$. Thus we can retrieve N_j from N_j mod R in $O(d^2)$ time. The time complexity for the update step, which includes matrix operations, gcd computations, and other arithmetic, is $\tilde{O}(m^2k^{\omega-1} + d^2k)$ or $\tilde{O}(m^2k^{\omega-1})$. Thus our recurrence becomes

(assuming Lemma 7.4 from section 7, which shows that the base case $T(1)$ takes $O(m^2n + mn^2 \log n)$ time)

$$T(k) = \begin{cases} 2T(k/2) + \tilde{O}(m^2k^{\omega-1}) & \text{if } k > 1, \\ m^2n + mn^2 \log n & \text{if } k = 1. \end{cases}$$

The recurrence solves to $T(k) = O(k(m^2n + mn^2 \log n) + k^{\omega-1}m^2 \cdot \text{poly}(\log m))$, and hence $T(d) = O(m^3n + m^2n^2 \log n) + \tilde{O}(m^{\omega+1})$, which is $O(m^3n + m^2n^2 \log n)$, because $m \leq n^2$ implies $\tilde{O}(m^{\omega+1})$ is always $o(m^3n)$. Thus we have shown the following theorem.

THEOREM 5.1. *A minimum cycle basis in a weighted directed graph with m edges and n vertices and nonnegative edge weights can be computed in $O(m^3n + m^2n^2 \log n)$ time.*

6. Faster arithmetic via randomization. Suppose we could perform each arithmetic operation in $O(1)$ time; then our recurrence relation for $T(k)$, $k > 1$ will be given by:

$$T(k) = 2T(k/2) + O(mk^{\omega-1}).$$

We would also like to show that $T(1)$ is $O(mn + n^2 \log n)$. Then this yields a running time of $O(n^2d \log n + nmd + m^\omega)$ or $O(m^2n + mn^2 \log n)$, since m^ω is $o(m^2n)$. Such a running time matches the complexity of minimum cycle basis computation in undirected graphs.

Now we will show that we *can* perform each arithmetic operation in $O(1)$ time. This will require working modulo a randomly chosen prime p. We will perform all arithmetic over the finite field $\mathbb{Z}_p$ and not over the rationals. The danger now is that the results of working in this field could be different from those obtained by working over rationals. Fortunately, the following theorem claims that the results remain the same, provided the prime p chosen satisfies certain properties. In the description below, let C_i, N_i denote the results obtained by the fast deterministic algorithm working over the rationals, and let C_i', N_i' be the counterparts obtained by the fast deterministic algorithm working over $\mathbb{Z}_p$.

THEOREM 6.1. *If $\langle C_i, N_i \rangle \neq 0 \pmod p$ for all i, $1 \leq i \leq d$, then $C_i' = C_i$ and $N_i' = N_i \pmod p$.*

Proof. Recall that the fast deterministic algorithm has a recursive structure. We use an inductive argument that mimics this recursion. At the very beginning, $N_i' = N_i \pmod p$ for all i, $1 \leq i \leq d$, as the N_i's are $\{0,1\}$ vectors. Each recursive subproblem (when working with rationals) then takes a contiguous subset $N_\ell, N_{\ell+1}, \ldots, N_h$ and computes $C_\ell, C_{\ell+1}, \ldots, C_h$ using only $N_\ell, N_{\ell+1}, \ldots, N_h$, modifying the latter in the process. We claim that if this recursive subproblem began with $N_\ell', N_{\ell+1}', \ldots, N_h'$ instead of $N_\ell, N_{\ell+1}, \ldots, N_h$, where

$$N_\ell' = N_\ell \pmod p,\ N_{\ell+1}' = N_{\ell+1} \pmod p, \ldots, N_h' = N_h \pmod p,$$

and subsequently worked modulo p, then it would still produce the same cycles $C_\ell, \ldots, C_h$ and in addition, after modification, the relations $N_\ell' = N_\ell \pmod p$, $N_{\ell+1}' = N_{\ell+1} \pmod p, \ldots, N_h' = N_h \pmod p$ will continue to hold. We will use induction on $\ell - h + 1$ to prove this claim. We will explicitly show the base case when $\ell - h + 1 = 1$. We will then assume that the claim is true for all recursive subproblems with $1 \leq \ell - h + 1 < t$ and then prove it for $\ell - h + 1 = t$.

Consider a subproblem of size 1 ($\ell - h + 1 = 1$, i.e., $\ell = h$). When working over rationals, N_ℓ does not change and C_ℓ is defined as the shortest cycle such that $\langle C_\ell, N_\ell \rangle \neq 0$. When working (mod p), N'_ℓ does not change and C'_ℓ is defined as the shortest cycle such that $\langle C'_\ell, N'_\ell \rangle \neq 0 \pmod{p}$. Assuming $N'_\ell = N_\ell \pmod{p}$ at the beginning of this subproblem, and given that p satisfies $\langle C_\ell, N_\ell \rangle \neq 0 \pmod{p}$, it follows that $C'_\ell = C_\ell$. This shows the base case.

Next, consider the three-step process used in the fast deterministic algorithm. First, consider the first recursive step which computes $C_\ell, \ldots, C_{mid}$. By the induction hypothesis, at the end of this step, we have $C'_\ell = C_\ell, \ldots, C'_{mid} = C_{mid}$ and $N'_\ell = N_\ell \pmod{p}, \ldots, N'_{mid} = N_{mid} \pmod{p}$. The vectors $N'_{mid+1}, \ldots, N'_h$ are untouched by this step and by virtue of the initial assignment, stay identical to $N_{mid+1}, \ldots, N_h$. Second, consider the bulk update step. This step modifies $N'_{mid+1} \ldots N'_h$ as a function of $N'_\ell, \ldots, N'_{mid}$ and $C'_\ell, \ldots, C'_{mid}$. Since $N'_\ell = N_\ell \pmod{p}, \ldots, N'_{mid} = N_{mid} \pmod{p}$ and $C'_1 = C_1, \ldots, C'_{mid} = C_{mid}$, it follows that $N'_{mid+1} = N_{mid+1} \pmod{p}, \ldots, N'_h = N_h \pmod{p}$ after the bulk update step. This sets up the necessary initial condition for the second recursive step which computes $C'_{mid+1}, \ldots, C'_h$ from $N'_{mid+1}, \ldots, N'_h$ alone. Applying the induction hypothesis again proves the theorem. ∎

Selecting the number p. It remains to show how p is chosen. Consider a pool of d^2 primes, each of which is bigger than d^2, and suppose we choose a prime at random from this pool. Lemma 6.3 shows that it satisfies the conditions of Theorem 6.1 with probability of at least 3/4. Let us first make the following definition.

DEFINITION 6.2. *Call a prime p* good *if $\langle C_i, N_i \rangle \neq 0 \pmod{p}$ for each $i \in \{1, \ldots, d\}$. Call a prime p* bad *if it is not good.*

LEMMA 6.3. *Let P be a set of d^2 primes, each of which is at least d^2. Then at least 3/4th of the set P is good.*

Proof. We will use Lemma 3.3 here. Lemma 3.3 shows that for all $1 \leq j \leq d$, $\langle C_j, N_j \rangle = \det(M)$, where M is the $j \times j$ matrix of $\tilde{C}_k$'s on the right-hand side of (3.1). Hadamard's inequality tells us that the absolute value of $\det(M)$ is at most $d^{d/2}$, since the entries in M are $0, \pm 1$. Hence for all $1 \leq i \leq d$, we have $0 \neq |\langle C_i, N_i \rangle| \leq d^{d/2}$. Since each prime in P is at least d^2, at most $d/4$ elements in P can be divisors of $\langle C_i, N_i \rangle$. So the number of primes in P that can divide at least one of $\langle C_1, N_1 \rangle, \langle C_2, N_2 \rangle, \ldots, \langle C_d, N_d \rangle$ is at most $d^2/4$. Hence the fraction of bad primes in P is at most $d^2/4d^2 \leq 1/4$. ∎

We now need to show how to compute the pool P of d^2 primes, each bigger than d^2. As mentioned earlier, the value of $\pi(r)$, the number of primes less than r, is given by $r/6 \log r \leq \pi(r) \leq 8r/\log r$. So the elements in P can be bounded by $100d^2 \log d$. Using sieving, we can compute the set of primes in the first $100d^2 \log d$ numbers in $O(d^2 \log^2 d)$ time. So the set P can be determined in $O(d^2 \log^2 d) = O(m^2 \log^2 n)$ time. Note that this term does not appear in the final complexity of $O(m^2 n + mn^2 \log n)$ because it is completely dominated by the $m^2 n$ term.

Arithmetic modulo p. Under the assumption that arithmetic on $O(\log m)$ bits takes unit time, it follows that addition, subtraction, and multiplication in $\mathbb{Z}_p$ can be implemented in unit time since p is $O(d^2 \log d) = O(m^2 \log m)$. However, we also need to implement division efficiently. Once p is chosen, we will compute the multiplicative inverses of all elements in $\mathbb{Z}_p^*$ by the extended Euclid's gcd algorithm by solving $ax = 1 (\mathrm{mod} p)$ for each $a \in \mathbb{Z}_p^*$. This takes time $O(\log p)$ for each element and hence $O(p \log p) = O(d^2 \log^2 d)$ for all of the elements. Thereafter, division in $\mathbb{Z}_p$ gets implemented as multiplication with the inverse of the divisor. The $O(d^2 \log^2 d)$

term does not appear in the final complexity of $O(m^2n + mn^2 \log n)$ because it is completely dominated by the m^2n term.

Lemma 7.3, proved in section 7, shows that $T(1)$, the time taken to compute the shortest C_i such that $\langle C_i, N_i \rangle \neq 0 (\bmod p)$, is $O(mn + mn^2 \log n)$. Hence Theorem 6.4 follows from the recurrence relation for $T(\cdot)$ (refer to the beginning of this section).

THEOREM 6.4. *A minimum cycle basis in a directed graph with n vertices and m edges, with nonnegative weights on its edges, can be computed with a probability of at least $3/4$ in $O(m^2n + mn^2 \log n)$ time.*

7. Computing nonorthogonal shortest cycles. Now we come to the second key routine required by our algorithm—given a directed graph G with nonnegative edge weights, compute a shortest cycle in G whose dot product with a given vector $N_i \in \mathbb{Z}^m$ is nonzero. We will first consider the problem of computing a shortest cycle C_p such that $\langle C_p, N_i \rangle \neq 0 \pmod p$ for a number $p = O(d^2 \log d)$. Recall that C_p can traverse edges of G in both forward and reverse directions; the vector representation of C_p has a 1 for every forward edge in the cycle, a -1 for every reverse edge, and a 0 for edges not present at all in the cycle. This vector representation is used for computing dot products with N_i. The weight of C_p itself is simply the sum of the weights of the edges in the cycle. We show how to compute C_p in $O(mn + n^2 \log n)$ time.

Definitions. To compute shortest paths and cycles, we will work with the undirected version of G. Directions will be used only to compute the *residue class* of a path or cycle, i.e., the dot product between the vector representation of this path or cycle and N_i modulo p. Let p_{uv} denote a shortest path between vertices u and v and let f_{uv} denote its length and r_{uv} its residue class. Let s_{uv} be the length of a shortest path, if any, between u and v in a residue class distinct from r_{uv}. Observe that the value of s_{uv} is independent of the choice of p_{uv}.

We will show how to compute f_{uv} and s_{uv} for all pairs of vertices u, v in $O(mn + n^2 \log n)$ time. As is standard, we will also compute paths realizing these lengths in addition to computing the lengths themselves. The following claim tells us how these paths can be used to compute a shortest nonorthogonal cycle—simply take each edge uv and combine it with s_{vu} to get a cycle. The shortest of all these cycles having a nonzero residue class is our required cycle.

LEMMA 7.1. *Let $C = u_0u_1 \dots u_ku_0$ be a shortest cycle whose residue class is nonzero modulo p and whose shortest edge is u_0u_1. Then the path $u_1u_2 \dots u_ku_0$ has a residue class different from the residue class of the edge u_1u_0, the length of the path $u_1u_2 \dots u_ku_0$ equals $s_{u_1u_0}$, and the length of the edge u_0u_1 equals $f_{u_1u_0}$.*

Proof. First, we show that the path $u_1u_2 \dots u_ku_0$ and the edge u_1u_0 have different residue classes. Let x denote the residue class of the path, and let y denote the residue class of the edge u_0u_1. Since C is in a nonzero residue class, $x + y \not\equiv 0 \pmod p$, so $x \not\equiv -y \pmod p$. Since the incidence vector corresponding to u_1u_0 is the negation of the incidence vector corresponding to u_0u_1, the residue class of the edge u_1u_0 is $-y$. Thus the claim follows.

Now, if the length of u_1u_0 is strictly greater than $f_{u_1u_0}$, then consider any shortest path π between u_1 and u_0 (which, of course, has length $f_{u_1u_0}$). Combining π with u_1u_0 yields a cycle and combining π with $u_1u_2 \dots u_ku_0$ yields another cycle. These cycles are in distinct residue classes and are shorter than C. This contradicts the definition of C. Therefore, the edge u_1u_0 has length $f_{u_1u_0}$.

Since $u_1u_2 \dots u_ku_0$ has a different residue class from the edge u_1u_0, the length of $u_1u_2 \dots u_ku_0$ cannot be smaller than $s_{u_1u_0}$, by the very definition of $s_{u_1u_0}$. Suppose,

for a contradiction, that the length $u_1u_2 \dots u_ku_0$ is strictly larger than $s_{u_1u_0}$. Then combining the path between u_1 and u_0 which realizes the length $s_{u_1u_0}$ along with the edge u_1u_0 yields a cycle which is shorter than C and which has a nonzero residue class modulo p. This contradicts the definition of C. The lemma follows. ∎

Computing f_{uv} and s_{uv}. We first find any one shortest path (among possibly many) between each pair of vertices u and v by Dijkstra's algorithm; this gives us p_{uv}, f_{uv}, and r_{uv} for each pair u, v. The time taken is $O(mn+n^2 \log n)$. For each pair u, v, we now need to find a shortest path between u, v with a residue class distinct from r_{uv}; the length of this path will be s_{uv}. Use q_{uv} to denote any such path. We show how a modified Dijkstra search can compute these paths in $O(mn + n^2 \log n)$ time. The following lemma shows the key prefix property of the q_{uv} paths needed for a Dijkstra-type algorithm.

LEMMA 7.2. *For any u and v, the path q_{uv} can be chosen from the set $\{p_{uw} \circ wv,\ q_{uw} \circ wv\ :\ wv \in E\}$. Here $p \circ e$ denotes the path p extended by the edge e.*

Proof. Consider any path π between u and v that realizes the value s_{uv}, i.e., it has length s_{uv} and residue class distinct from r_{uv}. Let w be the penultimate vertex on this path, and let π' be the prefix path from u to w. Clearly, π cannot be shorter than $p_{uw} \circ wv$. Hence, if the residue class of $p_{uw} \circ wv$ is distinct from r_{uv}, then we are done. So assume that $p_{uw} \circ wv$ has residue class r_{uv}. Then π' must have a residue class distinct from p_{uw} and hence q_{uw} exists. Also, the length of π' must be at least the length of q_{uw}, and the residue class of $q_{uw} \circ wv$ is distinct from the residue class of $p_{uw} \circ wv$ and hence distinct from r_{uv}. Thus $q_{uw} \circ wv$ realizes s_{uv}. ∎

We now show how to compute the s_{uv}'s for any fixed u in time $O(m + n \log n)$ with a Dijkstra-type algorithm. Repeating this for every source gives the result. The algorithm differs from Dijkstra's shortest path algorithm only in the initialization and update steps, which we describe below. We use the notation key_{uv} to denote the key used to organize the priority heap; key_{uv} will finally equal s_{uv}.

Initialization. We set key_{uv} to the minimal length of any path $p_{uw} \circ wv$ with residue class distinct from r_{uv}. If there is no such path, then we set it to ∞.

The update step. Suppose we have just removed w from the priority queue. We consider the u-w path of length key_{uw} which was responsible for the current key value of w. For each edge wv incident on w, we extend this path via the edge wv. We update key_{uv} to the length of this path provided its residue class is different from r_{uv}.

Correctness. We need to show that key_{uv} is set to s_{uv} in the course of the algorithm (note that one does not need to worry about the residue class since any path that updates key_{uv} in the course of the algorithm has residue class different from r_{uv}). This follows immediately from Lemma 7.2. If s_{uv} is realized by the path $p_{uw} \circ wv$ for some neighbor w, then key_{uv} is set to s_{uv} in the initialization step. If s_{uv} is realized by the path $q_{uw} \circ wv$ for some neighbor w, then key_{uv} is set to s_{uv} in the update step. This completes the proof of correctness. Thus we have given an $O(mn+n^2 \log n)$ algorithm to compute the s_{uv}'s for all $u, v \in V$. We have thus shown the following lemma.

LEMMA 7.3. *A shortest cycle C_p in G, whose dot product with N_i is nonzero modulo p, can be computed in $O(mn + n^2 \log n)$ time.*

We now consider the complexity of computing a shortest cycle C_i whose dot product with N_i is nonzero, instead of the condition that the dot product is nonzero modulo p. But any cycle C_i which satisfies $\langle C_i, N_i \rangle \neq 0$ satisfies $\langle C_i, N_i \rangle \neq 0 \pmod{p}$ for some $p \in \{p_1, \dots, p_{d/2}\}$, where $p_1, \dots, p_{d/2}$ are distinct primes, each of which is at least d. This follows from the isomorphism of the ring $\mathbb{Z}_{\prod p_i}$ to the ring $\mathbb{Z}_{p_1} \times$

$\mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_{d/2}}$. So any nonzero element whose absolute value is less than $\prod_{i=1}^{d/2} p_i$ is mapped to a tuple of its residues that is not the zero vector.

We have $|\langle C_i, N_i \rangle| \leq d^{d/2}$ (from Lemma 3.3 and Hadamard's inequality). Thus $|\langle C_i, N_i \rangle| < \prod_{i=1}^{d/2} p_i$. So if $\langle C_i, N_i \rangle$ is nonzero, then it is a nonzero element in $\mathbb{Z}_{\prod p_i}$, and so it satisfies $\langle C_i, N_i \rangle \neq 0 \pmod{p}$ for some p in $\{p_1, \ldots, p_{d/2}\}$. Thus a shortest cycle C_i such that $\langle C_i, N_i \rangle \neq 0$ is the shortest among all the cycles C_p, $p \in \{p_1, \ldots, p_{d/2}\}$, where C_p is a shortest cycle such that $\langle C_p, N_i \rangle \neq 0 \pmod{p}$. Hence, by Lemma 7.3, the time taken to compute C is $O(d \cdot (mn + n^2 \log n))$ or $O(m^2 n + mn^2 \log n)$. Thus we have shown Lemma 7.4.

LEMMA 7.4. *A shortest cycle C_i in G, whose dot product with N_i is nonzero, can be computed in $O(m^2 n + mn^2 \log n)$ time.*

8. Conclusions. We considered the minimum cycle basis problem in directed graphs with nonnegative edge weights. We presented an $O(m^3 n + m^2 n^2 \log n)$ deterministic algorithm and an $O(m^2 n + mn^2 \log n)$ randomized algorithm for this problem, where m is the number of edges and n is the number of vertices. These algorithms use the framework of computing cycles $C_1, \ldots, C_d$ and their supporting vectors $N_1, \ldots, N_d$ using fast matrix multiplication. However, this approach leads to large intermediate numbers and the cost of arithmetic becomes high. We overcome this problem in the randomized algorithm by working over a finite field $\mathbb{Z}_p$ for a small random prime p and by working over suitable rings $\mathbb{Z}_R$ in the deterministic algorithm. We also presented an efficient algorithm, based on Dijkstra's algorithm, to compute a shortest cycle whose dot product with a function on its edge set is nonzero.

REFERENCES

[1] T. M. APOSTOL, *Introduction to Analytic Number Theory*, Springer-Verlag, New York, 1997.

[2] C. BERGE, *Graphs*, North–Holland, Amsterdam, The Netherlands, 1985.

[3] F. BERGER, P. GRITZMANN, AND S. DE VRIES, *Minimum cycle bases for network graphs*, Algorithmica, 40 (2004), pp. 51–62.

[4] B. BOLLOBÁS, *Modern Graph Theory*, in Graduate Texts in Mathematics 184, Springer-Verlag, New York, 1998.

[5] A. C. CASSELL, J. C. HENDERSON, AND K. RAMACHANDRAN, *Cycle bases of minimal measure for the structural analysis of skeletal structures by the flexibility method*, Proc. Royal Society of London Series A, 350 (1976), pp. 61–70.

[6] T. F. COLEMAN AND A. POTHEN, *The null space problem* I. *Complexity*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 527–537.

[7] T. F. COLEMAN AND A. POTHEN, *The null space problem* II. *Algorithms*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 544–563.

[8] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplications via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[9] J. C. DE PINA, *Applications of Shortest Path Methods*, Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands, 1995.

[10] N. DEO, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall Series in Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[11] P. M. GLEISS, *Short Cycles: Minimum Cycle Bases of Graphs from Chemistry and Biochemistry*, Ph.D. thesis, Universität Wien, Wein, Germany, 2001.

[12] P. M. GLEISS, J. LEYDOLD, AND P. F. STADLER, *Circuit bases of strongly connected digraphs*, Discuss. Math. Graph Theory, 23 (2003), pp. 241–260.

[13] A. GOLYNSKI AND J. D. HORTON, *A polynomial time algorithm to find the minimum cycle basis of a regular matroid*, in Proceedings of SWAT, Lecture Notes in Comput. Sci. 2368, Springer, Berlin, 2002, pp. 200–209.

[14] R. HARIHARAN, T. KAVITHA, AND K. MEHLHORN, *A faster deterministic algorithm for minimum cycle basis in directed graphs*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 4051, Springer, Berlin, 2006, pp. 250–261.

[15] J. D. Horton, *A polynomial-time algorithm to find a shortest cycle basis of a graph*, SIAM J. Comput., 16 (1987), pp. 358–366.
[16] T. Kavitha, *An $\tilde{O}(m^2n)$ randomized algorithm to compute a minimum cycle basis of a directed graph*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 3580, Springer, Berlin, 2005, pp. 273–284.
[17] T. Kavitha and K. Mehlhorn, *Algorithms to compute minimum cycle bases in directed graphs*, Theory of Comput. Syst., 40 (2007), pp. 485–505.
[18] T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch, *A faster algorithm for minimum cycle bases of graphs*, in Proceedings of ICALP, Lecture Notes in Comput. Sci. 3142, Springer, Berlin, 2004, pp. 846–857.
[19] J. Leydold and P. F. Stadler, *Minimal cycle bases of outerplanar graphs*, Electron. J. Combin., 5 (1998), pp. 1–14.
[20] C. Liebchen, *Finding short integral cycle bases for cyclic timetabling*, in Proceedings of ESA, Lecture Notes in Comput. Sci. 2832, 2003, pp. 715–726.
[21] C. Liebchen and L. Peeters, *On Cyclic Timetabling and Cycles in Graphs*, Technical report 761/2002, TU Berlin, Berlin, Germany.
[22] C. Liebchen and R. Rizzi, *A greedy approach to compute a minimum cycle basis of a directed graph*, Inform. Process. Lett., 94 (2005), pp. 107–112.
[23] G. Tewari, C. Gotsman, and S. J. Gortler, *Meshing genus-1 point clouds using discrete one-forms*, Computers and Graphics, 30 (2006), pp. 917–926.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1448–1463

LINEAR EQUATIONS MODULO 2 AND THE L_1 DIAMETER OF CONVEX BODIES*

SUBHASH KHOT† AND ASSAF NAOR†

Abstract. We design a randomized polynomial time algorithm which, given a 3-tensor of real numbers $A = \{a_{ijk}\}_{i,j,k=1}^n$ such that for all $i,j,k \in \{1,\ldots,n\}$ we have $a_{ijk} = a_{ikj} = a_{kji} = a_{jik} = a_{kij} = a_{jki}$ and $a_{iik} = a_{ijj} = a_{iji} = 0$, computes a number $\mathrm{Alg}(A)$ which satisfies with probability at least $\frac{1}{2}$, $\Omega(\sqrt{\frac{\log n}{n}}t) \cdot \max_{x\in\{-1,1\}^n} \sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k \le \mathrm{Alg}(A) \le \max_{x\in\{-1,1\}^n} \sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k$. On the other hand, we show via a simple reduction from a result of Håstad and Venkatesh [*Random Structures Algorithms*, 25 (2004), pp. 117–149] that under the assumption $NP \not\subseteq DTIME(n^{(\log n)^{O(1)}})$, for every $\epsilon > 0$ there is no algorithm that approximates $\max_{x\in\{-1,1\}^n} \sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k$ within a factor of $2^{(\log n)^{1-\epsilon}}$ in time $2^{(\log n)^{O(1)}}$. Our algorithm is based on a reduction to the problem of computing the diameter of a convex body in $\mathbb{R}^n$ with respect to the L_1 norm. We show that it is possible to do so up to a multiplicative error of $O(\sqrt{\frac{n}{\log n}})$, while no randomized polynomial time algorithm can achieve accuracy $o(\sqrt{\frac{n}{\log n}})$. This resolves a question posed by Brieden et al. in [*Mathematika*, 48 (2001), pp. 63–105]. We apply our new algorithm to improve the algorithm of Håstad and Venkatesh for the Max-E3-Lin-2 problem. Given an overdetermined system $\mathcal{E}$ of N linear equations modulo 2 in $n \le N$ Boolean variables such that in each equation only three distinct variables appear, the goal is to approximate in polynomial time the maximum number of satisfiable equations in $\mathcal{E}$ minus $\frac{N}{2}$ (i.e., we subtract the expected number of satisfied equations in a random assignment). Håstad and Venkatesh obtained an algorithm which approximates this value up to a factor of $O(\sqrt{N})$. We obtain an $O(\sqrt{\frac{n}{\log n}})$ approximation algorithm. By relating this problem to the refutation problem for random $3-CNF$ formulas, we give evidence that obtaining a significant improvement over this approximation factor is likely to be difficult.

Key words. Max-E3-Lin-2, computational convex geometry, semidefinite programming, refutation of random SAT, Grothendieck's inequality

AMS subject classifications. 68W25, 90C22, 90C22

DOI. 10.1137/070691140

1. Introduction.

A function $f : \{-1,1\}^n \to \mathbb{R}$ has Fourier expansion $f(x) = \sum_{S\subseteq\{1,\ldots,n\}} \widehat{f}(S) \prod_{i\in S} x_i$. Assume that f has a succinct representation in phase space; i.e., only polynomially many of the Fourier coefficients $\widehat{f}(S)$ are nonzero. Can we then compute in polynomial time a good approximation of the maximum of f over the discrete cube $\{-1,1\}^n$? In other words, if we are given polynomially many Fourier coefficients, is there a way to approximate $\max_{x\in\{-1,1\}^n} f(x)$ while looking only at the values of f on a tiny part of the cube? As we shall see below, under widely believed complexity assumptions the answer to this question is generally negative. But, under some additional structural information on the support of the Fourier transform it is possible to achieve this goal, and when this occurs such phenomena have powerful algorithmic applications. Currently our understanding of this fundamental problem is far from satisfactory, and the purpose of the present paper is to investigate cases which have previously eluded researchers. As a result, we uncover new connections

*Received by the editors May 10, 2007; accepted for publication (in revised form) May 2, 2008; published electronically September 8, 2008.

http://www.siam.org/journals/sicomp/38-4/69114.html

†Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (khot@cims.nyu.edu, naor@cims.nyu.edu). The first author's research was supported in part by NSF CAREER award CCF-0643626 and a Microsoft New Faculty Fellowship. The second author's research was supported by NSF grants CCF-0635078 and DMS-0528387.

to problems in algorithmic convex geometry and combinatorial optimization.

The Fourier maximization problem described above has been investigated extensively in the quadratic case, partly due to its connections to various graph partitioning problems. In [3] it has been shown that a classical inequality of Grothendieck can be used to give a constant factor approximation algorithm for computing the maximum of functions $f : \{-1,1\}^n \times \{-1,1\}^m \to \mathbb{R}$ which have the form $f(x,y) = \sum_{i=1}^n \sum_{j=1}^m a_{ij} x_i y_j$. This algorithm has various applications, including an algorithmic version of Szemerédi's regularity lemma. In the nonbipartite case, several researchers [27, 25, 12] have discovered an algorithm which computes up to a factor $O(\log n)$ the maximum of functions $f : \{-1,1\}^n \to \mathbb{R}$ which have the form $f(x) = \sum_{i,j=1}^n a_{ij} x_i x_j$, where the matrix (a_{ij}) is assumed to be symmetric and vanish on the diagonal. This result was shown in [12] to imply the best-known approximation algorithms for graphing partitioning problems such as MAXCUTGAIN and correlation clustering. In [2] the structure of the "Fourier support graph," i.e., the pairs $\{i,j\} \in \{1,\ldots,n\}$ for which $a_{ij} \neq 0$, was taken into account. It was shown there that there exists an approximation algorithm which approximates the maximum of f up to a factor $O(\log \vartheta) = O(\log \chi)$, where ϑ is the Lovász theta function of the complement of the Fourier support graph and χ is the chromatic number of this graph. We refer to [2] for more information on this topic, as well as its connection to the evaluation of ground states of spin glasses.

Negative results on the performance of the above mentioned algorithms as well as complexity lower bounds were obtained in [3, 2, 5, 24, 1]. In particular, it was shown in [2] that the semidefinite relaxation that was used in the $O(\log n)$ algorithm discussed above had integrality gap $\Omega(\log n)$. Moreover, in [5] it was shown that unless $NP \subseteq DTIME(n^{O((\log n)^3)})$ there is no polynomial time algorithm which approximates the maximum of $\sum_{i,j=1}^n a_{ij} x_i x_j$ on $\{-1,1\}^n$ up to a factor smaller than $(\log n)^\gamma$, where γ is a universal constant. It was also shown in [5] that under the assumption of the existence of sufficiently strong PCPs it is also NP-hard to approximate this problem to within a factor of $O(\log n)$.

The motivation for the present paper is to study the case of functions whose Fourier expansion is supported on the third level. Specifically, given a 3-tensor of real numbers $A = \{a_{ijk}\}_{i,j,k=1}^n$ such that for all $i,j,k \in \{1,\ldots,n\}$ we have $a_{ijk} = a_{ikj} = a_{kji} = a_{jik} = a_{kij} = a_{jki}$ and $a_{iik} = a_{ijj} = a_{iji} = 0$, we wish to approximate the maximum of the function $f : \{-1,1\}^n \to \mathbb{R}$ given by $f(x) = \sum_{i,j,k=1}^n a_{ijk} x_i x_j x_k$. Despite being a modest goal, this problem has eluded researchers for some time, as the "obvious" semidefinite programming approach that was previously applied to the quadratic case does not generalize to the degree-3 case. As we shall see below, this issue reflects a major difference from the quadratic case: Unless $NP \subseteq DTIME(n^{(\log n)^{O(1)}})$, for every $\epsilon > 0$ there is no algorithm that approximates the maximum of f within a factor of $2^{(\log n)^{1-\epsilon}}$ in time $2^{(\log n)^{O(1)}}$. On the other hand, we will derive here a polynomial time algorithm which approximates the maximum of f to within a factor of $O(\sqrt{\frac{n}{\log n}})$. This algorithm is based on a novel connection between this problem and the problem of efficient computation of the diameter of convex bodies under the ℓ_1^n norm, which is the main new insight of the present paper. We shall now describe our new approach and its application to a fundamental problem in combinatorial optimization: the Max-E3-Lin-2 problem.

We associate with every 3-tensor A as above a convex body $K_A \subseteq \mathbb{R}^n$. The body K_A admits a polynomial time solution to the weak optimization problem for linear functionals (see [19, 18] for the relevant background on convex optimization).

Moreover, we show that the ℓ_1^n diameter of K_A, i.e., $\mathrm{diam}_1(K_A) := \max_{a,b\in K_A} \|a-b\|_1$, is within a constant factor of $\max_{x_i\in\{-1,1\}} \sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k$. This step is crucially based on an application of Grothendieck's inequality. We therefore reduce the level-3 Fourier maximization problem to the following question: Given a convex body $K \subseteq \mathbb{R}^n$ with a weak optimization oracle, approximate in oracle-polynomial time its ℓ_1^n diameter $\mathrm{diam}_1(K)$.

Such problems have been studied extensively in the literature, though mostly in the context of the Euclidean ℓ_2^n diameter (see, for example, [6, 8, 18, 10, 9, 31] and the references therein). In particular, a famous result of Bárány and Füredi states that no deterministic polynomial time algorithm can approximate the ℓ_2^n diameter of convex bodies up to a factor of $o(\sqrt{\frac{n}{\log n}})$. In the paper [10] of Brieden et al. it is shown that unlike the case of volume computation, randomization does not help when it comes to approximating the Euclidean diameter of convex bodies: The same lower bound holds also for the accuracy of randomized oracle-polynomial time algorithms. The paper [10] also studies the case of the ℓ_1^n diameter, or more generally the ℓ_p^n diameter, i.e., $\mathrm{diam}_p(K) := \max_{a,b\in K_A} \|a-b\|_p$. It is shown there that there is an oracle-polynomial time algorithm which approximates $\mathrm{diam}_1(K)$ to within a factor of $O(\sqrt{n})$, and no polynomial time algorithm can achieve accuracy better than $O(\frac{\sqrt{n}}{\log n})$. When $1 < p \le 2$ it is shown in [10] that $\mathrm{diam}_p(K)$ can be approximated within a factor $O(\frac{\sqrt{n}}{(\log n)^{(p-1)/p}})$, and no polynomial time algorithm can achieve accuracy better than $O_p(\sqrt{\frac{n}{\log n}})$. These bounds coincide only when $p = 2$, and the question of closing the gap in the remaining cases was raised in [10] (see also [9]). Here we resolve this problem by showing that the accuracy threshold for randomized oracle-polynomial algorithms that compute $\mathrm{diam}_p(K)$ is $\Theta(\sqrt{\frac{n}{\log n}})$ for all $1 \le p \le 2$. Our improved accuracy lower bound when $p = 1$ is a slight variant of the argument in [10]. The main issue is obtaining an improved approximation algorithm—our approach is different from the polyhedral approximation of the ℓ_p^n ball that was used in [10] (though we believe that the construction of [10] is of independent interest).

We apply the results described above to obtain a significant improvement to the Max-E3-Lin-2 algorithm of Håstad and Venkatesh [22]. This fundamental problem is described as follows. Consider a system $\mathcal{E}$ of N linear equations modulo 2 in n Boolean variables $z_1, \ldots, z_n$ such that in each equation only three distinct variables appear. We assume throughout that $N \ge n$ (thus avoiding degenerate cases). Let $\mathrm{MAXSAT}(\mathcal{E})$ be the maximum number of equations in $\mathcal{E}$ that can be satisfied simultaneously. A random assignment of these variables satisfies in expectation $\frac{N}{2}$ equations, so in the Max-E3-Lin-2 problem it is natural to ask for an approximation algorithm to $\mathrm{MAXSAT}(\mathcal{E}) - \frac{N}{2}$. This problem was studied extensively by Håstad and Venkatesh in [22], where the best known upper and lower bounds were obtained. In particular, using the powerful methods of Håstad [21] they show that unless $NP \subseteq DTIME(n^{(\log n)^{O(1)}})$, for every $\epsilon > 0$ there is no algorithm that approximates $\mathrm{MAXSAT}(\mathcal{E}) - \frac{N}{2}$ within a factor of $2^{(\log n)^{1-\epsilon}}$ in time $2^{(\log n)^{O(1)}}$. Moreover, they design a randomized polynomial time algorithm which approximates $\mathrm{MAXSAT}(\mathcal{E}) - \frac{N}{2}$ to within a factor of $O(\sqrt{N})$.

Let $\mathcal{E}$ be a system of linear equations as above. Write $a_{ijk}(\mathcal{E}) = 1$ if the equation $z_i + z_j + z_k = 0$ is in the system $\mathcal{E}$. Similarly write $a_{ijk}(\mathcal{E}) = -1$ if the equation $z_i + z_j + z_k = 1$ is in $\mathcal{E}$. Finally, write $a_{ijk}(\mathcal{E}) = 0$ if no equation in $\mathcal{E}$ corresponds to $z_i + z_j + z_k$. Assume that the assignment $(z_1, \ldots, z_k)$ satisfies m of the equations in

$\mathcal{E}$. Then $\sum_{i,j,k=1}^{n} a_{ijk}(\mathcal{E})(-1)^{z_i+z_j+z_k} = m - (N-m) = 2\left(m - \frac{N}{2}\right)$. It follows that

$$\max_{x_i\in\{-1,1\}} \sum_{i,j,k=1}^{n} a_{ijk}(\mathcal{E}) x_i x_j x_k = \max_{z_i\in\{0,1\}} \sum_{i,j,k=1}^{n} a_{ijk}(\mathcal{E})(-1)^{z_i+z_j+z_k} = 2\left(\mathrm{MAXSAT}(\mathcal{E}) - \frac{N}{2}\right).$$

Thus our algorithm yields an $O(\sqrt{\frac{n}{\log n}})$ approximation to the Max-E3-Lin-2 problem. Note that when $N = \Theta(n)$ our improvement over the Håstad–Venkatesh algorithm is only logarithmic, but typically N can be as large as $\Theta(N^3)$. The above reasoning also allows us to apply the Håstad–Venkatesh hardness result for Max-E3-Lin-2 that was described above to the level-3 Fourier maximization problem. In particular it follows that this problem is computationally much harder than the quadratic case, in which an $O(\log n)$ approximation is possible. Finally, our reasoning comes full circle to shed light on the problem of approximating the ℓ_1^n diameter $\mathrm{diam}_1(K)$. While the proof in [10] is essentially an "entropy argument" showing that there are simply too many convex bodies to allow an approximation factor better than $O(\sqrt{n})$, our reduction produces a concrete family of convex bodies for which computing the ℓ_1^n diameter within a factor of $2^{(\log n)^{1-\epsilon}}$ is hard.

2. A new algorithm for Max-E3-Lin-2. As described in the reduction that was presented in the introduction, our new algorithm for Max-E3-Lin-2 will follow from the more general algorithm for approximating the maximum of functions whose Fourier transform is supported on subsets of size 3. So, from now on let $\{a_{ijk}\}_{i,j,k=1}^{n}$ be real numbers such that for all $i,j,k \in \{1,\dots,n\}$ we have $a_{ijk} = a_{ikj} = a_{kji} = a_{jik} = a_{kij} = a_{jki}$ and $a_{iik} = a_{ijj} = a_{jii} = 0$. Our first lemma reduces the problem of maximizing $\sum_{i,j,k=1}^{n} a_{ijk}x_ix_jx_k$ to the analogous tripartite case. Note that such a result *is false* in the quadratic case. Indeed, Theorem 3.5 in [2] implies that the gap between $\max_{x_i\in\{-1,1\}} \sum_{i,j=1}^{n} a_{ij}x_ix_j$ and $\max_{x_i,y_j\in\{-1,1\}} \sum_{i,j=1}^{n} a_{ij}x_iy_j$ can be as large as $\Omega(\frac{n}{\log n})$. The key "trick" which allows us to prove that this cannot happen in the level-3 case is identity (1) below.

LEMMA 2.1. *The following inequalities hold true:*

$$\frac{1}{10} \max_{x_i,y_j,z_k\in\{-1,1\}} \sum_{i,j,k=1}^{n} a_{ijk}x_iy_jz_k \le \max_{x_i\in\{-1,1\}} \sum_{i,j,k=1}^{n} a_{ijk}x_ix_jx_k \le \max_{x_i,y_j,z_k\in\{-1,1\}} \sum_{i,j,k=1}^{n} a_{ijk}x_iy_jz_k.$$

Proof. Define

$$M = \max_{x_i,y_j,z_k\in\{-1,1\}} \sum_{i,j,k=1}^{n} a_{ijk}x_iy_jz_k$$

and

$$m = \max_{x_i\in\{-1,1\}} \sum_{i,j,k=1}^{n} a_{ijk}x_ix_jx_k.$$

Clearly $m \le M$, so we need to show that $M \le 10m$. To see this observe first that $\sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k$ is linear in x_i for each i. This implies that

$$m = \max_{|x_i|\le 1} \sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k.$$

Moreover, since $\sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k$ changes sign if we replace x_i by $-x_i$ for each i, we see that

$$m = \max_{|x_i|\le 1} \left| \sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k \right|.$$

Now, for each $i,j,k \in \{1,\ldots,n\}$ we have the identity

$$\begin{aligned}(1)\quad & 2x_iy_jy_k + 2x_jy_iy_k + 2x_ky_iy_j \\ &= (x_i+y_i)(x_j+y_j)(x_k+y_k) + (x_i-y_i)(x_j-y_j)(x_k-y_k) - 2x_ix_jx_k - 2y_iy_jy_k.\end{aligned}$$

Multiplying this identity by a_{ijk}, summing over all $i,j,k \in \{1,\ldots,n\}$, and using the symmetries of the coefficients a_{ijk}, we see that

$$\begin{aligned}6\sum_{i,j,k=1}^n a_{ijk}x_iy_jy_k &= 8\sum_{i,j,k=1}^n a_{ijk}\frac{x_i+y_i}{2}\cdot\frac{x_j+y_j}{2}\cdot\frac{x_k+y_k}{2} \\ &\quad + 8\sum_{i,j,k=1}^n a_{ijk}\frac{x_i-y_i}{2}\cdot\frac{x_j-y_j}{2}\cdot\frac{x_k-y_k}{2} \\ &\quad - 2\sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k - 2\sum_{i,j,k=1}^n a_{ijk}y_iy_jy_k.\end{aligned}$$

It follows that

$$M' := \max_{x_i,y_j\in\{-1,1\}} \left| \sum_{i,j,k=1}^n a_{ijk}x_iy_jy_k \right| \le \frac{20}{6}m = \frac{10}{3}m.$$

As before, because $\sum_{i,j,k=1}^n a_{ijk}x_iy_jy_k$ is linear in each of the variables x_i and y_j, we have the identity

$$M' = \max_{|x_i|,|y_j|\le 1} \left| \sum_{i,j,k=1}^n a_{ijk}x_iy_jy_k \right|.$$

Now, consider the identity

$$y_jz_k + y_kz_j = (y_j+z_j)(y_k+z_k) - y_jy_k - z_jz_k.$$

Multiplying by $a_{ijk}x_i$ and summing up, we get the identity

$$\begin{aligned}2\sum_{i,j,k=1}^n a_{ijk}x_iy_jz_k &= 4\sum_{i,j,k=1}^n a_{ijk}x_i\cdot\frac{y_j+z_j}{2}\cdot\frac{y_k+z_k}{2} - \sum_{i,j,k=1}^n a_{ijk}x_iy_jy_k \\ &\quad - \sum_{i,j,k=1}^n a_{ijk}x_iz_jz_k \le 6M' \le 20m.\end{aligned}$$

It follows that $M \leq 10m$, as required. □

Let $(\ell_2^n)_\infty^n$ denote the space of $\vec{v} = (v_1, \ldots, v_n) \in (\mathbb{R}^n)^n$, equipped with the norm

$$\|\vec{v}\|_{(\ell_2^n)_\infty^n} := \max_{1 \leq j \leq n} \|v_j\|_2.$$

Similarly we let $(\ell_2^n)_1^n$ denote the space of $\vec{v} = (v_1, \ldots, v_n) \in (\mathbb{R}^n)^n$, equipped with the norm

$$\|\vec{v}\|_{(\ell_2^n)_1^n} := \sum_{j=1}^n \|v_j\|_2.$$

Any $n \times n$ matrix $B = (b_{ij}) \in M_n(\mathbb{R})$ can be tensorized with the identity to yield an operator $B \otimes I : (\ell_2^n)_\infty^n \to (\ell_2^n)_1^n$ given by

$$((B \otimes I)\vec{v})_i := \sum_{j=1}^n b_{ij} v_j.$$

The operator norm of $B \otimes I$ is given by

$$\begin{aligned}
\|B \otimes I\|_{(\ell_2^n)_\infty^n \to (\ell_2^n)_1^n} &= \max \left\{ \sum_{i=1}^n \left\| \sum_{j=1}^n b_{ij} v_j \right\|_2 : \max_{1 \leq j \leq n} \|v_j\|_2 \leq 1 \right\} \\
&= \max \left\{ \sum_{i=1}^n \left\langle \sum_{j=1}^n b_{ij} v_j, u_i \right\rangle : \max_{1 \leq j \leq n} \|v_j\|_2 \leq 1 \ \wedge \ \max_{1 \leq i \leq n} \|u_i\|_2 \leq 1 \right\} \\
&= \max_{\|u_i\|_2, \|v_j\|_2 \leq 1} \sum_{i,j=1}^n b_{ij} \langle u_i, v_j \rangle . \qquad (2)
\end{aligned}$$

By Lemma 2.1, our goal is to approximate the value

$$\mathrm{Opt}(A) := \max_{x_i, y_j, z_k \in \{-1,1\}} \sum_{i,j,k=1}^n a_{ijk} x_i y_j z_k.$$

For every $x \in \{-1,1\}^n$, define an $n \times n$ matrix $A(x) \in M_n(\mathbb{R})$ by

$$A(x)_{jk} = \sum_{i=1}^n a_{ijk} x_i.$$

Since for each $x \in \mathbb{R}^n$ we have

$$\|A(x) \otimes I\|_{(\ell_2^n)_\infty^n \to (\ell_2^n)_1^n} \geq \max_{|y_j|, |z_k| \leq 1} \sum_{j,k=1}^n A(x)_{ij} y_j z_k,$$

it follows that $\mathrm{Opt}(A) \leq \max_{x \in \{-1,1\}^n} \|A(x) \otimes I\|_{(\ell_2^n)_\infty^n \to (\ell_2^n)_1^n}$. On the other hand, using (2), Grothendieck's inequality (see the discussion in [3]) says that

$$\begin{aligned}
\max_{x \in \{-1,1\}^n} \|A(x) \otimes I\|_{(\ell_2^n)_\infty^n \to (\ell_2^n)_1^n} &\leq \max_{x \in \{-1,1\}^n} K_G \max_{y, x \in \{-1,1\}^n} \sum_{j,k=1}^n A(x)_{jk} y_j z_k \\
&= K_G \cdot \mathrm{Opt}(A),
\end{aligned}$$

where $K_G \leq 2$ is Grothendieck's constant. It therefore suffices to approximate the value of

$$\max_{x\in\{-1,1\}^n} \|A(x)\otimes I\|_{(\ell_2^n)_\infty^n\to(\ell_2^n)_1^n}.$$

Define a norm $\|\cdot\|_A$ on $\mathbb{R}^n$ by $\|x\|_A := \|A(x)\otimes I\|_{(\ell_2^n)_\infty^n\to(\ell_2^n)_1^n}$. Then the unit ball $B_A := \{x\in\mathbb{R}^n : \|x\|_A \leq 1\}$ is a centrally symmetric convex body. Denote by $K_A = B_A^\circ$ the polar of B_A, i.e.,

$$K_A = \{y\in\mathbb{R}^n : \forall x\in B_A,\ \langle x,y\rangle \leq 1\}.$$

Then

$$\begin{aligned}\max_{x\in\{-1,1\}^n} \|A(x)\otimes I\|_{(\ell_2^n)_\infty^n\to(\ell_2^n)_1^n} &= \max_{\|x\|_\infty\leq 1}\|x\|_A = \max_{\|x\|_\infty\leq 1}\max_{y\in K_A}\langle x,y\rangle\\ &= \max_{y\in K_A}\max_{\|x\|_\infty\leq 1}\langle x,y\rangle = \max_{y\in K_A}\|y\|_1 = \frac{1}{2}\mathrm{diam}_1(K_A).\end{aligned}$$

We have thus reduced our original problem to approximating $\mathrm{diam}_1(K_A)$ in polynomial time. Note that (2) implies that the computation of $\|x\|_A$ is a semidefinite program. Therefore by the theory of Grötschel, Lovász, and Schrijver [19] it follows that linear functionals can be optimized on B_A in polynomial time. As shown in [19], this property is preserved under polarity; i.e., linear functionals can be optimized on $B_A^\circ = K_A$ in polynomial time. We have therefore reduced the problem of approximating $\max_{x_i\in\{-1,1\}}\sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k$ to the problem of approximating $\mathrm{diam}_1(K)$ in oracle-polynomial time, where K is a centrally symmetric convex body with a weak optimization oracle. This problem is resolved in section 3, thus proving the following theorem, which is our main result.

THEOREM 2.2. *There is a randomized polynomial time algorithm which, given a 3-tensor $A = \{a_{ijk}\}_{i,j,k=1}^n$ such that for all $i,j,k\in\{1,\ldots,n\}$ we have $a_{ijk} = a_{ikj} = a_{kji} = a_{jik} = a_{kij} = a_{jki}$ and $a_{iik} = a_{ijj} = a_{iji} = 0$, computes a number $\mathrm{Alg}(A)$, which satisfies with probability at least $\frac{1}{2}$*

$$\max_{x\in\{-1,1\}^n}\sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k \leq \mathrm{Alg}(A) \leq O\left(\sqrt{\frac{n}{\log n}}\right)\max_{x\in\{-1,1\}^n}\sum_{i,j,k=1}^n a_{ijk}x_ix_jx_k.$$

3. An approximation algorithm for the L_1 diameter. The main result of this section is the following theorem, which settles a problem posed by Brieden et al. in [10].

THEOREM 3.1. *Let $K\subseteq\mathbb{R}^n$ be a convex body with a weak optimization oracle. Then there exists a randomized algorithm which computes in oracle-polynomial time a number $\mathrm{Alg}(K)$ such that with probability at least $\frac{1}{2}$*

$$\frac{1}{2}\sqrt{\frac{\log n}{n}}\cdot\mathrm{diam}_1(K) \leq \mathrm{Alg}(K) \leq \mathrm{diam}_1(K).$$

On the other hand, no randomized oracle-polynomial time algorithm can compute $\mathrm{diam}_1(K)$ with accuracy $o(\sqrt{\frac{n}{\log n}})$.

Since we will be using Theorem 3.1 only when K is 0-symmetric, i.e., $K = -K$, we will prove it under this assumption. This is only for the sake of simplifying the

notation—identical arguments work in the general case. Our starting point is the following distributional inequality.

LEMMA 3.2. *For every $\delta \in (0, \frac{1}{2})$ there is a constant $c(\delta) > 0$ with the following property. Fix $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ and let $\epsilon_1, \ldots, \epsilon_n$ be independently and identically distributed (i.i.d.) symmetric Bernoulli random variables. Then*

$$\Pr\left(\sum_{j=1}^n a_j \epsilon_j \geq \sqrt{\frac{\delta \log n}{n}} \cdot \|a\|_1\right) \geq \frac{c(\delta)}{n^\delta}.$$

Proof. Write $X = \sum_{j=1}^n a_j \epsilon_j$. Assume first that

$$\frac{4\|a\|_2^2 \sqrt{n}}{\|a\|_1^2} > \frac{1}{12 n^\delta}. \tag{3}$$

The classical Paley–Zygmund inequality [28, 23, 4] states that for every $\theta \in (0,1)$ we have

$$\Pr\left(X^2 \geq \theta \mathbb{E}X^2\right) \geq (1-\theta)^2 \cdot \frac{\left(\mathbb{E}X^2\right)^2}{\mathbb{E}X^4} \geq \frac{(1-\theta)^2}{9}, \tag{4}$$

where we used the well-known (and easy) fact that $\mathbb{E}X^4 \leq 9\left(\mathbb{E}X^2\right)^2$.

The inclusion of events $\{X^2 \geq \theta\} \subseteq \{X \geq \sqrt{\theta}\} \cup \{-X \geq \sqrt{\theta}\}$, and the fact that X is symmetric, implies that $\Pr(X \geq \sqrt{\theta}) \geq \frac{1}{2}\Pr(X^2 \geq \theta)$. Since $\delta < \frac{1}{2}$ there is $n_0(\delta) \in \mathbb{N}$ such that for every $n \geq n_0(\delta)$ we have $\frac{48\delta \log n}{n^{\frac{1}{2}-\delta}} < \frac{1}{2}$. For such n we deduce that

$$\Pr\left(\sum_{j=1}^n a_j \epsilon_j \geq \sqrt{\frac{\delta \log n}{n}} \cdot \|a\|_1\right) \geq \frac{1}{2}\Pr\left(X^2 \geq \frac{\delta \log n}{n}\|a\|_1^2\right)$$
$$\overset{(3)}{\geq} \frac{1}{2}\Pr\left(X^2 \geq \frac{48\delta \log n}{n^{\frac{1}{2}-\delta}} \cdot \mathbb{E}X^2\right) \overset{(4)}{\geq} \frac{1}{72}.$$

Hence Lemma 3.2 holds assuming (3) and $n \geq n_0(\delta)$. By adjusting $c(\delta)$, the required result holds also when $n \leq n_0(\delta)$, so it remains to deal with the case

$$\frac{4\|a\|_2^2 \sqrt{n}}{\|a\|_1^2} \leq \frac{1}{12 n^\delta}. \tag{5}$$

Assuming (5), we define $S := \{j \in \{1, \ldots, n\} : \ |a_j| < \frac{2\|a\|_2^2}{\|a\|_1}\}$. Then

$$\|a\|_1 = \sum_{j \notin S} |a_j| + \sum_{j \in S} |a_j| \leq \frac{\|a\|_1}{2\|a\|_2^2} \sum_{j \notin S} a_j^2 + \sqrt{|S| \sum_{j \in S} a_j^2} \leq \frac{\|a\|_1}{2} + \sqrt{n \sum_{j \in S} a_j^2}.$$

Hence,

$$\sqrt{\sum_{j \in S} a_j^2} \geq \frac{\|a\|_1}{2\sqrt{n}}. \tag{6}$$

Write $Y = \sum_{j \in S} a_j \epsilon_j$ and $Z = X - Y$. For every $t \in \mathbb{R}$ we have $\{Y \geq 2t\} \subseteq \{Y + Z \geq t\} \cup \{Y - Z \geq t\}$. Since $Y + Z$ and $Y - Z$ have the same distribution as

X, it follows that $\Pr(X \geq t) \geq \frac{1}{2}\Pr(Y \geq 2t)$. Let g be a standard Gaussian random variable. By the Berry–Esseen inequality (see [20]; the constant we use below follows from [30]), we know that

$$\begin{aligned}\Pr(Y \geq 2t) &= \Pr\left(\frac{Y}{\sqrt{\mathbb{E}Y^2}} \geq \frac{2t}{\sqrt{\mathbb{E}Y^2}}\right) \geq \Pr\left(\frac{Y}{\sqrt{\mathbb{E}Y^2}} \geq \frac{t\sqrt{n}}{\|a\|_1}\right) \\ &\geq \Pr\left(g \geq \frac{t\sqrt{n}}{\|a\|_1}\right) - \max_{j\in S}\frac{|a_j|}{\sqrt{\sum_{k\in S} a_k^2}} \overset{(6)}{\geq} \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{t^2 n}{\|a\|_1^2}\right) - \frac{4\|a\|_2^2\sqrt{n}}{\|a\|_1^2}.\end{aligned}$$

Plugging $t = \sqrt{\frac{\delta \log n}{n}} \cdot \|a\|_1$ into the equation, we get that

$$\Pr\left(\sum_{j=1}^n a_j\epsilon_j \geq \sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_1\right) \geq \frac{1}{2}\Pr(Y \geq 2t) \geq \frac{1}{6n^\delta} - \frac{4\|a\|_2^2\sqrt{n}}{\|a\|_1^2} \overset{(5)}{\geq} \frac{1}{12n^\delta},$$

as required. □

Proof of Theorem 3.1. Let $\{\epsilon_{ij} : i \in \{1,\ldots,m\},\ j \in \{1,\ldots,n\}\}$ be i.i.d. symmetric Bernoulli random variables. Compute the number

$$\mathrm{Alg}(K) := 2\max_{1\leq i\leq m}\max_{a\in K}\sum_{j=1}^n a_j\epsilon_{ij}.$$

Then $\mathrm{Alg}(K) \leq 2\max_{a\in K}\|a\|_1 = \mathrm{diam}_1(K)$. Moreover, M can be computed using $O(m)$ oracle calls. Now, fix $a \in K$ such that $\|a\|_1 = \frac{1}{2}\mathrm{diam}_1(K)$. Using Lemma 3.2, we see that there exists a universal constant $c > 0$ for which

$$\begin{aligned}&\Pr\left(\mathrm{Alg}(K) > \frac{1}{2}\sqrt{\frac{\log n}{n}}\cdot\mathrm{diam}_1(K)\right) \\ &= 1 - \Pr\left(\bigcap_{i=1}^m\left\{\sum_{j=1}^n a_j\epsilon_{ij} < \frac{1}{2}\sqrt{\frac{\log n}{n}}\cdot\|a\|_1\right\}\right) \\ &\geq 1 - \left(1 - \frac{c}{\sqrt[4]{n}}\right)^m \\ &\geq 1 - \exp\left(-\frac{cm}{\sqrt[4]{n}}\right).\end{aligned}$$

Choosing $m = \left\lceil \frac{\sqrt[4]{n}}{c}\right\rceil$, we see that with probability at least $\frac{1}{2}$

$$\frac{1}{2}\sqrt{\frac{\log n}{n}}\cdot\mathrm{diam}_1(K) \leq \mathrm{Alg}(K) \leq \mathrm{diam}_1(K),$$

as required.

The algorithmic lower bound in Theorem 3.1 is essentially already contained in [10]—the authors simply overlooked an easy stronger upper bound on the volume of polytopes inscribed in the cube $[-1,1]^n$. In Proposition 1.10 in [10] the authors prove that for every 0-symmetric polytope $P \subseteq [-1,1]^n$ with at most $2k$ vertices, where $20\log_2\left(\frac{k}{n}+1\right) \leq n \leq k$,

$$(\mathrm{vol}(P))^{1/n} \leq O(1)\sqrt{1+\log n}\cdot\sqrt{\frac{\log\left(\frac{k}{n}+1\right)}{n}}. \tag{7}$$

The term $\sqrt{1+\log n}$ in (7) is precisely the reason that the lower bound in [10] for the accuracy of randomized algorithms which compute $\mathrm{diam}_1(K)$ was $\Omega(\frac{\sqrt{n}}{\log n})$ instead of $O(\sqrt{\frac{n}{\log n}})$. This term can be removed as follows.

Let B_2^n be the standard unit Euclidean ball of ℓ_2^n. Write $P = \mathrm{conv}\{\pm v_1, \ldots, \pm v_k\}$, where $v_i \in [-1,1]^n$. Then $\frac{v_i}{\sqrt{n}} \in B_2^n$, and by the results of [6, 11, 17] we deduce that

$$\left(\frac{\mathrm{vol}\left(\frac{1}{\sqrt{n}}P\right)}{\mathrm{vol}\,(B_2^n)}\right)^{1/n} \leq O(1)\sqrt{\frac{\log\left(\frac{k}{n}+1\right)}{n}}.$$

Since $(\mathrm{vol}\,(B_2^n))^{1/n} = \Theta(\frac{1}{\sqrt{n}})$ it follows that

$$(\mathrm{vol}(P))^{1/n} \leq O(1)\sqrt{\frac{\log\left(\frac{k}{n}+1\right)}{n}}. \qquad \square$$

3.1. The case of the L_p diameter, $1 < p < 2$. Fix $p \in (1,2)$, and define $q = \frac{p}{p-1} > 2$. Let $h_1, \ldots, h_n$ be i.i.d. random variables whose density is $\frac{q}{2\Gamma(1/q)}e^{-|t|^q}$. Let H be the random vector $(h_1, \ldots, h_n) \in \mathbb{R}^n$. Then the random variables $H/\|H\|_q$ and $\|H\|_q$ are independent [29] (see [7] for more information on this phenomenon). The following lemma is analogous to Lemma 3.2.

LEMMA 3.3. *There exist universal constants $\delta, c_1, c_2 > 0$ such that for every $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ we have*

$$\Pr\left(\left\langle \frac{H}{\|H\|_q}, a\right\rangle \geq \sqrt{\frac{\delta \log n}{n}} \cdot \|a\|_p\right) \geq \frac{c_1}{n^{c_2}}.$$

Proof. The random variable $\|H\|_q$ has density $\frac{q}{\Gamma(n/q)}u^{n-1}e^{-u^q}$ for $u > 0$ (see, for example, [26]). Hence for every $t \in (0,1)$ we have

$$\begin{aligned}\mathbb{E}e^{t\|H\|_q^q} &= \frac{q}{\Gamma(n/q)}\int_0^\infty e^{tu^q}\cdot u^{n-1}e^{-u^q}du \\ &= \frac{q}{\Gamma(n/q)}\int_0^\infty u^{n-1}e^{-[(1-t)^{1/q}u]^q}du = \frac{1}{(1-t)^{n/q}}.\end{aligned}$$

Since $q \geq 2$ it follows that

$$\begin{aligned}\Pr\left(\|H\|_q \geq n^{1/q}\right) &\leq e^{-n\left(1-\frac{1}{e}\right)}\mathbb{E}e^{\left(1-\frac{1}{e}\right)\|H\|_q^q} = e^{-n\left(1-\frac{1}{e}-\frac{1}{q}\right)} \\ &\leq e^{-n\left(\frac{1}{2}-\frac{1}{e}\right)} \leq e^{-n/8}.\end{aligned} \tag{8}$$

Using the independence of $H/\|H\|_q$ and $\|H\|_q$, we deduce that

$$\begin{aligned}
&\Pr\left(\left\langle \frac{H}{\|H\|_q}, a\right\rangle \geq \sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p\right)\\
&\geq \Pr\left(\left\langle \frac{H}{\|H\|_q}, a\right\rangle \geq \sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p\right)\Pr\left(\|H\|_q \leq n^{1/q}\right)\\
&= \Pr\left(\left\langle \frac{H}{\|H\|_q}, a\right\rangle \geq \sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p \ \wedge\ \|H\|_q \leq n^{1/q}\right)\\
&\geq \Pr\left(\langle H, a\rangle \geq n^{1/q}\sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p \ \wedge\ \|H\|_q \leq n^{1/q}\right)\\
&\geq 1-\Pr\left(\langle H, a\rangle < n^{1/q}\sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p\right)-\Pr\left(\|H\|_q > n^{1/q}\right)\\
&\overset{(8)}{\geq} \Pr\left(\langle H, a\rangle \geq n^{1/q}\sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p\right)-e^{-n/8}. \qquad (9)
\end{aligned}$$

As in the proof of Lemma 3.2 we write $X=\sum_{j=1}^n a_j h_j$. Let

$$S := \left\{ j\in\{1,\ldots,n\} :\ |a_j| \leq \frac{2^{\frac{1}{2-p}}\|a\|_2^{\frac{2}{2-p}}}{\|a\|_p^{\frac{p}{2-p}}}\right\}.$$

Then, using the definition of S and Hölder's inequality, we see that

$$\begin{aligned}
\|a\|_p^p = \sum_{j\notin S}|a_j|^p + \sum_{j\in S}|a_j|^p &\leq \frac{\|a\|_p^p}{2\|a\|_2^2}\sum_{j\notin S}a_j^2 + |S|^{\frac{2-p}{2}}\left(\sum_{j\in S}a_j^2\right)^{\frac{p}{2}}\\
&\leq \frac{\|a\|_p^p}{2} + n^{\frac{2-p}{2}}\left(\sum_{j\in S}a_j^2\right)^{\frac{p}{2}}.
\end{aligned}$$

It follows that

$$\sqrt{\sum_{j\in S}a_j^2} \geq \frac{\|a\|_p}{2^{\frac{1}{p}}n^{\frac{1}{p}-\frac{1}{2}}}. \qquad (10)$$

Set $Y:=\sum_{j\in S}a_j h_j$, and note that

$$\sqrt{\mathbb{E}Y^2} = \sqrt{\sum_{j\in S}a_j^2\mathbb{E}h_j^2} = \Omega(1)\sqrt{\sum_{j\in S}a_j^2} \overset{(10)}{\geq} c\frac{\|a\|_p}{n^{\frac{1}{p}-\frac{1}{2}}}, \qquad (11)$$

where $c>0$ is a universal constant. Using the Berry–Esseen inequality as in the proof

of Lemma 3.2, we see that

(12)
$$\Pr\left(\sum_{j=1}^{n} a_j h_j \geq n^{\frac{1}{q}}\sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p\right) \geq \frac{1}{2}\Pr\left(\frac{Y}{\sqrt{\mathbb{E}Y^2}} \geq 2cn^{1-\frac{1}{p}}\sqrt{\frac{\delta\log n}{n}}\cdot n^{\frac{1}{p}-\frac{1}{2}}\right)$$
$$\geq \frac{1}{6n^{4c^2\delta}} - \max_{j\in S}\frac{|a_j|}{\sqrt{\sum_{k\in S}a_k^2}}\cdot\mathbb{E}|h_1|^3 = \frac{1}{6n^{4c^2\delta}} - O\left(n^{\frac{1}{p}-\frac{1}{2}}\left(\frac{\|a\|_2}{\|a\|_p}\right)^{\frac{2}{2-p}}\right) \geq \frac{1}{12n^{4c^2\delta}},$$

provided that

$$n^{\frac{1}{p}-\frac{1}{2}}\left(\frac{\|a\|_2}{\|a\|_p}\right)^{\frac{2}{2-p}} \leq \frac{\tilde{c}}{n^{4c^2\delta}} \tag{13}$$

for some small enough constant $\tilde{c}$. But, assuming that (13) fails, and that δ is small enough and n is large enough, we may apply the Paley–Zygmund inequality to conclude that

$$\Pr\left(\sum_{j=1}^{n} a_j h_j \geq n^{\frac{1}{q}}\sqrt{\frac{\delta\log n}{n}}\cdot\|a\|_p\right) \geq \Pr\left(X^2 \geq \frac{\tilde{c}^{2-p}\delta\log n}{n^{2-\frac{p}{2}-\frac{1}{p}-8c^2\delta}}\cdot\mathbb{E}X^2\right)$$
$$\geq \Pr\left(X^2 \geq \frac{\delta\log n}{n^{\frac{1}{2}-8c^2\delta}}\cdot\mathbb{E}X^2\right) \geq \Omega(1), \tag{14}$$

where we used the fact that $p \in (1,2)$ and the easy bound $\sqrt[4]{\mathbb{E}X^4} = O(\sqrt{\mathbb{E}X^2})$. Combining (12) and (14) with (9) yields the required result. □

Now, arguing as in the proof of Theorem 3.1, given a 0-symmetric convex body $K \subseteq \mathbb{R}^n$ with a weak optimization oracle, we select m i.i.d. copies of H, $H_1, \ldots, H_m$, and define

$$\mathrm{Alg}(K) := 2\max_{1\leq i\leq m}\max_{a\in K}\left\langle\frac{H_i}{\|H_i\|_q}, a\right\rangle.$$

Arguing as in the proof of Theorem 3.1, with Lemma 3.2 replaced by Lemma 3.3, we see that for $m = \mathrm{poly}(n)$, with constant probability

$$\Omega(1)\sqrt{\frac{\log n}{n}}\cdot\mathrm{diam}_p(K) \leq \mathrm{Alg}(K) \leq \mathrm{diam}_p(K).$$

4. Discussion and open problems. We end this paper with some remarks and directions for future research.

• We assumed throughout that $a_{iik} = a_{ijj} = a_{iji} = 0$. This restriction, which also appeared in [2] as the condition that the Fourier support graph does not have self loops, is necessary since otherwise if $P \neq NP$, then there is no polynomial time algorithm that evaluates the maximum of $\sum_{i,j,k=1}^{n} a_{ijk}x_ix_jx_k$ over $x \in \{-1,1\}^n$ up to any factor (even one that grows with n arbitrarily fast)—see the discussion in Remark 3.2 in [2].

• It would be very interesting to investigate the maximization problem of $\sum_{i,j,k=1}^{n} a_{ijk}x_ix_jx_k$ in terms of the combinatorial structure of the Fourier support *hypergraph* given by $\{\{i,j,k\} : \ a_{ijk} \neq 0\}$. The results of [2] suggest that it might

be possible to achieve a better approximation guarantee in the presence of additional structural information of this type.

• A natural question that arises from our results is to study the maximization problem for

$$\sum_{\substack{S\subseteq\{1,\ldots,n\}\\|S|=k}} a_S \prod_{j\in S} x_j$$

when $k \geq 4$. Our methods do not immediately give good bounds in this case—it is quite easy to get an $O(n^{\frac{k}{2}-1}/(\log n)^{\frac{k}{2}-1})$ approximation algorithm for odd k by iterating our approach, and it would be desirable to get improved bounds. Such improvements, beyond their intrinsic interest, might have implications for the problem of refutation of random $k-CNF$ formulas [16, 15, 14] (see [13] for motivation of such questions). The connection between these two problems is explained in the following theorem.

THEOREM 4.1. *Suppose for every $\ell \in \{1,\ldots,k\}$ there is a deterministic polynomial time algorithm that approximates*

$$\max_{x_1,x_2,\ldots,x_n\in\{-1,1\}} \sum_{\substack{S\subseteq\{1,\ldots,n\}\\|S|=\ell}} a_S \prod_{j\in S} x_j$$

within factor $f(n)$. Then there is a polynomial time refutation procedure that refutes with high probability a random k-CNF formula with $2^{4k+1}nf(n)^2$ clauses.

Remark 1. Note that for $\ell = 1$ there is a (trivial) exact algorithm, and for $\ell = 2$ the result of [27, 25, 12] gives an $O(\log n)$-approximation. Therefore, as long as $f(n) \geq O(\log n)$, the hypothesis in Theorem 4.1 is required to hold only for $3 \leq \ell \leq k$.

The best known refutation procedure for random $3-CNF$ formulas works when they have $O(n^{3/2})$ clauses [16]. This can be viewed as evidence that obtaining an improvement over our approximation factor to $o(n^{1/4})$ is likely to be difficult.

Before proving Theorem 4.1 we shall introduce some notation. Let -1 represent logical TRUE and 1 represent logical FALSE. Let $\phi = \{C_1, C_2, \ldots, C_m\}$ be a k-CNF formula on variables $x := \{x_1, x_2, \ldots, x_n\}$. Let the set of indices of variables in the clause C_i be denoted as S_i, so that $|S_i| = k$. Define $\{\sigma_{ij} : 1 \leq i \leq m,\ j \in S_i\}$ as follows: $\sigma_{ij} = 1$ if x_j appears in clause C_i unnegated and $\sigma_{ij} = -1$ if x_j appears in clause C_i negated. Consider the expression

$$1 - \frac{1}{2^k} \prod_{j\in S_i} (1 + \sigma_{ij} x_j)\,.$$

For any $\{-1,1\}$-assignment to variables, this expression evaluates to 1 if the clause C_i is satisfied and to 0 if the clause C_i is not satisfied. Therefore, the fraction of satisfied clauses is

$$\frac{1}{m}\sum_{i=1}^{m}\left(1 - \frac{1}{2^k}\prod_{j\in S_i}(1+\sigma_{ij}x_j)\right). \tag{15}$$

For notational convenience, think of S_i as an ordered k-tuple of indices, and for $T \subseteq \{1,\ldots,k\}$, let $S_i[T]$ denote the subset of S_i given by the coordinates in T. With

this notation (15) can be rewritten as

$$1 - \frac{1}{2^k} - \frac{1}{m}\sum_{i=1}^{m} \sum_{\emptyset \neq T \subseteq \{1,\ldots,k\}} \prod_{j \in S_i[T]} \sigma_{ij} x_j = 1 - \frac{1}{2^k} + \sum_{\emptyset \neq T \subseteq \{1,\ldots,k\}} \Gamma_T(x), \tag{16}$$

where

$$\Gamma_T(x) := -\frac{1}{m}\sum_{i=1}^{m} \prod_{j \in S_i[T]} \sigma_{ij} x_j. \tag{17}$$

LEMMA 4.2. *If ϕ is satisfiable, then there exists a $\{-1,1\}$ assignment to variables of ϕ and a nonempty set $T \subseteq \{1,\ldots k\}$ such that $\Gamma_T(x) \geq \frac{1}{2^k(2^k-1)}$. In other words,*

$$\exists\, \emptyset \neq T \subseteq \{1,\ldots,k\}, \qquad \max_{x \in \{-1,1\}^n} \Gamma_T(x) \geq \frac{1}{2^k(2^k-1)}.$$

Proof. Since ϕ is satisfiable, there is an assignment x that satisfies every clause. For this assignment, the expression (16) has value 1. Thus, for some $T \subseteq [k], T \neq \emptyset$, it must be the case that $\Gamma_T(x) \geq \frac{1}{2^k(2^k-1)}$. ∎

LEMMA 4.3. *Let ϕ be a random k-CNF formula with $m \geq 2^{4k+1} n f(n)^2$ clauses. Then with probability $1 - 2^{-\Omega(n)}$ over the choice of the formula, for every $\{-1,1\}$-assignment to variables and every nonempty $T \subseteq \{1,\ldots,k\}$ we have $\Gamma_T(x) \leq \frac{1}{2^k f(n)}$. In other words,*

$$\forall \emptyset \neq T \subseteq \{1,\ldots,k\}, \qquad \max_{x \in \{-1,1\}^n} \Gamma_T(x) \leq \frac{1}{2^k f(n)}.$$

Proof. Fix any $\{-1,1\}$-assignment to the variables and a nonempty set $T \subseteq \{1,\ldots,k\}$. We will show that with probability $1 - e^{-n}$ over the choice of ϕ we have $\Gamma_T(x) \leq \frac{1}{2^k f(n)}$. Taking the union bound over all possible $\{-1,1\}$-assignments to variables and all choices for T implies the statement of the lemma.

Note that when ϕ is random, the signs σ_{ij} are random and independent, and therefore an inspection of the definition (17) shows that $\Gamma_T(x)$ is an average of m independent Bernoulli random variables. By the Chernoff bound,

$$\Pr\left[\Gamma_T(x) \geq \frac{1}{2^k f(n)}\right] \;\leq\; \exp\left(-\frac{1}{2} \cdot \frac{m}{(2^k f(n))^2}\right) \leq e^{-n}. \qquad \blacksquare$$

Proof of Theorem 4.1. The refutation procedure is very simple. Given a formula ϕ, use the $f(n)$-approximation algorithm to compute, for every nonempty $T \subseteq \{1,\ldots,k\}$, a number $\mathrm{Alg}(\Gamma_T)$ such that

$$\frac{1}{f(n)}\left(\max_{x \in \{-1,1\}^n} \Gamma_T(x)\right) \leq \mathrm{Alg}(\Gamma_T) \leq \max_{x \in \{-1,1\}^n} \Gamma_T(x).$$

If there is some $T \neq \emptyset$ for which $\mathrm{Alg}(\Gamma_T) \geq \frac{1}{2^k(2^k-1)f(n)}$, then say YES. Otherwise, say NO. Lemma 4.2 shows that this procedure always says YES if ϕ is satisfiable. Lemma 4.3 shows that the procedure says NO on a $1 - 2^{-\Omega(n)}$ fraction of random formulas. ∎

REFERENCES

[1] A. Alon and E. Berger, *The Grothendieck constant of random and pseudo-random graphs*, Discrete Optim., 5 (2008), pp. 323–327.

[2] N. Alon, K. Makarychev, Y. Makarychev, and A. Naor, *Quadratic forms on graphs*, Invent. Math., 163 (2006), pp. 499–522.

[3] N. Alon and A. Naor, *Approximating the cut-norm via Grothendieck's inequality*, SIAM J. Comput., 35 (2006), pp. 787–803.

[4] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience, New York, 2000.

[5] S. Arora, E. Berger, G. Kindler, E. Hazan, and S. Safra, *On non-approximability for quadratic programs*, in Proceedings of the 46th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Piscataway, NJ, 2005, pp. 206–215.

[6] I. Bárány and Z. Füredi, *Computing the volume is difficult*, Discrete Comput. Geom., 2 (1987), pp. 319–326.

[7] F. Barthe, M. Csörnyei, and A. Naor, *A note on simultaneous polar and Cartesian decomposition*, in Geometric Aspects of Functional Analysis, Lecture Notes in Math. 1807, Springer-Verlag, Berlin, 2003, pp. 1–19.

[8] H. L. Bodlaender, P. Gritzmann, V. Klee, and J. van Leeuwen, *Computational complexity of norm-maximization*, Combinatorica, 10 (1990), pp. 203–225.

[9] A. Brieden, *Geometric optimization problems likely not contained in* $\mathbb{APX}$, Discrete Comput. Geom., 28 (2002), pp. 201–209.

[10] A. Brieden, P. Gritzmann, R. Kannan, V. Klee, L. Lovász, and M. Simonovits, *Deterministic and randomized polynomial-time approximation of radii*, Mathematika, 48 (2001), pp. 63–105.

[11] B. Carl and A. Pajor, *Gel'fand numbers of operators with values in a Hilbert space*, Invent. Math., 94 (1988), pp. 479–504.

[12] M. Charikar and A. Wirth, *Maximizing quadratic programs: Extending Grothendieck's inequality*, in Proceedings of the 45th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Piscataway, NJ, 2004, pp. 54–60.

[13] U. Feige, *Relations between average case complexity and approximation complexity*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 534–543.

[14] U. Feige, J. H. Kim, and E. Ofek, *Witnesses for non-satisfiability of dense random* 3*CNF formulas*, in Proceedings of the 47th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Piscataway, NJ, 2006, pp. 497–508.

[15] U. Feige and E. Ofek, *Easily refutable subformulas of large random* 3*CNF formulas*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 3142, Springer-Verlag, Berlin, 2004, pp. 519–530.

[16] J. Friedman, A. Goerdt, and M. Krivelevich, *Recognizing more unsatisfiable random k-SAT instances efficiently*, SIAM J. Comput., 35 (2005), pp. 408–430.

[17] E. D. Gluskin, *Extremal properties of orthogonal parallelepipeds and their applications to the geometry of Banach spaces*, Mat. Sb. (N.S.), 136 (1988), pp. 85–96.

[18] P. Gritzmann and V. Klee, *Computational convexity*, in Handbook of Discrete and Computational Geometry, CRC Press Ser. Discrete Math. Appl., CRC, Boca Raton, FL, 1997, pp. 491–515.

[19] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, 2nd ed., Algorithms Combin. 2, Springer-Verlag, Berlin, 1993.

[20] P. Hall, *Rates of Convergence in the Central Limit Theorem*, Res. Notes Math. 62, Pitman (Advanced Publishing Program), Boston, MA, 1982.

[21] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.

[22] J. Håstad and S. Venkatesh, *On the advantage over a random assignment*, Random Structures Algorithms, 25 (2004), pp. 117–149.

[23] J.-P. Kahane, *Some Random Series of Functions*, 2nd ed., Cambridge Stud. Adv. Math. 5, Cambridge University Press, Cambridge, UK, 1985.

[24] S. Khot and R. O'Donnell, *SDP gaps and UGC-hardness for MAXCUTGAIN*, in Proceedings of the 47th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Piscataway, NJ, 2006, pp. 217–226.

[25] A. Megretski, *Relaxations of quadratic programs in operator theory and system analysis*, in Systems, Approximation, Singular Integral Operators, and Related Topics (Bordeaux, 2000), Oper. Theory Adv. Appl. 129, Birkhäuser, Basel, 2001, pp. 365–392.

[26] A. Naor, *The surface measure and cone measure on the sphere of* l_p^n, Trans. Amer. Math. Soc., 359 (2007), pp. 1045–1079.

[27] A. Nemirovski, C. Roos, and T. Terlaky, *On maximization of quadratic form over intersection of ellipsoids with common center*, Math. Program., 86 (1999), pp. 463–473.

[28] R. E. A. C. Paley and A. Zygmund, *A note on analytic functions in the unit circle*, Proc. Camb. Phil. Soc., 28 (1932), pp. 266–272.

[29] G. Schechtman and J. Zinn, *On the volume of the intersection of two L_p^n balls*, Proc. Amer. Math. Soc., 110 (1990), pp. 217–224.

[30] P. van Beek, *An application of Fourier methods to the problem of sharpening the Berry-Esseen inequality*, Z. Wahrscheinlichkeitstheorie und Verw. Gebiete, 23 (1972), pp. 187–196.

[31] K. R. Varadarajan, S. Venkatesh, and J. Zhang, *On approximating the radii of point sets in high dimensions*, in Proceedings of the 43rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Piscataway, NJ, 2002, pp. 561–569.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1464–1483

COMBINATION CAN BE HARD: APPROXIMABILITY OF THE UNIQUE COVERAGE PROBLEM*

ERIK D. DEMAINE†, URIEL FEIGE‡, MOHAMMADTAGHI HAJIAGHAYI†, AND MOHAMMAD R. SALAVATIPOUR§

Abstract. We prove semilogarithmic inapproximability for a maximization problem called *unique coverage*: given a collection of sets, find a subcollection that maximizes the number of elements covered exactly once. Specifically, assuming that NP $\not\subseteq$ BPTIME(2^{n^ε}) for an arbitrary $\varepsilon > 0$, we prove $O(1/\log^\sigma n)$ inapproximability for some constant $\sigma = \sigma(\varepsilon)$. We also prove $O(1/\log^{1/3-\varepsilon} n)$ inapproximability for any $\varepsilon > 0$, assuming that refuting random instances of 3SAT is hard on average; and we prove $O(1/\log n)$ inapproximability under a plausible hypothesis concerning the hardness of another problem, balanced bipartite independent set. We establish an $\Omega(1/\log n)$-approximation algorithm, even for a more general (budgeted) setting, and obtain an $\Omega(1/\log B)$-approximation algorithm when every set has at most B elements. We also show that our inapproximability results extend to envy-free pricing, an important problem in computational economics. We describe how the (budgeted) unique coverage problem, motivated by real-world applications, has close connections to other theoretical problems, including max cut, maximum coverage, and radio broadcasting.

Key words. unique coverage, hardness of approximation, wireless networks

AMS subject classifications. 68Q25, 68W25

DOI. 10.1137/060656048

1. Introduction.

In this paper we consider the approximability of the following natural maximization analogue of the set cover problem:

> **Unique coverage problem.** Given a universe $U = \{e_1, \ldots, e_n\}$ of elements, and given a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ to maximize the number of elements that are *uniquely covered*, i.e., appear in exactly one set of $\mathcal{S}'$.

We also consider a generalized form of this problem that is useful for several applications (detailed in section 2):

> **Budgeted unique coverage problem.** Given a universe $U = \{e_1, \ldots, e_n\}$ of elements, a profit p_i for each element e_i, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, a cost c_i of each subset S_i, and a budget B, find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$, whose total cost is at most the budget B, to maximize the total profit of elements that are *uniquely covered*, i.e., appear in exactly one set of $\mathcal{S}'$.

Motivation. Logarithmic inapproximability for minimization problems is by now commonplace, starting in 1993 with a result for the celebrated set cover problem

*Received by the editors April 3, 2006; accepted for publication (in revised form) May 28, 2008; published electronically September 8, 2008. A preliminary version of this paper appeared in *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006.
http://www.siam.org/journals/sicomp/38-4/65604.html

†Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (edemaine@mit.edu, hajiagha@mit.edu).

‡Department of Computer Science and Applied Mathematics, The Weizmann Institute, Rehovot 76100, Israel (uriel.feige@weizmann.ac.il).

§Department of Computing Science, University of Alberta, Edmonton, AB, T6G 2E8, Canada (mreza@cs.ualberta.ca). This author's research was supported by NSERC and a faculty startup grant.

[42], which has since been improved to the optimal constant [18] and to assume just $P \neq NP$ [47], and has been used to prove other tight (not necessarily logarithmic) inapproximability results for a variety of minimization problems; see, e.g., [34, 25, 13]. In contrast, for maximization problems, $\log n$ inapproximability seems more difficult, and relatively few results are known. The only three such results of which we are aware are $(1+\varepsilon)/\ln n$ inapproximability for domatic number unless $NP \subseteq DTIME(n^{O(\log \log n)})$ [21]; $1/\log^{1/2-\varepsilon} n$ inapproximability for the maximum edge-disjoint paths problems unless $NP \subseteq ZPTIME(n^{\text{polylog } n})$ [5, 6, 48]; and $1/\log^{1-\varepsilon} n$ inapproximability for the maximum edge-disjoint cycles problems unless $NP \subseteq ZPTIME(n^{\text{polylog } n})$ [38]. Although these problems are interesting, they are rather specific, and we lack a central maximization problem analogous to set cover to serve as a basis for further reduction to many other maximization problems.

The unique coverage problem defined above is a natural maximization version of set cover which was brought to our attention from its applications in wireless networks. In one (simplified) application, we have a certain budget to build/place some transmitters at a subset of some specified set of possible locations. Our goal is to maximize the clients that are "covered" by (i.e., are within the range of) exactly one transmitter; these are the clients that receive signal without interference; see section 2.1 for details. Another closely related application is the radio broadcast problem, in which a message (starting from one node of the network) is to be sent to all the nodes in the network in rounds. In each round, some of the nodes that have already received the message send it to their neighbors, and a node receives a message only if it receives it from exactly one of its neighbors. The goal is to find the minimum number of rounds to broadcast the message to all the nodes; see section 2.5 for details. Therefore, every single round of a radio broadcast can be seen as a unique coverage problem. These applications and others are studied in more detail in section 2.

Known results. To the best of our knowledge, there is no explicit study in the literature of the unique coverage problem and its budgeted version. However, the closely related radio broadcast problem has been studied extensively in the past, and implicitly includes an $\Omega(1/\log n)$-approximation algorithm for the basic (unbudgeted) unique coverage problem; see section 2.5 for details.

Concurrently and independently of our work, Guruswami and Trevisan [28] study the so-called 1-in-k SAT problem, which includes the unique coverage problem (but not its budgeted version) as a special case. In particular, they show that there is an approximation algorithm that achieves an approximation ratio of $1/e$ on satisfiable instances (in which all items can be covered by mutually disjoint sets).

Our results. On the positive side, we give an $\Omega(1/\log n)$-approximation for the budgeted unique coverage problem. We also show that if each set has a bound B on the ratio between the maximum profit of a set (where the profit of a set is the sum of profits of its elements) and the minimum profit of an element, then budgeted unique coverage has an $\Omega(1/\log B)$-approximation. Section 4 proves these results.

The main focus of this paper is proving the following inapproximability results. We show that it is hard to approximate the unique coverage problem within a factor of $\Omega(1/\log^{\sigma} n)$ for some constant σ depending on ε, assuming that $NP \not\subseteq BPTIME(2^{n^{\varepsilon}})$ for some constant $\varepsilon > 0$. This inapproximability can be strengthened to $\Omega(1/\log^{1/3-\varepsilon} n)$ (for any constant $\varepsilon > 0$) under the assumption that refuting random instances of 3SAT is hard on average (hardness of R3SAT as in [19]). The inapproximability can be further strengthened to $1/(\varepsilon \log n)$ for some constant $\varepsilon > 0$ under a plausible hardness

hypothesis about a problem called balanced bipartite independent set; see Hypothesis 3.22. Section 3 proves all of these results.

Our hardness results have other implications regarding the hardness of some well-studied problems. In particular, for the problem of unlimited-supply single-minded (envy-free) pricing, a recent result [27] proves an $\Omega(1/\log n)$-approximation, but no inapproximability result better than APX-hardness is known. As we show in section 2.2, our hardness results for the unique coverage problem imply the same hardness-of-approximation bounds for this version of envy-free pricing. For the radio broadcast problem, as we discuss in section 2.5, there is essentially a gap of $\Omega(\log n)$ between the approximation and inapproximability factors ($O(\log^2 n)$ vs. $\Omega(\log n)$). We believe that our technique to prove hardness of unique coverage may shed some light on how to obtain a hardness of approximation beyond $\Omega(\log n)$ for this problem.

More generally, there are many maximization problems for which the best known approximation factor is $\Omega(1/\log n)$—see, e.g., [27, 9, 43]—and it is not known whether an $\Omega(1)$-factor approximation is possible. Often (as indeed is the case with unique coverage) these problems naturally decompose into $\Theta(\log n)$ subproblems, where at least an $\Omega(1/\log n)$ fraction of the optimum's value comes from one of these subproblems. In isolation, each subproblem can be approximated up to a constant factor, leading to an $\Omega(1/\log n)$-approximation algorithm for the whole problem. It may appear that this isolation approach is too naïve to give the best possible approximation, and that by a clever combination of the subproblems, it should be possible to get an $\Omega(1)$-approximation algorithm. Our hardness results show to the contrary that such intelligent combination can be hard, in the sense that the naïve isolation approach cannot be substantially improved, and suggest how one might obtain better hardness results for these problems.

2. Applications and related problems.

2.1. Wireless networks. Our original motivation for the budgeted unique coverage problem is a real-world application arising in wireless networks.[1] We are given a map of the densities of mobile clients throughout a service region (e.g., the plane with obstacles). We are also given a collection of candidate locations for wireless base stations, each with a specified building cost and a specified coverage region (typically a cone or a disk, possibly obstructed by obstacles). This collection may include multiple options for base stations at the same location, e.g., different powers and different orientations of antennas. The goal is to choose a set of base stations and options to build, subject to a budget on the total building cost, in order to maximize the density of served clients.

The difficult aspect of this problem (and what distinguishes it from maximum coverage—see section 2.4) is interference between base stations. In the simplest form, there is a limit k on the number of base stations that a mobile client can reasonably hear without conflict between the signals; any client within range of more than k base stations cannot communicate because of interference and thus is not serviced. More generally, a mobile client's reception is better when it is within range of fewer base stations, and our goal is to maximize total reception. To capture these desires, the instance specifies the *satisfaction* s_i of a client within range of exactly i base stations,

[1] The application arises in the context of cellular networks at Bell Labs. The problem we consider here is a somewhat simplified theoretical formulation of this application. In the real application, the interference patterns are more complicated, but this problem seems to be the cleanest theoretical formulation.

such that $s_0 = 0$ and $s_1 \geq s_2 \geq s_3 \geq \cdots \geq 0$. The goal is to choose a set of base stations and options, again subject to the budget constraint, in order to maximize the total satisfaction weighted by client densities.

When all s_i's are equal, we just have the maximum coverage problem (section 2.4). When $s_1 = 1$ and $s_i = 0$ for all $i \neq 1$, this problem can be formulated as a budgeted unique coverage problem, by standard discretization of the density map. More generally, for any assignment of s_i's, the problem can be formulated as a generalization of budgeted unique coverage, the *budgeted low-coverage problem*. In this problem, we are also given satisfaction factors s_i for an element being covered exactly i times, zero for $i = 0$ and nonincreasing for $i > 0$, and the goal is to maximize the total satisfaction, i.e., the sum over all elements of the product of the element's profit (here, density) and its satisfaction factor (the appropriate s_i). We show that our approximation algorithms for the budgeted unique coverage problem apply more generally to the budgeted low-coverage problem, yielding an $\Omega(1/\log n)$-approximation where n is the total number of options for base stations.

While similar problems about base-station placement have been considered before, very few works consider maximization forms of the problem, which is the focus of this paper. Lev-Tov and Peleg [41] consider the following very specialized form of the problem: base stations are unit disks in the plane, and the goal is to maximize the number of uniquely receiving clients. For this problem they give an $n^{O(\sqrt{n})}$-time algorithm, where n is the number of candidate disks. In the application of interest, we believe that it is more natural to allow clients to be covered more than once but reduce (or eliminate) the satisfaction of these clients; this removal of an artificial constraint may enable substantially better solutions to the problem. Other work [31, 22, 8] solves the problem of assigning powers to base stations such that, when each client prefers a unique base station, we do not violate the capacities of the base stations, provided the number of clients is at most the total capacity of the network.

2.2. Envy-free pricing. Fundamental to "fair" equilibrium pricing in economics is the notion of envy-free pricing [49, 26]. This concept has recently received attention in computer science [1, 27] in the new trend toward an algorithmic understanding of economic game theory; see, e.g., [14, 15] for related work.

The following version of envy-free pricing was considered in [27]. A single seller prices m different items, $I_1, \ldots, I_m$, each with a specified quantity (limited or unlimited *supply*). Each of n buyers b_i $(1 \leq i \leq n)$ wishes to purchase a subset of items (a *bundle*), and the seller knows the maximum price that each buyer is willing to pay for each bundle (the *valuation*). A buyer's *utility* is the difference between the valuation and the price of the bundle (sum of the prices of the items in the bundle) as sold to the buyer. The seller must choose the item prices, price p_i for item I_i, and which bundles are sold to which buyers in such a way that is *envy-free*: each buyer should be sold a bundle that has the maximum utility among all bundles. The goal is to maximize the seller's *profit*, i.e., the total price of the sold bundles.

Among other results, Guruswami et al. [27] give an $\Omega(1/(\log n + \log m))$-approximation algorithm for the unlimited-supply *single-minded* bidder problem, where each buyer b_i considers only one particular bundle B_i and buys it if the cost is less than the valuation. They also give a constant-factor hardness-of-approximation result for this problem, via a reduction from max-cut. Single-minded bidders were considered before in the context of combinatorial auctions and mechanism design [7, 45, 40]. The unlimited-supply assumption in combination with single-mindedness simplifies the problem, as the notion of *envy* does not play a role in this case. The general version

of the envy-free pricing problem is of course at least as difficult as this special case.

We now show that unlimited-supply single-minded (envy-free) pricing is as hard to approximate as the unique coverage problem. The reduction is as follows. Each set S_i in the collection (for the instance of unique coverage) maps to an item I_i (for the instance of envy-free pricing). Each element e_i of the universe U maps to a buyer b_i. Buyer b_i has a valuation of 1 for one bundle, B_i, namely, the set of items I_j that correspond to sets S_j containing the element e_i. In this context, every price assignment is envy-free, because we have unlimited supply for each item so the seller can always sell each buyer its desired bundle (if the buyer wants). Because the valuations are all 1, we can assume that all prices are between 0 and 1. By randomized rounding (see Lemma A.1), we can assume that all prices are either 0 or 1, at a loss of a constant factor in profit. In this case, each buyer b_i will buy its bundle precisely if at most one item is priced at 1, and the rest of the items are priced at 0. If all items in a bundle are priced at 0, then the seller makes no profit; if exactly one item is priced at 1 and the rest are priced at 0, then the seller profits by 1. Thus the effective goal is to assign prices of 0 or 1 in order to maximize the number of bundles for which exactly one item is priced at 1, which is identical to the original unique coverage problem.

Therefore our hardness-of-approximation results apply to unlimited-supply single-minded (envy-free) pricing and establish semilogarithmic inapproximability.

2.3. Max-cut. Recall the max-cut problem: given a graph G, find a cut $(S, \overline{S})$, where $S \subseteq V(G)$ and $\overline{S} = V(G) - S$, that maximizes the number of edges with one endpoint in S and the other endpoint in $\overline{S}$. The max-cut problem can be seen to be equivalent to a special case of the unique coverage problem in which every element is in exactly two sets. Simply view every vertex as a set and every edge as an element.

Max-cut is 0.878567-approximable [24], 0.941177-inapproximable assuming P $\neq$ NP [29], and 0.878568-inapproximable assuming the unique games conjecture [33]. From these results one can immediately obtain constant-factor hardness for unique coverage, but in this paper we show that unique coverage is in fact much harder.

2.4. Maximum coverage. Our budgeted unique coverage problem is also closely related to the *budgeted maximum coverage* variation of set cover: given a collection of subsets S of a universe U, where each element in U has a specified weight and each subset has a specified cost, and given a budget B, find a subcollection $S' \subseteq S$ of sets, whose total cost is at most B, in order to maximize the total weight of elements covered by S'. For this problem, there is a $(1 - 1/e)$-approximation [30, 34], and this is the best constant approximation ratio possible unless P = NP [18, 34].

At first glance, one might expect the greedy $(1 - 1/e)$-approximation algorithm to work for unique coverage as well: the only difference between the two problems is whether we count elements that are covered (contained in at least one set) or uniquely covered (contained in exactly one set). Of course, we show that the (in)approximability of the two problems is quite different. Indeed, a natural class of greedy algorithms can be very bad for unique coverage. Consider the collection of sets $S_i = \{i, k+1, k+2, \ldots, n\}$ for $i = 1, 2, \ldots, k$, with an infinite budget B. Consider a greedy algorithm that repeatedly chooses a set to add to the cover, according to some (arbitrary) rule, with one of two stopping conditions: either when the budget is exhausted, or when the number of uniquely covered elements goes down. Then the approximation ratio is $\Theta(1/n)$ with the first stopping condition if $k = 2$, and with the second stopping condition if $k = n - 2$.

2.5. Radio broadcast. The unique coverage problem is closely related to a single "round" of the *radio broadcast* problem [10]. This problem considers a *radio network*, i.e., a network of processors (nodes) that communicate synchronously in rounds. In each round, a node can either transmit to all of its neighbors in an undirected graph (representing the communicability between pairs of nodes) or not transmit. A node receives a message if exactly one of its neighbors transmits a message in the round; otherwise the messages are lost because of radio interference. In the radio broadcast problem, initially one node has a message, and the goal is to propagate this message to all nodes in the network.

Radio broadcast is one of the most important communication primitives in radio networks, and the problem has been studied extensively in the literature. In summary, the current best algorithms for approximating the minimum number of rounds are a (multiplicative) $O(\log^2 n)$-approximation [12, 10, 36, 35] and an additive $O(\log^2 n)$-approximation [23]. Alon et al. [3] show that, even for graphs with diameter 3, $\Omega(\log^2 n)$ rounds can be necessary. The problem has also been considered in the context of distributed algorithms [39, 37] and low-energy ad-hoc wireless networks [4]. Elkin and Kortsarz prove a lower bound of inapproximability of a (multiplicative) $\Omega(\log n)$ [16] and an additive $\Omega(\log^2 n)$ [17] assuming NP $\not\subseteq$ BPTIME($n^{O(\log\log n)}$).

The unique coverage problem (but not the budgeted version) can be considered as a single round of a greedy algorithm for the radio broadcast problem, which maximizes the number of nodes that receive the message in each step. Specifically, consider the bipartite subgraph where one side consists of all nodes that currently have the message and the other side consists of all nodes that do not yet have the message. In one round of the greedy algorithm, the goal is to find a subset of nodes in the first side to transmit in order to maximize the number of nodes in the second side that (uniquely) receive the message. This problem is equivalent to unique coverage, viewing nodes on the first side as sets and the nodes on the second side as elements of the universe.

One implication of the radio broadcasting work on unique coverage is an implicit $\Omega(1/\log n)$-approximation algorithm for the (unbudgeted) unique coverage problem. Namely, there is a randomized broadcasting algorithm that, in each round, guarantees transmission to an $\Omega(1/\log r)$ fraction of the r neighbors of nodes that currently have the message. Because r is an obvious upper bound on the number of successful transmissions of the message, this result is an $\Omega(1/\log r) = \Omega(1/\log n)$-approximation in this special case; see, e.g., [10].

To avoid the possibility of misunderstanding, let us point out that the known hardness-of-approximation results for radio broadcast [16, 17] do not give (neither explicitly nor implicitly) any useful hardness-of-approximation result for the unique coverage problem (not even a constant factor). Likewise, our hardness-of-approximation results for the unique coverage problem do not by themselves imply any new hardness-of-approximation results for radio broadcast. However, they do introduce a component that may be useful in future hardness-of-approximation results for the radio broadcast problem, as they show that the greedy broadcast policy might need to lose a semilogarithmic factor already in a single round (a fact not used in [16, 17]).

3. Inapproximability. In this section we prove that it is hard to approximate unique coverage within a factor of $\Omega(1/\log^c n)$ for some constant c, $0 < c \leq 1$. Our main result is a general reduction from a variation of the balanced bipartite independent set (BBIS) problem (defined below) to the unique coverage problem. From this reduction and the known hardness results for BBIS, we can derive an $O(1/\log^c n)$ hardness for unique coverage. Under a plausible assumption about the hardness of BBIS, this bound can be improved to $O(1/\log n)$.

We consider the natural graph-theoretic model of the unique coverage problem. Define the bipartite graph $H(V \cup W, F)$ with a vertex $v_i \in V$ for every set $S_i \in \mathcal{S}$ and a vertex $w_j \in W$ for every element $e_j \in U$, and with an edge $f = (v_i, w_j) \in F$ precisely if $e_j \in S_i$. Then the unique coverage problem asks to find a subset $V' \subseteq V$ such that the subgraph induced by $V' \cup W$ has the maximum number of degree-1 vertices in W. We call the degree-1 vertices *uniquely covered* by the vertices in V'.

DEFINITION 3.1. *Given a bipartite graph $G(A \cup B, E)$ with $|A| = |B| = n$, the BBIS problem asks to find the largest value of k such that there are sets $\tilde{A} \subseteq A$ and $\tilde{B} \subseteq B$ with $|\tilde{A}| = |\tilde{B}| = k$ where the subgraph $\tilde{G}$ of G induced by $\tilde{A} \cup \tilde{B}$ is an independent set.*

As detailed below, this problem has known hardness results (see [19, 32]). In order to prove hardness of the unique coverage problem, we define a variation of BBIS. Then we give a reduction from this variation of BBIS. Before stating the main result, we need to define what we mean by an (a, b)-BIS (bipartite independent set). Let $G(A \cup B, E)$ be a given a bipartite graph. If the subgraph $\tilde{G}$ induced by $\tilde{A} \subseteq A$ and $\tilde{B} \subseteq B$, with $|\tilde{A}| = a$ and $|\tilde{B}'| = b$, is an independent set, then we call it an (a, b)-*BIS*.

DEFINITION 3.2. *Given bipartite graph $G(A \cup B, E)$ with $|A| = |B| = n$, and given parameters γ, γ', δ, and δ' satisfying $0 < \gamma' < \gamma \leq 1$ and $0 \leq \delta < \delta' \leq 1$, the* BBIS$(\gamma, \gamma', \delta, \delta')$ *problem is to distinguish between two cases:*

1. Yes instance: *G has an $(n^{\gamma}, n/\log^{\delta} n)$-BIS.*
2. No instance: *G has no $(n^{\gamma'}, n/\log^{\delta'} n)$-BIS.*

The main theorem of this section is the following.

THEOREM 3.3. *There is a polynomial-time probabilistic reduction from BBIS to the unique coverage problem with the following properties. Given a bipartite graph $G(A \cup B, E)$ with $|A| = |B| = n$ and given constants γ, γ', δ, and δ' satisfying $0 < \gamma' < \gamma \leq 1$ and $0 \leq \delta < \delta' \leq 1$, the algorithm constructs in randomized polynomial time an instance $H(V \cup W, F)$ of unique coverage with $|W| = \Theta((\gamma - \gamma')n \log n)$ and $|V| = n$ satisfying the following two properties:*

1. *If G is a Yes instance of* BBIS$(\gamma, \gamma', \delta, \delta')$*, then H has a solution of size $\Omega((\gamma - \gamma')n \log^{1-\delta} n)$.*
2. *If G is a No instance of* BBIS$(\gamma, \gamma', \delta, \delta')$*, then H has no solution of size $O((\gamma - \gamma')n \log^{1-\delta'} n)$.*

COROLLARY 3.4. *Assuming that* BBIS$(\gamma, \gamma', \delta, \delta')$ *is hard for constants $\gamma, \gamma', \delta, \delta'$, we get a hardness of approximation within a factor of $\Omega(1/\log^{\delta'-\delta} n)$ for unique coverage.*

This theorem is proved in section 3.1. Next we show how the known hardness results for BBIS can be used to derive explicit hardness results for unique coverage. In particular, the following theorems follow from Theorem 3.3.

THEOREM 3.5. *Let $\varepsilon > 0$ be an arbitrarily small constant. Assuming that* NP $\not\subseteq$ BPTIME$(2^{n^{\varepsilon}})$*, it is hard to approximate the unique coverage problem within a factor of $\Omega(1/\log^{\sigma} n)$ for some constant $\sigma = \sigma(\varepsilon)$.*

Feige [19] makes the following hypothesis about average-case hardness of 3SAT.

> **R3SAT hardness hypothesis [19]:** Let ϕ be a 3SAT formula with n variables and $m = \Delta n$ clauses where every clause is generated independently at random by selecting three literals independently at random. For arbitrary large constant Δ, there is no polynomial-time algorithm that accepts if ϕ is satisfiable and refutes at least half of the times for those formulas that are not satisfiable.

Under this hypothesis, we can prove the same hardness result with an explicit value for σ.

THEOREM 3.6. *Assuming R3SAT hardness hypothesis, unique coverage is hard to approximate within a factor of* $\Omega(1/\log^{1/3-\sigma} n)$ *for an arbitrarily small constant* $\sigma > 0$.

Under a stronger (yet plausible) hardness assumption, explained in section 3.2, we close the gap between the approximation factor and the hardness of approximation, up to the constant multiplicative factor, by proving an $O(1/\log n)$-hardness result for unique coverage.

THEOREM 3.7. *Assuming a specific hardness of factor* $\Omega(n^{\varepsilon})$ *for BBIS for some constant* $\varepsilon > 0$ *(Hypothesis* 3.22*), it is hard to approximate the unique coverage problem within a factor of* $\Omega(1/\log n)$ *where the constant in the* Ω *term depends on* ε.

Theorems 3.5 to 3.7 are proved in section 3.2.

3.1. Reduction from BBIS to unique coverage and proof of Theorem 3.3.

Construction. Consider an instance of BBIS$(\gamma, \gamma', \delta, \delta')$: a bipartite graph $G(A \cup B, E)$ with $|A| = |B| = n$, and constants γ, γ', δ, and δ' with $0 < \gamma' < \gamma \leq 1$ and $0 \leq \delta < \delta' \leq 1$. We construct a graph $H(V \cup W, F)$ as an instance of unique coverage as follows.

First we construct a random graph $G'(A' \cup B', E')$ where A' is a copy of A and B' is a copy of B. For every $a \in A'$ and $b \in B'$ we place the edge (a, b) in E' with probability $1/n^{\gamma}$. So the expected degree of every vertex in G' is $n^{1-\gamma}$.

Now to construct H, let V be a copy of A. Then with $\gamma'' = \frac{\gamma - \gamma'}{7}$, create $p = \gamma'' \log n$ copies of B, named $W_1, \ldots, W_p$. We define a bipartite graph $H_i(V \cup W_i, F_i)$, for every $1 \leq i \leq p$, and at the end $H = \bigcup_{i=1}^{p} H_i$. Note that $|V| = n$ and $|W| = pn$. The set of edges F_i (in H_i) consists of the union of two edge sets: (i) the edges of the random graph G' induced on the vertices $V \cup W_i$ (V as A' and W_i as B'), plus (ii) the edges of another random graph G_i, where G_i is defined recursively as follows. Initially, G_1 is G induced on $V \cup W_1$. For every $i \geq 2$, G_i is obtained from G_{i-1} by deleting every edge independently with probability $\frac{1}{2}$. The edges of G' in H_i are called *type*-1 *edges* and the rest of the edges of H_i (which come from G_i) are called *type*-2 *edges* of H_i. In a solution to $V' \subseteq V$ for instance H, we say a vertex of W is uniquely covered by a type-1 (type-2) edge if that vertex is adjacent to exactly one vertex of V' and that edge is a type-1 (type-2) edge.

Proof overview. Here is the general idea of the proof. Intuitively, the balanced independent sets in G relate to the elements that will be uniquely covered by type-1 edges in H. The removal of edges (randomly) from G_i to G_{i+1} is to ensure that not too many vertices are uniquely covered by type-2 edges. More specifically, we will show that the number of vertices uniquely covered by type-2 edges (edges that were originally in G) in this instance is $O(n)$. So let us focus on the vertices uniquely covered by type-1 edges (i.e., edges from the random graph G' in each H_i).

First suppose that G is a Yes instance; i.e., it has an $(n^{\gamma}, \frac{n}{\log^{\delta} n})$-BIS, say $A^* \cup B^*$ (with $A^* \subseteq A$ and $B^* \subseteq B$). Because the expected degree of every vertex in G' is $n^{1-\gamma}$, the expected number of type-1 edges coming out of A^* (in G') is n, and because these edges are selected at random, we expect a fraction of $1/e$ of the vertices in B' (in G') and in particular a fraction of $1/e$ of the vertices in B^* to have degree 1. This implies that the type-1 edges in each H_i uniquely cover a linear number of vertices of B^* (at least in expectation); i.e., it gives a solution of size $\Omega(\frac{n}{\log^{\delta} n})$ in H_i. Because

$H = \bigcup_{i=1}^{p} H_i$ and $p = \gamma'' \log n$, we have a total of $\Omega(\gamma'' n \log^{1-\delta} n)$ vertices uniquely covered by type-1 edges.

Now suppose that G is a No instance; i.e., it has no $(n^{\gamma'}, n/\log^{\delta'} n)$-BIS. We will show that although we delete edges to construct G_i from G_{i-1}, the last (and most sparse) graph G_p will not have "too large" a bipartite independent set with high probability. This property will be used to show that, in every graph H_i, the number of vertices uniquely covered by type-1 edges in any solution of H is at most $O(n/\log^{\delta'} n)$ with high probability. Thus, the total number of vertices uniquely covered in H (by type-1 or type-2 edges) in any solution is at most $O(\gamma'' n \log^{1-\delta'} n + n)$ with high probability. Because $\delta' \leq 1$, this creates a hardness gap of $\Omega(1/\log^{\delta'-\delta} n)$.

Now we give the details of the proof of Theorem 3.3. We use the following simplified version of the Chernoff bound.

LEMMA 3.8 (Chernoff bound). *For independent* 0/1 *random variables* $X_1, \ldots, X_n$, $X = \sum_{i=1}^{n} X_i$, $\mu = E[X]$, *and any* $0 < \delta < 1$, *we have*

$$\Pr[|X - E[X]| > \delta\mu] \leq e^{-\delta^2 \mu/3}.$$

LEMMA 3.9. *For every selection of vertices* $V' \subseteq V$ *as a solution for instance* H, *the number of vertices uniquely covered by type-2 edges in any solution to* H *is* $O(n)$ *with high probability.*

Proof. Let $b \in B$ be an arbitrary vertex (in G) and assume that $w_1, \ldots, w_p$ are its corresponding vertices in $W_1, \ldots, W_p$. Consider any subset $V' \subseteq V$. Assuming that V' is a solution to unique coverage, we compute the probability that exactly i vertices out of $w_1, \ldots, w_p$ are uniquely covered by type-2 edges (of the vertices of V'). Assume that j is the first index for which w_j is uniquely covered by a type-2 edge, and $w_j, \ldots, w_{j+i-1}$ are the copies that are uniquely covered by a type-2 edge. Because every edge is deleted with probability $\frac{1}{2}$ from G_t to G_{t+1} (for $1 \leq t < p$), the probability that a single edge survives i rounds is 2^{-i}. Let X_b be the number of copies of b (from $w_1, \ldots, w_p$) that are uniquely covered by a type-2 edge (by the vertices of V') and define $X = \sum_{b \in B} X_b$. Therefore,

$$\mathrm{E}[X] = \sum_{b \in B} \mathrm{E}[X_b] = n \sum_{i=1}^{p} \frac{i}{2^i} \leq 3n.$$

Using the Chernoff bound (Lemma 3.8), we obtain

$$\Pr[X \geq 6n] \leq e^{-4n}.$$

Because there are 2^n subsets V', a union bound shows that the probability that, for at least one of those sets, the number of vertices in W that are uniquely covered by type-2 edges is $\geq 6n$ is at most $2^n \cdot e^{-4n} \leq e^{-\Omega(n)}$. This completes the proof of the lemma. ∎

Completeness. Suppose that G is a Yes instance; i.e., it has a $(n^{\gamma}, n/\log^{\delta} n)$-BIS, say, $A^* \cup B^*$ where $A^* \subseteq A$ and $B^* \subseteq B$. Assume that V' and W_i' are the subsets of vertices in H_i and A'' and B'' are the subsets of vertices in G' corresponding to A^* and B^*, respectively. Because G_i is obtained from G by deleting edges, there are no type-2 edges in $V' \cup W_i'$ in H_i (for any $1 \leq i \leq p$). Therefore, every vertex $w \in W_i'$ (for all values of $1 \leq i \leq p$) has degree 1 if and only if the corresponding vertex $w \in B''$ (in G') has degree 1. For every $w \in B''$, let X_w be a 0/1 random variable that is 1

if and only if $w \in B''$ has degree 1 (and so w is uniquely covered by a type-1 edge in H_i for all $1 \le i \le p$). With $X = \sum_{w \in B''} X_w$,

$$\begin{aligned}\mathrm{E}[X] &= \sum_{w \in B''} \Pr[X_w = 1] \\ &= |B''| \cdot \binom{|A''|}{1} \cdot \frac{1}{n^\gamma} \left(1 - \frac{1}{n^\gamma}\right)^{|A''|-1} \\ &\ge \frac{|B''|}{e} \ge \frac{n}{e \log^\delta n}.\end{aligned}$$

A simple application of the Chernoff bound shows that $\Pr[X \le \frac{n}{6 \log^\delta n}] \le e^{-\Omega(n/\log^\delta n)}$. Therefore, if we select the subset of vertices in V (in H) corresponding to A^* (in G), then, with high probability, there are at least $p \cdot \frac{n}{6 \log^\delta n} = \Omega(\gamma'' n \log^{1-\delta} n)$ vertices in W uniquely covered (by type-1 edges). Thus, we have proved the following.

COROLLARY 3.10. *If G is a Yes instance, then, with high probability, H has a unique cover of size $\Omega(\gamma'' n \log^{1-\delta} n)$.*

Soundness. Suppose that G is a No instance; i.e., it has no $(n^{\gamma'}, n/\log^{\delta'} n)$-BIS. Our goal is to show that, with high probability, every solution to unique coverage for H has size $O(\gamma'' n \log^{1-\delta'} n)$. Because by Lemma 3.9 the number of vertices uniquely covered by type-2 edges is $O(n)$, we only need to prove that, with high probability, the number of vertices uniquely covered by type-1 edges is at most $O(\gamma'' n \log^{1-\delta'} n)$.

Consider any solution to unique coverage for H. By construction of the H_i's, it is easy to see that, for every vertex $b \in B$ (in G), if the corresponding vertex in W_i is uniquely covered by a type-1 edge in H_i, then all the corresponding vertices of b in the W_j's, for $i \le j \le p$, are also uniquely covered by a type-1 edge. Therefore, if we prove that the number of vertices uniquely covered by type-1 edges in H_p is upper bounded (with high probability) by $O(n/\log^{\delta'} n)$, then because $p = \gamma'' \log n$, we obtain the claimed upper bound for the total number of vertices uniquely covered by type-1 edges.

Suppose that $V' \subseteq V$ and $W' \subseteq W_p$ are such that all the vertices in W' are uniquely covered by V', and the edges that cover them are all type-1 edges. It is easy to see that $V' \cup W'$ must be a bipartite independent set in G_p (otherwise there is some type-2 edge incident to some vertex $w \in W'$ and therefore w is not uniquely covered).

LEMMA 3.11. *If $V' \cup W'$ (with $V' \subseteq V$ and $W' \subseteq W_p$) is a bipartite independent set in G_p, then with high probability, either $|V'| < n^{(\gamma+\gamma')/2}$ or $|W'| < 2n/\log^{\delta'} n$; i.e., G_p has no $(n^{(\gamma+\gamma')/2}, 2n/\log^{\delta'} n)$-BIS.*

Proof. Suppose that $V' \subseteq V$ and $W' \subseteq W_p$ satisfy $|V'| = n^{(\gamma+\gamma')/2}$ and $|W'| = 2n/\log^{\delta'} n$. Partition V' into $q = n^{(\gamma-\gamma')/2}$ subsets $V'_1, \ldots, V'_q$, each of size $n^{\gamma'}$. Let A^*_i and B^* $(1 \le i \le q)$ be the subset of vertices of A and B (in G) corresponding to V'_i and W', respectively. Consider the subgraph of G induced by $A^*_i \cup B^*$. Because $|A^*_i| = n^{\gamma'}$, $|B^*| = 2n/\log^{\delta'} n$, and because G has no $(n^{\gamma'}, n/\log^{\delta'} n)$-BIS, it follows that at least $n/\log^{\delta'} n$ vertices in B^* must be connected to the vertices in A^*_i. Therefore, the total number of edges in the subgraph induced by $B^* \cup \bigcup_{i=1}^q A^*_i$ is at least $q \cdot n/\log^{\delta'} n = \Omega(n^{1+(\gamma-\gamma')/2}/\log^{\delta'} n)$. Because $G_1 = G$, $V' \cup W'$ forms an independent set in G_p only if all of these $\Omega(n^{1+(\gamma-\gamma')/2}/\log^{\delta'} n)$ edges are deleted while G_p is created. Because in creating G_{i+1} from G_i, edges are deleted with probability $\frac{1}{2}$, we

have

$$\Pr[V' \cup W' \text{ is an independent set in } G_p] \le (1-2^{-p})^{\Omega(n^{1+(\gamma-\gamma')/2}/\log^{\delta'} n)}. \tag{3.1}$$

The number of such subsets $V' \cup W'$ is

$$\binom{n}{n^{(\gamma+\gamma')/2}}\binom{n}{2n/\log^{\delta'} n}. \tag{3.2}$$

Thus, using (3.1) and (3.2), the expected number of bipartite independent sets $V' \cup W'$ with $|V'| = n^{(\gamma'+\gamma)/2}$ and $|W'| = 2n/\log^{\delta'} n$ in G_p is at most

$$\begin{aligned}
&(1-2^{-p})^{\Omega(n^{1+(\gamma-\gamma')/2}/\log^{\delta'} n)}\binom{n}{n^{(\gamma+\gamma')/2}}\binom{n}{2n/\log^{\delta'} n}\\
&\le (1-n^{-(\gamma-\gamma')/7})^{\Omega(n^{1+(\gamma-\gamma')/2}/\log^{\delta'} n)} \cdot \left(\frac{en}{n^{(\gamma+\gamma')/2}}\right)^{n^{(\gamma+\gamma')/2}} \cdot \left(\frac{en}{2n/\log^{\delta'} n}\right)^{2n/\log^{\delta'} n}\\
&\le e^{-\Omega(n^{1+(\gamma-\gamma')/2-(\gamma-\gamma')/7}/\log^{\delta'} n)} \cdot e^{O(n^{(\gamma+\gamma')/2}\log n)} \cdot e^{O(n\log\log n/\log^{\delta'} n)}\\
&\le e^{-\Omega(n^{1+(\gamma-\gamma')/3})} \cdot e^{O(n/\log^{\delta'/2} n)}\\
&\le e^{-\Omega(n^{1+(\gamma-\gamma')/3})}.
\end{aligned}$$

Therefore, with probability $1 - e^{-\Omega(n^{1+(\gamma-\gamma')/3})}$, for every bipartite independent set $V' \cup W'$ of G_p, either $|V'| < n^{(\gamma+\gamma')/2}$ or $|W'| < 2n/\log^{\delta'} n$; i.e., G_p has no $(n^{(\gamma+\gamma')/2}, 2n/\log^{\delta'} n)$-BIS. ∎

LEMMA 3.12. *With high probability, for every selection of vertices $V' \subseteq V$ as a solution to instance H, for every H_i ($1 \le i \le p$), the number of vertices uniquely covered by type-1 edges is at most $O(n/\log^{\delta'} n)$.*

Proof. Let $V' \subseteq V$ be any fixed solution to instance H. Clearly, for every vertex uniquely covered by a type-1 edge in W_i, its corresponding copy is also uniquely covered (by a type-1 edge) in W_j for every $i \le j \le p$. So let us focus on the number of vertices uniquely covered by type-1 edges in H_p. If $W'_p \subseteq W_p$ is the set of vertices uniquely covered by type-1 edges in H_p (by V'), then $V' \cup W'_p$ is a $(|V'|, |W'_p|)$-BIS in G_p. We are going to use Lemma 3.11 to prove that, with high probability (for all possible choices of V'), the size of W'_p is small.

First consider the case that $|V'| \ge n^{(\gamma+\gamma')/2}$ and V', W'_p form a $(|V'|, |W'_p|)$-BIS in G_p. By Lemma 3.11, the probability that $V' \ge n^{(\gamma+\gamma')/2}$ and $|W'_p| \ge 2n/\log^{\delta'} n$ is at most $e^{-\Omega(n^{1+(\gamma-\gamma')/3})}$. The number of solutions to H (i.e., subsets V') that satisfy the bound on V' is clearly at most 2^n. Thus, the probability that there is a solution V' such that $V' \ge n^{(\gamma+\gamma')/2}$ and $|W'_p| \ge 2n/\log^{\delta'} n$ and V', W'_p forms a $(|V'|, |W'_p|)$-BIS in G_p is at most

$$2^n \cdot e^{-\Omega(n^{1+(\gamma-\gamma')/3})} = e^{-\Omega(n^{1+(\gamma-\gamma')/3})}. \tag{3.3}$$

Now consider the case that $|V'| < n^{(\gamma+\gamma')/2}$ (and of course $|W'_p| \le n$). In this case, we show that, with high probability, $|W'_p| \le O(n^{1-(\gamma-\gamma')/3})$, which is clearly $O(n/\log^{\delta'} n)$. Consider an arbitrary vertex $w \in W'_p$ and let X_w be a 0/1 random variable that is 1 if and only if w is incident to exactly one type-1 edge. With

$X = \sum_{w \in W'_p} X_w$,

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{w \in W'_p} \Pr[X_w = 1] \\
&= \sum_{w \in W'_p} \binom{|V'|}{1} \cdot \frac{1}{n^\gamma} \left(1 - \frac{1}{n^\gamma}\right)^{|V'|-1} \\
&< n \cdot n^{(\gamma+\gamma')/2} \cdot \frac{1}{n^\gamma} \\
&\leq n^{1-(\gamma-\gamma')/2}.
\end{aligned}$$

Using the Chernoff bound,

$$\Pr[X \geq n^{1-(\gamma-\gamma')/3}] \leq e^{-\Omega(n^{1-(\gamma-\gamma')/6})}.$$

The number of solutions V' such that $|V'| < n^{(\gamma+\gamma')/2}$ is at most $\binom{n}{n^{(\gamma+\gamma')/2}} \leq \left(\frac{en}{n^{(\gamma+\gamma')/2}}\right)^{n^{(\gamma+\gamma')/2}} \leq e^{O(n^{(\gamma+\gamma')/2} \cdot \log n)}$. Thus, the probability that there is a solution V' such that $|V'| < n^{(\gamma+\gamma')/2}$ and $|W'_p| > O(n^{1-(\gamma-\gamma')/3})$ and V', W'_p forms a $(|V'|, |W'_p|)$-BIS in G_p is at most

$$e^{O(n^{(\gamma+\gamma')/2} \cdot \log n)} \cdot e^{-\Omega(n^{1-(\gamma-\gamma')/6})} \leq e^{-\Omega(n^{1-(\gamma-\gamma')/6})}, \tag{3.4}$$

since $1 > \frac{2\gamma}{3} + \frac{\gamma'}{3}$.

Therefore, using (3.3) and (3.4), with high probability over all possible solutions V', $|W'_p| \leq O(n/\log^{\delta'} n)$ as wanted. □

COROLLARY 3.13. *If G is a No instance, then, with high probability, every solution to unique coverage for H has size at most $O(\gamma'' n \log^{1-\delta'} n)$.*

Proof. From Lemma 3.12 and because $p = \gamma'' \log n$, it follows that, with high probability, the number of vertices uniquely covered by type-1 edges is at most $O(\gamma'' n \log^{1-\delta'} n)$. Combining this bound with Lemma 3.9 shows that if G is a No instance (i.e., has no $(n^{\gamma'}, n/\log^{\delta'} n)$-BIS), then the size of any solution to unique coverage for H is at most $O(\gamma'' n \log^{1-\delta'} n)$. □

Proof of Theorem 3.3. The proof follows easily from Corollaries 3.10 and 3.13 and the assumption that $BBIS(\gamma, \gamma', \delta, \delta')$ is hard. □

3.2. Proving specific hardness results for unique coverage. In this subsection we prove Theorems 3.5, 3.6, and 3.7. In order to prove these theorems, we will prove some hardness results for BBIS$(\gamma, \gamma', \delta, \delta')$ and then combine them with Theorem 3.3.

Recently, two hardness results for BBIS were proved by Feige [19] and Khot [32] under different complexity assumptions. Feige [19] proved a constant factor hardness result for BBIS under the R3SAT hardness hypothesis (for more details see [19]).

THEOREM 3.14 (see [19]). *For every $\varepsilon > 0$ and a given bipartite graph $G(A \cup B, E)$ with $|A| = |B| = n$, deciding between the following two cases is hard, under the complexity assumption that refuting random instances of 3SAT is hard on average:*

1. *G has a BBIS of size at least $(\frac{1}{4} - \varepsilon)n$.*
2. *Every BBIS of G has size smaller than $(\frac{1}{8} + \varepsilon)n$.*

More recently, Khot [32] proved a similar result, for some (unspecified) constants α and β instead of $(\frac{1}{4}-\varepsilon)$ and $(\frac{1}{8}+\varepsilon)$, respectively, but under a more plausible assumption that NP problems do not have subexponential-time algorithms. More specifically, he proved the following PCP (probabilistically checkable proof) theorem.

THEOREM 3.15 (see [32]). *For every $\varepsilon > 0$ an integer $d = O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$ exists such that there is a PCP verifier for SAT instances of size n such that the following hold:*

1. *The proof Π for the verifier has size 2^{n^ε}.*
2. *The verifier queries a set Q of size d bits from Π.*
3. *Every query bit is uniformly distributed over Π.*
4. *Completeness: If SAT is a Yes instance, Π is a correct proof, and Π_0 is the set of 0-bits in the proof (it contains half the bits from the proof), then*

$$\Pr[Q \subseteq \Pi_0] \geq \left(1 - O\left(\frac{1}{d}\right)\right)\frac{1}{2^{d-1}},$$

   where the probability is taken over the random tests of the verifier.
5. *Soundness: If SAT is a No instance and Π_* is any set of half the bits from Π, then*

$$\Pr[Q \subseteq \Pi_*] = \frac{1}{2^d} \pm O\left(\frac{1}{2^{20d}}\right).$$

A direct application of Theorem 3.15 implies the following (see [32]).[2]

THEOREM 3.16. *Let $\varepsilon > 0$ be an arbitrary constant and $d = O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$. Consider an instance Φ of SAT with n variables. Let $\alpha = \frac{3}{4}\frac{1}{2^{d-1}}$ and $\beta = (1-\frac{1}{2^{5d}})\alpha$. We can construct a bipartite graph $G(A \cup B, E)$ as an instance of BBIS from the PCP verifier of Theorem 3.15 with $|A| = |B| = N$ where $N = 2^{n^\varepsilon}$ such that the following hold:*

1. Yes instance: *If Φ is a Yes instance, then G has a BBIS of size αN.*
2. No instance: *If Φ is a No instance, then no BBIS of G has size βN.*

COROLLARY 3.17. *Assuming that* NP $\not\subseteq$ BPTIME(2^{n^ε}), *it is hard to distinguish between the Yes and No cases in the above theorem.*

In order to get a hardness for BBIS$(\gamma, \gamma', \delta, \delta')$, we need a stronger version of Theorem 3.16. For this, we boost the gap in Theorem 3.16 using the standard technique of graph products (see, for example, [20, 11]). Note that Theorem 1.2 in [32] amplifies the gap in Theorem 3.16 using the same technique. However, we require a gap which is asymmetric with respect to the sizes of sets selected on different parts; i.e., the bipartite independent set is not necessarily balanced. In particular, the gap created on one side (say A) is polynomial, whereas the gap created on the other side (that is, B) is polylogarithmic. Our proof is very similar to that of Theorem 1.2 in [32]. We need the following definition for our proof.

DEFINITION 3.18. *For a bipartite graph $G(A \cup B, E)$ and integers $K_A, K_B \geq 2$ the bipartite graph $G^{(K_A,K_B)}$ is defined as follows:*

1. *The vertex set of G^{K_A,K_B} is $A' \cup B'$, where $A' \cap B' = \emptyset$, $A' = A^{K_A}$, and $B' = B^{K_B}$; i.e., A' and B' are the sets of all K_A-tuples from A and all K_B-tuples from B, respectively.*
2. *Two vertices $(a_1, \ldots, a_{K_A}) \in A'$ and $(b_1, \ldots, b_{K_B}) \in B'$ are adjacent in $G^{(K_A,K_B)}$ if and only if $\exists i, j$, $1 \leq i \leq K_A$, $1 \leq j \leq K_B$, $(a_i, b_j) \in E$.*

[2] Khot defines the bi-clique problem and proves this theorem for bi-clique.

Suppose that $G(A \cup B, E)$ is a bipartite graph with $|A| = |B| = N$, $0 < \alpha < 1$ is a constant, and K_A, K_B are integers such that

$$\frac{1}{\alpha^{K_A}}, \frac{1}{\alpha^{K_B}} \in O(N). \tag{3.5}$$

Let $G^*(A^* \cup B^*, E^*)$ be a random subgraph of $G^{K_A,K_B}(A' \cup B', E')$ with $|A^*| = |B^*| = M$, where $M = N^3$ and every vertex of G^{K_A,K_B} is selected uniformly at random but with different probabilities for A' and B'.

LEMMA 3.19. *If $G(A \cup B, E)$ has a BBIS of size αN, then, with high probability, G^* has a $(\frac{1}{2}\alpha^{K_A}M, \frac{1}{2}\alpha^{K_B}M)$-BIS.*

Proof. Let $A_I \subseteq A$ and $B_I \subseteq B$ be subsets that form a BBIS of size αN in G. Clearly, the subgraph of G^{K_A,K_B} induced on $A_I^{K_A} \cup B_I^{K_B}$ is an independent set. Because the vertices of G^* are selected randomly, each vertex of A^* belongs to $A_I^{K_A}$ with probability α^{K_A}. Also, each vertex of B^* belongs to $B_I^{K_B}$ with probability α^{K_B}. Therefore, $\mathrm{E}[|A^* \cap A_I^{K_A}|] = \alpha^{K_A}|A^*|$ and $\mathrm{E}[B^* \cap B_I^{K_B}] = \alpha^{K_B}|B^*|$. Using the Chernoff bound and (3.5),

$$\Pr\left[|A^* \cap A_I^{K_A}| \leq \frac{1}{2}\alpha^{K_A}|A^*|\right] \leq 2^{-\Omega(N^2)}.$$

Similarly, with high probability, $|B^* \cap B_I^{K_B}| \geq \frac{1}{2}\alpha^{K_B}|B^*|$. Therefore, with high probability, G^* has a $(\frac{1}{2}\alpha^{K_A}|A^*|, \frac{1}{2}\alpha^{K_B}|B^*|)$-BIS. ∎

LEMMA 3.20. *If $G(A \cup B, E)$ has no BBIS of size βN, then, with high probability, G^* does not have any $(2\beta^{K_A}M, 2\beta^{K_B}M)$-BIS.*

Proof. First, note that every maximal bipartite independent set of G^{K_A,K_B} is of the form $A_I^{K_A} \cup B_I^{K_B}$, where $A_I \cup B_I$ is a bipartite independent set in G. Consider a fixed maximal bipartite independent set of G^{K_A,K_B}, say $A_I^{K_A} \cup B_I^{K_B}$. Either $|A_I^{K_A}| < \beta^{K_A}N^{K_A}$ or $|B_I^{K_B}| < \beta^{K_B}N^{K_B}$. Without loss of generality, assume $|A_I^{K_A}| < \beta^{K_A}N^{K_A}$. Because the elements in A^* and B^* are selected uniformly randomly, $\mathrm{E}[|A^* \cap A_I^{K_A}|] < \beta^{K_A}|A^*|$. Using the Chernoff bound,

$$\Pr\left[|A^* \cap A_I^{K_A}| \geq 2\beta^{K_A}|A^*|\right] \leq 2^{-\Omega(N^2)}.$$

An almost identical argument applies if $|B_I^{K_B}| < \beta^{K_B}N^{K_B}$. Because there are at most $2^{O(N)}$ possible maximal bipartite independent sets in G, using union bound, the probability of having a $(2\beta^{K_A}|A^*|, 2\beta^{K_B}|B^*|)$-BIS in G^* is $o(1)$. ∎

Let Φ be an instance of SAT and let $\varepsilon > 0$ be an arbitrary small constant. Define d, α, β, and $G(A \cup B, E)$ as in Theorem 3.16, with $|A| = |B| = N$. Also let $M = N^3$, $K_A = -\frac{(1-\gamma)\log M}{\log \alpha}$, and $K_B = -\frac{\delta \log\log M}{\log \alpha}$, for some constants $0 < \gamma, \delta < 1$ such that $1/\alpha^{K_A} \in O(N)$. Construct the graph G^{K_A,K_B} and the random subgraph of it $G^*(A^* \cup B^*, E^*)$, where $|A^*| = |B^*| = M$, as explained above. By Theorem 3.16 and Lemmas 3.19 and 3.20 we have the following:

1. If Φ is a Yes instance, then, by Theorem 3.16, G has a BBIS of size αN. So, by Lemma 3.19, with high probability, G^* has a $(\frac{1}{2}\alpha^{K_A}M, \frac{1}{2}\alpha^{K_B}M)$-BIS. By definition of K_A and K_B, this is a $(\frac{M^\gamma}{2}, \frac{M}{2\log^\delta M})$-BIS in G^*.
2. If Φ is a No instance, then, by Theorem 3.16, G has no BBIS of size βN. So, by Lemma 3.20, with high probability, G^* has no $(2\beta^{K_A}M, 2\beta^{K_B}M)$-BIS. With $\ell = \log_\alpha(\beta/\alpha)$, $\gamma' = \gamma - \ell(1-\gamma)$, and $\delta' = \delta(1+\ell)$, this means that, with high probability, G^* has no $(2M^{\gamma'}, \frac{2M}{\log^{\delta'} M})$-BIS.

Therefore, we have proved the following amplified version of Theorem 3.16.

THEOREM 3.21. *Let $G(A \cup B, E)$ be a given bipartite graph with $|A| = |B| = n$, together with an arbitrary small constant $\varepsilon > 0$, and $d = O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$, $\alpha = \frac{3}{4}\frac{1}{2^{d-1}}$, $\beta = (1 - \frac{1}{2^{5d}})\alpha$. Furthermore, let $0 < \gamma' < \gamma \leq 1$ and $0 \leq \delta < \delta' \leq 1$ be such that δ is any constant and with $\ell = \log_\alpha(\beta/\alpha)$: $\gamma' = \gamma - \ell(1-\gamma)$, and $\delta' = \delta(1+\ell)$. Then it is hard to distinguish between the following two cases unless* NP $\subseteq$ BPTIME(2^{n^ε})*:*

1. *G has a $(\frac{n^\gamma}{2}, \frac{n}{2\log^\delta n})$-BIS.*
2. *G has no $(2n^{\gamma'}, \frac{2n}{\log^{\delta'} n})$-BIS.*

Proof of Theorem 3.5. Given a bipartite graph $G(A \cup B, E)$ with $|A| = |B| = n$ as an instance of bipartite independent set and parameters ε, α, β, δ, γ, δ', and γ' as in Theorem 3.21 we construct $H(V \cup W, F)$ as explained in the proof of Theorem 3.3. We choose $\delta = \frac{1}{1+\ell}$, where $\ell = \log_\alpha(\beta/\alpha)$. Therefore $\delta' = 1$ and by Corollaries 3.10 and 3.13 unless NP $\subseteq$ BPTIME(2^{n^ε}) it is hard to approximate unique coverage within a factor of $\Omega(1/\log^{\delta'-\delta} n)$. Because $\delta' - \delta = \frac{\ell}{1+\ell}$ and ℓ is a function of ε, this completes the proof of the theorem. ∎

Proof of Theorem 3.6. If our starting point to prove Theorem 3.21 is Theorem 3.14 instead of Theorem 3.16, then we have $\alpha = \frac{1}{4} - \varepsilon$ and $\beta = \frac{1}{8} + \varepsilon$, and $\ell = \frac{1}{2} + \varepsilon'$, where $\varepsilon' = \varepsilon'(\varepsilon)$ is a constant. Then the same argument as in the proof of Theorem 3.5 proves a hardness of $O(1/\log^{\frac{1/2+\varepsilon'}{1+1/2+\varepsilon'}} n)$ which is $O(1/\log^{\frac{1}{3}-\sigma} n)$ for some $\sigma = \sigma(\varepsilon)$. ∎

We now turn to the proof of Theorem 3.7. It is based on the following hypothesis.

HYPOTHESIS 3.22. *Given a bipartite graph $G(A \cup B, E)$ with size $|A| = |B| = n$ as an instance of BBIS and for absolute constants $1 \geq \gamma > \gamma' > 0$ it is hard to distinguish the following two cases:*

1. *G has an $(n^\gamma, \Omega(n))$-BIS.*
2. *G has no $(n^{\gamma'}, n/\log n)$-BIS.*

Now we show how Hypothesis 3.22 would imply an $O(1/\log n)$-hardness for unique coverage.

Proof of Theorem 3.7. Given a bipartite graph $G(A \cup B, E)$ with size $|A| = |B| = n$ and $1 \geq \gamma > \gamma' > 0$ we construct $H(V \cup W, F)$, the instance of unique coverage, as in the reduction of Theorem 3.3.

1. If G has an $(n^\gamma, \Omega(n))$-BIS, then H has a unique coverage of size $\Omega((\gamma - \gamma')n\log^{1-\delta} n)$ with $\delta = 0$, which is $\Omega(n \log n)$.
2. If G has no $(n^{\gamma'}, n/\log n)$-BIS, then every unique coverage solution for H has size at most $O((\gamma - \gamma')n\log^{1-\delta'} n)$ with $\delta' = 1$, which is $O(n)$.

This implies that, assuming Hypothesis 3.22, it is hard to distinguish between the two cases above, and hence hard to approximate unique coverage within a factor of $\Omega(1/\log n)$. ∎

The authors suspect that Hypothesis 3.22 will be difficult to refute in the near future. The BBIS problem appears to be at least as hard to approximate as maximum independent set in general graphs. (This is not a theorem, but merely an empirical observation concerning currently known approximation algorithms.) For the latter problem, despite extensive work, no known polynomial-time algorithm can distinguish between graphs with independent sets of size $\Omega(n/k)$ and graphs with no independent set of size $n^{1/k}$, where k is some sufficiently large constant. It is plausible (though not certain) that any refutation of Hypothesis 3.22 would lead to major improvements in the approximation ratio for maximum independent sets in general graphs.

4. Approximation algorithms.

4.1. $\Omega(1/\log m)$-approximation. In this section we develop our main logarithmic approximation algorithm.

THEOREM 4.1. *There is an $\Omega(1/\log\rho) = \Omega(1/\log m)$-approximation algorithm for the budgeted unique coverage problem, where ρ is one more than the ratio of the maximum number of sets in which an element appears over the minimum number of sets in which an element appears.*

Proof. First we find an $(1-1/e)$-approximate solution $\mathcal{S}'$ to the maximum coverage problem with the same universe, profits, sets, costs, and budget [34]. Because the total profit of uniquely covered elements is always at most the total profit of all covered elements, the optimum solution value OPT to the unique coverage problem must be at most the optimum solution value to the maximum coverage problem. Thus the total profit of covered elements in $\mathcal{S}'$ is within an $1-1/e$ factor of an upper bound on OPT. Symbolically, if $p(S)$ denotes the total profit of elements in set S and $\bigcup\mathcal{S}'$ denotes the union $\bigcup_{S\in\mathcal{S}'} S$, then $p(\bigcup\mathcal{S}') \geq (1-1/e)\,\mathrm{OPT}$.

We cluster the elements in $\bigcup\mathcal{S}'$ into $\lg\rho$ groups as follows: an element is in *group* i if it is covered by between 2^i and $2^{i+1}-1$ sets. The group i^* with the most total profit must have at least a $1/\lg\rho$ fraction of $p(\bigcup\mathcal{S}') \geq (1-1/e)\,\mathrm{OPT}$. Now we randomly discard sets from $\mathcal{S}'$, keeping a set with probability $1/2^{i^*}$. We claim that, in expectation, the resulting collection $\mathcal{S}''$ uniquely covers a constant fraction of the elements in group i^*, which is $\Omega(\mathrm{OPT}/\log\rho)$.

Fix an element x in group i^*, and suppose that it was covered d times in $\mathcal{S}'$, $2^{i^*} \leq d \leq 2^{i^*+1}-1$. The probability that x is covered exactly once by $\mathcal{S}''$ is $(d/2^{i^*})(1-1/2^{i^*})^{d-1}$. (There is a factor of d for the choice of which set covers x, a $1/2^{i^*}$ probability that this set is kept, and a $1-1/2^{i^*}$ probability that each of the $d-1$ other sets is discarded.) By our bounds on d, the probability that x is covered exactly once by $\mathcal{S}''$ is at least $(1-1/2^{i^*})^{2^{i^*+1}} \geq 1/e^2$.

The expected total profit of elements covered exactly once by $\mathcal{S}''$ is at least $\sum\{p_x/e^2 \mid x \text{ in group } i^*\}$, which is $1/e^2$ times the total profit of elements in group i^*, which we argued is at least $(1-1/e)\,\mathrm{OPT}/\lg\rho$. Therefore the expected profit of our randomized solution is at least $(1/e^2 - 1/e^3)\,\mathrm{OPT}/\lg\rho = \Omega(\mathrm{OPT}/\log\rho)$.

We can derandomize this algorithm by the standard method of conditional expectation [44]. For each set in $\mathcal{S}'$, we decide whether to keep it in $\mathcal{S}''$ by trying both options, and choosing the option that maximizes the conditional expectation of the total profit of elements in group i^* uniquely covered by $\mathcal{S}''$. The conditional expectations can be computed easily in polynomial time according to the analysis above. ∎

The approximate solution computed by this algorithm is not only within an $\Omega(1/\log m)$ factor of the optimal unique coverage, but also within an $\Omega(1/\log m)$ of the optimal maximum coverage. As a consequence, we also obtain an $\Omega(1/\log m)$-approximation for the more general problem of budgeted low-coverage described in section 2.1.

4.2. Approximation with bounded set size. In this section we consider the unique coverage problem with a bound B on the maximum set size, or more generally, the budgeted unique coverage problem with a bound B on the ratio between the maximum profit of a set and the minimum profit of an element (recall that the profit of a set is the sum of profits of its elements). In both cases we obtain an approximation ratio of $\Omega(1/\log B)$. In particular, $B \leq n$, so this algorithm is an $\Omega(1/\log n)$-approximation.

THEOREM 4.2. *There is an $\Omega(1/\log B)$-approximation algorithm for instances of the budgeted unique coverage problem in which the minimum element profit is* 1 *and the total profit of every set is at most B.*

Proof. As before, we first find an $(1-1/e)$-approximate solution $\mathcal{S}'$ to the maximum coverage problem with the same universe, profits, sets, costs, and budget [34]. As argued in the proof of Theorem 4.1, $p(\bigcup \mathcal{S}') \geq (1-1/e)\,\mathrm{OPT}$, where $p(S)$ denotes the total profit of elements in set S, $\bigcup \mathcal{S}'$ denotes the union $\bigcup_{S\in\mathcal{S}'} S$, and OPT denotes the optimum solution value to the unique coverage problem.

We modify $\mathcal{S}'$ to be minimal by removing any sets that do not uniquely cover any elements. Thus the set of covered elements remains the same, so the same upper bound on OPT holds. Let X be the set of elements covered by exactly one set of $\mathcal{S}'$. Because $\mathcal{S}'$ is minimal, each set must uniquely cover at least one element in X, so $|X| \geq |\mathcal{S}'|$. Because every element has profit at least 1, $p(X) \geq |X| \geq |\mathcal{S}'|$.

If $p(\bigcup \mathcal{S}') \leq 2|\mathcal{S}'| \leq 2p(X)$, then $\mathcal{S}'$ is already an $\Omega(1)$-approximate solution to the budgeted unique coverage problem. If $p(\bigcup \mathcal{S}') > 2|\mathcal{S}'|$, then we claim that the total profit of elements covered at most B times by $\mathcal{S}'$ is at least $p(\bigcup \mathcal{S}')/2$. Otherwise, the elements covered more than B times by $\mathcal{S}'$ would be at least $p(\bigcup \mathcal{S}')/2$, and thus the total profit of the sets would satisfy $\sum_{S\in\mathcal{S}'} p(S) > Bp(\bigcup \mathcal{S}')/2 > B|\mathcal{S}'|$, contradicting that every set in $\mathcal{S}$ (and thus $\mathcal{S}'$) has total profit at most B. Now we apply Theorem 4.1 above to the elements covered at most B times by $\mathcal{S}'$, for which $\rho \leq B$. Thus we obtain an $\Omega(1/\log B)$-approximation for this subproblem, whose optimal solution value is at least $(1-1/e)\,\mathrm{OPT}/2$. ∎

We note that, when every set has cardinality at most $B = 3$ and every element appears in exactly two sets ($\rho = 1$), the unique coverage problem is APX-hard even in this restricted case [46, 2].

Appendix. Randomized rounding for envy-free pricing. In this section we prove the necessary lemma about randomized rounding needed in section 2.2 for the reduction from unique coverage to unlimited-supply single-minded envy-free pricing.

LEMMA A.1. *In the setting of single-minded envy-free pricing, suppose all valuations are* 1. *Then there is a price assignment that uses prices of just* 0 *and* 1 *and whose profit is within a constant factor of optimal.*

Proof. Consider the optimal assignment of prices p_i to items I_i. If any price p_i is larger than 1, we set it to 1 at no cost. Now we round by setting the new price p_i' of item I_i to 1 with probability $\frac{1}{2}p_i$ and to 0 otherwise. We claim that if $u_i = \sum_{I_j\in B_i} p_j < 1$ (i.e., the optimal solution profits u_i from buyer b_i), then the probability that the seller profits 1 from buyer b_i is at least $\frac{1}{2e}u_i$.

The probability that the seller profits 1 from buyer b_i, who desires bundle B_i, is $\sum_{I_j\in B_i} \frac{1}{2}p_j \prod_{I_j\neq I_k\in B_i}(1-\frac{1}{2}p_k)$. This quantity can be rewritten as $\prod_{I_k\in B_i}(1-\frac{1}{2}p_k)\sum_{I_j\in B_i}\frac{1}{2}p_j/(1-\frac{1}{2}p_j)$. Because $\sum_{I_j\in B_i} p_j \leq \frac{1}{2}$, it is easy to show that the quantity is minimized when all of the p_j's, $I_j \in B_i$, are equal. Thus the probability of profit from b_i is at least $(1-\frac{1}{2}u_i/|B_i|)^{|B_i|}\frac{1}{2}u_i/(1-\frac{1}{2}u_i|B_i|)$. Because $1-x \geq e^{-2x}$ for $0 \leq x \leq \frac{1}{2}$, this probability is at least $e^{-u_i}\frac{1}{2}u_i \geq e^{-1}\frac{1}{2}u_i$, as claimed.

Thus the expected total profit in the modified solution is at least $\sum_i \frac{1}{2e}u_i/e$, which is $\frac{1}{2e}$ times the profit of the optimal solution. We can derandomize this algorithm by the standard method of conditional expectation [44]; see the proof of Theorem 4.1. ∎

Acknowledgments. We thank David Abusch-Magder, who mentioned to us a real-world application in wireless networks at Bell Labs, of which unique coverage is

a natural simplification. We also thank David Karger for initial discussions about the problem. The third author also thanks Nikhil Bansal, Venkatesan Guruswami, Jason Hartline, Kamal Jain, and Kunal Talwar for fruitful discussions. Finally, we thank the anonymous referees for their helpful comments.

REFERENCES

[1] G. Aggarwal, T. Feder, R. Motwani, and A. Zhu, *Algorithms for multi-product pricing*, in Proceedings of the 31st International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 3142, Springer-Verlag, Berlin, 2004, pp. 72–83.

[2] P. Alimonti and V. Kann, *Hardness of approximating problems on cubic graphs*, in Proceedings of the 3rd Italian Conference on Algorithms and Complexity, Lecture Notes in Comput. Sci. 1203, Springer-Verlag, Berlin, 1997, pp. 288–298.

[3] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg, *A lower bound for radio broadcast*, J. Comput. System Sci., 43 (1991), pp. 290–298.

[4] C. Ambühl, A. E. F. Clementi, M. D. Ianni, N. Lev-Tov, A. Monti, D. Peleg, G. Rossi, and R. Silvestri, *Efficient algorithms for low-energy bounded-hop broadcast in ad-hoc wireless networks*, in Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2996, Springer-Verlag, Berlin, 2004, pp. 418–427.

[5] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang, *Hardness of the undirected edge-disjoint paths problem with congestion*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, 2005, pp. 226–244.

[6] M. Andrews and L. Zhang, *Hardness of the undirected edge-disjoint paths problem*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 276–283.

[7] A. Archer, C. Papadimitriou, K. Talwar, and É. Tardos, *An approximate truthful mechanism for combinatorial auctions with single parameter agents*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2003, pp. 205–214.

[8] V. Bahl, M. Hajiaghayi, K. Jain, V. S. Mirrokni, L. Qui, and A. Saberi, *Cell Breathing in Wireless LAN: Algorithms and Evaluation*, manuscript, 2005.

[9] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, *Approximation algorithms for deadline-TSP and vehicle routing with time-windows*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 166–174.

[10] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126.

[11] P. Berman and G. Schnitger, *On the complexity of approximating the independent set problem*, Inform. and Comput., 96 (1992), pp. 77–94.

[12] I. Chlamtac and O. Weinstein, *The wave expansion approach to broadcasting in multi-hop radio networks*, IEEE Trans. Commun., 39 (1991), pp. 426–433.

[13] J. Chuzhoy, S. Guha, E. Halperi, S. Khanna, G. Kortsarz, and J. S. Naor, *Asymmetric k-center is* $\log^* n$*-hard to approximate*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 21–27.

[14] X. Deng, C. Papadimitriou, and S. Safra, *On the complexity of equilibria*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 67–71.

[15] N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani, *Market equilibrium via a primal-dual-type algorithm*, in Proceedings of the 43rd Symposium on Foundations of Computer Science, 2002, pp. 389–395.

[16] M. Elkin and G. Kortsarz, *Combinatorial logarithmic approximation algorithm for directed telephone broadcast problem*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 438–447.

[17] M. Elkin and G. Kortsarz, *Polylogarithmic inapproximability of the radio broadcast problem*, in Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, 2004, pp. 105–116.

[18] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[19] U. Feige, *Relations between average case complexity and approximation complexity*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, pp. 534–543.

[20] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.

[21] U. Feige, M. M. Halldórsson, G. Kortsarz, and A. Srinivasan, *Approximating the domatic number*, SIAM J. Comput., 32 (2002), pp. 172–195.

[22] Y. Fukuda, T. Abe, and Y. Oie, *Decentralized access point selection architecture for wireless LANs: Deployability and robustness for wireless LANs*, in Proceedings of the IEEE Vehicular Technology Conference, 2004, pp. 1103–1107.

[23] L. Gasieniec, D. Peleg, and Q. Xin, *Faster communication in known topology radio networks*, in Proceedings of the 24th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, 2005, pp. 129–137.

[24] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.

[25] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[26] F. Gul and E. Stacchetti, *Walrasian equilibrium with gross substitutes*, J. Econom. Theory, 87 (1999), pp. 95–124.

[27] V. Guruswami, J. D. Hartline, A. R. Karlin, D. Kempe, C. Kenyon, and F. McSherry, *On profit-maximizing envy-free pricing*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 2005, pp. 1164–1173.

[28] V. Guruswami and L. Trevisan, *The complexity of making unique choices: Approximating 1-in-k SAT*, in Proceedings of 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, Lecture Notes in Comput. Sci. 3624, Springer-Verlag, Berlin, 2005, pp. 99–110.

[29] J. Håstad, *Some optimal inapproximability results*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 1–10.

[30] D. S. Hochbaum, *Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems*, in Approximation Algorithms for NP-Hard Problems, PWS Publishing, Boston, 1997, pp. 94–143.

[31] G. Judd and P. Steenkiste, *Fixing* 802.11 *access point selection*, in ACM SIGCOMM Poster, August 2002.

[32] S. Khot, *Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique*, in Proceedings of the 45th Annual Symposium on Foundations of Computer Science, 2004, pp. 136–145.

[33] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell, *Optimal inapproximability results for MAX-CUT and other* 2*-variable CSPs?*, SIAM J. Comput., 37 (2007), pp. 319–357.

[34] S. Khuller, A. Moss, and J. S. Naor, *The budgeted maximum coverage problem*, Inform. Process. Lett., 70 (1999), pp. 39–45.

[35] D. R. Kowalski and A. Pelc, *Broadcasting in undirected ad hoc radio networks*, in Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing, 2003, pp. 73–82.

[36] D. R. Kowalski and A. Pelc, *Centralized deterministic broadcasting in undirected multi-hop radio networks*, in Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, 2004, pp. 171–182.

[37] D. R. Kowalski and A. Pelc, *Time of deterministic broadcasting in radio networks with local knowledge*, SIAM J. Comput., 33 (2004), pp. 870–891.

[38] M. Krivelevich, Z. Nutov, M. R. Salavatipour, J. Verstraete, and R. Yuster, *Approximation algorithms and hardness results for cycle packing problems*, ACM Trans. Algorithms, 3 (2007), article 48.

[39] E. Kushilevitz and Y. Mansour, *An* $\Omega(D \log(N/D))$ *lower bound for broadcast in radio networks*, SIAM J. Comput., 27 (1998), pp. 702–712.

[40] D. Lehmann, L. I. O'Callaghan, and Y. Shoham, *Truth revelation in approximately efficient combinatorial auctions*, in Proceedings of the 1st ACM Conference on Electronic Commerce, 1999, pp. 96–102.

[41] N. Lev-Tov and D. Peleg, *Exact algorithms and approximation schemes for base station placement problems*, in Proceedings of the 8th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 2368, Springer-Verlag, Berlin, 2002, pp. 90–99.

[42] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.

[43] A. Moss and Y. Rabani, *Approximation algorithms for constrained for constrained node weighted Steiner tree problems*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, pp. 373–382.

[44] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[45] A. Mu'alem and N. Nisan, *Truthful approximation mechanisms for restricted combinatorial auctions: Extended abstract*, in Proceedings of the 18th National Conference on Artificial Intelligence, 2002, pp. 379–384.

[46] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.

[47] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 475–484.

[48] M. R. Salavatipour and J. Verstraete, *Disjoint cycles: Integrality gap, hardness, and approximation*, in Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization, 2005, pp. 51–65.

[49] L. Walras, *Elements of Pure Economics*, Allen and Unwin, London, 1954.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1484–1503

APPROXIMATION ALGORITHM AND PERFECT SAMPLER FOR CLOSED JACKSON NETWORKS WITH SINGLE SERVERS*

S. KIJIMA† AND T. MATSUI‡

Abstract. In this paper, we propose the first fully polynomial-time randomized approximation scheme (FPRAS) for closed Jackson networks with single servers. Our algorithm is based on the Markov chain Monte Carlo (MCMC) method, and our scheme returns an approximate solution, for which the size of error satisfies a given error rate. We propose two Markov chains: one is for approximate sampling, and the other is for perfect sampling based on the monotone coupling from the past algorithm.

Key words. Markov chain Monte Carlo, Jackson networks, rapidly mixing, path coupling, perfect sampling, coupling from the past, FPRAS

AMS subject classifications. 37A25, 65C40, 65C05, 60K25

DOI. 10.1137/06064980X

1. Introduction. In this paper, we propose the first fully polynomial-time randomized approximation scheme for basic queueing networks, called closed Jackson networks with single servers. Our scheme is based on the Markov chain Monte Carlo (MCMC) method and returns an approximate value of the normalizing constant of the steady-state distribution of the numbers of customers at nodes. The complexity of our algorithm is bounded by a polynomial of n and the logarithm of K where n is the number of nodes and K is the number of customers in a network. Thus our scheme is a polynomial-time approximation scheme. We propose two Markov chains, both of which are rapidly mixing. One is for approximate sampling, while the other is for perfect sampling based on the coupling from the past (CFTP) algorithm.

The Jackson network, proposed by Jackson in 1957 [13], is one of the basic models in queueing network theory and is of considerable importance. The Jackson network consists of a set of nodes, each of which has one or more servers. In the network, customers receive a service from a server at a node according to an exponentially distributed service time, move stochastically to a next node after the service, and wait their turn in a line on a first-come-first-served (FCFS) basis. It is well known that the steady-state distribution of customers in a Jackson network is a product form [13, 11, 10].

We say a network is closed if no customers leave or enter the network. By computing the normalizing constant of the product form solution of a given closed queueing network, we can evaluate useful parameters such as network throughput and rates of utilization of stations [10]. Buzen's algorithm [5], which computes the normalizing constant of closed queueing networks, is well known. However, Buzen's algorithm runs in pseudopolynomial time that depends on K, the number of customers in a closed network, and not on the logarithm of K. To deal with networks where the

*Received by the editors January 13, 2006; accepted for publication (in revised form) May 28, 2008; published electronically September 19, 2008.
http://www.siam.org/journals/sicomp/38-4/64980.html
†Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502, Japan (kijima@kurims.kyoto-u.ac.jp).
‡Department of Information and System Engineering, Faculty of Science and Engineering, Chuo University, Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan (matsui@ise.chuo-u.ac.jp).

number of customers as well as the server capacities are very large, we need a weakly polynomial-time (approximation) algorithm.

The MCMC method is practical for computing the normalizing constant of a distribution. Chen and O'Cinneide [6] proposed a randomized algorithm based on MCMC, but their algorithm runs in weakly polynomial time only in some very special cases. Ozawa [23] proposed a perfect sampler for closed Jackson networks with single servers, and his chain mixes in pseudopolynomial time.

When we construct a randomized approximation algorithm based on MCMC, we need to consider the accuracy of the obtained value. For any given ε and δ with $0 < \varepsilon, \delta < 1$, a fully polynomial-time randomized approximation scheme (FPRAS) provides an algorithm which finds an approximate solution Z satisfying

$$\Pr[|Z - A| \leq \varepsilon A] \geq 1 - \delta$$

where A is the exact solution, and whose running time is bounded by a polynomial of the input size of the instance (the number of nodes n and *logarithm* of the number of customers K), ε^{-1}, and $\ln(\delta^{-1})$ [15].

In many practical situations, each node of a network has a single server. In this paper, we are concerned with a closed Jackson network with single servers. We propose an FPRAS based on MCMC for calculating the normalizing constant. We make use of a Markov chain which has the product form solution of a given closed Jackson network as a unique stationary distribution and show that the chain mixes in $\mathrm{O}(n^2 \ln K)$, where n is the number of nodes and K is the number of customers in the closed Jackson network. Here we note that the chain is not a simulation of customers' movements in a queueing network, but just has a unique stationary distribution which is the same as the product form solution for the given network.

We also propose a second Markov chain and show that this chain is *monotone* and rapidly $\mathrm{O}(n^3 \ln K)$ mixing. An ordinary sampling via Markov chain is an approximate sampler, whereas Propp and Wilson [24] devised the monotone CFTP algorithm which realizes a perfect (exact) sampling from stationary distribution in probabilistically finite time by ingeniously simulating the chain (see also [12, 18, 21]). Our chain provides an efficient perfect sampler based on monotone CFTP. One of the great advantages of utilizing perfect sampling is that we need not be concerned with the error rate ε. Another is that a perfect sampler becomes faster than any approximate sampler based on a Markov chain when we need a sample which accurately follows a particular distribution.

2. Closed Jackson networks. We denote the set of real numbers (nonnegative, positive real numbers) by $\mathbb{R}$ ($\mathbb{R}_+$, $\mathbb{R}_{++}$), and the set of integers (nonnegative, positive integers) by $\mathbb{Z}$ ($\mathbb{Z}_+$, $\mathbb{Z}_{++}$), respectively. A closed Jackson network with single servers is a queueing network model satisfying the following:

(i) The network has $n \in \mathbb{Z}_{++}$ nodes. Each node contains exactly one server; thus at most one customer can receive a service on a node at a time.

(ii) In each node, customers are served one by one on an FCFS basis. The servicing time on node $i \in \{1, \ldots, n\}$ is exponentially distributed with mean $1/\mu_i \in \mathbb{R}_{++}$.

(iii) Once served in node $i \in \{1, \ldots, n\}$, a customer goes to node $j \in \{1, \ldots, n\}$ with probability $W_{ij} \in \mathbb{R}_+$. We assume that the matrix $W = (W_{ij})$ of transition probabilities of customers is irreducible and aperiodic, and thus ergodic.

(iv) No customers leave or enter the network. Thus, there are always $K \in \mathbb{Z}_{++}$ customers in the network.

In queueing network theory, it is well known that a closed Jackson network has a *product form solution* as a steady-state distribution of customers in a network. Let us consider the set of nonnegative integer points

$$\Xi(K) \stackrel{\text{def.}}{=} \left\{ \boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{Z}_+^n \;\middle|\; \sum_{i=1}^n x_i = K \right\}$$

contained in an $n-1$-dimensional simplex. Clearly, a state of customers in the network with K customers is represented by $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \Xi(K)$. We abbreviate $\Xi(K)$ to Ξ when doing so causes no confusion. Since matrix W of the transition probabilities of customers is ergodic, 1 is an eigenvalue and its corresponding eigenvector is unique, excluding a constant factor. Let $\theta \in \mathbb{R}_{++}^n$ be an eigenvector for W corresponding to the eigenvalue 1, i.e., $\theta W = \theta$. The steady-state distribution $J : \Xi \to \mathbb{R}_{++}$ for the closed Jackson network is a product form defined by

$$J(\boldsymbol{x}) = \frac{1}{G(K)} \prod_{i=1}^n \alpha_i^{x_i} \left(\equiv \frac{1}{G(K)} \prod_{i=1}^n \left(\frac{\theta_i}{\mu_i} \right)^{x_i} \right), \tag{2.1}$$

where $\alpha_i \stackrel{\text{def.}}{=} \theta_i/\mu_i$ and $G(K) \stackrel{\text{def.}}{=} \sum_{\boldsymbol{x} \in \Xi(K)} \prod_{i=1}^n \alpha_i^{x_i}$ is the normalizing constant [13].

3. Randomized approximation scheme. In the following we consider a closed Jackson network with n nodes, K customers, and parameters $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{Z}_{++}$ which has the product form solution (2.1) for any $\boldsymbol{x} \in \Xi(K)$.

3.1. Rapidly mixing Markov chain. Now we propose a new Markov chain $\mathcal{M}_{\mathrm{A}}(K)$ with state space $\Xi(K)$. A transition of $\mathcal{M}_{\mathrm{A}}(K)$ from a current state $X \in \Xi(K)$ to a next state X' is defined as follows. First, we choose a pair of distinct indices (nodes) $\{j_1, j_2\} \subseteq \{1, \ldots, n\}$ uniformly at random. Next, put $k = X_{j_1} + X_{j_2}$, and choose $l \in \{0, 1, \ldots, k\}$ with probability

$$\frac{\alpha_{j_1}^l \alpha_{j_2}^{k-l}}{\sum_{s=0}^k \alpha_{j_1}^s \alpha_{j_2}^{k-s}} \quad \left(= \frac{\alpha_{j_1}^l \alpha_{j_2}^{k-l} \prod_{j \notin \{j_1, j_2\}} \alpha_j^{X_j}}{\sum_{s=0}^k \alpha_{j_1}^s \alpha_{j_2}^{k-s} \prod_{j \notin \{j_1, j_2\}} \alpha_j^{X_j}} \right) \tag{3.1}$$

and set

$$X_i' = \begin{cases} l & (\text{for } i = j_1), \\ k - l & (\text{for } i = j_2), \\ X_i & (\text{otherwise}). \end{cases}$$

The Markov chain $\mathcal{M}_{\mathrm{A}}(K)$ is irreducible and aperiodic, so ergodic, and hence has a unique stationary distribution. Also, $\mathcal{M}_{\mathrm{A}}(K)$ satisfies the detailed balance equation, and thus the stationary distribution is the product form solution $J(\boldsymbol{x})$.

Given a pair of probability distributions ν_1 and ν_2 on a finite state space Ω, the *total variation distance* between ν_1 and ν_2 is defined by $\mathrm{d}_{\mathrm{TV}}(\nu_1, \nu_2) \stackrel{\text{def.}}{=} \frac{1}{2} \sum_{x \in \Omega} |\nu_1(x) - \nu_2(x)|$. The *mixing time* of an ergodic Markov chain is defined by

$$\tau(\varepsilon) \stackrel{\text{def.}}{=} \max_{x \in \Xi} \{ \min\{ t \mid \forall s \geq t,\ \mathrm{d}_{\mathrm{TV}}(\pi, P_x^s) \leq \varepsilon \} \} \qquad (0 < \forall \varepsilon < 1),$$

where π is the stationary distribution and P_x^s is the probability distribution of the chain at time period $s \geq 0$ with initial state x at time period 0. In the following, we discuss the mixing time of $\mathcal{M}_{\mathrm{A}}(K)$.

0	$\alpha_i^0\alpha_j^k/A$	$\alpha_i^1\alpha_j^{k-1}/A$	$\cdots$	$\alpha_i^k\alpha_j^0/A$	1
0	$\alpha_i^0\alpha_j^{k+1}/A'$	$\alpha_i^1\alpha_j^k/A'$	$\alpha_i^2\alpha_j^{k-1}/A'$	$\cdots$ $\alpha_i^{k+1}\alpha_j^0/A'$	1

FIG. 3.1. *Alternating inequalities for a pair of indices (i,j) and a nonnegative integer k. In the figure, $A \stackrel{\text{def.}}{=} \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}$ and $A' \stackrel{\text{def.}}{=} \sum_{s=0}^{k+1} \alpha_i^s \alpha_j^{k+1-s}$ are normalizing constants.*

Here, we consider the cumulative distribution function $g_{ij}^k : \{0,1,\dots,k\} \to \mathbb{R}_+$ of the transition probability (3.1) after a pair of indices is chosen, defined by

$$g_{ij}^k(l) \stackrel{\text{def.}}{=} \frac{\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}}{A_{ij}^k} = \begin{cases} \frac{\alpha_i^{l+1} - \alpha_j^{l+1}}{\alpha_i^{k+1} - \alpha_j^{k+1}} \cdot \alpha_j^{k-l} & (\alpha_i \neq \alpha_j), \\ \frac{l}{k+1} & (\alpha_i = \alpha_j) \end{cases}$$

for $l \in \{0,1,\dots,k\}$, where $A_{ij}^k \stackrel{\text{def.}}{=} \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}$ is a normalizing constant. We also define $g_{ij}^k(-1) \stackrel{\text{def.}}{=} 0$ for convenience. We can simulate the Markov chain $\mathcal{M}_{\mathrm{A}}(K)$ efficiently by using the function g_{ij}^k as follows. First, choose a pair $\{i,j\}$ of indices with the probability $2/(n(n-1))$. Next, put $k = X_i + X_j$, generate a uniformly random real number $\Lambda \in [0,1)$, choose a unique integer l satisfying $g_{ij}^k(l-1) \leq \Lambda < g_{ij}^k(l)$, and set $X_i' = l$ and $X_j' = k - l$, keeping the value of the other indices. We can execute a transition of $\mathcal{M}_{\mathrm{A}}$ efficiently by employing an ordinary binary search technique.

The following lemma gives a property of functions g_{ij}^k, which plays a key role in this paper.

LEMMA 3.1. *The function g_{ij}^k satisfies the following* "alternating inequalities"*:*

$$g_{ij}^{k+1}(l) \leq g_{ij}^k(l) \leq g_{ij}^{k+1}(l+1) \quad \forall k \in \{1,\dots,K\},\ \forall l \in \{1,\dots,k\}. \tag{3.2}$$

Figure 3.1 is an illustration of inequalities (3.2) for a fixed k.

Proof. First, we prove the former inequality $g_{ij}^{k+1}(l) \leq g_{ij}^k(l)$ as follows:

$$\begin{aligned} \frac{g_{ij}^k(l)}{g_{ij}^{k+1}(l)} &= \frac{\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}}{A_{ij}^k} \cdot \frac{A_{ij}^{k+1}}{\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k+1-s}} \\ &= \frac{A_{ij}^{k+1}}{\alpha_j A_{ij}^k} = \frac{\sum_{s=0}^{k+1} \alpha_i^s \alpha_j^{k+1-s}}{\alpha_j \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}} = \frac{\sum_{s=0}^{k+1} \alpha_i^s \alpha_j^{k+1-s}}{\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k+1-s}} \geq 1. \end{aligned}$$

Next, we prove the latter inequality $g_{ij}^k(l) \leq g_{ij}^{k+1}(l+1)$ as follows (see, the appendix for details):

$$\begin{aligned} \frac{g_{ij}^{k+1}(l+1)}{g_{ij}^k(l)} &= \frac{\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k+1-s} + \alpha_i^{l+1}\alpha_j^{k-l}\right)}{\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k+1-s} + \alpha_i^{k+1}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right)} \\ &= \frac{\left(\alpha_i^{l+1}\alpha_j^{k-l}\right)^{-1} \alpha_j \left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}}{\left(\alpha_i^{l+1}\alpha_j^{k-l}\right)^{-1} \alpha_j \left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \sum_{s=k-l}^{k} \alpha_i^s \alpha_j^{k-s}} \geq 1. \end{aligned}$$

Thus we obtain the claim. ∎

The above alternating inequalities imply the following.

THEOREM 3.2. *For $0 < \forall\varepsilon < 1$, the mixing time $\tau(\varepsilon)$ of Markov chain $\mathcal{M}_{\mathrm{A}}(K)$ satisfies*

$$\tau(\varepsilon) \leq \frac{n(n-1)}{2} \ln(K\varepsilon^{-1}).$$

Proof. Let $G = (\Xi, \mathcal{E})$ be an undirected simple graph with vertex set Ξ and edge set $\mathcal{E}$ defined as follows. A pair of vertices $\{\boldsymbol{x}, \boldsymbol{y}\}$ is an edge of G if and only if $(1/2)\sum_{i=1}^{n} |x_i - y_i| = 1$. Clearly the graph G is connected. We define the length $l_{\mathrm{A}}(e)$ of every edge $e \in \mathcal{E}$ by $l_{\mathrm{A}}(e) \stackrel{\text{def.}}{=} 1$. For each pair $(\boldsymbol{x}, \boldsymbol{y}) \in \Xi^2$, we define the distance $d_{\mathrm{A}}(\boldsymbol{x}, \boldsymbol{y})$ by the length of the shortest path between $\boldsymbol{x}$ and $\boldsymbol{y}$ on G with respect to l_{A}. Clearly, the diameter of G, defined by $\max_{\boldsymbol{x},\boldsymbol{y}\in\Xi}\{d_{\mathrm{A}}(\boldsymbol{x}, \boldsymbol{y})\}$, is bounded by K.

We define a joint process $(X, Y) \mapsto (X', Y')$ for any pair $\{X, Y\} \in \mathcal{E}$ as follows. Pick a distinct pair of indices $\{i_1, i_2\}$ uniformly at random. Then set $k_X = X_{i_1} + X_{i_2}$ and $k_Y = Y_{i_1} + Y_{i_2}$, generate a uniform random number $\Lambda \in [0, 1)$, choose l_X and l_Y satisfying $g_{i_1 i_2}^{k_X}(l_X - 1) \leq \Lambda < g_{i_1 i_2}^{k_X}(l_X)$ and $g_{i_1 i_2}^{k_Y}(l_Y - 1) \leq \Lambda < g_{i_1 i_2}^{k_Y}(l_Y)$, and set $X'_{i_1} = l_X$, $X'_{i_2} = k_X - l_X$, $Y'_{i_1} = l_Y$, and $Y'_{i_2} = k_Y - l_Y$. Now we show that

$$\forall\{X, Y\} \in \mathcal{E},\ \mathrm{E}[d_{\mathrm{A}}(Y', Y')] \leq \beta \cdot d_{\mathrm{A}}(X, Y),\ \text{where } \beta = 1 - \frac{2}{n(n-1)}.$$

Since $\{X, Y\} \in \mathcal{E}$, there exists a distinct pair of indices $\{j_1, j_2\}$ satisfying $|X_j - Y_j| = 1$ for $j \in \{j_1, j_2\}$, and $|X_j - Y_j| = 0$ for $j \notin \{j_1, j_2\}$. We will consider the following three cases.

Case 1. When neither index j_1 nor j_2 is chosen, i.e., $\{i_1, i_2\} \cap \{j_1, j_2\} = \emptyset$, we put $k = X_{i_1} + X_{i_2}$, and it is easy to see that $\Pr(X'_{i_1} = l) = \Pr(Y'_{i_1} = l)$ for any $l \in \{0, \ldots, k\}$ since $Y_{i_1} + Y_{i_2} = k$. Thus $X'_{i_1} = Y'_{i_1}$ and $X'_{i_2} = Y'_{i_2}$ hold. Hence $d_{\mathrm{A}}(X', Y') = d_{\mathrm{A}}(X, Y)$.

Case 2. When both indices j_1 and j_2 are chosen, i.e., $\{i_1, i_2\} = \{j_1, j_2\}$, in the same way as Case 1, both $X'_{i_1} = Y'_{i_1}$ and $X'_{i_2} = Y'_{i_2}$ hold. Hence $d_{\mathrm{A}}(X', Y') = 0$.

Case 3. When exactly one of j_1 and j_2 is chosen, i.e., $|\{i_1, i_2\} \cap \{j_1, j_2\}| = 1$, without loss of generality, we can assume that $i_1 = j_1$ and that $X_{i_1} = Y_{i_1} + 1$. Let $k + 1 = X_{i_1} + X_{i_2}$. Then $Y_{i_1} + Y_{i_2} = k$ obviously. We parameterize the transition of Markov chain $\mathcal{M}_{\mathrm{A}}(K)$ with a uniformly random number $\Lambda \in [0, 1)$ by using the functions $g_{i_1 i_2}^{k+1}$ and $g_{i_1 i_2}^{k}$. Set $X_{i_1} = l_1$ such that $g_{i_1 i_2}^{k+1}(l_1 - 1) \leq \Lambda < g_{i_1 i_2}^{k+1}(l_1)$ and $Y_{i_1} = l_2$ such that $g_{i_1 i_2}^{k}(l_2 - 1) \leq \Lambda < g_{i_1 i_2}^{k}(l_2)$ for the common random number Λ. From Lemma 3.1, the alternating inequalities hold, and thus we can see that $l_1 = l_2$ or $l_1 = l_2 + 1$. In either case, we can set $[X'_{i_1} = Y'_{i_1} + 1$ and $X'_{i_2} = Y'_{i_2}]$ or $[X'_{i_1} = Y'_{i_1}$ and $X'_{i_2} = Y'_{i_2} + 1]$. Hence $d_{\mathrm{A}}(X', Y') = d_{\mathrm{A}}(X, Y)$.

Considering that Case 2 occurs with probability $2/(n(n-1))$, we obtain that

$$\mathrm{E}[d_{\mathrm{A}}(X', Y')] \leq \left(1 - \frac{2}{n(n-1)}\right) d_{\mathrm{A}}(X, Y).$$

Since the diameter of G is bounded by K, the path coupling theorem (see Theorem 3.3 described below) implies that the mixing time $\tau(\varepsilon)$ of $\mathcal{M}_{\mathrm{A}}(K)$ satisfies

$$\tau(\varepsilon) \leq \frac{n(n-1)}{2} \ln(K\varepsilon^{-1}). \qquad \square$$

The path coupling theorem proposed by Bubley and Dyer [4] is a useful technique for bounding the mixing time (see also [1, 3, 7, 22]).

THEOREM 3.3 (path coupling theorem [4]). *Let $\mathcal{M}$ be a finite ergodic Markov chain with state space Ω. Let $H = (\Omega, \mathcal{F})$ be a connected undirected graph with vertex set Ω and edge set $\mathcal{F} \subseteq \binom{\Omega}{2}$. Let $l : \mathcal{F} \to \mathbb{R}_{++}$ be a positive length defined on the edge set. For any pair of vertices $\{x, y\}$ of H, the distance between x and y, denoted by $d(x, y)$ and/or $d(y, x)$, is the length of a shortest path between x and y, where the length of a path is the sum of the lengths of edges in the path. Suppose that there exists a joint process $(X, Y) \mapsto (X', Y')$ with respect to $\mathcal{M}$ whose marginals are a faithful copy of $\mathcal{M}$ and satisfying*

$$0 < \exists\beta < 1,\ \forall\{X, Y\} \in \mathcal{F},\ \mathrm{E}[d(X', Y')] \le \beta d(X, Y).$$

Then the mixing time $\tau(\varepsilon)$ of the Markov chain $\mathcal{M}$ satisfies $\tau(\varepsilon) \le (1-\beta)^{-1} \ln(\varepsilon^{-1} D/d)$, where $d \stackrel{\text{def.}}{=} \min\{d(x, y) \mid \forall x, \forall y \in \Omega,\ x \ne y\}$ and $D \stackrel{\text{def.}}{=} \max\{d(x, y) \mid \forall x, \forall y \in \Omega\}$.

The above theorem differs from the original theorem in [4] in that the integrality of the edge lengths is not assumed. We dropped the integrality and introduced the minimum distance d. Theorem 3.3 can be proved by slightly modifying the original proof.

3.2. Monte Carlo integration. In this section, we give an FPRAS for calculating the normalizing constant $G(K)$ of product form solution for a closed Jackson network. Our approximation scheme is a standard Jerrum–Sinclair-type recursive algorithm [16, 15] but we need to exercise caution at some points.

Here we suppose that $K \ge 1$. We arrange the indices (nodes in the network) satisfying the following condition.

Condition 1. $\alpha_1 = \max_i \alpha_i$.

We define a set $\Xi'(K) \subset \Xi(K)$ by $\Xi'(K) \stackrel{\text{def.}}{=} \left\{\boldsymbol{x} \in \Xi(K) \mid x_1 \ge \left\lceil \frac{K}{n} \right\rceil\right\}$ and $G'(K) \stackrel{\text{def.}}{=} \sum_{\boldsymbol{x} \in \Xi'(K)} \prod_{i=1}^{n} \alpha_i^{x_i}$. It is not difficult to see that Condition 1 implies $G'(K)/G(K) \ge 1/n$.

Considering that α_i ($\forall i \in \{1, 2, \ldots, n\}$) is independent of K, it is easy to see that

$$\begin{aligned} G'(K) &= \sum_{\boldsymbol{x} \in \Xi'(K)} \left(\alpha_1^{\lceil K/n \rceil} \cdot \left(\prod_{i=2}^{n} \alpha_i^{x_i} \right) \cdot \alpha_1^{x_1 - \lceil K/n \rceil} \right) \\ &= \sum_{\boldsymbol{x} \in \Xi(K - \lceil K/n \rceil)} \left(\alpha_1^{\lceil K/n \rceil} \cdot \left(\prod_{i=2}^{n} \alpha_i^{x_i} \right) \cdot \alpha_1^{x_1} \right) \\ &= \alpha_1^{\lceil K/n \rceil} \cdot G(K - \lceil K/n \rceil). \end{aligned}$$

Thus, we can compute $G(K)$ by

$$G(K) = \frac{G(K)}{G'(K)} \cdot \alpha_1^{\lceil K/n \rceil} \cdot G(K - \lceil K/n \rceil)$$

if we know the value of $G(K)/G'(K)$ and $G(K - \lceil K/n \rceil)$. By applying the above equation recursively, $G(0) = 1$ implies that

$$G(K) = G(0) \prod_{j=1}^{R} \frac{G(K_{j-1})}{G(K_j)} = \prod_{j=1}^{R} \left(\alpha_1^{K_{j-1} - K_j} \cdot \frac{G(K_{j-1})}{G'(K_{j-1})} \right) = \alpha_1^K \prod_{j=0}^{R-1} \frac{G(K_j)}{G'(K_j)},$$

where we define $K_0 \stackrel{\text{def.}}{=} K$, and $K_j \stackrel{\text{def.}}{=} K_{j-1} - \left\lceil \frac{K_{j-1}}{n} \right\rceil$ for $j = 1, 2, \ldots, R$ while $K_j \ge 0$, and let $R \in \mathbb{Z}_{++}$ be the minimum index satisfying $K_R = 0$.

LEMMA 3.4. *The number of recursions R satisfies that $R \leq n \ln K + 1$ for any $K \in \mathbb{Z}_{++}$.*

We give a proof of Lemma 3.4 in the appendix. Since we already have an approximate sampler via the Markov chain $\mathcal{M}_{\mathrm{A}}(K)$, we only need to estimate $G(K_j)/G'(K_j)$ for $j \in \{0, 1, \ldots, R-1\}$ by the Monte Carlo method. The whole algorithm is as follows.

ALGORITHM 1 (randomize approximation scheme with approximate sampler).

Step 1. *Set $j = 1$, $K' = K$.*
While $K' \geq 1$,
do→
Generate Q_{A} samples, each of which is obtained by simulating $\mathcal{M}_{\mathrm{A}}(K')$ for $T_{\mathrm{A}}(K')$ steps.
Let U_j be the number of samples $\boldsymbol{x}$ which satisfy $x_1 \geq K'/n$.
Set $Z_j := (U_j + 1)/(Q_{\mathrm{A}} + 1)$.
Set $K' := K' - \left\lceil \frac{K'}{n} \right\rceil$ and set $j := j + 1$.
←od
Step 2. *Set $Z := \alpha_1^K \prod_j (1/Z_j)$. Output Z.*

Our algorithm generates Q_{A} samples by simulating $\mathcal{M}_{\mathrm{A}}(K_j)$ for $T_{\mathrm{A}}(K_j)$ steps for each sample. By setting $Q_{\mathrm{A}} = 144nR^2\varepsilon^{-2}\ln(2R/\delta)$ and $T_{\mathrm{A}}(K') = \left\lceil \frac{n(n-1)}{2} \ln \frac{6nRK'}{\varepsilon} \right\rceil$, we obtain the following theorem.

THEOREM 3.5. *If we set $Q_{\mathrm{A}} = 144nR^2\varepsilon^{-2}\ln(2R/\delta)$ and $T_{\mathrm{A}}(K') = \left\lceil \frac{n(n-1)}{2} \ln \frac{6nRK'}{\varepsilon} \right\rceil$, then our randomized approximation scheme (Algorithm 1) returns Z satisfying*

$$\Pr\left[|Z - G(K)| \leq \varepsilon G(K)\right] \geq 1 - \delta.$$

In our proof of the above theorem, we need the following modified Chernoff bound [17].

LEMMA 3.6. *Let X_i $(1 \leq i \leq M)$ be independently and identically distributed (i.i.d.) random variables such that $X_i = 1$ with probability p, and $X_i = 0$ with probability $1 - p$. Let $U = \sum_{i=1}^{M} X_i$ and $0 < \lambda < 1$. If $M \geq (4 + 2\sqrt{3})/p\lambda$, then the inequality*

$$\Pr\left[\left|\frac{U+1}{M+1} - p\right| \geq \lambda p\right] \leq 2\mathrm{e}^{-\frac{1}{4}\lambda^2 Mp}$$

holds.

Also we can estimate the error of bias.

THEOREM 3.7. *If we set $Q_{\mathrm{A}} = 144nR^2\varepsilon^{-2}\ln(2R/\delta)$ and $T_{\mathrm{A}}(K') = \left\lceil \frac{n(n-1)}{2} \ln \frac{6nRK'}{\varepsilon} \right\rceil$, then the bias of the expectation of the obtained approximate solution is bounded as follows:*

$$\frac{|\mathrm{E}[Z] - G(K)|}{G(K)} \leq \frac{\varepsilon}{4} + R\mathrm{e}^{-120R^2\varepsilon^{-2}\ln(2R/\delta)} \leq \left(\frac{1}{4} + \frac{1}{10^{36}}\right)\varepsilon.$$

Proofs of the above results are given in the appendix.

4. Perfect sampler.

4.1. Monotone coupling from the past. Here we briefly review CFTP [24]. Suppose that we have an ergodic Markov chain $\mathcal{M}$ with a finite state space Ω and a transition matrix P. The transition rule of the Markov chain $X \mapsto X'$ can be described by a deterministic function $\phi : \Omega \times [0, 1) \to \Omega$, called an *update function*, as

follows. Given a random number Λ uniformly distributed over $[0,1)$, update function ϕ satisfies that $\Pr(\phi(x,\Lambda)=y)=P(x,y)$ for any $x,y\in\Omega$. We can realize the Markov chain by setting $X'=\phi(X,\Lambda)$. Clearly, update functions corresponding to the given transition matrix P are not unique. The result of transitions of the chain from the time t_1 to t_2 ($t_1<t_2$) with a sequence of random numbers $\boldsymbol{\lambda}=(\lambda[t_1],\lambda[t_1+1],\dots,\lambda[t_2-1])\in[0,1)^{t_2-t_1}$ is denoted by $\Phi_{t_1}^{t_2}(x,\boldsymbol{\lambda}):\Omega\times[0,1)^{t_2-t_1}\to\Omega$, where $\Phi_{t_1}^{t_2}(x,\boldsymbol{\lambda})\stackrel{\text{def.}}{=}\phi(\phi(\cdots\phi(x,\lambda[t_1]),\dots,\lambda[t_2-2]),\lambda[t_2-1])$. We say that a sequence $\boldsymbol{\lambda}\in[0,1)^{|T|}$ satisfies the *coalescence condition*, when $\exists y\in\Omega$, $\forall x\in\Omega$, $y=\Phi_T^0(x,\boldsymbol{\lambda})$.

Suppose that there exists a partial order "$\succeq$" on the set of states Ω. A transition rule expressed by a deterministic update function ϕ is called *monotone* (with respect to "$\succeq$") if $\forall\lambda\in[0,1)$, $\forall x,\forall y\in\Omega$, $x\succeq y\Rightarrow\phi(x,\lambda)\succeq\phi(y,\lambda)$. We also say that a chain is *monotone* if the chain has a *monotone* update function. Here we suppose that there exists a unique pair of states $(x_{\max},x_{\min})$ in a partially ordered set $(\Omega,\succeq)$, satisfying $x_{\max}\succeq x\succeq x_{\min}$ $\forall x\in\Omega$.

With these preparations, a standard monotone CFTP algorithm is expressed as follows.

ALGORITHM 2 (monotone CFTP algorithm [24]).

Step 1. *Set the starting time period $T:=-1$ to go back, and set $\boldsymbol{\lambda}$ to be the empty sequence.*

Step 2. *Generate random real numbers $\lambda[T],\lambda[T+1],\dots,\lambda[\lceil T/2\rceil-1]\in[0,1)$, and insert them to the head of $\boldsymbol{\lambda}$ in order, i.e., put $\boldsymbol{\lambda}:=(\lambda[T],\lambda[T+1],\dots,\lambda[-1])$.*

Step 3. *Start two chains from $x_{\max}$ and $x_{\min}$, respectively, at time period T, and run each chain to time period 0 according to the update function ϕ with the sequence of numbers in $\boldsymbol{\lambda}$. (Here we note that every chain uses the common sequence $\boldsymbol{\lambda}$.)*

Step 4. [*Coalescence check.*] *The state obtained at time period 0 is denoted by $\Phi_T^0(x,\boldsymbol{\lambda})$.*

(a) *If $\exists y\in\Omega$, $y=\Phi_T^0(x_{\max},\boldsymbol{\lambda})=\Phi_T^0(x_{\min},\boldsymbol{\lambda})$, then return y.*

(b) *Else, update the starting time period $T:=2T$, and go to Step 2.*

THEOREM 4.1 (monotone CFTP theorem [24]). *Suppose that a Markov chain defined by an update function ϕ is monotone with respect to a partially ordered set of states $(\Omega,\succeq)$, and $\exists x_{\max},\exists x_{\min}\in\Omega$, $\forall x\in\Omega$, $x_{\max}\succeq x\succeq x_{\min}$. Then the monotone CFTP algorithm (Algorithm 2) terminates with probability 1, and the obtained value is a realization of a random variable exactly distributed according to the stationary distribution.*

Theorem 4.1 says that Algorithm 2 is a (probabilistically) finite time algorithm for infinite time simulation.

4.2. Monotone Markov chain. In this section we propose a new Markov chain $\mathcal{M}_{\mathrm{P}}$ for a given Jackson network. The transition rule of $\mathcal{M}_{\mathrm{P}}$ is defined by the following *update function* $\phi:\Xi\times[1,n)\to\Xi$. For a current state $X\in\Xi$, the next state $X'=\phi(X,\lambda)\in\Xi$ with respect to a random number $\lambda\in[1,n)$ is defined by

$$X_i'=\begin{cases} l & (\text{for } i=\lfloor\lambda\rfloor),\\ k-l & (\text{for } i=\lfloor\lambda\rfloor+1),\\ X_i & (\text{otherwise}),\end{cases}$$

where $k=X_{\lfloor\lambda\rfloor}+X_{\lfloor\lambda\rfloor+1}$ and $l\in\{0,1,\dots,k\}$ satisfies

$$g^k_{\lfloor\lambda\rfloor(\lfloor\lambda\rfloor+1)}(l-1)\le\lambda-\lfloor\lambda\rfloor<g^k_{\lfloor\lambda\rfloor(\lfloor\lambda\rfloor+1)}(l).$$

Our chain $\mathcal{M}_{\mathrm{P}}$ is a modification of $\mathcal{M}_{\mathrm{A}}$, obtained by restricting ourselves to choosing only a consecutive pair of indices. Clearly, $\mathcal{M}_{\mathrm{P}}$ is ergodic. The chain has a unique stationary distribution $J(\boldsymbol{x})$ defined in section 2.

In the following, we show the monotonicity of $\mathcal{M}_{\mathrm{P}}$. Here we introduce a partial order "$\succeq$" on Ξ. For any state $\boldsymbol{x} \in \Xi$, we define *cumulative sum vector* $c_{\boldsymbol{x}} = (c_{\boldsymbol{x}}(0), c_{\boldsymbol{x}}(1), \ldots, c_{\boldsymbol{x}}(n)) \in \mathbb{Z}_+^{n+1}$ by

$$c_{\boldsymbol{x}}(i) \stackrel{\text{def.}}{=} \begin{cases} 0 & (\text{for } i = 0), \\ \sum_{j=1}^{i} x_j & (\text{for } i \in \{1, 2, \ldots, n\}). \end{cases}$$

For any pair of states $\boldsymbol{x}, \boldsymbol{y} \in \Xi$, we say $\boldsymbol{x} \succeq \boldsymbol{y}$ if and only if $c_{\boldsymbol{x}} \geq c_{\boldsymbol{y}}$. Next, we define two special states $x_{\max}, x_{\min} \in \Xi(K)$ by $x_{\max} \stackrel{\text{def.}}{=} (K, 0, \ldots, 0)$ and $x_{\min} \stackrel{\text{def.}}{=} (0, \ldots, 0, K)$. Then we can see easily that $\forall \boldsymbol{x} \in \Xi(K)$, $x_{\max} \succeq \boldsymbol{x} \succeq x_{\min}$.

THEOREM 4.2. *Markov chain $\mathcal{M}_{\mathrm{P}}$ is monotone on the partially ordered set $(\Xi(K), \succeq)$, i.e., $\forall \lambda \in [1, n)$, $\forall X, \forall Y \in \Xi(K)$, $X \succeq Y \Rightarrow \phi(X, \lambda) \succeq \phi(Y, \lambda)$.*

Outline of proof (a detailed proof appears in the appendix). We say that a state $X \in \Xi$ *covers* $Y \in \Xi$ (at j), denoted by $X \cdot\!\succ Y$ (or $X \cdot\!\succ_j Y$), when

$$X_i - Y_i = \begin{cases} +1 & (\text{for } i = j), \\ -1 & (\text{for } i = j + 1), \\ 0 & (\text{otherwise}). \end{cases}$$

We show that if a pair of states $X, Y \in \Xi$ satisfies $X \cdot\!\succ_j Y$, then $\forall \lambda \in [1, n)$, $\phi(X, \lambda) \succeq \phi(Y, \lambda)$. We denote $\phi(X, \lambda)$ by X' and $\phi(Y, \lambda)$ by Y' for simplicity. For any index $i \neq \lfloor \lambda \rfloor$, it is easy to see that $c_X(i) = c_{X'}(i)$ and $c_Y(i) = c_{Y'}(i)$, and so $c_{X'}(i) - c_{Y'}(i) = c_X(i) - c_Y(i) \geq 0$ since $X \succeq Y$. We can show that $c_{X'}(\lfloor \lambda \rfloor) \geq c_{Y'}(\lfloor \lambda \rfloor)$ by considering the following three cases:

Case 1. $[\lfloor \lambda \rfloor \neq j - 1$ and $\lfloor \lambda \rfloor \neq j + 1]$.

Case 2. $[\lfloor \lambda \rfloor = j - 1]$.

Case 3. $[\lfloor \lambda \rfloor = j + 1]$.

For any pair of states X, Y satisfying $X \succeq Y$, it is easy to see that there exists a sequence of states $Z_1, Z_2, \ldots, Z_r$ with an appropriate length that satisfies $X = Z_1 \cdot\!\succ Z_2 \cdot\!\succ \cdots \cdot\!\succ Z_r = Y$. Then, by applying the above property repeatedly, we obtain $\phi(X, \lambda) = \phi(Z_1, \lambda) \succeq \phi(Z_2, \lambda) \succeq \cdots \succeq \phi(Z_r, \lambda) = \phi(Y, \lambda)$.

Since $\mathcal{M}_{\mathrm{P}}$ is a monotone chain, we can design a perfect sampler based on monotone CFTP. We could also employ Wilson's read-once algorithm [25] and Fill's interruptible algorithm [8, 9], each of which also gives a perfect sampler.

4.3. Expected running time. Here, we assume a condition which gives the expected polynomial-time monotone CFTP algorithm.

Condition 2. Parameters are arranged in nonincreasing order, i.e., $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$.

THEOREM 4.3. *Under Condition* 2*, the expected running time of our perfect sampler is bounded by* $\mathrm{O}(n^3 \ln K)$*, where n is the number of nodes and K is the number of customers in a closed Jackson network.*

We can show Theorem 4.3 by estimating the expectation of *coalescence time* $T_* \in \mathbb{Z}_{++}$ defined by $T_* \stackrel{\text{def.}}{=} \min\{t > 0 \mid \exists y \in \Xi,\ \forall x \in \Xi,\ y = \Phi_{-t}^0(x, \boldsymbol{\Lambda})\}$. Note that T_* is a random variable.

Outline of proof. Let $G = (\Xi, \mathcal{E})$ be the graph defined in the proof of Theorem 3.2 in section 3. For each edge $e = \{X, Y\} \in \mathcal{E}$, there exists a unique pair of indices $j_1, j_2 \in \{1, 2, \ldots, n\}$ called a *supporting pair* of e satisfying

$$|X_j - Y_j| = \begin{cases} 1 & (\text{for } j \in \{j_1, j_2\}), \\ 0 & (\text{otherwise}). \end{cases}$$

We define the length of $l_{\mathrm{P}}(e)$ of an edge $e = \{X, Y\} \in \mathcal{E}$ by $l_{\mathrm{P}}(e) \stackrel{\text{def.}}{=} (1/(n-1)) \sum_{i=1}^{j^*-1} (n-i)$, where $j^* = \max\{j_1, j_2\} \geq 2$ and $\{j_1, j_2\}$ is the supporting pair of e. Note that $1 \leq \min_{e \in \mathcal{E}} l_{\mathrm{P}}(e) \leq \max_{e \in \mathcal{E}} l_{\mathrm{P}}(e) \leq n/2$. For each pair $X, Y \in \Xi$, we define the distance $d_{\mathrm{P}}(X, Y)$ to be the length of a shortest path between X and Y on G. Clearly, the length between $x_{\max}$ and $x_{\min}$ is bounded by Kn. Here we denote $X' = \phi(X, \lambda)$ and $Y' = \phi(Y, \Lambda)$ with the uniform real random number $\Lambda \in [1, n)$; then Condition 2 implies

$$\mathrm{E}[d_{\mathrm{P}}(X', Y')] \leq \left(1 - \frac{1}{n(n-1)^2}\right) d_{\mathrm{P}}(X, Y)$$

for any pair $\{X, Y\} \in \mathcal{E}$, and we can prove the claim.

A detailed proof appears in the appendix.

5. Concluding remarks. We proposed FPRAS for closed Jackson networks with single servers. Our scheme is based on MCMC, and we made use of two rapidly mixing Markov chains. One is for approximate sampling, while the other is for perfect sampling. Though we omit the details here, we can also construct an FPRAS using the perfect sampler in the same way as we did with the approximate sampler. It is also possible to show that it suffices to take $36nR^2\varepsilon^{-2} \ln(2R/\delta)$ samples in each iteration. In [19], we discussed a class of functions which satisfies alternating inequalities and showed that this class contains all logarithmic separable concave functions.

We plan to extend our FPRAS to closed Jackson networks with multiple servers [20]. We also hope to extend it to closed BCMP networks [2].

Appendix. Proofs.

LEMMA 3.1. *The function g_{ij}^k satisfies the following* "alternating inequalities"*:*

$$g_{ij}^{k+1}(l) \leq g_{ij}^k(l) \leq g_{ij}^{k+1}(l+1) \quad \forall k \in \{1, \ldots, K\},\ \forall l \in \{1, \ldots, k\}.$$

Proof. First, we prove the former inequality $g_{ij}^{k+1}(l) \leq g_{ij}^k(l)$ as follows:

$$\frac{g_{ij}^k(l)}{g_{ij}^{k+1}(l)} = \frac{\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}}{A_{ij}^k} \frac{A_{ij}^{k+1}}{\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k+1-s}} = \frac{A_{ij}^{k+1}}{\alpha_j A_{ij}^k}$$
$$= \frac{\sum_{s=0}^{k+1} \alpha_i^s \alpha_j^{k+1-s}}{\alpha_j \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}} = \frac{\sum_{s=0}^{k+1} \alpha_i^s \alpha_j^{k+1-s}}{\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k+1-s}} \geq 1.$$

Next, we prove the latter inequality $g_{ij}^k(l) \leq g_{ij}^{k+1}(l+1)$ as follows:

$$\frac{g_{ij}^{k+1}(l+1)}{g_{ij}^k(l)} = \frac{A_{ij}^k}{A_{ij}^{k+1}} \frac{\sum_{s=0}^{l+1} \alpha_i^s \alpha_j^{k+1-s}}{\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}} = \frac{\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l+1} \alpha_i^s \alpha_j^{k+1-s}\right)}{\left(\sum_{s=0}^{k+1} \alpha_i^s \alpha_j^{k+1-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right)}$$

$$
\begin{aligned}
&= \frac{\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k+1-s} + \alpha_i^{l+1}\alpha_j^{k-l}\right)}{\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k+1-s} + \alpha_i^{k+1}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right)} \\
&= \frac{\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\alpha_j \sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \alpha_i^{l+1}\alpha_j^{k-l}\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)}{\left(\alpha_j \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \alpha_i^{k+1}\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right)} \\
&= \frac{\left(\alpha_i^{l+1}\alpha_j^{k-l}\right)^{-1}\alpha_j\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}}{\left(\alpha_i^{l+1}\alpha_j^{k-l}\right)^{-1}\alpha_j\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \alpha_i^{k-l}\alpha_j^{l-k}\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}} \\
&= \frac{\left(\alpha_i^{l+1}\alpha_j^{k-l}\right)^{-1}\alpha_j\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}}{\left(\alpha_i^{l+1}\alpha_j^{k-l}\right)^{-1}\alpha_j\left(\sum_{s=0}^{k} \alpha_i^s \alpha_j^{k-s}\right)\left(\sum_{s=0}^{l} \alpha_i^s \alpha_j^{k-s}\right) + \sum_{s=k-l}^{k} \alpha_i^s \alpha_j^{k-s}} \geq 1.
\end{aligned}
$$

Thus we obtain the claim. □

LEMMA 3.4. *The number of recursions R satisfies that $R \leq n \ln K + 1$ for any $K \in \mathbb{Z}_{++}$.*

Proof. If $n = 1$, then $R = 1$; hence we obtain the claim. If $n \geq 2$ and $K = 1, 2$, then $R = 1, 2$, respectively; hence we also obtain the claim. In the following, we consider the case $n \geq 2$ and $K \geq 3$. We define R' by

$$
R' \stackrel{\text{def.}}{=} \min\left\{ r \;\middle|\; K\left(\frac{n-1}{n}\right)^r < 1 \right\}.
$$

Then clearly $R \leq R'$, since $K' - \lceil K'/n \rceil \leq K'(n-1)/n$ for any $K' \in \mathbb{Z}_{++}$. Thus it is enough to show that

$$
K\left(\frac{n-1}{n}\right)^{n \ln K} \leq 1.
$$

Considering $\ln K > 0$,

$$
\left(\frac{n-1}{n}\right)^{n \ln K} = \left(\left(1 - \frac{1}{n}\right)^n\right)^{\ln K} \leq \left(\frac{1}{\mathrm{e}}\right)^{\ln K} = 1/K.
$$

Thus we obtain the claim. □

THEOREM 3.5. *If we set $Q_{\mathrm{A}} = 144nR^2\varepsilon^{-2}\ln(2R/\delta)$ and $T_{\mathrm{A}}(K') = \left\lceil \frac{n(n-1)}{2} \ln \frac{6nRK'}{\varepsilon} \right\rceil$, then our randomized approximation scheme (Algorithm 1) returns Z satisfying*

$$
\Pr\left[|Z - G(K)| \leq \varepsilon G(K)\right] \geq 1 - \delta.
$$

Proof. In the following, we denote $\omega_j \stackrel{\text{def.}}{=} G'(K_j)/G(K_j)$ and $\widehat{\omega}_j \stackrel{\text{def.}}{=} \mathrm{E}[U_j/Q_{\mathrm{A}}]$ for $j \in \{1, 2, \ldots, R\}$, for simplicity.

(i) We show that $1 \leq \forall j \leq R$, the inequality $\widehat{\omega}_j \geq 1/n - \varepsilon/(6nR)$ holds. By the hypothesis of the theorem, $|\omega_j - \widehat{\omega}_j| \leq \varepsilon/(6nR)$. Since $\omega_j \geq 1/n$, we can see that $\widehat{\omega}_j \geq \omega_j - \varepsilon/(6nR) \geq 1/n - \varepsilon/(6nR)$ for $1 \leq j \leq R$.

(ii) We show that $|\omega_j - \widehat{\omega}_j| \leq \frac{\varepsilon\widehat{\omega}_j}{6R-\varepsilon}$. By using the result of (i), we have

$$
|\omega_j - \widehat{\omega}_j| \leq \frac{\varepsilon}{6nR} = \frac{\varepsilon}{6nR} \cdot \frac{1}{\widehat{\omega}_j} \cdot \widehat{\omega}_j \leq \frac{\varepsilon}{6nR} \cdot \frac{\widehat{\omega}_j}{\left(\frac{1}{n} - \frac{\varepsilon}{6nR}\right)} = \frac{\varepsilon}{6R - \varepsilon} \cdot \widehat{\omega}_j.
$$

(iii) We show that $\Pr[|Z_j - \widehat{\omega}_j| > (\varepsilon/(6R-\varepsilon))] \le \delta/R$. By employing the modified Chernoff bound in Lemma 3.6, we have

$$\Pr\left[\left|\tfrac{U+1}{M+1} - p\right| \ge \lambda p\right] \le 2\mathrm{e}^{-\frac{1}{4}\lambda^2 Mp}.$$

By substituting the parameters $\lambda = \varepsilon/(6R-\varepsilon)$ and $p = \widehat{\omega}_j \ge 1/n - \varepsilon/(6nR)$, clearly $Q_{\mathrm{A}} \ge (4+2\sqrt{3})/(p\lambda)$ holds. We put $M = Q_{\mathrm{A}} = 144nR^2\varepsilon^{-2}\ln(2R/\delta)$ and $U = U_j$. Then we obtain that $Z_j = \frac{U_j+1}{Q_{\mathrm{A}}+1}$ satisfies

$$\begin{aligned}\Pr\left[|Z_j - \widehat{\omega}_j| > \frac{\varepsilon}{6R-\varepsilon}\widehat{\omega}_j\right] &\le 2\mathrm{e}^{-\left(\frac{\varepsilon}{6R-\varepsilon}\right)^2 \frac{1}{4} 144nR^2\varepsilon^{-2}\left(\ln\frac{2R}{\delta}\right)\widehat{\omega}_j} \\ &\le 2\mathrm{e}^{-\left(\frac{\varepsilon}{6R-\varepsilon}\right)^2 36nR^2\varepsilon^{-2}\left(\ln\frac{2R}{\delta}\right)\left(\frac{6R-\varepsilon}{6nR}\right)} \le 2\mathrm{e}^{-\left(\frac{6R}{6R-\varepsilon}\right)\left(\ln\frac{2R}{\delta}\right)} \le 2\mathrm{e}^{-\left(\ln\frac{2R}{\delta}\right)} = \frac{\delta}{R}.\end{aligned}$$

(iv) We show that $|(Z_1\cdots Z_R)^{-1} - (\omega_1\cdots\omega_R)^{-1}| \le \varepsilon(\omega_1\cdots\omega_R)^{-1}$ with probability higher than $1-\delta$. By using the result of (ii), we obtain that

$$\frac{6R-2\varepsilon}{6R-\varepsilon}\cdot\widehat{\omega}_j \le \omega_j \le \frac{6R}{6R-\varepsilon}\cdot\widehat{\omega}_j. \tag{A.1}$$

From the result of (iii), we obtain that

$$\frac{6R-\varepsilon}{6R}\cdot Z_j \le \widehat{\omega}_j \le \frac{6R-\varepsilon}{6R-2\varepsilon}\cdot Z_j, \tag{A.2}$$

with probability higher than $1-\delta/R$. By combining (A.1) and (A.2), and considering $Z_j > 0$, the pair of ω_j and Z_j satisfy that

$$\frac{6R-2\varepsilon}{6R} \le \frac{\omega_j}{Z_j} \le \frac{6R}{6R-2\varepsilon} \quad \text{with probability higher than } 1-\delta/R.$$

The above inequality holds for each $1 \le j \le R$ and each Z_j follows i.i.d.; thus with probability higher than $1-\delta$, the inequalities

$$\left(1-\frac{\varepsilon}{3R}\right)^R \le \frac{\omega_1\cdots\omega_R}{Z_1\cdots Z_R} \le \left(\frac{1}{1-\frac{\varepsilon}{3R}}\right)^R \tag{A.3}$$

hold. The right-hand side of inequality (A.3) satisfies

$$\left(\frac{1}{1-\frac{\varepsilon}{3R}}\right)^R = \left(1+\frac{\varepsilon}{3R-\varepsilon}\right)^R \le \left(1+\frac{\varepsilon}{3R-1}\right)^R \le \mathrm{e}^{\frac{R}{3R-1}\varepsilon} \le \mathrm{e}^{\frac{1}{2}\varepsilon} \le 1+\varepsilon.$$

The left-hand side satisfies

$$\left(1-\frac{\varepsilon}{3R}\right)^R = \left(\frac{1}{1-\frac{\varepsilon}{3R}}\right)^{-R} \ge \frac{1}{1+\varepsilon} \ge 1-\varepsilon.$$

Thus, inequality (A.3) is transformed into

$$1-\varepsilon \le \frac{\omega_1\cdots\omega_R}{Z_1\cdots Z_R} \le 1+\varepsilon$$

accordingly. Then we have the result that with probability higher than $1-\delta$,

$$|(Z_1\cdots Z_R)^{-1} - (\omega_1\cdots\omega_R)^{-1}| \le \varepsilon(\omega_1\cdots\omega_R)^{-1}.$$

(v) Since $\alpha_1^K(\omega_1 \cdots \omega_R)^{-1} = G(K)$ and $\alpha_1^K(Z_1 \cdots Z_R)^{-1} = Z$, we obtain the desired result that

$$\Pr\left[|Z - G(K)| \le \varepsilon G(K)\right] \ge 1 - \delta. \qquad \square$$

LEMMA 3.6. *Let X_i $(1 \le i \le M)$ be i.i.d. random variables such that $X_i = 1$ with probability p, and $X_i = 0$ with probability $1 - p$. Let $U = \sum_{i=1}^{M} X_i$ and $0 < \lambda < 1$. If $M \ge (4 + 2\sqrt{3})/p\lambda$, then*

1. $\Pr\left[p - \frac{U+1}{M+1} \ge \lambda p\right] \le \mathrm{e}^{-\frac{1}{2}\lambda^2 Mp}$,
2. $\Pr\left[\frac{U+1}{M+1} - p \ge \lambda p\right] \le \mathrm{e}^{-\frac{1}{4}\lambda^2 Mp}$, *and*
3. $\Pr\left[\left|\frac{U+1}{M+1} - p\right| \ge \lambda p\right] \le 2\mathrm{e}^{-\frac{1}{4}\lambda^2 Mp}$

hold.

Proof. 1. If $p - \frac{U+1}{M+1} \ge \lambda p$, then $p - \frac{U}{M} \ge \lambda p$. This implies that

$$\Pr\left[p - \frac{U+1}{M+1} \ge \lambda p\right] \le \Pr\left[p - \frac{U}{M} \ge \lambda p\right] \le \mathrm{e}^{-\frac{1}{2}\lambda^2 Mp},$$

where the last inequality is obtained by a Chernoff bound.

2. By using a Chernoff bound, the condition $M \ge (4 + 2\sqrt{3})/p\lambda$ implies that

$$\begin{aligned}\Pr\left[\frac{U+1}{M+1} - p \ge \lambda p\right] &\le \Pr\left[\frac{U}{M} + \frac{1}{M} - p \ge \lambda p\right] = \Pr\left[\frac{U}{M} - p \ge \left(\lambda - \frac{1}{Mp}\right)p\right] \\ &\le \mathrm{e}^{-\frac{1}{3}\left(\lambda - \frac{1}{Mp}\right)^2 Mp} = \mathrm{e}^{-\frac{1}{3}\left(1 - \frac{1}{Mp\lambda}\right)^2 \lambda^2 Mp} \le \mathrm{e}^{-\frac{1}{4}\lambda^2 Mp}.\end{aligned}$$

3. By using the result of 1 and 2, clearly we have

$$\begin{aligned}\Pr\left[\left|\frac{U+1}{M+1} - p\right| \ge \lambda p\right] &= \Pr\left[p - \frac{U+1}{M+1} \ge \lambda p\right] + \Pr\left[\frac{U+1}{M+1} - p \ge \lambda p\right] \\ &\le \mathrm{e}^{-\frac{1}{2}\lambda^2 Mp} + \mathrm{e}^{-\frac{1}{4}\lambda^2 Mp} \le 2\mathrm{e}^{-\frac{1}{4}\lambda^2 Mp}. \qquad \square\end{aligned}$$

THEOREM 3.7. *If we set $Q_{\mathrm{A}} = 144nR^2\varepsilon^{-2}\ln(2R/\delta)$ and $T_{\mathrm{A}}(K') = \lceil \frac{n(n-1)}{2}\ln(\frac{6nRT'}{\varepsilon})\rceil$, then the bias of the expectation of the approximate solution Z obtained by Algorithm 1 is bounded as follows:*

$$\frac{|\mathrm{E}[Z] - G(K)|}{G(K)} \le \frac{\varepsilon}{4} + R\mathrm{e}^{-120R^2\varepsilon^{-2}\ln(2R/\delta)} \le \left(\frac{1}{4} + \frac{1}{10^{36}}\right)\varepsilon.$$

Proof. Since $Z_1, \ldots, Z_R$ are independent, the equalities

$$\mathrm{E}[Z] = \alpha_1^K \mathrm{E}\left[\prod_{i=1}^{R} \frac{1}{Z_i}\right] = \alpha_1^K \prod_{i=1}^{R} \mathrm{E}\left[\frac{1}{Z_i}\right]$$

hold. Now, we have

$$\begin{aligned}\mathrm{E}\left[\frac{1}{Z_i}\right] &= \sum_{U_i=0}^{Q_{\mathrm{A}}} \frac{Q_{\mathrm{A}}+1}{U_i+1}\binom{Q_{\mathrm{A}}}{U_i}\widehat{\omega_i}^{U_i}(1-\widehat{\omega_i})^{Q_{\mathrm{A}}-U_i} \\ &= \frac{1}{\widehat{\omega_i}}\sum_{U_i=0}^{Q_{\mathrm{A}}}\binom{Q_{\mathrm{A}}+1}{U_i+1}\widehat{\omega_i}^{U_i+1}(1-\widehat{\omega_i})^{Q_{\mathrm{A}}-U_i} \\ &= \frac{1}{\widehat{\omega_i}}\left\{1 - (1-\widehat{\omega_i})^{Q_{\mathrm{A}}+1}\right\}.\end{aligned}$$

Let $\gamma_i \overset{\text{def.}}{=} (1-\widehat{\omega}_i)^{Q_{\mathrm{A}}+1}$; then the equality

$$\mathrm{E}[Z] = \alpha_1^K \prod_{i=1}^{R} \frac{1}{\widehat{\omega}_i}(1-\gamma_i) \tag{A.4}$$

holds, where we note that $0 \le \gamma_i < 1$. From (A.4), $|G(K) - \mathrm{E}[Z]|$ satisfies

$$\begin{aligned}
|G(K) - \mathrm{E}[Z]| &= \left| \alpha_1^K \prod_{i=1}^{R} \tfrac{1}{\omega_i} - \alpha_1^K \prod_{i=1}^{R} \tfrac{1}{\widehat{\omega}_i}(1-\gamma_i) \right| \\
&= \alpha_1^K \left| \prod_{i=1}^{R} \frac{1}{\omega_i}(1-\gamma_i) + \left(\prod_{i=1}^{R} \frac{1}{\omega_i} \right) \left\{ 1 - \prod_{i=1}^{R}(1-\gamma_i) \right\} - \prod_{i=1}^{R} \frac{1}{\widehat{\omega}_i}(1-\gamma_i) \right| \\
&\le \alpha_1^K \left| \prod_{i=1}^{R} \frac{1}{\omega_i} - \prod_{i=1}^{R} \frac{1}{\widehat{\omega}_i} \right| \prod_{i=1}^{R}(1-\gamma_i) + \alpha_1^K \left(\prod_{i=1}^{R} \frac{1}{\omega_i} \right) \left\{ 1 - \prod_{i=1}^{R}(1-\gamma_i) \right\} \\
&\le \alpha_1^K \left(\prod_{i=1}^{R} \frac{1}{\omega_i} \right) \left| 1 - \prod_{i=1}^{R} \frac{\omega_i}{\widehat{\omega}_i} \right| + \alpha_1^K \left(\prod_{i=1}^{R} \frac{1}{\omega_i} \right) \left\{ 1 - \prod_{i=1}^{R}(1-\gamma_i) \right\} \\
&= G(K) \left| 1 - \prod_{i=1}^{R} \frac{\omega_i}{\widehat{\omega}_i} \right| + G(K) \left\{ 1 - \prod_{i=1}^{R}(1-\gamma_i) \right\}.
\end{aligned}$$

If $1 - \prod_{i=1}^{R}(\omega_i/\widehat{\omega}_i) \le 0$, we have that

$$\begin{aligned}
\left| 1 - \prod_{i=1}^{R} \frac{\omega_i}{\widehat{\omega}_i} \right| &= \prod_{i=1}^{R} \frac{\omega_i}{\widehat{\omega}_i} - 1 \le \left(1 + \frac{\varepsilon}{6R-\varepsilon} \right)^R - 1 = 1 + \sum_{k=1}^{R} \binom{R}{k} \left(\frac{\varepsilon}{6R-\varepsilon} \right)^k - 1 \\
&\le \sum_{k=1}^{R} \left(\frac{R}{6R-\varepsilon} \right)^k \varepsilon^k \le \varepsilon \sum_{k=1}^{R} \left(\frac{1}{5} \right)^k \le \varepsilon \sum_{k=1}^{\infty} \left(\frac{1}{5} \right)^k \le \frac{\varepsilon}{4},
\end{aligned}$$

since $R \ge 1 \ge \varepsilon$. Otherwise, $1 - \prod_{i=1}^{R}(\omega_i/\widehat{\omega}_i) > 0$ implies that

$$\left| 1 - \prod_{i=1}^{R} \frac{\omega_i}{\widehat{\omega}_i} \right| = 1 - \prod_{i=1}^{R} \frac{\omega_i}{\widehat{\omega}_i} \le 1 - \left(1 - \frac{\varepsilon}{6R-\varepsilon} \right)^R \le \left(1 + \frac{\varepsilon}{6R-\varepsilon} \right)^R - 1 \le \frac{\varepsilon}{4}.$$

Thus

$$|G(K) - \mathrm{E}[Z]| \le \frac{\varepsilon}{4} G(K) + G(K) \left\{ 1 - \prod_{i=1}^{R}(1-\gamma_i) \right\}.$$

Since $0 \le \gamma_i < 1$ $(\forall i)$, it is easy to show that $1 - \prod_{i=1}^{R}(1-\gamma_i) \le \sum_{i=1}^{R} \gamma_i$ by induction on R. Accordingly, we have

$$\begin{aligned}
1 - \prod_{i=1}^{R}(1-\gamma_i) &\le \sum_{i=1}^{R} \gamma_i = \sum_{i=1}^{R} (1-\widehat{\omega}_i)^{Q_{\mathrm{A}}+1} \le R \left\{ 1 - \left(\frac{1}{n} - \frac{\varepsilon}{6nR} \right) \right\}^{Q_{\mathrm{A}}} \\
&= R \left\{ 1 - \left(\frac{1 - \frac{\varepsilon}{6R}}{n} \right) \right\}^{Q_{\mathrm{A}}} \le R \left(1 - \frac{5}{6n} \right)^{Q_{\mathrm{A}}} \le R \left(1 - \frac{5}{6n} \right)^{144nR^2\varepsilon^{-2}\ln(2R/\delta)} \\
&\le R \left(\mathrm{e}^{-1} \right)^{\frac{5}{6n} 144nR^2\varepsilon^{-2}\ln(2R/\delta)} = \mathrm{e}^{-120R^2\varepsilon^{-2}\ln(2R/\delta)}.
\end{aligned}$$

Hence the bias is bounded as follows

$$|G(K) - \mathrm{E}[Z]| \le \frac{\varepsilon}{4} G(K) + R\mathrm{e}^{-120R^2\varepsilon^{-2}\ln(2R/\delta)} G(K)$$
$$\le G(K)\left(\frac{\varepsilon}{4} + R\mathrm{e}^{-120R^2\varepsilon^{-2}\ln(2R/\delta)}\right). \qquad \square$$

THEOREM 4.2. *Markov chain $\mathcal{M}_{\mathrm{P}}$ is monotone on the partially ordered set $(\Xi(K), \succeq)$, i.e., $\forall\lambda \in [1,n)$, $\forall X, \forall Y \in \Xi(K)$, $X \succeq Y \Rightarrow \phi(X,\lambda) \succeq \phi(Y,\lambda)$.*

Proof. We say that a state $X \in \Xi$ *covers* $Y \in \Xi$ (at j), denoted by $X \cdot\succ Y$ (or $X \cdot\succ_j Y$), when

$$X_i - Y_i = \begin{cases} +1 & (\text{for } i = j), \\ -1 & (\text{for } i = j+1), \\ 0 & (\text{otherwise}). \end{cases}$$

We show that if a pair of states $X, Y \in \Xi$ satisfies $X \cdot\succ_j Y$, then $\forall\lambda \in [1,n)$, $\phi(X,\lambda) \succeq \phi(Y,\lambda)$. We denote $\phi(X,\lambda)$ by X' and $\phi(Y,\lambda)$ by Y' for simplicity. For any index $i \neq \lfloor\lambda\rfloor$, it is easy to see that $c_X(i) = c_{X'}(i)$ and $c_Y(i) = c_{Y'}(i)$, and so $c_{X'}(i) - c_{Y'}(i) = c_X(i) - c_Y(i) \ge 0$ since $X \succeq Y$. In the following, we show that $c_{X'}(\lfloor\lambda\rfloor) \ge c_{Y'}(\lfloor\lambda\rfloor)$.

Case 1. In the case that $\lfloor\lambda\rfloor \neq j-1$ and $\lfloor\lambda\rfloor \neq j+1$, if we put $k = X_{\lfloor\lambda\rfloor} + X_{\lfloor\lambda\rfloor+1}$, then it is easy to see that $Y_{\lfloor\lambda\rfloor} + Y_{\lfloor\lambda\rfloor+1} = k$. Accordingly, $X'_{\lfloor\lambda\rfloor} = Y'_{\lfloor\lambda\rfloor} = l$, where l satisfies

$$g^k_{\lfloor\lambda\rfloor(\lfloor\lambda\rfloor+1)}(l-1) \le \lambda - \lfloor\lambda\rfloor < g^k_{\lfloor\lambda\rfloor(\lfloor\lambda\rfloor+1)}(l),$$

and hence $c_{X'}(\lfloor\lambda\rfloor) = c_{Y'}(\lfloor\lambda\rfloor)$.

Case 2. Consider the case that $\lfloor\lambda\rfloor = j-1$. Let $k+1 = X_{j-1} + X_j$. Then $Y_{j-1} + Y_j = k$, since $X \cdot\succ_j Y$. From the definition of the cumulative sum vector,

$$\begin{aligned} c_{X'}(\lfloor\lambda\rfloor) - c_{Y'}(\lfloor\lambda\rfloor) &= c_{X'}(j-1) - c_{Y'}(j-1) \\ &= c_{X'}(j-2) + X'_{j-1} - c_{Y'}(j-2) - Y'_{j-1} \\ &= c_X(j-2) + X'_{j-1} - c_Y(j-2) - Y'_{j-1} \\ &= X'_{j-1} - Y'_{j-1}. \end{aligned}$$

Thus, it is enough to show that $X'_{j-1} \ge Y'_{j-1}$. Now suppose that $l \in \{0,1,\ldots,k\}$ satisfies $g^k_{(j-1)j}(l-1) \le \lambda - \lfloor\lambda\rfloor < g^k_{(j-1)j}(l)$ for λ. Then $g^{k+1}_{(j-1)j}(l-1) \le \lambda - \lfloor\lambda\rfloor < g^{k+1}_{(j-1)j}(l+1)$, since the alternating inequalities imply that $g^{k+1}_{(j-1)j}(l-1) \le g^k_{(j-1)j}(l-1) < g^{k+1}_{(j-1)j}(l) \le g^{k+1}_{(j-1)j}(l+1)$. Thus we have that if $Y'_{j-1} = l$, then $X'_{j-1} = l$ or $l+1$. In other words,

$$\begin{pmatrix} X'_{j-1} \\ Y'_{j-1} \end{pmatrix} \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \ldots, \begin{pmatrix} k \\ k \end{pmatrix}, \begin{pmatrix} k+1 \\ k \end{pmatrix} \right\}$$

and $X'_{j-1} \ge Y'_{j-1}$ in all cases. Accordingly, we have that $c_{X'}(\lfloor\lambda\rfloor) \ge c_{Y'}(\lfloor\lambda\rfloor)$.

Case 3. Consider the case that $\lfloor\lambda\rfloor = j+1$. We can show $c_{X'}(\lfloor\lambda\rfloor) \ge c_{Y'}(\lfloor\lambda\rfloor)$ in a similar way to Case 2.

For any pair of states X, Y satisfying $X \succeq Y$, it is easy to see that there exists a sequence of states $Z_1, Z_2, \ldots, Z_r$ with appropriate length satisfying $X = Z_1 \cdot\succ Z_2 \cdot\succ \cdots \cdot\succ Z_r = Y$. Then applying the above claim repeatedly, we obtain that $\phi(X,\lambda) = \phi(Z_1,\lambda) \succeq \phi(Z_2,\lambda) \succeq \cdots \succeq \phi(Z_r,\lambda) = \phi(Y,\lambda)$. $\square$

THEOREM 4.3. *Under Condition* 2*, the expected running time of our perfect sampler is bounded by* $\mathrm{O}(n^3 \ln K)$*, where* n *is the number of nodes and* K *is the number of customers in a closed Jackson network.*

To show the above theorem, we need the following three lemmas.

LEMMA A.1. *Under Condition* 2*, the mixing rate* τ*, defined by* $\tau \stackrel{\text{def.}}{=} \tau(1/\mathrm{e})$*, of our Markov chain* $\mathcal{M}_{\mathrm{P}}$ *satisfies*

$$\tau \le n(n-1)^2(1+\ln Kn).$$

Proof. Let $G = (\Xi, \mathcal{E})$ be an undirected simple graph with vertex set Ξ and edge set $\mathcal{E}$ defined as follows. A pair of vertices $\{X, Y\}$ is an edge if and only if $(1/2)\sum_{i=1}^{n} |X_i - Y_i| = 1$. Clearly, the graph G is connected. For each edge $e = \{X, Y\} \in \mathcal{E}$, there exists a unique pair of indices $j_1, j_2 \in \{1, \dots, n\}$, called a *supporting pair* of e, satisfying

$$|X_i - Y_i| = \begin{cases} 1 & (i \in \{j_1, j_2\}), \\ 0 & (\text{otherwise}). \end{cases}$$

We define the length $l_{\mathrm{P}}(e)$ of an edge $e = \{X, Y\} \in \mathcal{E}$ by $l_{\mathrm{P}}(e) \stackrel{\text{def.}}{=} (1/(n-1)) \sum_{i=1}^{j^*-1} (n-i)$, where $j^* = \max\{j_1, j_2\} \ge 2$ and $\{j_1, j_2\}$ is the supporting pair of e. Note that $1 \le \min_{e \in \mathcal{E}} l_{\mathrm{P}}(e) \le \max_{e \in \mathcal{E}} l_{\mathrm{P}}(e) \le n/2$. For each pair $X, Y \in \Xi$, we define the distance $d(X, Y)$ to be the length of a shortest path between X and Y on G. Clearly, the diameter of G, i.e., $\max_{(X,Y) \in \Xi^2} d(X, Y)$, is bounded by $Kn/2$, since $d(X, Y) \le (n/2)\sum_{i=1}^{n} (1/2)|X_i - Y_i| \le (n/2)K$ for any $(X, Y) \in \Xi^2$. The definition of edge length implies that for any edge $\{X, Y\} \in \mathcal{E}$, $d(X, Y) = l_{\mathrm{P}}(\{X, Y\})$.

We define a joint process $(X, Y) \to (X', Y')$ as $(X, Y) \to (\phi(X, \Lambda), \phi(Y, \Lambda))$ with a uniform real random number $\Lambda \in [1, n)$ and the update function ϕ defined in subsection 3.2. Now we show that

$$\mathrm{E}[d(X', Y')] \le \beta \cdot d(X, Y), \text{ where } \beta = 1 - 1/(n(n-1)^2), \tag{A.5}$$

for any pair $\{X, Y\} \in \mathcal{E}$. In the following, we denote the supporting pair of $\{X, Y\}$ by $\{j_1, j_2\}$. Without loss of generality, we can assume that $j_1 < j_2$ and $X_{j_2} + 1 = Y_{j_2}$.

Case 1. When $\lfloor \Lambda \rfloor = j_2 - 1$, we will show that

$$\mathrm{E}[d(X', Y') \mid \lfloor \Lambda \rfloor = j_2 - 1] \le d(X, Y) - (1/2)(n - j_2 + 1)/(n-1).$$

In the case $j_1 = j_2 - 1$, $X' = Y'$ with conditional probability 1. Hence $d(X', Y') = 0$. In the following, we consider the case $j_1 < j_2 - 1$. Put $k' = X_{j_2-1} + X_{j_2}$ and $k'' = Y_{j_2-1} + Y_{j_2}$. Since $X_{j_2} + 1 = Y_{j_2}$, $k' + 1 = k''$ holds. From the definition of the update function of our Markov chain, we have the following:

$$\begin{aligned} X'_{j_2-1} = l &\Leftrightarrow [g^{k'}_{(j_2-1)j_2}(l-1) \le \Lambda - \lfloor \Lambda \rfloor < g^{k'}_{(j_2-1)j_2}(l)], \\ Y'_{j_2-1} = l &\Leftrightarrow [g^{k'+1}_{(j_2-1)j_2}(l-1) \le \Lambda - \lfloor \Lambda \rfloor < g^{k'+1}_{(j_2-1)j_2}(l)]. \end{aligned}$$

Now, the alternating inequalities

$$\begin{aligned} 0 &< g^{k'+1}_{(j_2-1)j_2}(0) = g^{k'}_{(j_2-1)j_2}(0) \le g^{k'+1}_{(j_2-1)j_2}(1) \le g^{k'}_{(j_2-1)j_2}(1) \le \cdots \\ &\le g^{k'+1}_{(j_2-1)j_2}(k') \le g^{k'}_{(j_2-1)j_2}(k') = g^{k'+1}_{(j_2-1)j_2}(k'+1) = 1 \end{aligned}$$

hold. Thus we have

$$\begin{pmatrix} X'_{j_2-1} \\ Y'_{j_2-1} \end{pmatrix} \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \ldots, \begin{pmatrix} k' \\ k' \end{pmatrix}, \begin{pmatrix} k' \\ k'+1 \end{pmatrix} \right\}.$$

If $X'_{j_2-1} = Y'_{j_2-1}$, the supporting pair of $\{X', Y'\}$ is $\{j_1, j_2\}$ and so $d(X', Y') = d(X, Y)$. If $X'_{j_2-1} \neq Y'_{j_2-1}$, the supporting pair of $\{X', Y'\}$ is $\{j_1, j_2 - 1\}$ and so $d(X', Y') = d(X, Y) - (n - j_2 + 1)/(n-1)$.

Lemma A.2 (proved later) implies that if $\alpha_{j_2-1} \geq \alpha_{j_2}$, then

$$\begin{aligned} &\Pr[X'_{j_2-1} \neq Y'_{j_2-1} \mid \lfloor \Lambda \rfloor = j_2 - 1] - \Pr[X'_{j_2-1} = Y'_{j_2-1} \mid \lfloor \Lambda \rfloor = j_2 - 1] \\ &= \sum_{l=0}^{k'} \left(g^{k'}_{(j_2-1),j_2}(l) - g^{k'+1}_{(j_2-1),j_2}(l) \right) \\ &\quad - \sum_{l=1}^{k'} \left(g^{k'+1}_{(j_2-1),j_2}(l) - g^{k'}_{(j_2-1),j_2}(l-1) \right) - g^{k'+1}_{(j_2-1),j_2}(0) \geq 0. \end{aligned}$$

Hence

$$\begin{aligned} \Pr[X'_{j_2-1} = Y'_{j_2-1} \mid \lfloor \Lambda \rfloor = j_2 - 1] &\leq (1/2), \\ \Pr[X'_{j_2-1} \neq Y'_{j_2-1} \mid \lfloor \Lambda \rfloor = j_2 - 1] &\geq (1/2). \end{aligned}$$

Thus we obtain that

$$\begin{aligned} \mathrm{E}[d(X', Y') | \lfloor \Lambda \rfloor = j_2 - 1] &\leq (1/2) d(X, Y) + (1/2)(d(X, Y) - (n - j_2 + 1)/(n-1)) \\ &= d(X, Y) - (1/2)(n - j_2 + 1)/(n-1). \end{aligned}$$

Case 2. When $\lfloor \Lambda \rfloor = j_2$, we can show that $\mathrm{E}[d(X', Y') | \lfloor \Lambda \rfloor = j_2] \leq d(X, Y) + (1/2)(n - j_2)/(n-1)$ in a similar way to Case 1.

Case 3. When $\lfloor \Lambda \rfloor \neq j_2 - 1$ and $\lfloor \Lambda \rfloor \neq j_2$, it is easy to see that the supporting pair $\{j'_1, j'_2\}$ of $\{X', Y'\}$ satisfies $j_2 = \max\{j'_1, j'_2\}$. Thus $d(X, Y) = d(X', Y')$.

The probability of appearance of Case 1 is equal to $1/(n-1)$, and that of Case 2 is less than or equal to $1/(n-1)$. From the above,

$$\begin{aligned} \mathrm{E}[d(X', Y')] &\leq d(X, Y) - \frac{1}{n-1} \cdot \frac{1}{2} \cdot \frac{n - j_2 + 1}{n-1} + \frac{1}{n-1} \cdot \frac{1}{2} \cdot \frac{n - j_2}{n-1} \\ &= d(X, Y) - \frac{1}{2(n-1)^2} \\ &\leq \left(1 - \frac{1}{2(n-1)^2} \cdot \frac{1}{\max_{\{X,Y\} \in \mathcal{E}} \{d(X, Y)\}} \right) d(X, Y) \\ &= \left(1 - \frac{1}{n(n-1)^2} \right) d(X, Y). \end{aligned}$$

Since the diameter of G is bounded by $Kn/2$, Theorem 3.3 implies that the mixing rate τ satisfies $\tau \leq n(n-1)^2(1 + \ln(Kn/2))$. □

Lemma A.2. *When $\alpha_i \geq \alpha_j > 0$, the inequality*

$$\sum_{l=0}^{k} \left(g^k_{ij}(l) - g^{k+1}_{ij}(l) \right) - \sum_{l=1}^{k} \left(g^{k+1}_{ij}(l) - g^k_{ij}(l-1) \right) - g^{k+1}_{ij}(0) \geq 0$$

holds.

Proof. We can transform the left-hand side as

$$\begin{aligned}
&\sum_{l=0}^{k}\left(g_{ij}^{k}(l)-g_{ij}^{k+1}(l)\right)-\sum_{l=1}^{k}\left(g_{ij}^{k+1}(l)-g_{ij}^{k}(l-1)\right)-g_{ij}^{k+1}(0)\\
&=\sum_{l=0}^{k}\left(g_{ij}^{k}(l)-g_{ij}^{k+1}(l)\right)-\sum_{l=0}^{k-1}\left(g_{ij}^{k+1}(k-l)-g_{ij}^{k}(k-l-1)\right)-g_{ij}^{k+1}(0)\\
&=\sum_{l=0}^{k-1}\left(g_{ij}^{k}(l)-g_{ij}^{k+1}(l)-g_{ij}^{k+1}(k-l)+g_{ij}^{k}(k-l-1)\right)+1-g_{ij}^{k+1}(k)-g_{ij}^{k+1}(0),
\end{aligned}$$

and we can see that

$$\begin{aligned}
1-g_{ij}^{k+1}(k)-g_{ij}^{k+1}(0)&=1-\frac{\sum_{s=0}^{k}\alpha_i^s\alpha_j^{k+1-s}}{\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}-\frac{\sum_{s=0}^{0}\alpha_i^s\alpha_j^{k+1-s}}{\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}\\
&=\frac{\alpha_i^{k+1}}{\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}-\frac{\alpha_j^{k+1}}{\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}\geq 0,
\end{aligned}$$

since $\alpha_i \geq \alpha_j$ (Condition 2). Thus it is sufficient to show that

$$g_{ij}^{k}(l)-g_{ij}^{k+1}(l)-g_{ij}^{k+1}(k-l)+g_{ij}^{k}(k-l-1)\geq 0 \quad \text{for any } l\ (0\leq l\leq k-1).$$

By transforming the left-hand side, we can see that

$$\begin{aligned}
&g_{ij}^{k}(l)-g_{ij}^{k+1}(l)-g_{ij}^{k+1}(k-l)+g_{ij}^{k}(k-l-1)\\
&=g_{ij}^{k}(l)-g_{ij}^{k+1}(l)-\frac{\sum_{s=0}^{k-l}\alpha_i^s\alpha_j^{k+1-s}}{\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}+\frac{\sum_{s=0}^{k-l-1}\alpha_i^s\alpha_j^{k-s}}{\sum_{s=0}^{k}\alpha_i^s\alpha_j^{k-s}}\\
&=g_{ij}^{k}(l)-g_{ij}^{k+1}(l)-\left(1-\frac{\sum_{s=k-l+1}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}{\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}\right)+\left(1-\frac{\sum_{s=k-l}^{k}\alpha_i^s\alpha_j^{k-s}}{\sum_{s=0}^{k}\alpha_i^s\alpha_j^{k-s}}\right)\\
&=\frac{\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k-s}}{A_{ij}^{k}}-\frac{\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k+1-s}}{A_{ij}^{k+1}}+\frac{\sum_{s=k-l+1}^{k+1}\alpha_i^s\alpha_j^{k+1-s}}{A_{ij}^{k+1}}-\frac{\sum_{s=k-l}^{k}\alpha_i^s\alpha_j^{k-s}}{A_{ij}^{k}}\\
&=\frac{\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k-s}}{A_{ij}^{k}}-\frac{\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k+1-s}}{A_{ij}^{k+1}}+\frac{\sum_{s=0}^{l}\alpha_i^{k+1-s}\alpha_j^s}{A_{ij}^{k+1}}-\frac{\sum_{s=0}^{l}\alpha_i^{k-s}\alpha_j^s}{A_{ij}^{k}}\\
&=\left(\frac{1}{A_{ij}^{k}}-\frac{\alpha_j}{A_{ij}^{k+1}}\right)\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k-s}+\left(\frac{\alpha_i}{A_{ij}^{k+1}}-\frac{1}{A_{ij}^{k}}\right)\sum_{s=0}^{l}\alpha_i^{k-s}\alpha_j^s\\
&=\frac{\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k-s}}{A_{ij}^{k}A_{ij}^{k+1}}\left(\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}-\sum_{s=0}^{k}\alpha_i^s\alpha_j^{k+1-s}\right)\\
&\quad+\frac{\sum_{s=0}^{l}\alpha_i^{k-s}\alpha_j^s}{A_{ij}^{k}A_{ij}^{k+1}}\left(\sum_{s=1}^{k+1}\alpha_i^s\alpha_j^{k+1-s}-\sum_{s=0}^{k+1}\alpha_i^s\alpha_j^{k+1-s}\right)\\
&=\frac{1}{A_{ij}^{k}A_{ij}^{k+1}}\left(\alpha_i^{k+1}\sum_{s=0}^{l}\alpha_i^s\alpha_j^{k-s}-\alpha_j^{k+1}\sum_{s=0}^{l}\alpha_i^{k-s}\alpha_j^s\right)
\end{aligned}$$

$$= \frac{1}{A_{ij}^k A_{ij}^{k+1}} \sum_{s=0}^{l} \left(\alpha_i^{k+1+s} \alpha_j^{k-s} - \alpha_i^{k-s} \alpha_j^{k+1+s} \right)$$
$$= \frac{1}{A_{ij}^k A_{ij}^{k+1}} \sum_{s=0}^{l} \left(\alpha_i^{k-s} \alpha_j^{k-s} \left(\alpha_i^{2s+1} - \alpha_j^{2s+1} \right) \right) \geq 0,$$

since $\alpha_i \geq \alpha_j$ (Condition 2). Thus we obtain the claim. ∎

Next we estimate the expectation of the coalescence time of $\mathcal{M}_{\mathrm{P}}$.

LEMMA A.3. *Under Condition* 2*, the coalescence time* T_* *of* $\mathcal{M}_{\mathrm{P}}$ *satisfies* $\mathrm{E}[T_*] = \mathrm{O}(n^3 \ln Kn)$.

Proof. Let $G = (\Xi, \mathcal{E})$ be the undirected graph and $d(X, Y)$ $\forall X, \forall Y \in \Xi$ be the metric on G, both of which are defined in the proof of Lemma A.1. We define $D \stackrel{\text{def.}}{=} d(x_{\max}, x_{\min})$ and $\tau_0 \stackrel{\text{def.}}{=} n(n-1)^2(1 + \ln D)$. By using inequality (A.5) obtained in the proof of Lemma A.1, we have

$$\begin{aligned}
\Pr[T_* > \tau_0] &= \Pr\left[\Phi_{-\tau_0}^0(x_{\max}, \mathbf{\Lambda}) \neq \Phi_{-\tau_0}^0(x_{\min}, \mathbf{\Lambda})\right] = \Pr\left[\Phi_0^{\tau_0}(x_{\max}, \mathbf{\Lambda}) \neq \Phi_0^{\tau_0}(x_{\min}, \mathbf{\Lambda})\right] \\
&\leq \sum_{(X,Y) \in \Xi^2} d(X, Y) \Pr\left[X = \Phi_0^{\tau_0}(x_{\max}, \mathbf{\Lambda}), Y = \Phi_0^{\tau_0}(x_{\min}, \mathbf{\Lambda})\right] \\
&= \mathrm{E}\left[d\left(\Phi_0^{\tau_0}(x_{\max}, \mathbf{\Lambda}), \Phi_0^{\tau_0}(x_{\min}, \mathbf{\Lambda})\right)\right] \leq \left(1 - \frac{1}{n(n-1)^2}\right)^{\tau_0} d(x_{\max}, x_{\min}) \\
&= \left(1 - \frac{1}{n(n-1)^2}\right)^{n(n-1)^2(1+\ln D)} D \leq \mathrm{e}^{-1} \mathrm{e}^{-\ln D} D = \frac{1}{\mathrm{e}}.
\end{aligned}$$

By the *submultiplicativity* of coalescence time (see [24]), for any $k \in \mathbb{Z}_+$, the inequality $\Pr[T_* > k\tau_0] \leq (\Pr[T_* > \tau_0])^k \leq (1/\mathrm{e})^k$ holds. Thus

$$\begin{aligned}
\mathrm{E}[T_*] &= \sum_{t=0}^{\infty} t \Pr[T_* = t] \leq \tau_0 + \tau_0 \Pr[T_* > \tau_0] + \tau_0 \Pr[T_* > 2\tau_0] + \cdots \\
&\leq \tau_0 + \tau_0/\mathrm{e} + \tau_0/\mathrm{e}^2 + \cdots = \tau_0/(1 - 1/\mathrm{e}) \leq 2\tau_0.
\end{aligned}$$

Clearly $D \leq Kn$. Then we obtain the result that $\mathrm{E}[T_*] = \mathrm{O}(n^3 \ln Kn)$. ∎

Proof of Theorem 4.3. Let T_* be the coalescence time of our chain. Clearly T_* is a random variable. Put $m = \lceil \log_2 T_* \rceil$. Algorithm 2 terminates when we set the starting time period $T = -2^m$ to the $(m+1)$st iteration. Then the total number of simulated transitions is bounded by $2(2^0 + 2^1 + 2^2 + \cdots + 2^K) < 2 \cdot 2 \cdot 2^m \leq 8T_*$, since we need to execute two chains from both $x_{\max}$ and $x_{\min}$. Thus the expectation of total number of transitions of $\mathcal{M}$ is bounded by $\mathrm{O}(\mathrm{E}[8T_*]) = \mathrm{O}(n^3 \ln(Kn))$. ∎

REFERENCES

[1] D. ALDOUS, *Random walks on finite groups and rapidly mixing Markov chains*, in Séminarie de Probabilitiés XVII 1981/1982, D. Dold and B. Eckmann, eds., Lecture Notes in Math. 986, Springer-Verlag, New York, 1983, pp. 243–297.

[2] F. BASKETT, K. M. CHANDY, R. R. MUNTZ, AND F. G. PALACIOS, *Open, closed and mixed networks of queues with different classes of customers*, J. ACM, 22 (1975), pp. 248–260.

[3] R. BUBLEY, *Randomized Algorithms: Approximation, Generation, and Counting*, Springer-Verlag, New York, 2001.

[4] R. BUBLEY AND M. DYER, *Path coupling: A technique for proving rapid mixing in Markov chains*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1997, pp. 223–231.

[5] J. P. Buzen, *Computational algorithms for closed queueing networks with exponential servers*, Comm. ACM, 16 (1973), pp. 527–531.
[6] W. Chen and C. A. O'Cinneide, *Towards a polynomial-time randomized algorithm for closed product-form networks*, ACM Trans. Modeling Comput. Simul., 8 (1998), pp. 227–253.
[7] M. Dyer and C. Greenhill, *Polynomial-time counting and sampling of two-rowed contingency tables*, Theoret. Comput. Sci., 246 (2000), pp. 265–278.
[8] J. Fill, *An interruptible algorithm for perfect sampling via Markov chains*, Ann. Appl. Probab., 8 (1998), pp. 131–162.
[9] J. Fill, M. Machida, D. Murdoch, and J. Rosenthal, *Extension of Fill's perfect rejection sampling algorithm to general chains*, Random Structures Algorithms, 17 (2000), pp. 290–316.
[10] E. Gelenbe and G. Pujolle, *Introduction to Queueing Networks*, 2nd ed., John Wiley & Sons, New York, 1998.
[11] W. J. Gordon and G. F. Newell, *Closed queueing systems with exponential servers*, Oper. Res., 15 (1967), pp. 254–265.
[12] O. Häggström, *Finite Markov Chains and Algorithmic Application*, London Math. Soc. Stud. Texts 52, Cambridge University Press, Cambridge, UK, 2002.
[13] J. R. Jackson, *Networks of waiting lines*, J. Oper. Res. Soc. Amer., 5 (1957), pp. 518–521.
[14] J. R. Jackson, *Jobshop-like queueing systems*, Management Sci., 10 (1963), pp. 131–142.
[15] M. Jerrum, *Counting, Sampling and Integrating: Algorithms and Complexity*, Lectures in Math., ETH Zürich, Birkhäuser Verlag, Basel, 2003.
[16] M. Jerrum and A. Sinclair, *The Markov chain Monte Carlo method: An approach to approximate counting and integration*, in Approximation Algorithm for NP-hard Problems, D. Hochbaum, ed., PWS, Boston, MA, 1996, pp. 482–520.
[17] S. Kijima and T. Matsui, *Approximate counting scheme for $m \times n$ contingency tables*, IEICE Trans. Inform. Systems, E87-D (2004), pp. 308–314.
[18] S. Kijima and T. Matsui, *Polynomial time perfect sampling algorithm for two-rowed contingency tables*, Random Structures Algorithms, 29 (2006), pp. 243–256.
[19] S. Kijima and T. Matsui, *Rapidly mixing chain and perfect sampler for logarithmic separable concave distributions on simplex*, in 2005 International Conference on Analysis of Algorithms, Discrete Math. Theor. Comput. Sci. Proc. AD, Assoc. Discrete Math. Theor. Comput. Sci., Nancy, France, 2005, pp. 369–380.
[20] S. Kijima and T. Matsui, *Randomized Approximation Scheme and Perfect Sampler for Closed Jackson Networks with Multiple Servers*, Mathematical Engineering Technical Report 2006-34, University of Tokyo, Tokyo, Japan, 2006. Available online at http://www.keisu.t.u-tokyo.ac.jp/Research/techrep.0.html.
[21] T. Matsui and S. Kijima, *Polynomial time perfect sampler for discretized Dirichlet distribution*, in The Grammar of Technology Development, H. Tsubaki, K. Nishina, and S. Yamada, eds., Springer Tokyo, Tokyo, 2008, pp. 179–199.
[22] T. Matsui, M. Motoki, and N. Kamatani, *Polynomial time approximate sampler for discretized Dirichlet distribution*, in Algorithms and Computation, Lecture Notes in Comput. Sci. 2906, Springer-Verlag, Berlin, 2003, pp. 676–685.
[23] T. Ozawa, *Perfect simulation of a closed Jackson network*, in Proceedings of the ORSJ Queueing Symposium, Hikone, Japan, 2004.
[24] J. Propp and D. Wilson, *Exact sampling with coupled Markov chains and applications to statistical mechanics*, Random Structures Algorithms, 9 (1996), pp. 223–252.
[25] D. Wilson, *How to couple from the past using a read-once source of randomness*, Random Structures Algorithms, 16 (2000), pp. 85–113.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1504–1518

APPROXIMATION ALGORITHMS FOR BICLUSTERING PROBLEMS*

LUSHENG WANG†, YU LIN‡, AND XIAOWEN LIU†

Abstract. One of the main goals in the analysis of microarray data is to identify groups of genes and groups of experimental conditions (including environments, individuals, and tissues) that exhibit similar expression patterns. This is the so-called biclustering problem. In this paper, we consider two variations of the biclustering problem: the *consensus submatrix problem* and the *bottleneck submatrix problem*. The input of the problems contains an $m \times n$ matrix A and integers l and k. The *consensus submatrix problem* is to find an $l \times k$ submatrix with $l < m$ and $k < n$ and a consensus vector such that the sum of distances between the rows in the submatrix and the consensus vector is minimized. The *bottleneck submatrix problem* is to find an $l \times k$ submatrix with $l < m$ and $k < n$, an integer d and a center vector such that the distance between every row in the submatrix and the vector is at most d and d is minimized. We show that both problems are NP-hard and give randomized approximation algorithms for special cases of the two problems. Using standard techniques, we can derandomize the algorithms to get polynomial time approximation schemes for the two problems. To the best of our knowledge, this is the first time that approximation algorithms with guaranteed ratios are presented for microarray data analysis.

Key words. approximation algorithms, computational biology, microarray data analysis, genes

AMS subject classifications. 68W25, 90C05, 90C27, 92B05

DOI. 10.1137/060664112

1. Introduction.

In the past few years, microarray techniques have been widely used in biological research. Microarray techniques have helped to illuminate mechanisms of diseases, identify disease subphenotypes, predict disease progression, assign functions to previously unannotated genes, group genes into functional pathways, and predict activities of new compounds [1]. Microarray data analysis is an important problem in computational biology [2]. For this large-scale data, classifying genes into different groups under certain conditions is a first step to gain more sophisticated knowledge of different biological pathways or functions. Several clustering or classification techniques such as k-means [3, 4], self-organizing maps [5, 6], hierarchical clustering [7, 8, 9], principal component analysis, and singular value decomposition [10, 11, 12] have been extensively applied to identify groups of similarly expressed genes and conditions from gene expression data.

It is known that many activation patterns are common to a group of genes only under specific experimental conditions. We should expect subsets of genes to be coregulated and coexpressed only under certain experimental conditions, but to behave almost independently under other conditions, according to our general understanding of cellular processes [22, 23, 24]. The fact is that we need to discover local patterns in the microarray matrix. The basic model for biclustering is as follows: given an

*Received by the editors July 3, 2006; accepted for publication (in revised form) August 16, 2007; published electronically September 25, 2008. This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project CityU 120905).

http://www.siam.org/journals/sicomp/38-4/66411.html

†Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (lwang@cs.cityu.edu.hk, liuxw@cs.cityu.edu.hk).

‡Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, and Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China (linyu@cs.cityu.edu.hk).

$m \times n$ matrix A, where each element $a_{i,j} \in \{0, 1\}$, the problem here is to find an $l \times k$ submatrix with all elements identical to 1 such that $l \times k$ is maximized.

Let $\Sigma = \{\pi_1, \pi_2, \ldots, \pi_{|\Sigma|}\}$ be a fixed size alphabet of symbols. A vector over Σ is a sequence of symbols in Σ. Let A be an $m \times n$ matrix, where each row corresponds to a gene and each column corresponds to a condition. Each element $a_{i,j}$ in A represents the expression level of gene i under condition j. Such a matrix A is defined by its set of m rows, $X = \{x_1, x_2, \ldots, x_m\}$, and its set of n columns, $Y = \{y_1, y_2, \ldots, y_n\}$. Let $P = \{p_1, \ldots, p_l\}$ be a subset of $\{1, 2, \ldots, m\}$ indicating rows in X and $Q = \{q_1, \ldots, q_k\}$ be a subset of $\{1, 2, \ldots, n\}$ indicating columns in Y. The $l \times k$ submatrix $A_{P,Q}$ induced by the pair (P, Q) contains the elements $a_{i,j}$, where $i \in P$ and $j \in Q$. We treat each row in the matrix or submatrix as a vector over Σ. Define $x_i|^Q = a_{i,q_1} a_{i,q_2} \ldots a_{i,q_k}$. Let p and p' be two vectors of the same length over Σ. We use $d(p, p')$ to denote the number of mismatches between the two vectors. Throughout this paper, we will study the following two problems:

1. *The consensus submatrix problem*: Given an $m \times n$ matrix A, and integers l and k, find a subset P of l rows, a subset Q of k columns in matrix A, and a consensus vector z of length k such that the consensus score $\sum_{i=1}^{l} d(x_{p_i}|^Q, z)$ is minimized.
2. *The bottleneck submatrix problem*: Given an $m \times n$ matrix A, and integers l and k, find a subset P of l rows, a subset Q of k, columns in matrix A, a center vector z of length k, and an integer d such that for every $p_i \in P$, $d(x_{p_i}|^Q, z) \leq d$ and the bottleneck score d is minimized.

In some applications, the number of conditions/samples ranges from 50 to 550 and the number of genes is about a few thousand. The interesting submatrices (biclusters) may contain tens of conditions/samples. In this case, we have $k = \Omega(n)$. In practice, there are errors in the microarray data. In [22, 29], the input matrix contains 12.3% missing values (see Data Preparation in [22]). In the $l \times k$ submatrix, if we assume that 12.3% values are missing, then the total consensus score $\sum_{i=1}^{l} d(x_{p_i}|^Q, z)$ is $\Omega(lk)$ and the bottleneck score d is $\Omega(k)$. Moreover, the expression levels of coregulated and coexpressed genes in the submatrices are not 100% numerically identical. Therefore, in some cases we can assume that for the consensus submatrix problem $\sum_{i=1}^{l} d(x_{p_i}|^Q, z) = \Omega(lk)$ and for the bottleneck submatrix problem $d = \Omega(k)$.

Throughout this paper, we assume that for the consensus submatrix problem $\sum_{i=1}^{l} d(x_{p_i}|^Q, z) = \Omega(lk)$ and for the bottleneck submatrix problem $d = \Omega(k)$. Moreover, we further assume that $k = \Omega(n)$.

1.1. Real examples. Here we present a few real examples to show that our assumptions on d and k are reasonable in some real applications.

Example 1. The first paper using the biclustering approach for microarray data analysis is [22]. In this paper, the authors analyze the human data originated from [29]. There are $4,026$ genes and 96 conditions with 12.3% missing values in the input matrix. (For more on microarray data errors, see [13, 14, 15, 16, 17, 18, 19, 20].) For the interesting submatrices, the number of conditions ranges from 13 to 96 (most of them > 20). (See the caption of Figure 5 in [22].) Here the assumption that $k = \Omega(n)$ does hold. Moreover, with 12.3% missing values (thus $\sum_{i=1}^{l} d(x_{p_i}|^Q, z) = \Omega(lk)$ and $d = \Omega(k)$), they can still find some biologically interesting submatrices.

Example 2. Koyutürk, Szpankowski, and Grama study a dataset containing 84 samples for human breast cancer [28]. They obtain a biologically interesting submatrix containing 62 samples and 141 genes. Figure 1 shows the submatrix after binary quantization. From Figure 1, we can see that in the gray (red in the color version)

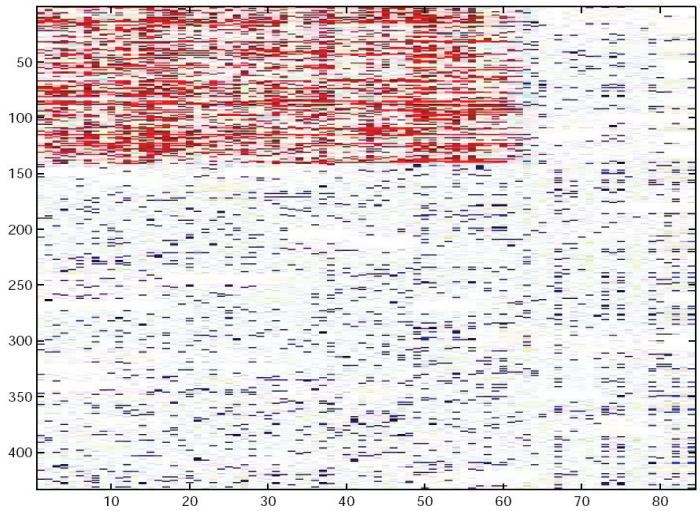


FIG. 1. *The interesting submatrix originally from* [*M. Koyutürk, W. Szpankowski, and A. Grama, Biclustering gene-feature matrices for statistically significant dense patterns, in Proceedings of the* 2004 *Annual IEEE Bioinformatics Conference,* 2004*, pp.* 480–484]. ©2004 *IEEE.*

submatrix, there are still quite a lot of dark and white (blue and white in the color version) dots indicating that the expression levels of genes are not completely identical. (This may be caused by the performance of genes or by data errors.) Based on Figure 1, it is easy to see that there are at least 10% errors. Again, this example shows that our assumptions on d and k are reasonable in some applications.

1.2. Relations to previous work. The basic model for biclustering is to find a submatrix $A_{P,Q}$ with all elements identical to a constant value [23]:

$$a_{i,j} = \mu \quad \text{for all} \quad i \in P, j \in Q.$$

If the submatrix is error free, both the consensus score and the bottleneck score are clearly 0 for the new problems that we propose in the paper.

In practice, it is interesting to find submatrices such that all elements in a row have the same constant value [21, 24]. That is,

$$a_{i,j} = c_i \text{ for } \quad j \in Q.$$

In this case, all the columns in the submatrix are identical. Again, it is clear that both the consensus score and the bottleneck score are 0 if the submatrix is error-free.

For the additive model [22, 23], we are interested in submatrices satisfying

$$a_{i,j} = a_{i',j} + c(i,i') \quad \text{for all} \quad i,i' \in P, \quad j \in Q. \tag{1}$$

That is, for two elements $a_{i,j}$ and $a_{i',j}$ in row i and row i', the difference is a constant $c(i',i)$.

Now we will show that our model can also handle the additive model. Let r be a row in the error-free submatrix. We construct a new matrix A' as follows:

$$a'_{i,j} = a_{i,j} - a_{r,j} \quad \text{for all} \quad i \in X, \quad j \in Y.$$

Then, the error-free submatrix is converted into a new submatrix $A'_{P,Q}$ with the element

$$\begin{aligned} a'_{i,j} &= a_{i,j} - a_{r,j} \\ &= c(i,r) \quad \text{for all} \;\; i \in P, \;\; j \in Q. \end{aligned}$$

That is, in the resulting submatrix, all elements in a row have the same value. Thus, the additive model degenerates to the second case. Therefore, our models can also handle the additive model by trying all rows in A as row r.

Cheng and Church were the first to use the biclustering approach for microarray data analysis [22]. They introduced the *mean squared residue score* H to measure the coherence of the rows and columns in the submatrix.

$$H(P,Q) = \frac{1}{|P||Q|} \sum_{i \in P, j \in Q} (a_{i,j} - a_{i,Q} - a_{P,j} + a_{P,Q})^2,$$

where

$$a_{i,Q} = \frac{1}{|Q|} \sum_{j \in Q} a_{i,j}, \quad a_{P,j} = \frac{1}{|P|} \sum_{i \in P} a_{i,j}, \quad \text{and} \quad a_{P,Q} = \frac{1}{|P||Q|} \sum_{i \in P, j \in Q} (a_{i,j}).$$

Clearly, the H score is 0 for the first two cases if the submatrix is error free. We can show that for the additive model, the H score is also 0 if the submatrix is error free.

In this paper, we design randomized approximation algorithms for both problems. We have a new idea to randomly select $\Theta(\log m)$ columns in the optimal set of columns $Q_{opt} \subseteq Y$ when Q_{opt} is not known. For the bottleneck submatrix problem, we use linear programming and randomized rounding to successfully select a good approximation Q of Q_{opt} and set the letters for the center vector at the columns in Q. Using standard techniques, we can derandomize the randomized algorithms to get polynomial time approximation schemes (PTAS) for the two problems. To the best of our knowledge, this is the first time that approximation algorithms with guaranteed ratios have been presented for microarray analysis.

The paper is organized as follows. In section 2, we prove that both problems are NP-hard. In section 3, we give the algorithm for the consensus submatrix problem. The algorithm for the bottleneck submatrix problem is given in section 4.

2. NP-hardness result. In this section, we will show that both the consensus submatrix problem and the bottleneck submatrix problem are NP-hard. The reduction is from the maximum edge biclique problem. The maximum edge biclique problem was proved to be NP-hard in [25]. A biclique is a complete bipartite subgraph, where every vertex of the first set is connected to every vertex of the second set.

The maximum edge biclique problem. Given a bipartite graph $G = (V_1 \cup V_2, E)$ and a positive integer t, does G contain a biclique with t edges?

THEOREM 1. *The consensus submatrix problem and the bottleneck submatrix problem are NP-hard.*

Proof. We use a reduction from the maximum edge biclique problem. Given a graph $G = (V_1 \cup V_2, E)$ and a positive integer t, where $|V_1| = m$, $|V_2| = n$, we construct an $m \times n$ matrix $A = \{a_{i,j}\}$ as follows: $a_{i,j} = 1$ if and only if $(v_1(i), v_2(j)) \in E$, and otherwise $a_{i,j} = 0$. After constructing A, we get a new matrix A' of size $2m \times n$ by

adding m rows, each containing n 1's, to A. Thus, for an element $a'_{i,j}$ in A', $a'_{i,j} = a_{i,j}$ if $1 \leq i \leq m$. Otherwise, $a'_{i,j} = 1$. (Here we need A' to ensure that every element in the consensus/center vector is 1.)

The input of both the consensus submatrix and the bottleneck submatrix problems contains the matrix A' and two integers $l+m$ and k with $t = l \times k$. The number of pairs for l and k is at most t^2.

It is easy to see that there exists an $l \times k$ biclique for the maximum edge biclique problem if and only if there is a size $l \times k$ submatrix of A in which every element is 1.

Now, we want to show that there is an $l \times k$ submatrix of A such that every element is 1 if and only if there is an $(l+m) \times k$ submatrix of A' such that every element is 1.

(if) Given an $l \times k$ submatrix of A such that every element is 1, we can obtain an $(l+m) \times k$ submatrix of A' such that every element is 1 by adding the m newly added rows with every element being 1 in A'.

(only if) Consider an $(l+m) \times k$ submatrix of A' such that every element is 1. There are at least l rows in the $(l+m) \times k$ submatrix that are also in A since there are m newly added rows in A'. Thus, we can select l rows in the $(l+m) \times k$ submatrix that are also in A. In this way, we get an $l \times k$ submatrix of A such that every element is 1.

Obviously, every element in the $(l+m) \times k$ submatrix of A' is 1 if and only if both the consensus score and the bottleneck score for the submatrix are 0 and every element in the consensus/center vector is 1. □

The proof also suggests that it is NP-hard to decide whether the score is 0 in both problems. Therefore, there is no approximation algorithm with *any* guaranteed ratio for both problems when the optimal score is 0.

3. The consensus submatrix problem. In this section, we will present the randomized approximation algorithm for the consensus submatrix problem. Let P_{opt}, Q_{opt}, and z_{opt} be the set of rows, the set of columns, and the consensus vector of an optimal solution. The optimal consensus score is H_{opt}. By assumption, $H_{opt} = \Omega(kl)$, i.e., there is a constant c' such that $H_{opt} \times c' = kl$. Again, by assumption, $k = \Omega(n)$, i.e., there is a constant c such that $k \times c = n$.

Before we present the algorithm, we first introduce the basic ideas of the algorithm. By enumerating all size k subsets of Y and all length k vectors, we can find Q_{opt} and z_{opt} at some moment. It is easy to see that if we know exactly Q_{opt} and z_{opt}, then we can find the corresponding P_{opt} in polynomial time to minimize the consensus score. However, this straightforward approach costs exponential time. Here we use a random sampling technique to randomly select $\Theta(\log m)$ columns in Q_{opt}, and enumerate all possible vectors of length $\Theta(\log m)$ for those columns. (If $n < \Theta(\log m)$, we will select all the n columns and the running time is still polynomial in terms of the input size.) At some point, we know $\Theta(\log m)$ bits of z_{opt} and we can use the partial z_{opt} to select the l rows which are closest to z_{opt} in those $\Theta(\log m)$ bits. After that we can construct a consensus vector z as follows. For each column, choose the (majority) letter that appears the most in each of the l letters in the l selected rows. Then for each of the n columns, we can calculate the number of mismatches between the majority letter and the l letters in the l selected rows. By selecting the best k columns, we can get a good solution.

The remaining difficulty is how to randomly select $\Theta(\log m)$ columns in Q_{opt} while Q_{opt} is unknown. Our new idea is to randomly select a set B of $\lceil (c+1)(\frac{4 \log m}{\epsilon^2} + 1) \rceil$ columns from A and enumerate all size $\lceil \frac{4 \log m}{\epsilon^2} \rceil$ subsets of B in time $O(m^{\frac{4(c+1)}{\epsilon^2}})$,

Algorithm 1 for the Consensus Submatrix Problem

Input: an $m \times n$ matrix A, integers l and k, and a number $\epsilon > 0$.

Output: a size l subset P of rows, a size k subset Q of columns and a length k consensus vector z.

Step 1: Randomly and independently select a set B of $\lceil (c+1)(\frac{4\log m}{\epsilon^2}+1)\rceil$ columns from A. (If $n < \lceil (c+1)(\frac{4\log m}{\epsilon^2}+1)\rceil$, we will select all the n columns and the running time is still polynomial in terms of the input size.)

(1.1) **for** every size $\lceil \frac{4\log m}{\epsilon^2}\rceil$ subset R of B **do**

(1.2) **for** every $z|^R \in \Sigma^{|R|}$ **do**

(a) Select the best l rows $P = \{p_1, \ldots, p_l\}$ that minimize $d(z|^R, x_i|^R)$.

(b) **for** each column j **do**

Compute $f(j) = \sum_{i=1}^{l} d(s_j, a_{p_i,j})$, where s_j is the majority element of the l rows in P in column j.

Select the best k columns $Q = \{q_1, \ldots, q_k\}$ with minimum value $f(j)$ and let $z(Q) = s_{q_1} s_{q_2} \ldots s_{q_k}$.

(c) Calculate $H = \sum_{i=1}^{l} d(x_{p_i}|^Q, z)$ of this solution.

Step 2: Output P, Q and z with minimum H.

FIG. 2. *Algorithm* 1.

which is polynomial in terms of the input size $O(mn)$. We can show that with high probability, we can get a set of $\lceil \frac{4\log m}{\epsilon^2}\rceil$ columns randomly selected from Q_{opt}.

Now we describe the complete algorithm in Figure 2.

The following lemma, originally from [26], will be used in our proofs.

LEMMA 1. *Let $X_1, X_2, \ldots, X_n$ be n independent random* 0-1 *variables, where X_i takes* 1 *with probability p_i, $0 < p_i < 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Then for any $0 < \epsilon \le 1$,*

$$\mathbf{Pr}(X > \mu + \epsilon\, n) < \exp\left(-\frac{1}{3} n\epsilon^2\right),$$

$$\mathbf{Pr}(X < \mu - \epsilon\, n) \le \exp\left(-\frac{1}{2} n\epsilon^2\right).$$

LEMMA 2. *With probability at most $m^{-\frac{2}{\epsilon^2 c^2 (c+1)}}$, no subset R of size $\lceil \frac{4\log m}{\epsilon^2}\rceil$ used in Step* 1 *of Algorithm* 1 *satisfies $R \subseteq Q_{opt}$.*

Proof. Obviously, if the subset B in Step 1 of Algorithm 1 contains at least $\lceil \frac{4\log m}{\epsilon^2}\rceil$ columns in Q_{opt}, there is a size $\lceil \frac{4\log m}{\epsilon^2}\rceil$ subset $R \subseteq Q_{opt}$. Now we consider the probability that the subset B contains less than $\lceil \frac{4\log m}{\epsilon^2}\rceil$ columns in Q_{opt}. Let b be the number of columns in B that are also in Q_{opt}. Recall that $k \times c = n$. If we randomly select a column, the probability that the column is in Q_{opt} is $\frac{1}{c}$. Let μ be the expected number of columns in B that are in Q_{opt}. We have $\mu = \frac{|B|}{c} = \frac{1}{c}\lceil (c+1)(\frac{4\log m}{\epsilon^2}+1)\rceil$. Let $X_1, X_2, \ldots, X_{|B|}$ be $|B|$ independent random 0/1 variables, where $X_i = 1$ with probability $\frac{1}{c}$ indicating that the selected column is in Q_{opt}. Thus,

$$b = \sum_{i=1}^{|B|} X_i \tag{2}$$

and

$$\mu = E\left(\sum_{i=1}^{|B|} X_i\right) = \frac{1}{c}\left\lceil (c+1)\left(\frac{4\log m}{\epsilon^2}+1\right)\right\rceil. \tag{3}$$

From the algorithm,

$$|B| = \left\lceil (c+1)\left(\frac{4\log m}{\epsilon^2}+1\right)\right\rceil. \tag{4}$$

Based on Lemma 1, we have

$$\begin{aligned}
\mathbf{Pr}\left(b < \left\lceil \frac{4\log m}{\epsilon^2}\right\rceil\right) &\le \mathbf{Pr}\left(b < \frac{4\log m}{\epsilon^2}+1\right)\\
&\le \mathbf{Pr}\left(b < \left(\frac{1}{c}-\frac{1}{c(c+1)}\right)\left\lceil (c+1)\left(\frac{4\log m}{\epsilon^2}+1\right)\right\rceil\right)\\
&= \mathbf{Pr}\left(\sum_{i=1}^{|B|} X_i < \mu - \frac{1}{c(c+1)}|B|\right) \quad \text{(from (2), (3), and (4))}\\
&\le \exp\left(-\frac{1}{2c^2(c+1)^2}|B|\right)\\
&\le \exp\left(-\frac{1}{2c^2(c+1)^2}(c+1)\left(\frac{4\log m}{\epsilon^2}\right)\right)\\
&= \exp\left(-\frac{2\log m}{\epsilon^2 c^2(c+1)}\right) = m^{-\frac{2}{\epsilon^2 c^2(c+1)}}.
\end{aligned}$$

Therefore, with probability at most $m^{-\frac{2}{\epsilon^2 c^2(c+1)}}$, no subset R of size $\lceil \frac{4\log m}{\epsilon^2}\rceil$ used in Step 1 of Algorithm 1 satisfies $R \subseteq Q_{opt}$. □

LEMMA 3. *Assume* $|R| = \lceil \frac{4\log m}{\epsilon^2}\rceil$ *and* $R \subseteq Q_{opt}$. *Let* $\rho = \frac{k}{|R|}$. *With probability at most* m^{-1}, *there is a row* x_i *in* X *satisfying*

$$\frac{d(z_{opt}, x_i|^{Q_{opt}}) - \epsilon k}{\rho} > d(z_{opt}|^R, x_i|^R). \tag{5}$$

With probability at most $m^{-\frac{1}{3}}$, *there is a row* x_i *in* X *satisfying*

$$d(z_{opt}|^R, x_i|^R) > \frac{d(z_{opt}, x_i|^{Q_{opt}}) + \epsilon k}{\rho}. \tag{6}$$

Proof. We will calculate $\mathbf{Pr}(d(z_{opt}|^R, x_i|^R) < \frac{d(z_{opt}, x_i|^{Q_{opt}})-\epsilon k}{\rho})$ first. From Algorithm 1, R is a subset of B, a set of randomly independently selected columns. Thus, R is also a set of randomly independently selected columns. Therefore, we can treat $d(z_{opt}|^R, x_i|^R)$ as the sum of $|R|$ independent random 0/1 variables $\sum_{j=1}^{|R|} X_j$. As $R \subseteq Q_{opt}$, $X_j = 1$ indicates a mismatch between z_{opt} and x_i at the jth position in R. Clearly $E[d(z_{opt}|^R, x_i|^R)] = d(z_{opt}, x_i|^{Q_{opt}})/\rho$. From Lemma 1, we have

$$\begin{aligned}
&\mathbf{Pr}\left(d(z_{opt}|^R, x_i|^R) < \frac{d(z_{opt}, x_i|^{Q_{opt}}) - \epsilon k}{\rho}\right)\\
&= \mathbf{Pr}(d(z_{opt}|^R, x_i|^R) < E[d(z_{opt}|^R, x_i|^R)] - \epsilon|R|)\\
&\le \exp\left(-\frac{1}{2}\epsilon^2|R|\right) \le m^{-2},
\end{aligned}$$

where the last inequality is due to the setting $|R| = \lceil \frac{4 \log m}{\epsilon^2} \rceil$ in Step 1 of Algorithm 1. Considering all the m x_i's, the probability that an $x_i \in X$ satisfies (5) is at most $m \times m^{-2} = m^{-1}$.

Similarly, we have

$$\mathbf{Pr}\left(d(z_{opt}|^R, x_i|^R) > \frac{d(z_{opt}, x_i|^{Q_{opt}}) + \epsilon k}{\rho} \right) < m^{-\frac{4}{3}}.$$

Considering all the m x_i's, the probability that an $x_i \in X$ satisfies (6) is at most $m \times m^{-\frac{4}{3}} = m^{-\frac{1}{3}}$. □

LEMMA 4. *When $R \subseteq Q_{opt}$ and $z|^R = z_{opt}|^R$, with probability at most $2m^{-\frac{1}{3}}$, the set of rows $P = \{p_1, \ldots, p_l\}$ selected in Step* 1(a) *of Algorithm* 1 *satisfies $\sum_{i=1}^{l} d(z_{opt}, x_{p_i}|^{Q_{opt}}) > H_{opt} + 2\epsilon kl$.*

Proof. Let $P_{opt} = \{p_1^*, \ldots, p_l^*\}$ be the l rows in the optimal solution; we have

$$\sum_{i=1}^{l} d(z_{opt}, x_{p_i^*}|^{Q_{opt}}) = H_{opt}. \tag{7}$$

From Lemma 3, with probability at most m^{-1}, a row $x_{p_i} \in \{p_1, p_2, \ldots, p_l\}$ satisfies

$$\sum_{i=1}^{l} d(z_{opt}|^R, x_{p_i}|^R) < \sum_{i=1}^{l} \frac{d(z_{opt}, x_{p_i}|^{Q_{opt}}) - \epsilon k}{\rho}. \tag{8}$$

Again from Lemma 3, with probability at most $m^{-\frac{1}{3}}$, a row $x_{p_i^*} \in \{p_1^*, p_2^*, \ldots, p_l^*\}$ satisfies

$$\sum_{i=1}^{l} d(z_{opt}|^R, x_{p_i^*}|^R) > \sum_{i=1}^{l} \frac{d(z_{opt}, x_{p_i^*}|^{Q_{opt}}) + \epsilon k}{\rho}. \tag{9}$$

By trying all length $|R|$ vectors in Step 1 of Algorithm 1, we can assume that we know $z_{opt}|^R$. In Step 1(a), we select the best set $P = \{p_1, \ldots, p_l\}$ with minimum $d(z_{opt}|^R, x_i|^R)$. Thus, for any $R \subseteq Q_{opt}$, we have

$$\sum_{i=1}^{l} d(z_{opt}|^R, x_{p_i}|^R) \leq \sum_{i=1}^{l} d(z_{opt}|^R, x_{p_i^*}|^R). \tag{10}$$

From (8), (9), and (10), if $R \subseteq Q_{opt}$ and $z|^R = z_{opt}|^R$, with probability at most $m^{-1} + m^{-\frac{1}{3}} \leq 2m^{-\frac{1}{3}}$, there exists a set of rows $P = \{p_1, \ldots, p_l\}$ obtained in Step 1(a) of Algorithm 1 such that

$$\begin{aligned} \sum_{i=1}^{l} d(z_{opt}, x_{p_i}|^{Q_{opt}}) &> \sum_{i=1}^{l} (d(z_{opt}, x_{p_i^*}|^{Q_{opt}}) + 2\epsilon k) \\ &= H_{opt} + 2\epsilon kl. \end{aligned}$$

The last inequality is from (7). □

THEOREM 2. *For any $\delta > 0$, with probability at least $1 - m^{-\frac{8c'^2}{\delta^2 c^2 (c+1)}} - 2m^{-\frac{1}{3}}$, Algorithm* 1 *outputs a solution with consensus score at most $(1+\delta)H_{opt}$ in $O(nm^{O(\frac{1}{\delta^2})})$ time.*

Proof. When $R \subseteq Q_{opt}$ and $z|^R = z_{opt}|^R$, in Step 1(b), we can construct a Q and $z(Q)$ such that

$$\sum_{i=1}^{l} d(z(Q), x_{p_i}|^Q) \leq \sum_{i=1}^{l} d(z_{opt}, x_{p_i}|^{Q_{opt}}). \tag{11}$$

From Lemma 2, we know that with probability at most $m^{-\frac{2}{\epsilon^2 c^2 (c+1)}}$, there is no subset R with size $\lceil \frac{4 \log m}{\epsilon^2} \rceil$ in Step 1 of Algorithm 1 such that $R \subseteq Q_{opt}$. Combining with Lemma 4, we know that with probability at most $m^{-\frac{2}{\epsilon^2 c^2 (c+1)}} + 2m^{-\frac{1}{3}}$, in the execution of Algorithm 1, any set of rows $P = \{p_1, \ldots, p_l\}$ obtained in Step 1(a) satisfies

$$\sum_{i=1}^{l} d(z_{opt}, x_{p_i}|^{Q_{opt}}) > H_{opt} + 2\epsilon kl.$$

In other words, with probability at least $1 - m^{-\frac{2}{\epsilon^2 c^2 (c+1)}} - 2m^{-\frac{1}{3}}$, in the execution of Algorithm 1, we can get a set of rows $P = \{p_1, \ldots, p_l\}$ in Step 1(a) that satisfies $\sum_{i=1}^{l} d(z_{opt}, x_{p_i}|^{Q_{opt}}) \leq H_{opt} + 2\epsilon kl$.

From (11), we have

$$\sum_{i=1}^{l} d(z(Q), x_{p_i}|^Q) \leq \sum_{i=1}^{l} d(z_{opt}, x_{p_i}|^{Q_{opt}}) \leq H_{opt} + 2\epsilon kl.$$

Recall that $H_{opt} \times c' = kl$. Set $\epsilon = \frac{\delta}{2c'}$. So with probability at least $1 - m^{-\frac{8c'^2}{\delta^2 c^2 (c+1)}} - 2m^{-\frac{1}{3}}$, Algorithm 1 outputs a solution with consensus score at most $(1+\delta)H_{opt}$.

For the time complexity, Step 1(a), Step 1(b), and Step 1(c) take $O(mn)$ time. Step 1.1 is repeated $O(2^{\frac{4(c+1)\log m}{\epsilon^2}}) = O(m^{O(\frac{1}{\epsilon^2})}) = O(m^{O(\frac{1}{\delta^2})})$ times. Step 1.2 is repeated $O(m^{O(\frac{\log |\Sigma|}{\epsilon^2})}) = O(m^{O(\frac{1}{\delta^2})})$ times as $\epsilon = \frac{\delta}{2c'}$ and $|\Sigma|$ is a fixed constant. Thus, the total running time is $O(nm^{O(\frac{1}{\delta^2})})$. ∎

THEOREM 3. *There exists a PTAS for the consensus submatrix problem.*

Proof. Algorithm 1 can be derandomized by the standard method. For instance, instead of randomly and independently choosing $\Theta(\log m)$ columns from the n columns in Step 1, we can pick the vertices encountered on a random walk of length $\Theta(\log m)$ on a constant degree expander [27]. Obviously, the number of such random walks on a constant degree expander is polynomial in terms of m. Thus, by enumerating all random walks of length $\Theta(\log m)$, we have a polynomial time deterministic algorithm. (Also see [30].) ∎

4. The bottleneck submatrix problem. In this section, we present the randomized approximation algorithm for the bottleneck submatrix problem. Let P_{opt}, Q_{opt}, and z_{opt} be the set of rows, the set of columns, and the center vector of an optimal solution. The optimal bottleneck score is d_{opt}. By assumption, $d_{opt} = \Omega(k)$ and $k = \Omega(n)$, i.e., there are constants c'' and c such that $d_{opt} \times c'' = k$ and $k \times c = n$.

Similar to Algorithm 1, we can use a random sampling technique to know $\Theta(\log m)$ bits of z_{opt}. Then we can use the partial z_{opt} to select the l rows which are closest to z_{opt} in those $\Theta(\log m)$ bits as in Step 1(a) of Algorithm 1. From Lemma 3, we know that using $\Theta(\log m)$ bits in R, we can get a good estimation of $d(z_{opt}, x_i|^{Q_{opt}})$ for each

x_i in X. Thus, if we can correctly select Q_{opt} from the n given columns, then we can get a good approximation solution. However, Step 1(b) in Algorithm 1 does not work for the bottleneck score in selecting a good approximation of Q_{opt}. Thus, we use the linear programming and randomized rounding technique to select k columns in the matrix.

Linear programming formulation. Given a set of rows $P = \{p_1, \ldots, p_l\}$, we want to find a set of k columns Q and a vector z such that the bottleneck score is minimized. This problem is equivalent to the following optimization problem:

$$\begin{cases} \min d; \\ d(z, x_{p_i}|^Q) \le d, \quad i = 1, 2, \ldots, l, \quad Q \subseteq Y, \quad |Q| = k, z \in \Sigma^k. \end{cases} \tag{12}$$

Let $\Sigma = \{\pi_1, \pi_2, \ldots, \pi_{|\Sigma|}\}$. We introduce 0/1 variable $y_{i,j}$ $(i = 1, 2, \ldots, n, j = 1, 2, \ldots, |\Sigma|)$ to indicate the membership of column i in Q and the corresponding bit of z. We have $y_{i,j} = 1$ if and only if column i is in Q and the corresponding bit in z is π_j. For any $a, b \in \Sigma$, $\chi(a, b) = 0$ if $a = b$ and $\chi(a, b) = 1$ if $a \ne b$. We can formulate (12) as 0/1 integer linear programming:

$$\begin{cases} \min d; \\ \displaystyle\sum_{i=1}^{n} \sum_{j=1}^{|\Sigma|} y_{i,j} = k, \\ \displaystyle\sum_{j=1}^{|\Sigma|} y_{i,j} \le 1, \quad i = 1, 2, \ldots, n, \\ \displaystyle\sum_{i=1}^{n} \sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y_{i,j} \le d, \quad s = 1, 2, \ldots, l. \end{cases} \tag{13}$$

Here $y_{i,j}$ is used to achieve two tasks: (1) to decide whether column i is selected and (2) if column i is selected, to choose the letter in the center vector z on this column.

We can obtain a fraction solution $y_{i,j} = \overline{y_{i,j}} (i = 1, 2, \ldots, n, j = 1, 2, \ldots, |\Sigma|)$ for (13) in polynomial time. After we get the fraction solution, we do randomized rounding to get an integer solution.

Randomized rounding. Given a fraction solution $y_{i,j} = \overline{y_{i,j}}$ $(i = 1, 2, \ldots, n, j = 1, 2, \ldots, |\Sigma|)$ with cost $\overline{d}$, for each $1 \le i \le n, 1 \le j \le |\Sigma|$, we randomly select column i to Q with probability $\sum_{j=1}^{|\Sigma|} \overline{y_{i,j}}$ and randomly set the bit of z in this column according to the distribution $\overline{y_{i,j}}$ for $j = 1, 2, \ldots, |\Sigma|$. In terms of programming, we can generate a random number ρ in (0,1) for every column i. If $\rho < \sum_{j=1}^{|\Sigma|} \overline{y_{i,j}}$, we select this column into Q and let the bit of z corresponding to this column be π_t if and only if $\sum_{j=1}^{t-1} \overline{y_{i,j}} \le \rho < \sum_{j=1}^{t} \overline{y_{i,j}}$. If $\rho \ge \sum_{j=1}^{|\Sigma|} \overline{y_{i,j}}$, this column is not selected. Hence we get a 0/1 integer solution $y' = \{y'_{1,1}, \ldots, y'_{1,|\Sigma|}, \ldots, y'_{n,1}, \ldots, y'_{n,|\Sigma|}\}$.

In this randomized rounding process, we have to do two things: (1) select k' columns, where $k' \ge k - \delta d_{opt}$, and (2) get the integral value for each $y_{i,j}$ such that the distance (restricted on the k' selected columns) between any row in P and the center vector thus obtained is at most γd_{opt}. Here $\delta > 0$ and $\gamma > 0$ are two parameters used to control the errors.

LEMMA 5. *When* $\frac{n\gamma^2}{3(cc')^2} \ge 2 \log m$, *for any* $\gamma, \delta > 0$, *with probability at most* $\exp(-\frac{n\delta^2}{2(cc')^2}) + m^{-1}$, *the integer solution* $y' = \{y'_{1,1}, \ldots, y'_{1,|\Sigma|}, \ldots, y'_{n,1}, \ldots, y'_{n,|\Sigma|}\}$

violates at least one of the following inequalities,

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} y'_{i,j}\right) > k - \delta d_{opt}, \tag{14}$$

and for every row $x_{p_s}(s = 1, 2, \ldots, l)$*,*

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j}\right) < \overline{d} + \gamma d_{opt}. \tag{15}$$

Proof. From $k \times c = n$ and $d_{opt} \times c'' = k$, we have $d_{opt} \times cc'' = n$. For each $1 \le i \le n$, the randomized rounding process ensures that at most one $y'_{i,a} = 1$ for $\pi_a \in \Sigma$. Since the rounding is independent for different i's, $(\sum_{j=1}^{|\Sigma|} y'_{i,j})$'s are independent 0/1 random variables for $1 \le i \le n$, and $(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j})$'s are independent 0/1 random variables for $1 \le i \le n$ in every row x_{p_s}. It is easy to see that

$$E\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} y'_{i,j}\right)\right) = k,$$

and for every row $x_{p_s}(s = 1, 2, \ldots, l)$,

$$E\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j}\right)\right) \le \overline{d}.$$

From Lemma 1, we have

$$\begin{aligned}\mathbf{Pr}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} y'_{i,j}\right) < k - \delta d_{opt}\right) &= \mathbf{Pr}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} y'_{i,j}\right) < E\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} y'_{i,j}\right)\right) - \frac{\delta n}{cc''}\right) \\ &\le \exp\left(-\frac{n\delta^2}{2(cc'')^2}\right),\end{aligned}$$

and for every row $x_{p_s}(s = 1, 2, \ldots, l)$,

$$\begin{aligned}&\mathbf{Pr}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j}\right) > \overline{d} + \gamma d_{opt}\right) \\ &\le \mathbf{Pr}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j}\right) > E\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j}\right)\right) + \frac{\gamma}{cc''} n\right) \\ &< \exp\left(-\frac{n\gamma^2}{3(cc'')^2}\right).\end{aligned}$$

Considering all l rows in P, the probability that at least one row in P satisfies $\sum_{i=1}^{n}(\sum_{j=1}^{|\Sigma|} \chi(\pi_j, x_{p_s,i}) y'_{i,j}) > \overline{d} + \gamma d_{opt}$ is at most $l \times \exp(-\frac{n\gamma^2}{3(cc'')^2}) \le m \times \exp(-\frac{n\gamma^2}{3(cc'')^2})$. Thus, with probability at most $\exp(-\frac{n\delta^2}{2(cc'')^2}) + m \times \exp(-\frac{n\gamma^2}{3(cc'')^2})$, the integer solution $y' = \{y'_{1,1}, \ldots, y'_{1,|\Sigma|}, \ldots, y'_{n,1}, \ldots, y'_{n,|\Sigma|}\}$ violates at least one of (14) and (15). When $\frac{n\gamma^2}{3(cc'')^2} \ge 2 \log m$, we get the desired probability. ∎

Algorithm 2 for the Bottleneck Submatrix Problem

Input: a matrix $A \in \Sigma^{m\times n}$, integers l, k, a row $z \in \Sigma^n$ and numbers $\epsilon > 0$, and $\gamma > 0$.

Output: a size l subset P of rows, a size k subset Q of columns and a length k center vector z.

if $\frac{n\gamma^2}{3(cc'')^2} \le 2\log m$ **then**

try all size k subset Q of the n columns and all center vectors of length k to solve the problem.

if $\frac{n\gamma^2}{3(cc'')^2} > 2\log m$ **then**

Step 1: randomly and independently select a set B of $\lceil \frac{4(c+1)\log m}{\epsilon^2} \rceil$ columns from A.

for every $\lceil \frac{4\log m}{\epsilon^2} \rceil$ size subset R of B **do**

for every $z|^R \in \Sigma^{|R|}$ **do**

(a) Select the best l rows $P = \{p_1, \ldots, p_l\}$ that minimize $d(z|^R, x_i|^R)$.

(b) Solve the optimization problem (12) by linear programming and randomized rounding to get Q and z.

Step 2: Output P,Q and z with minimum bottleneck score d.

FIG. 3. *Algorithm* 2.

When $\frac{n\gamma^2}{3(cc'')^2} < 2\log m$, we try all subsets of X with size k and all length k vectors in polynomial time and get the best solution.

From Lemma 5, we know that in the randomized rounding process, with high probability, we selected k' columns in Q, where $(1-\epsilon)k \le k'$. Our aim is to exactly select k columns. If $k' > k$, we can arbitrarily delete $k'-k$ columns from Q and obtain the set of k columns $Q' \subseteq Q$. If $k' < k$, we can arbitrarily select $k-k'$ columns outside Q and add them to Q to get a set of k columns $Q' \supset Q$. By doing so, the extra error introduced is at most ϵk. Since $d = \Omega(n)$, the error ϵk is small and we still can get a PTAS.

Now we describe the complete algorithm in Figure 3.

Similar to Lemma 4, we have the following lemma.

LEMMA 6. *When $R \subseteq Q_{opt}$ and $z|^R = z_{opt}|^R$, with probability at most $2m^{-\frac{1}{3}}$, the set of rows $P = \{p_1, \ldots, p_l\}$ obtained in Step* 1(a) *of Algorithm* 2 *satisfies $d(z_{opt}, x_{p_i}|^{Q_{opt}}) > d_{opt} + 2\epsilon k$ for some row $x_{p_i}(1 \le i \le l)$.*

Proof. Let $P_{opt} = \{p_1^*, \ldots, p_l^*\}$ be the l rows in the optimal solution. We have

$$\max_{p_i^* \in P_{opt}} d(z_{opt}, x_{p_i^*}|^{Q_{opt}}) = d_{opt}. \tag{16}$$

From Lemma 3, with probability at most m^{-1}, a row $x_{p_i} \in \{p_1, p_2, \ldots, p_l\}$ satisfies

$$d(z_{opt}|^R, x_{p_i}|^R) < \frac{d(z_{opt}, x_{p_i}|^{Q_{opt}}) - \epsilon k}{\rho}. \tag{17}$$

Again from Lemma 3, with probability at most $m^{-\frac{1}{3}}$, a row $x_{p_i^*} \in \{p_1^*, p_2^*, \ldots, p_l^*\}$ satisfies

$$d(z_{opt}|^R, x_{p_i^*}|^R) > \frac{d(z_{opt}, x_{p_i^*}|^{Q_{opt}}) + \epsilon k}{\rho}. \tag{18}$$

By trying all length $|R|$ vectors in Step 1 of Algorithm 2, we can assume that we know $z_{opt}|^R$. In Step 1(a), we select the best set $P = \{p_1, \ldots, p_l\}$ with minimum $d(z_{opt}|^R, x_i|^R)$. Thus, for any $R \subseteq Q_{opt}$, for every $p_i \in P$, we have

$$d(z_{opt}|^R, x_{p_i}|^R) \leq \max_{p_i^* \in P_{opt}} d(z_{opt}|^R, x_{p_i^*}|^R). \tag{19}$$

From (17), (18), and (19), if $R \subseteq Q_{opt}$ and $z|^R = z_{opt}|^R$, with probability at most $m^{-1} + m^{-\frac{1}{3}} \leq 2m^{-\frac{1}{3}}$, there exists a set of rows $P = \{p_1, \ldots, p_l\}$ obtained in Step 1(a) of Algorithm 2 such that for some row $x_{p_i} (1 \leq i \leq l)$,

$$\begin{aligned} d(z_{opt}, x_{p_i}|^{Q_{opt}}) &> \max_{p_i^* \in P_{opt}} d(z_{opt}, x_{p_i^*}|^{Q_{opt}}) + 2\epsilon k \\ &= d_{opt} + 2\epsilon k. \end{aligned}$$

The last inequality is from (16). ∎

From Lemmas 2, 5, and 6, we have the following theorem.

THEOREM 4. *With probability at least* $1 - m^{-\frac{2}{\epsilon^2 c^2 (c+1)}} - 2m^{-\frac{1}{3}} - \exp(-\frac{n\delta^2}{2(cc'')^2}) - m^{-1}$, *Algorithm* 2 *runs in time* $O(n^{O(1)} m^{O(\frac{1}{\epsilon^2} + \frac{1}{\gamma^2})})$ *and obtains a solution with the bottleneck score at most* $(1 + 2c''\epsilon + \gamma + \delta) d_{opt}$ *for any fixed* $\epsilon,\ \gamma,\ \delta > 0$.

Proof. From Lemma 2, we know that with probability at most $m^{-\frac{2}{\epsilon^2 c^2 (c+1)}}$, there is no subset R with size $\lceil \frac{4 \log m}{\epsilon^2} \rceil$ in Step 1 of Algorithm 2 such that $R \subseteq Q_{opt}$. Combining with Lemma 6, we know that with probability at most $m^{-\frac{8c''^2}{\epsilon'^2 c^2 (c+1)}} + 2m^{-\frac{1}{3}}$, in the execution of Algorithm 2, for any set of rows $P = \{p_1, \ldots, p_l\}$ obtained in Step 1(a), problem (12) has a solution $Q = Q_{opt}$ and $z = z_{opt}$ with the bottleneck score $d_P > (1 + 2c''\epsilon) d_{opt}$ as $k = 2c'' d_{opt}$. In Step 1(b), we can get a fraction solution $y_{i,j} = \overline{y_{i,j}}$ $(i = 1, 2, \ldots, n, j = 1, 2, \ldots, |\Sigma|)$ with cost $\overline{d} < d_P$. Thus, with probability at most $m^{-\frac{8c''^2}{\epsilon'^2 c^2 (c+1)}} + 2m^{-\frac{1}{3}}$,

$$\overline{d} > (1 + 2c''\epsilon) d_{opt}. \tag{20}$$

From Lemma 5, when $\frac{n\gamma^2}{3(cc'')^2} \geq 2 \log m$, we know that with probability at most $\exp(-\frac{n\delta^2}{2(cc'')^2}) + m^{-1}$, the integer solution $y' = \{y'_{1,1}, \ldots, y'_{1,|\Sigma|}, \ldots, y'_{n,1}, \ \ldots, y'_{n,|\Sigma|}\}$ violates at least one of (14) and (15). Thus the bottleneck score

$$d' > (\delta + \gamma) d_{opt} + \overline{d}. \tag{21}$$

From (20) and (21), when $\frac{n\gamma^2}{3(cc'')^2} \geq 2 \log m$, with probability at most $m^{-\frac{2}{\epsilon^2 c^2 (c+1)}} + 2m^{-\frac{1}{3}} + \exp(-\frac{n\delta^2}{2(cc'')^2}) + m^{-1}$,

$$d' > (1 + 2c''\epsilon + \gamma + \delta) d_{opt}.$$

In other words, when $\frac{n\gamma^2}{3(cc'')^2} \geq 2 \log m$, with probability at least $1 - m^{-\frac{2}{\epsilon^2 c^2 (c+1)}} - 2m^{-\frac{1}{3}} - \exp(-\frac{n\delta^2}{2(cc'')^2}) - m^{-1}$, in the execution of Algorithm 2, we can get a solution in Step 1(b) with the bottleneck score at most $(1 + 2c''\epsilon + \gamma + \delta) d_{opt}$.

For the time complexity, Step 1(a), Step 1(b), and Step 1(c) take $O((mn|\Sigma|)^{O(1)})$ time. Step 1 is repeated at most $O(m^{O(\frac{\log |\Sigma|}{\epsilon^2})}) = O(m^{O(\frac{1}{\epsilon^2})})$ times. Thus, the total time required is $O(n^{O(1)} m^{O(\frac{1}{\epsilon^2})})$.

When $\frac{n\gamma^2}{3(cc')^2} < 2\log m$, we can solve the problem in $O(n^{O(1)}m^{O(\frac{1}{\gamma^2})})$ time by enumerating all possible sets Q and all possible center vectors z. □

THEOREM 5. *There exists a PTAS for the bottleneck submatrix problem.*

Proof. For Step 1(b), we can use the technique in [26] to derandomize it. The derandomization of the random sampling step is the same as in Algorithm 1. □

5. Conclusion. We have designed PTASs for both the consensus submatrix and the bottleneck submatrix problems. To the best of our knowledge, this is the first time that approximation algorithms with guaranteed performance ratios have been presented for microarray data analysis. It is important to point out that the running time here is very high and the algorithms may not work in practice.

Acknowledgment. We thank the referees for their helpful suggestions.

REFERENCES

[1] R. B. STOUGHTON, *Applications of DNA microarrays in biology*, Annu. Rev. Biochem, 74 (2005), pp. 53–82.

[2] D. B. ALLISON, X. CUI, G. P. PAGE, AND M. SABRIPOU, *Microarray data analysis: From disarray to consolidation and consensus*, Nature Rev. Genetics, 7 (2006), pp. 55–65.

[3] S. TAVAZOIE, J. D. HUGHES, M. J. CAMPBELL, R. J. CHO, AND G. M. CHURCH, *Systematic determination of genetic network architecture*, Nature Genetics, 22 (1999), pp. 281–285.

[4] F. X. WU, W. J. ZHANG, AND A. J. KUSALIK, *A genetic K-means clustering algorithm applied to gene expression data*, Lecture Artificial Intelligence, 2671 (2003), pp. 520–526.

[5] P. TAMAYO, D. SLONIM, J. MESIROV, Q. ZHU, S. KITAREEWAN, E. DMITROVSKY, E. S. LANDER, AND T. R. GOLUB, *Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation*, Proc. Natl. Acad. Sci. USA, 96 (1999), pp. 2907–2912.

[6] H. RESSOM, D. WANG, AND P. NATARAJAN, *Clustering gene expression data using adaptive double self-organizing map*, Physiol. Genomics, 14 (2003) pp. 35–46.

[7] M. B. EISEN, P. T. SPELLMAN, P. O. BROWN, AND D. BOTSTEIN, *Cluster analysis and display of genome-wide expression patterns*, Proc. Natl. Acad. Sci. USA, 95 (1998), pp. 14863–14868.

[8] V. R. IYER, M. B. EISEN, D. T. ROSS, G. SCHULER, T. MOORE, J. C. F. LEE, J. M. TRENT, L. M. STAUDT, J. HUDSON, M. S. BOGUSKI, D. LASHKARI, D. SHALON, D. BOTSTEIN, AND P. O. BROWN, *The transcriptional program in the response of human fibroblasts to serum*, Science, 283 (1999), pp. 83–87.

[9] J. QIN, D. P. LEWIS, AND W. S. NOBLE, *Kernel hierarchical gene clustering from microarray expression data*, Bioinformatics, 19 (2003), pp. 2097–2104.

[10] O. ALTER, P. O. BROWN, AND D. BOTSTEIN, *Generalized singular value decomposition for comparative analysis of genome-scale expression data sets of two different organisms*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 3351–3356.

[11] N. S. HOLTER, M. MITRA, A. MARITAN, M. CIEPLAK, J. R. BANAVAR, AND N. V. FEDOROFF, *Fundamental patterns underlying gene expression profiles: Simplicity from complexity*, Proc. Natl. Acad. Sci. USA, 97 (2000), pp. 8409–8414.

[12] K. C. LI, M. YAN, AND S. S. YUAN, *A simple statistical model for depicting the cdc*15*-synchronized yeast cell-cycle regulated gene expression data*, Statist. Sinica, 12 (2002), pp. 141–158.

[13] B. TJADEN, *An approach for clustering gene expression data with error information*, BMC Bioinform., 7 (2006), p. 17.

[14] B. H. MECHAM, D. Z. WETMORE, Z. SZALLASI, Y. SADOVSKY, I. KOHANE, AND T. J. MARIANI, *Increased measurement accuracy for sequence-verified microarray probes*, Physiol. Genomics, 18 (2004), pp 308–315.

[15] D. M. ROCKE AND B. DUBIN, *A model for measurement error for gene expression arrays*, J. Comput. Biol., 8 (2001), pp. 557–569.

[16] S. DRAGHICI, P. KHATRI, A. C. EKLUND, AND Z. SZALLASI, *Reliability and reproducibility issues in DNA microarray measurements*, TRENDS in Genetics, 22 (2006), pp. 101–109.

[17] J. P. BRODY, B. A. WILLIAMS, B. J. WOLD, AND S. R. QUAKE, *Significance and statistical errors in the analysis of DNA microarray data*, Proc. Natl. Acad. Sci. USA, 99 (2002), pp. 12975–12978.

[18] E. Purdom and S. P. Holmes, *Error distribution for gene expression data*, Statist. Appl. Genetics Molecular Biol., 4 (2005).
[19] H. Cho and J. K. Lee, *Bayesian hierarchical error model for analysis of gene expression data*, Bioinformatics, 20 (2004), pp. 2016–2025.
[20] X. Liu and L. Wang, *Computing the maximum similarity bi-clusters of gene expression data*, Bioinformatics, 23 (2007), pp. 50–56.
[21] G. Getz, E. Levine, and E. Domany, *Coupled two–way clustering analysis of gene microarray data*, Proc. Natl. Acad. Sci. USA, 2000, pp. 12079–12084.
[22] Y. Cheng and G. M. Church, *Biclustering of expression data*, in Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB), 2000, pp. 93–103.
[23] S. C. Madeira and A. L. Oliveira, *Biclustering algorithms for biological data analysis: A survey*, IEEE/ACM Trans. Comput. Biol. Bioinform., 1 (2004), pp. 24–45.
[24] S. Lonardi, W. Szpankowski, and Q. Yang, *Finding biclusters by random projections*, in Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Istanbul, Turkey, 2004, pp. 102–116.
[25] R. Peeters, *The maximum edge biclique problem is NP-complete*, Discrete Appl. Math., 131 (2003), pp. 651–654.
[26] M. Li, B. Ma, and L. Wang, *On the closest string and substring problems*, J. ACM, 49 (2002), pp. 157–171.
[27] D. Gillman, *A Chernoff bound for random walks on expander graphs*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 680–691.
[28] M. Koyutürk, W. Szpankowski, and A. Grama, *Biclustering gene-feature matrices for statistically significant dense patterns*, in Proceedings of the 2004 Annual IEEE Bioinformatics Conference, 2004, pp. 480–484.
[29] S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church, *Systematic determination of genetic network architecture*, Nature Genetics, 22 (1999), pp. 281–285.
[30] S. Arora, D. Karger, and M. Karpinski, *Polynomial time approximation schemes for dense instances of NP-hard problems*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 284–293.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1519–1532

A DETERMINISTIC SUBEXPONENTIAL ALGORITHM FOR SOLVING PARITY GAMES*

MARCIN JURDZIŃSKI†, MIKE PATERSON†, AND URI ZWICK‡

Abstract. The existence of polynomial-time algorithms for the solution of parity games is a major open problem. The fastest known algorithms for the problem are *randomized* algorithms that run in subexponential time. These algorithms are all ultimately based on the randomized subexponential simplex algorithms of Kalai and of Matoušek, Sharir, and Welzl. Randomness seems to play an essential role in these algorithms. We use a completely different, and elementary, approach to obtain a *deterministic* subexponential algorithm for the solution of parity games. The new algorithm, like the existing randomized subexponential algorithms, uses only polynomial space, and it is almost as fast as the randomized subexponential algorithms mentioned above.

Key words. analysis of algorithms and problem complexity, specification and verification, 2-player games, games on graphs, discrete-time games

AMS subject classifications. 68Q25, 68Q60, 91A05, 91A43, 91A50

DOI. 10.1137/070686652

1. Introduction.

A parity game [11, 15] is played on a directed graph (V, E) by two players, *Even* and *Odd*, who move a token from vertex to vertex along the edges of the graph so that an infinite path is formed. A partition (V_0, V_1) is given of the set V of vertices: player Even moves if the token is at a vertex of V_0, and player Odd moves if the token is at a vertex of V_1. Finally, a priority function $p : V \to \mathbb{N}$ is given. The players compete for the parity of the highest priority occurring infinitely often: player Even wins if $\limsup_{i\to\infty} p(v_i)$ is even, while player Odd wins if it is odd, where $v_0, v_1, v_2, \ldots$ is the infinite path formed by the players.

The algorithmic problem of solving parity games is, given a parity game $G = (V_0, V_1, E, p)$ and an initial vertex $v_0 \in V$, to determine whether player Even has a winning strategy in the game if the token is initially placed on vertex v_0. Algorithms for solving parity games [33, 20, 32, 15, 2] usually compute the winning sets win_0 and win_1, i.e., the sets of vertices from which players Even and Odd, respectively, have a winning strategy. By the determinacy theorem for parity games [11, 15] the winning sets win_0 and win_1 form a partition of the set of vertices V. None of these algorithms is known to run in polynomial time and the existence of a polynomial-time algorithm for the solution of parity games is a long-standing open problem [12, 15].

The original motivation for the study of parity games comes from the area of formal verification of systems by temporal logic model checking [5, 15]. The problem of solving parity games is polynomial-time equivalent to the nonemptiness problem of ω-automata on infinite trees with Rabin-chain acceptance conditions [12], and to the model checking problem of the modal μ-calculus (modal fixpoint logic) which is a formalism of great expressiveness and succinctness in formal specification and

*Received by the editors March 28, 2007; accepted for publication (in revised form) July 18, 2008; published electronically November 12, 2008. This paper is an updated and extended version of the SODA'06 paper [22]. The work was partially supported by the London Mathematical Society, by DIMAP (the Centre for Discrete Mathematics and its Applications, EPSRC grant EP/D063191/1), and by EPSRC grant EP/E022030/1.

http://www.siam.org/journals/sicomp/38-4/68665.html

†Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK (Marcin.Jurdzinski@dcs.warwick.ac.uk, msp@dcs.warwick.ac.uk).

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (zwick@cs.tau.ac.il).

validation [10, 15]. The model checking problem is a fundamental algorithmic problem in automated hardware and software verification [10, 5].

Another important motivation to study the problem of solving parity games is its intriguing complexity theoretic status: the problem is known to be in NP ∩ co-NP [12] and even in UP ∩ co-UP [19] but, as mentioned, despite considerable efforts of the community [12, 20, 32, 15, 2, 28] no polynomial-time algorithm has been found so far. (The complexity class UP, a.k.a. unambiguous NP, is defined to contain all problems that can be recognized by an unambiguous nondeterministic polynomial-time Turing machine. A Turing machine is unambiguous if for every input it has at most one accepting computation. Clearly, the inclusions P ⊆ UP ⊆ NP hold.) Moreover, parity games are polynomial-time reducible to mean payoff games [34], simple stochastic games [6], and discounted payoff games [6, 34]. A stochastic generalization of parity games was also studied [9, 3]. The problems of solving all those games are in UP ∩ co-UP as well [19, 3]. Condon has shown that simple stochastic games are complete (with respect to log-space reductions) in the class of log-space randomized alternating Turing machines [6].

The task of solving parity, mean payoff, discounted payoff, and simple stochastic games can be also viewed as a search problem: given a game graph, compute optimal strategies for both players. The value functions used in strategy improvement algorithms [7, 25, 32, 2] witness membership of all those optimal strategies search problems in PLS (i.e., the class of polynomial local search problems) [17]. On the other hand, the problem of computing optimal strategies in simple stochastic games can be reduced in polynomial time to solving a P-matrix linear complementarity problem [14, 31, 21], and to finding a Brouwer fixpoint [18], and hence it is also in PPAD (see [29] for a definition of PPAD). It follows that there are polynomial-time reductions from the problems of computing optimal strategies in parity, mean payoff, discounted payoff, and simple stochastic games to the problem of finding Nash equilibria in bimatrix games [8, 4].

Let $n = |V|$ and $m = |E|$ be the numbers of vertices and edges of a parity game graph and let d be the number of different priorities assigned to vertices by the priority function $p : V \to \mathbb{N}$. For parity games with a small number of priorities, more specifically if $d = O(n^{1/2})$, the progress-measure lifting algorithm [20] gave, until recently, the best time complexity of $O(dm(2n/d)^{d/2})$. This has been improved by Schewe [30] to $O(m(\kappa n/d)^{d/3})$, where $\kappa \leq (2e)^{3/2}$. If $d = \Omega(n^{(1/2)+\varepsilon})$, then the randomized algorithm of Björklund, Sandberg, and Vorobyov [2] has a better (expected) running-time bound of $n^{O(\sqrt{n/\log n})}$.

The main contribution of this paper is a *deterministic* algorithm for solving parity games which achieves roughly the same complexity as the randomized algorithm of Björklund, Sandberg, and Vorobyov [2]: the complexity of our algorithm is $n^{O(\sqrt{n/\log n})}$ if the out-degree of all vertices is bounded, and is $n^{O(\sqrt{n})}$ otherwise. The new algorithm uses only polynomial space.

The randomized algorithm of Björklund, Sandberg, and Vorobyov [2] is based on the randomized algorithm of Ludwig [25] for simple stochastic games, which in turn is inspired by the subexponential randomized simplex algorithms for linear programming and LP-type problems by Kalai [23] and by Matoušek, Sharir, and Welzl [26]. For games with out-degree two these algorithms are instantiations of the Random-Facet algorithm for finding the unique sink in an acyclic unique sink orientation (AUSO) of a hypercube [13]. The nodes of a hypercube correspond to positional strategies for one of the players, and the orientation of an edge connecting two positional strategies

that differ at exactly one vertex is determined by which of the two strategies has a better value when the opponent plays a best-response strategy [7, 25, 32, 2].

In contrast, our deterministic algorithm for parity games is obtained by a modification of a more elementary algorithm of McNaughton and Zielonka for parity games [33, 15]. The methods we use are thus very different from those of Ludwig and Björklund et al. Our method is applicable, so it seems, only to parity games, while the randomized algorithms for finding the unique sink in an AUSO [13] and for solving an LP-type problem [26] can be applied to a number of problems, including computing the values of parity, mean payoff, discounted payoff, and simple stochastic games [2, 1, 16].

The recent improvement of the complexity of solving parity games with a small number of priorities due to Schewe [30] was inspired by the preliminary version of this paper published at SODA'06 [22]. Schewe refined our technique of searching and removing dominions while running the classical recursive algorithm [27, 33]: instead of a brute-force search he used a modification of the progress measure lifting algorithm [20].

2. Definitions. A *parity game* $G = (V_0, V_1, E, p)$ is composed of two disjoint sets of vertices V_0 and V_1, a set of directed edges $E \subseteq V \times V$, where $V = V_0 \cup V_1$, and a *priority* function $p : V_0 \cup V_1 \to \mathbb{N}$, defined on its vertices. Every vertex $u \in V$ has at least one outgoing edge $(u, v) \in E$. The game is played by two players: *Even*, also referred to as Player 0, and *Odd*, also referred to as Player 1.

The game starts at some vertex $v_0 \in V$. The players construct an infinite path (a play) as follows. Let u be the last vertex added so far to the path. If $u \in V_0$, then Player 0 chooses an edge $(u, v) \in E$. Otherwise, if $u \in V_1$, then Player 1 chooses an edge $(u, v) \in E$. In either case, vertex v is added to the path, and a new edge is then chosen by either Player 0 or Player 1. As each vertex has at least one outgoing edge, the path constructed can always be continued.

Let $v_0, v_1, v_2, \ldots$ be the infinite path constructed by the two players, and let $p(v_0), p(v_1), p(v_2), \ldots$ be the sequence of the priorities of the vertices on the path. Player 0 wins the play if the largest priority seen infinitely many times is even, and Player 1 wins otherwise. Observe that removing an arbitrary finite prefix of a play in a parity game does not change the winner; we refer to this property of parity games as *prefix independence*.

A *strategy* for Player i in a game G specifies, for every finite path $v_0, v_1, \ldots, v_k$ in G that ends in a vertex $v_k \in V_i$, an edge $(v_k, v_{k+1}) \in E$. A strategy is said to be a *positional* strategy if the edge $(v_k, v_{k+1}) \in E$ chosen depends only on v_k, the last vertex visited. A strategy for Player i is said to be a *winning* strategy if using this strategy ensures a win for Player i, no matter which strategy is used by the other player. The determinacy theorem for parity games [11, 15] says that for every parity game G and every start vertex v_0, either Player 0 has a winning strategy or Player 1 has a winning strategy. (This claim is not immediate as the games considered are infinite.) Furthermore, if a player has a winning strategy from a vertex in a parity game, then she also has a winning positional strategy from this vertex.

In the parity game G illustrated in Figure 2.1, the initial vertex is labeled a, *Even*'s vertices are represented as squares (even number of sides), and *Odd*'s as triangles. The numbers within the vertices show priorities. Note that each player's vertices may have both even and odd parities. As an example of a play in G, if each player were to choose the double-headed arrow out of each of their vertices, then the infinite path formed would be $a, d, b, e, d, b, e, \ldots$, and the largest priority seen infinitely often would be 4 at vertex e. So *Even* would win this play.

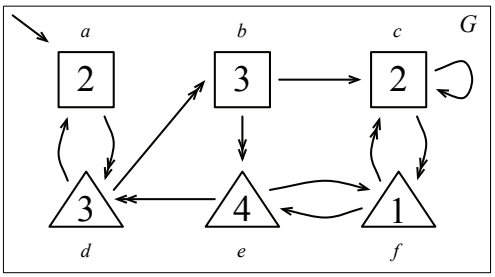


FIG. 2.1. *A parity game G. Double-headed arrows show a positional strategy for each player.*

The winning set for Player i, denoted by $win_i(G)$, is the set of vertices of the game from which Player i has a winning strategy. By the determinacy theorem for parity games [11, 15] we have that $win_0(G) \cup win_1(G) = V$.

3. Overview of the new algorithm. The previously known deterministic algorithm [33, 15] on which our improvement is built will be described fully in section 5. It has a recursive structure: solving a game with n vertices may require two recursive calls to smaller games. In the worst case, each of these games may have $n-1$ vertices, resulting in a running time satisfying the recurrence $T(n) \leq 2T(n-1)+O(n^2)$, which yields $T(n) = O(2^n)$. We offer no improvement in the first of the two recursive calls, but we do take advantage of a special feature of the second of these.

We introduce the notion of a *dominion*. An i-dominion, as the name suggests, is a set of vertices D *ruled* by Player i, in the sense that Player i can win from every vertex of D, without leaving D and without allowing the other player to leave D. One example of an i-dominion would be the whole of $win_i(G)$, but there may well be other "i-closed" subsets of $win_i(G)$ which are i-dominions. Although finding i-dominions can be just as hard as finding $win_i(G)$, we show that searching for small enough dominions is feasible, though taking time exponential in the size of dominion sought.

The significance of dominions for our algorithm depends on two properties. First, every i-dominion found can be easily removed at small cost, leaving a smaller game to be solved. Second, the second of the recursive calls is to a game resulting from the removal of a dominion. Therefore, if we look for and then remove all small dominions before entering the recursive calls, we can be sure that the second recursive call is to a substantially reduced game. The corresponding recurrence is then of the form $T(n) \leq T(n-1) + T(n-\ell) + O(n^\ell)$ for some $\ell = \ell(n)$.

With an appropriate choice of $\ell(n)$ to achieve a balance between the time to search for dominions of size up to ℓ and the savings from avoidance of the worst cases of the second recursive call, we achieve our subexponential algorithm with running time $n^{O(\sqrt{n})}$.

In the next section we prepare for the algorithms by introducing some key notions (*i-closed* and *reachability* set) and proving some of their properties. Lemmas 4.5 and 4.6 lay the foundation for the exponential algorithm. They show that we can begin to solve a game G by considering the set A of vertices with highest priority which Player i (say) would like the play to visit infinitely often, and identifying the set A^* from which Player i can guarantee to reach A at least once. From Lemma 4.6, we see that by first solving the smaller game G' based on vertices in $V(G) \setminus A^*$ we can identify a subset U of the winning set of the opponent of Player i, say Player j, in G.

By Lemma 4.5 we then know that the set U^*, from which Player j can guarantee to reach U at least once, is also included in the winning set of Player j in G. Moreover, Lemma 4.5 establishes that the winning set of Player i in G is equal to her winning set in the smaller game based on vertices in $V(G) \setminus U^*$, and hence the task of solving the game G is reduced to the task of solving the smaller game. For an illustration, see Figure 5.2, where $A^* = \mathit{reach}_i(A)$, $U = W'_j$, and $U^* = \mathit{reach}_j(W'_j)$.

After giving details of the exponential algorithm in section 5, we show in section 6 how to find dominions. In sections 7 and 8 we integrate this search-and-remove process into our new algorithm and analyze the resulting running time.

4. Preliminaries. The results presented in this section are well known [15] and form the basis of algorithms by McNaughton [27] and Zielonka [33]. We include our detailed exposition of them here in order to fix the terminology and to make the paper self-contained.

A set $B \subseteq V$ is said to be *i-closed*, where $i \in \{0,1\}$, if for every $u \in B$ the following hold:

- if $u \in V_i$, then there is some $(u,v) \in E$, such that $v \in B$; and
- if $u \in V_{\neg i}$, then for every $(u,v) \in E$, we have $v \in B$.

(We use $\neg i$ for the element $(1-i)$ in $\{0,1\}$.) In other words, a set B is i-closed if Player i can always choose to stay in B while Player $\neg i$ cannot escape from it, i.e., B is a "trap" for Player $\neg i$.

LEMMA 4.1. *For each $i \in \{0,1\}$, the set $\mathit{win}_i(G)$ is i-closed.*

Proof. The proof is straightforward from the definitions and prefix independence of parity games. □

Let $A \subseteq V$ be an arbitrary set. The *i-reachability set* of A, denoted $\mathit{reach}_i(A)$, contains all vertices in A and all vertices from which Player i has a strategy to enter the set A at least once; we call such a strategy an *i-reachability strategy to* set A. (See Figure 4.1 for a simple example.)

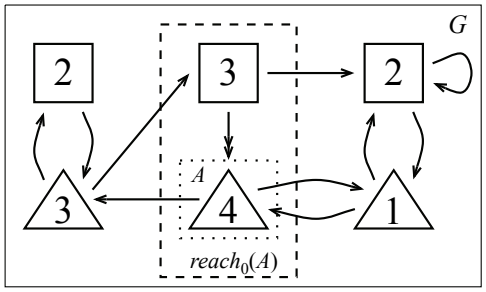


FIG. 4.1. *The 0-reachability set of A and a positional 0-reachability strategy to set A.*

LEMMA 4.2. *For every set $A \subseteq V$ and $i \in \{0,1\}$, the set $V \setminus \mathit{reach}_i(A)$ is $(\neg i)$-closed.*

Proof. Let $u \in V \setminus \mathit{reach}_i(A)$. Recall that every vertex has at least one outgoing edge; hence if $u \in V_{\neg i}$, then there must be an edge $(u,v) \in E$ from vertex u into the set $V \setminus \mathit{reach}_i(A)$, i.e., such that $v \notin \mathit{reach}_i(A)$, since otherwise vertex u would be in $\mathit{reach}_i(A)$. Similarly, if $u \in V_i$, then all edges from vertex u must go into the set $V \setminus \mathit{reach}_i(A)$. Therefore, the set $V \setminus \mathit{reach}_i(A)$ is $(\neg i)$-closed. □

LEMMA 4.3. *For every set $A \subseteq V$ and $i \in \{0,1\}$, the set $\mathrm{reach}_i(A)$ can be computed in $O(m)$ time, where $m = |E|$ is the number of edges in the game.*

Proof. The vertices of A are in $reach_i(A)$ so we initialize $B \leftarrow A$. We then iteratively add to B every vertex of V_i that has at least one edge going into B, and every vertex of $V_{\neg i}$ all of whose edges go into B. We stop when no new vertices can be added to B.

Some care is needed to keep the time in $O(m)$. One method is to maintain an adjacency list giving, for each vertex, the *incoming* edges together with a count of the number of outgoing edges. At each step we take an edge (u, v) entering B and delete it (from the list of edges entering v and from the count of edges leaving u): if $u \in B$, then nothing more is done; otherwise, if $u \in V_i$, then u is added to B; otherwise (when $u \in V_{\neg i} \setminus B$), if there are no other edges from u, then u is added to B.

It is easy to see that this process can be performed in $O(m)$ time, and that when it ends we have $B = reach_i(A)$, as required. □

If $B \subseteq V$ is such that for every vertex $u \in V \setminus B$ there is an edge (u, v) with $v \in V \setminus B$, then the *subgame* $G \setminus B$ is the game obtained from G by removing the vertices of B and all the edges that touch them. We will only be using B's for which $V \setminus B$ is an i-closed set for some i. In such cases $G \setminus B$ is always well-defined. The next lemmas show some useful properties of subgames.

LEMMA 4.4. *Let G' be a subgame of G and let $i \in \{0, 1\}$. If V', the vertex set of G', is i-closed in G, then $win_i(G') \subseteq win_i(G)$.*

Proof. A winning strategy for Player i from the set $win_i(G')$ in the subgame G' is also winning for her from the same set in the original game G. Player $\neg i$ cannot escape to $V \setminus V'$, since the set V' is i-closed in G. □

The following lemma implies that if we know an arbitrary nonempty subset U of the winning set of a player, say Player j, in a game G, then computing the winning sets of both players in G can be reduced to computing their winning sets in the smaller game $G \setminus reach_j(U)$.

LEMMA 4.5. *Let G be a parity game, let $i \in \{ 0, 1 \}$ and $j = \neg i$. If $U \subseteq win_j(G)$ and $U^* = reach_j(U)$, then $win_j(G) = U^* \cup win_j(G \setminus U^*)$ and $win_i(G) = win_i(G \setminus U^*)$.*

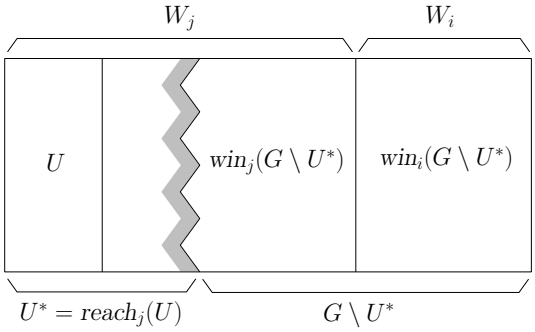


FIG. 4.2. *Diagram illustrating Lemma* 4.5.

Proof. Let $W_j = U^* \cup win_j(G \setminus U^*)$ and $W_i = win_i(G \setminus U^*)$; see Figure 4.2. Since (W_i, W_j) is a partition of V, it suffices to show that $W_i \subseteq win_i(G)$ and $W_j \subseteq win_j(G)$. By Lemma 4.2, $V \setminus U^*$, the vertex set of $G \setminus U^*$, is i-closed. The first inclusion then follows from Lemma 4.4.

To show the second inclusion, we exhibit a strategy for Player j that is winning for her from the set W_j in the game G. By the assumption that $U \subseteq win_j(G)$, there is a strategy σ for Player j in the game G which is winning for her from all vertices in U. Let τ be a winning strategy for Player j from the set $win_j(G \setminus U^*)$ in the

subgame $G \setminus U^*$. A strategy π for Player j in the game G is made by composing strategies τ and σ in the following way: if the play so far is contained in the set $win_j(G \setminus U^*)$, then follow strategy τ; otherwise use the j-reachability strategy to the set U and "restart" the play following the strategy σ thenceforth. The strategy π is well-defined because, by Lemma 4.1, Player i can escape from $win_j(G \setminus U^*)$ only into the set U^*. By prefix independence of parity games, the strategy π is a winning strategy for Player j, because if it ever switches from following τ to following σ, then an infinite suffix of the play is winning for Player j. □

The next lemma complements Lemma 4.5 by providing an algorithmic method which either finds a nonempty subset of the winning set of a player, say Player j, in a parity game G, or (if it returns an empty set) concludes that Player $\neg j$ can win from every vertex in G.

LEMMA 4.6. *Let G be a parity game. Let $d = d(G)$ be the highest priority and let $A = A_d(G)$ be the set of vertices of highest priority. Let $i = d \bmod 2$ and $j = \neg i$. Let $G' = G \setminus reach_i(A)$. Then we have $win_j(G') \subseteq win_j(G)$. Also, if $win_j(G') = \emptyset$, then $win_i(G) = V(G)$, i.e., Player i wins from every vertex of G.*

(As an example, consider Figures 4.1 and 4.3, with $i = 0$ and $j = 1$.)

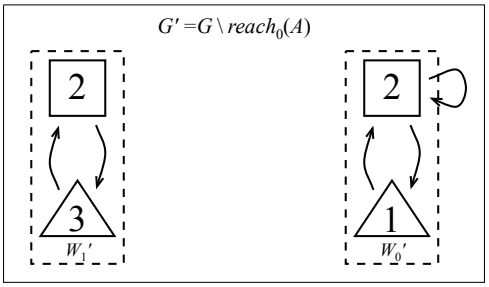


FIG. 4.3. *The game $G' = G \setminus reach_0(A)$ and winning sets $W'_i = win_i(G')$ for $i = 0, 1$.*

Proof. That $win_j(G') \subseteq win_j(G)$ follows from Lemmas 4.2 and 4.4.

Suppose now that $win_j(G') = \emptyset$. Let τ be a winning strategy for Player i from $win_i(G')$ (which, by determinacy, is equal to $V \setminus reach_i(A)$) in the subgame G'. We construct a strategy π for Player i in the following way: if a play so far is contained in the set $win_i(G')$, then follow strategy τ; otherwise the current vertex is in $reach_i(A)$ so follow the i-reachability strategy to the set A; moreover, each time the play reenters the set $win_i(G')$ "restart" the play and follow strategy τ, etc. If a play following the strategy π visits $reach_i(A)$ (and hence A) infinitely often, then it is winning for Player i because $i = d \bmod 2$. Otherwise, it has an infinite suffix played according to strategy τ, and hence it is winning for Player i by prefix independence of parity games. □

5. An exponential algorithm. A simple exponential-time algorithm for the solution of parity games is given in Figure 5.1. This algorithm originates from the work of McNaughton [27] and was first presented for parity games by Zielonka [33, 15]. Algorithm **win**(G) receives a parity game G and returns the pair of winning sets $(win_0(G), win_1(G))$ for the two players.

Algorithm **win**(G) is based on Lemmas 4.5 and 4.6. It starts by letting d be the largest priority in G and by letting A be the set of vertices having this highest priority. Let $i = d \bmod 2$ be the index of the player associated with the highest priority, and

```
algorithm win(G)
if V(G) = ∅ then return (∅, ∅)
d ← d(G) ; A ← A_d(G)
i ← d mod 2 ; j ← ¬i
(W'_0, W'_1) ← win(G \ reach_i(A))
if W'_j = ∅ then
   (W_i, W_j) ← (V(G), ∅)
else
   (W''_0, W''_1) ← win(G \ reach_j(W'_j))
   (W_i, W_j) ← (W''_i, V(G) \ W''_i)
endif
return (W_0, W_1)
```

FIG. 5.1. *An exponential algorithm for solving parity games.*

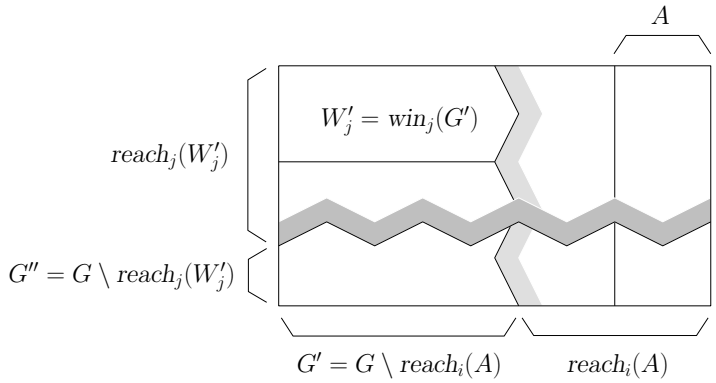


FIG. 5.2. *A game G and its subgames $G' = G \setminus reach_i(A)$ and $G'' = G \setminus reach_j(W'_j)$.*

let $j = \neg i$ be the index of the other player. The algorithm first finds the winning sets (W'_0, W'_1) of the smaller game $G' = G \setminus reach_i(A)$, using a recursive call; see Figure 5.2. By Lemma 4.6, if $W'_j = \emptyset$, then Player i wins from all vertices of G and we are done. Otherwise, again by Lemma 4.6, we know that $W'_j \subseteq win_j(G)$. The algorithm then finds the winning sets (W''_0, W''_1) of the smaller game $G'' = G \setminus reach_j(W'_j)$ by a second recursive call. By Lemma 4.5, we then know that $win_i(G) = W''_i$ and $win_j(G) = reach_j(W'_j) \cup W''_j = V(G) \setminus W''_i$.

A small detailed illustration of the main steps of the algorithm is given in Figures 4.1, 4.3, 5.3, and 5.4.

THEOREM 5.1. *Algorithm* $\mathbf{win}(G)$ *correctly finds the winning sets of the parity game G. Its running time is $O(2^n)$, where $n = |V(G)|$ is the number of vertices in G.*

Proof. The correctness of the algorithm follows from Lemmas 4.5 and 4.6, as argued above. Let $T(n)$ be the maximum running time of algorithm $\mathbf{win}(G)$ for a game on at most n vertices. Algorithm $\mathbf{win}(G)$ makes two recursive calls $\mathbf{win}(G')$ and $\mathbf{win}(G'')$ on games with at most $n-1$ vertices. Other than that, it performs only $O(n^2)$ operations. (The most time-consuming operations are the computations of the sets $reach_i(A)$ and $reach_j(W'_j)$.) Thus $T(n) \leq 2T(n-1) + O(n^2)$. It is easy to see then that $T(n) = O(2^n)$. □

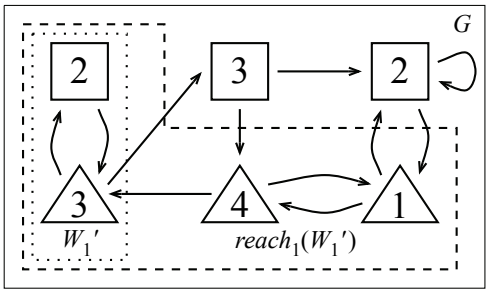


FIG. 5.3. *The* 1*-reachability set of* W_1'.

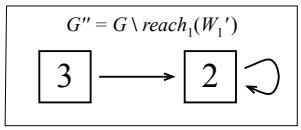


FIG. 5.4. *The game* $G'' = G \setminus reach_1(W_1')$.

6. Finding small dominions. A set $D \subseteq V(G)$ is said to be an *i-dominion* if Player i can win from every vertex of D without ever leaving D. Note, in particular, that an i-dominion must be i-closed. A set $D \subseteq V(G)$ is said to be a *dominion* if it is either a 0-dominion or a 1-dominion. By prefix independence of parity games, the winning set $win_i(G)$ of Player i is an i-dominion.

LEMMA 6.1. *Let* G *be a parity game on* n *vertices and let* $\ell \leq n/3$. *There is an* $O(n^\ell)$*-time algorithm that finds a nonempty dominion in* G *of size at most* ℓ, *or determines that no such dominion exists.*

Proof. If $\ell \leq n/3$, then, for all $j \leq \ell$, we have that $\binom{n}{j}/\binom{n}{j-1} > 2$. The number $\sum_{j=1}^{\ell}\binom{n}{j}$ of subsets of V of size at most ℓ is therefore at most $2\binom{n}{\ell}$. For each such subset U we check, in $O(\ell^2)$ time, whether it is 0-closed or 1-closed. If both tests fail, then U is clearly not a dominion. If U is i-closed, for some $i \in \{0,1\}$, we form the game $G[U]$, which is the game G restricted to U. This is well-defined since U is i-closed. We now apply the exponential algorithm of the previous section to $G[U]$ and find out, in $O(2^\ell)$ time, whether Player i can win from all the vertices of $G[U]$. If so, then U is an i-dominion; otherwise it is not. The total running time of the algorithm is therefore $O(\binom{n}{\ell}2^\ell) = O(n^\ell)$, as required. ∎

In a game with bounded out-degrees we can find small dominions even faster. For simplicity, the lemma below and the analysis in section 8 are stated for games in which the out-degree of every vertex is exactly two. Note, however, that for every constant b, every game on n vertices with out-degrees at most b can be easily converted into an equivalent game on at most $n(b-1)$ vertices with out-degrees exactly two, by replacing each higher-degree vertex by a binary tree.

LEMMA 6.2. *Let* G *be a parity game on* n *vertices in which the out-degree of each vertex is two. There is an* $O(n2^\ell \ell \log \ell)$*-time algorithm that finds a nonempty dominion in* G *of size at most* ℓ, *or determines that no such dominion exists.*

Proof. Assume, without loss of generality, that the vertices of G are numbered from 1 to n. Let $u \in V$ be a vertex of G and let (u, v_0) and (u, v_1) be the two edges emanating from v, where $v_0 \leq v_1$. We say that (u, v_0) is the 0th outgoing edge of v, while (u, v_1) is the 1st outgoing edge of v.

The algorithm generates at most $O(n2^\ell)$ 0-closed sets of size at most ℓ that are candidates for being 0-dominions. For every vertex $v \in V$ and a binary sequence $\langle a_1, \ldots, a_\ell \rangle \in \{0,1\}^\ell$, construct a set $U \subset V$ as follows. Start with $U = \{v\}$ and $r = 1$. Vertices added to U are initially unmarked. As long as there is still an unmarked vertex in U, pick the smallest such vertex $u \in U$ and mark it. If $u \in V_0$, then add the endpoint of the a_rth outgoing edge of u to U, if it is not already there,

and increment r. If $u \in V_1$, then add the endpoints of both the outgoing edges of u to U. If at some stage $|U| > \ell$, then discard the set U and restart the construction with the next binary sequence.

If the process above ends with $|U| \leq \ell$, then a 0-closed set of size at most ℓ has been found. Furthermore, for every vertex $u \in U \cap V_0$, one of the outgoing edges of u was selected. This corresponds to a suggested strategy for Player 0 in the game $G[U]$. Our algorithm therefore considers by exhaustive search all 0-closed sets of at most ℓ vertices, and for each set considers all possible positional strategies for Player 0.

Using an algorithm of King, Kupferman, and Vardi [24] we can check, in $O(\ell \log \ell)$ time, whether a given U and proposed strategy is indeed a winning strategy for Player 0 from all the vertices of U. Thus, if there is a 0-dominion of size at most ℓ in G, then the algorithm will find one. Finding 1-dominions of size at most ℓ can be done in an analogous manner. □

The algorithm described in Lemma 6.1 finds some i-dominion D if there is a dominion of size at most ℓ. We denote this algorithm by **dominion**(G, ℓ), and suppose that it returns either the pair (D, i) if successful, or $(\emptyset, -1)$ if not.

7. The new subexponential algorithm. The new algorithm for solving parity games is given in Figure 7.1. The algorithm **new-win** starts by trying to find a dominion of size at most ℓ, where $\ell = \lceil \sqrt{2n} \rceil$ (and $\ell = \lceil \sqrt{n \log n} \rceil$ for games with bounded out-degree) is a parameter chosen to minimize the running time of the whole algorithm. If such a small i-dominion is found, then it is easy to remove it, as well as its i-reachability set, from the game and recurse on what is left over. If no small

```
algorithm new-win(G)
if V(G) = ∅ then return (∅, ∅)
n ← |V(G)| ; ℓ ← ⌈√(2n)⌉
(D, i) ← dominion(G, ℓ); j ← ¬i
if D = ∅ then
    (W_0, W_1) ← old-win(G)
else
    (W'_0, W'_1) ← new-win(G \ reach_i(D))
    (W_j, W_i) ← (W'_j, V(G) \ W'_j)
endif
return(W_0, W_1)
```

```
algorithm old-win(G)
d ← d(G) ; A ← A_d(G)
i ← d mod 2 ; j ← ¬i
(W'_0, W'_1) ← new-win(G \ reach_i(A))
if W'_j = ∅ then
    (W_i, W_j) ← (V(G), ∅)
else
    (W''_0, W''_1) ← new-win(G \ reach_j(W'_j))
    (W_i, W_j) ← (W''_i, V(G) \ W''_i)
endif
return (W_0, W_1)
```

FIG. 7.1. *The new subexponential algorithm for solving parity games.*

dominion is found, then **new-win**(G) simply calls algorithm **old-win**(G), which is almost identical to the exponential algorithm **win**(G) of section 5. The only difference between **old-win**(G) and **win**(G) is that the recursive calls are made to **new-win**(G) and not to **win**(G).

THEOREM 7.1. *Algorithm* **new-win**(G) *correctly finds the winning sets of a parity game G. Its running time on a game with n vertices is $n^{O(\sqrt{n})}$.*

Proof. The correctness of the algorithm is immediate. We next analyze its running time. Let $T(n)$ be the maximum running time of **new-win**(G) on a game with at most n vertices.

Algorithm **new-win**(G) tries to find dominions of size at most $\ell = \lceil \sqrt{2n} \rceil$. By Lemma 6.1 this takes $O(n^\ell)$ time. If a nonempty dominion is found, then the algorithm simply proceeds on the remaining game, which has at most $n-1$ vertices, and the remaining running time is therefore at most $T(n-1)$. Otherwise, a call to **old-win**(G) is made. This results in a call to **new-win**$(G \setminus \mathit{reach}_i(A))$, which takes at most $T(n-1)$ time. If the set W'_j returned by the call is empty, then we are done. Otherwise, $W'_j = \mathit{win}_j(G \setminus \mathit{reach}_i(A))$, and this is equal to $\mathit{win}_j(G)$ by Lemma 4.5. Therefore W'_j is a j-dominion of G. We are in the case that there is no small dominion in G, so we know that $|W'_j| > \ell$, and therefore the second recursive call **new-win**$(G \setminus \mathit{reach}_j(W'_j))$ takes at most $T(n-\ell)$ time. Thus we get

$$T(n) \;\le\; O(n^\ell) + T(n-1) + T(n-\ell).$$

This recurrence relation, with $\ell = \lceil \sqrt{2n} \rceil$, is analyzed in Theorem 8.1, where it is shown that $T(n) = n^{O(\sqrt{n})}$. ∎

A slightly better bound is achieved for graphs with out-degree two.

THEOREM 7.2. *Consider the algorithm* **new-win**(G) *in which the variable ℓ is set to $\lceil \sqrt{n \log n} \rceil$. If the game G has n vertices and the out-degree of each of them is two, then the running time of the modified algorithm is $n^{O(\sqrt{n/\log n})}$.*

Proof. Note that if $\ell = \lceil \sqrt{n \log n} \rceil$, then $O(n 2^\ell \ell \log \ell) = n^{O(\sqrt{n/\log n})}$. Therefore, by Lemma 6.2 and by the analysis in the proof of the previous theorem, the time complexity $T(n)$ satisfies the following recurrence:

$$T(n) \;\le\; n^{O(\sqrt{n/\log n})} + T(n-1) + T(n-\ell).$$

The recurrence is analyzed in the following section in Theorem 8.2, where we show that $T(n) = n^{O(\sqrt{n/\log n})}$. ∎

8. Solving the recurrence relations. In this section we analyze the recurrence relations used in the previous section to bound the running time of the new algorithm.

We start by analyzing the recurrence relation used to bound the running time of the algorithm for game graphs with arbitrary out-degrees.

THEOREM 8.1. *If $T(n)$ is a positive function such that, for every $n > 3$,*

$$T(n) \;\le\; O(n^\ell) + T(n-1) + T(n-\ell),$$

where $\ell = \lceil \sqrt{2n} \rceil$, then $T(n) = n^{O(\sqrt{n})}$.

Proof. For every integer n we construct a binary tree $\mathcal{T}_n$ in the following way. The root of $\mathcal{T}_n$ is labeled by n. A node labeled by a number $k > 3$ has two children: a left child labeled by $k-1$ and a right child labeled by $k - \lceil \sqrt{2k} \rceil$. Nodes labeled by the numbers 1, 2, and 3 are leaves. A node labeled by k has a cost of $k^{O(\sqrt{2k})}$

associated with it. It is easy to see that the sum of the costs of the nodes of $\mathcal{T}_n$ is an upper bound on $T(n)$.

Clearly, the length of every path in $\mathcal{T}_n$ from the root to a leaf is at most n. We say that such a path makes a *right turn* when it descends from a vertex to its right child. We next claim that each such path makes at most $\lfloor\sqrt{2n}\rfloor$ right turns. This follows immediately from the observation that the function $f(n) = n - \lceil\sqrt{2n}\rceil$ can be iterated on n at most $\lfloor\sqrt{2n}\rfloor$ times before reaching the value of 3 or less. This observation can be proved by induction, based on the fact that if $\frac{1}{2}j^2 < n \leq \frac{1}{2}(j+1)^2$, then $n - \lceil\sqrt{2n}\rceil \leq \frac{1}{2}j^2$. (Initially we have $j = \lfloor\sqrt{2n}\rfloor$ and finally, with $1 \leq n \leq 3$, we have $j \geq 1$.)

As each leaf of $\mathcal{T}_n$ is determined by the positions of the right turns on the path leading to it from the root, we get that the number of leaves in $\mathcal{T}_n$ is at most $\binom{n}{\lfloor\sqrt{2n}\rfloor}$. The total number of nodes in $\mathcal{T}_n$ is therefore at most $2\binom{n}{\lfloor\sqrt{2n}\rfloor}$. As the cost of each node is at most $n^{O(\sqrt{2n})}$, we immediately get that

$$T(n) \;\leq\; 2\binom{n}{\lfloor\sqrt{2n}\rfloor} n^{O(\sqrt{2n})} \;=\; n^{O(\sqrt{n})}\,,$$

as claimed. □

A more careful analysis, in which the $O(n^\ell)$ term in the recurrence relation is replaced by $O\left(\binom{n}{\ell}2^\ell\right)$, can be used to show that $T(n) = O((cn)^{\sqrt{n/2}})$, for some constant $c > 0$, and that the choice $\ell = \lceil\sqrt{2n}\,\rceil$ is essentially optimal.

The running time of the algorithm for graphs with out-degree two satisfies a tighter recurrence relation, which is analyzed similarly in the next theorem.

THEOREM 8.2. *If $T(n)$ is a positive function such that, for every $n > 3$,*

$$T(n) \leq n^{O(\sqrt{n/\log n}\,)} + T(n-1) + T(n-\ell),$$

where $\ell = \lceil\sqrt{n \log n}\,\rceil$, then $T(n) = n^{O(\sqrt{n/\log n}\,)}$.

Proof. The proof is similar to the proof of Theorem 8.1. For every integer n we again construct a tree $\mathcal{T}_n$. A node labeled by a number $k > 2$ now has a left child labeled by $k-1$ and a right child labeled by $k - \lceil\sqrt{k \log k}\,\rceil$. The cost of a node labeled by k is now $k^{O(k/\log k)}$. Every root-to-leaf path in $\mathcal{T}_n$ is again of length at most n, and it can now make at most $O(\sqrt{n/\log n})$ right turns. Thus, the number of nodes in $\mathcal{T}_n$ is at most $n^{O(\sqrt{n/\log n})}$. As the cost of each node is also $n^{O(\sqrt{n/\log n})}$, we get that $T(n) = n^{O(\sqrt{n/\log n})}$, as claimed. □

9. Concluding remarks. We have obtained the first deterministic subexponential algorithm for solving parity games. Our algorithm does not seem to extend in an obvious way to the solution of the more general mean payoff games and simple stochastic games. On the other hand, the techniques that we have introduced in this paper have recently inspired a notable improvement in the running-time complexity of parity games with a small number of priorities [30].

Acknowledgments. We thank a SODA'06 referee for suggestions that resulted in a significant simplification of the proofs of Theorems 8.1 and 8.2, and the journal referees whose comments helped us improve the presentation.

REFERENCES

[1] H. Björklund and S. Vorobyov, *A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games*, Discrete Appl. Math., 155 (2007), pp. 210–229.

[2] H. Björklund, S. Sandberg, and S. Vorobyov, *A discrete subexponential algorithm for parity games*, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 2607, Springer, Berlin, 2003, pp. 663–674.

[3] K. Chatterjee, M. Jurdziński, and T. A. Henzinger, *Quantitative stochastic parity games*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2004, pp. 114–123.

[4] X. Chen and X. Deng, *Settling the complexity of two-player Nash equilibrium*, in Proceedings of the 47th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, 2006, pp. 261–272.

[5] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, Cambridge, MA, 1999.

[6] A. Condon, *The complexity of stochastic games*, Inform. and Comput., 96 (1992), pp. 203–224.

[7] A. Condon, *On algorithms for simple stochastic games*, in Advances in Computational Complexity Theory, AMS, Providence, RI, 1993, pp. 51–73.

[8] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, *The complexity of computing a Nash equilibrium*, in Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 2006, pp. 71–78.

[9] L. de Alfaro and R. Majumdar, *Quantitative solution of omega-regular games*, J. Comput. System Sci., 68 (2004), pp. 374–397.

[10] E. A. Emerson, *Model checking and mu-calculus*, in Descriptive Complexity and Finite Models, N. Immerman and P. G. Kolaitis, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 31, AMS, Providence, RI, 1996, pp. 185–214.

[11] E. A. Emerson and C. Jutla, *Tree automata, μ-calculus, and determinacy*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, 1991, pp. 368–377.

[12] E. A. Emerson, C. S. Jutla, and A. P. Sistla, *On model-checking for fragments of μ-calculus*, in Computer-Aided Verification (CAV), Lecture Notes in Comput. Sci. 697, Springer, Berlin, 1993, pp. 385–396.

[13] B. Gärtner, *The Random-Facet simplex algorithm on combinatorial cubes*, Random Structures Algorithms, 20 (2002), pp. 353–381.

[14] B. Gärtner and L. Rüst, *Simple stochastic games and P-matrix generalized linear complementarity problems*, in Fundamentals of Computation Theory (FCT), Lecture Notes in Comput. Sci. 3623, Springer, Berlin, 2005, pp. 209–220.

[15] E. Grädel, W. Thomas, and T. Wilke, eds., *Automata, Logics, and Infinite Games. A Guide to Current Research*, Lecture Notes in Comput. Sci. 2500, Springer, Berlin, 2002.

[16] N. Halman, *Discrete and Lexicographic Helly Theorems and Their Relations to LP-type Problems*, Ph.D. thesis, Tel Aviv University, Tel Aviv, Israel, 2004.

[17] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, J. Comput. System Sci., 37 (1988), pp. 79–100.

[18] B. Juba, *On the Hardness of Simple Stochastic Games*, M.S. thesis, Carnegie Mellon University, Pittsburgh, PA, 2005.

[19] M. Jurdziński, *Deciding the winner in parity games is in $UP \cap co\text{-}UP$*, Inform. Process. Lett., 68 (1998), pp. 119–124.

[20] M. Jurdziński, *Small progress measures for solving parity games*, in Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 1770, Springer, Berlin, 2000, pp. 290–301.

[21] M. Jurdziński and R. Savani, *A simple P-matrix linear complementarity problem for discounted games*, in Computability in Europe (CiE), Lecture Notes in Comput. Sci. 5028, Springer, Berlin, 2008, pp. 283–293.

[22] M. Jurdziński, M. Paterson, and U. Zwick, *A deterministic subexponential algorithm for solving parity games*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2006, pp. 117–123.

[23] G. Kalai, *A subexponential randomized simplex algorithm (extended abstract)*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, New York, 1992, pp. 475–482.

[24] V. King, O. Kupferman, and M. Y. Vardi, *On the complexity of parity word automata*, in Foundations of Software Science and Computation Structures (FoSSaCS), Lecture Notes

in Comput. Sci. 2030, Springer, Berlin, 2001, pp. 276–286.
[25] W. Ludwig, *A subexponential randomized algorithm for the simple stochastic game problem*, Inform. and Comput., 117 (1995), pp. 151–155.
[26] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.
[27] R. McNaughton, *Infinite games played on finite graphs*, Ann. Pure Appl. Logic, 65 (1993), pp. 149–184.
[28] J. Obdržálek, *Fast mu-calculus model checking when tree-width is bounded*, in Computer-Aided Verification (CAV), Lecture Notes in Comput. Sci. 2725, Springer, Berlin, 2003, pp. 80–92.
[29] C. H. Papadimitriou, *On the complexity of the parity argument and other inefficient proofs of existence*, J. Comput. System Sci., 48 (1994), pp. 498–532.
[30] S. Schewe, *Solving parity games in big steps*, in Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Lecture Notes in Comput. Sci. 4855, Springer, Berlin, 2007, pp. 449–460.
[31] O. Svensson and S. Vorobyov, *Linear complementarity and P-matrices for stochastic games*, in Perspectives of Systems Informatics, Andrei Ershov Memorial Conference (PSI 2006), Lecture Notes in Comput. Sci. 4378, Springer, Berlin, 2007, pp. 409–423.
[32] J. Vöge and M. Jurdziński, *A discrete strategy improvement algorithm for solving parity games (extended abstract)*, in Computer-Aided Verification (CAV), Lecture Notes in Comput. Sci. 1855, Springer, Berlin, 2000, pp. 202–215.
[33] W. Zielonka, *Infinite games on finitely coloured graphs with applications to automata on infinite trees*, Theoret. Comput. Sci., 200 (1998), pp. 135–183.
[34] U. Zwick and M. Paterson, *The complexity of mean payoff games on graphs*, Theoret. Comput. Sci., 158 (1996), pp. 343–359.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1533–1573

LINEAR-TIME ALGORITHMS FOR DOMINATORS AND OTHER PATH-EVALUATION PROBLEMS*

ADAM L. BUCHSBAUM†, LOUKAS GEORGIADIS‡, HAIM KAPLAN§, ANNE ROGERS¶, ROBERT E. TARJAN‖, AND JEFFERY R. WESTBROOK**

Abstract. We present linear-time algorithms for the classic problem of finding dominators in a flowgraph, and for several other problems whose solutions require evaluating a function defined on paths in a tree. Although all these problems had linear-time solutions previously, our algorithms are simpler, in some cases substantially. Our improvements come from three new ideas: a refined analysis of path compression that gives a linear bound if the compressions favor certain nodes; replacement of random-access table look-up by a radix sort; and a more careful partitioning of a tree into easily managed parts. In addition to finding dominators, our algorithms find nearest common ancestors off-line, verify and construct minimum spanning trees, do interval analysis of a flowgraph, and build the component tree of a weighted tree. Our algorithms do not require the power of a random-access machine; they run in linear time on a pointer machine. The genesis of our work was the discovery of a subtle error in the analysis of a previous allegedly linear-time algorithm for finding dominators. That algorithm was an attempt to simplify a more complicated algorithm, which itself was intended to correct errors in a yet earlier algorithm. Our work provides a systematic study of the subtleties in the dominators problem, the techniques needed to solve it in linear time, and the range of application of the resulting methods. We have tried to make our techniques as simple and as general as possible and to understand exactly how earlier approaches to the dominators problem were either incorrect or overly complicated.

Key words. dominators, flowgraphs, pointer machine, random-access machine, set union, path compression, nearest common ancestors, minimum spanning trees, interval analysis, component tree, data structures, analysis of algorithms

AMS subject classifications. 05C85, 68N20, 68P05, 68Q05, 68Q25, 68W05

DOI. 10.1137/070693217

1. Introduction.

The story of this paper begins with the publication in 1979 [43] of an almost-linear-time algorithm for the problem of finding immediate domina-

*Received by the editors May 30, 2007; accepted for publication (in revised form) July 18, 2008; published electronically November 12, 2008. This work is partially covered by the extended abstracts *Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators*, in Proceedings of the 30th ACM Symposium on Theory of Computing, 1998, pp. 279–288, and *Finding dominators revisited*, in Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 862–871. The work of the second and fifth authors was partially done at Princeton University and partially supported by the National Science Foundation under the Aladdin Project, grant CCR-0122581. The work of the fifth author was also partially supported by NSF grants CCF-0830676 and CCF-0832797. The information contained herein does not necessarily reflect the opinion or policy of the federal government and no official endorsement should be inferred.
http://www.siam.org/journals/sicomp/38-4/69321.html

†Madison, NJ (alb@adambuchsbaum.com). This author's was work done while a member of AT&T Labs.

‡HP Labs, 1501 Page Mill Rd, Palo Alto, CA 94304. Current address: Informatics and Telecommunications Engineering Department, University of Western Macedonia, Kozani, Greece (lgeorg@uowm.gr).

§School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel (haimk@math.tau.ac.il). This author's work was partially done while a member of AT&T Labs.

¶Department of Computer Science, University of Chicago, 1100 E. 58th Street, Chicago, IL 60637 (amr@cs.uchicago.edu). This author's work was partially done while a member of AT&T Labs.

‖Department of Computer Science, Princeton University, Princeton NJ 08544, and HP Labs, 1501 Page Mill Rd., Palo Alto, CA 94304 (ret@cs.princeton.edu).

**Los Angeles, CA (jwestbrook@acm.org). This author's work was partially done while a member of AT&T Labs.

tors in a flowgraph. Computing dominators is a fundamental problem in the theory of directed graphs and is a crucial first step in global optimizations of computer code. Although versions of the 1979 dominators algorithm are very fast in practice [34] and are widely used, the question remained of whether dominators could be computed in linear time, in either the random-access-machine (RAM) model or the less powerful pointer-machine model. Harel [37] claimed to have a linear-time RAM algorithm, but Alstrup et al. [10] found problems in his work. They developed a correct but very complicated algorithm that uses powerful bit-manipulation techniques. Buchsbaum et al. [16] claimed to present a simpler linear-time algorithm, but Georgiadis and Tarjan [32] found a flaw in their analysis, which, when corrected, results in a nonlinear-time bound. They also presented a way to repair and modify the algorithm so that it does indeed run in linear time, on a pointer machine. Buchsbaum et al. [16, Corrig.] gave a different fix for the RAM model.

Our paper addresses the question of exactly what techniques are needed to compute dominators in linear time and explores the range of application of these techniques. We clarify and extend the conference papers [15, 32] to provide not only a linear-time algorithm for finding dominators but linear-time algorithms for a variety of related problems as well. We avoid the use of bit-manipulation techniques, so that all our algorithms can run on a pointer machine. We describe the techniques needed (in different combinations) to obtain these results as well as the key difficulties in the dominators problem.

2. Overview. We study six problems—off-line computation of nearest common ancestors (NCAs), verification and construction of minimum spanning trees (MSTs), interval analysis of flowgraphs, finding dominators in flowgraphs, and building the component tree of a weighted tree—that directly or indirectly require the evaluation of a function defined on paths in a tree. Each of these problems has a linear-time algorithm on a RAM. Some of these algorithms are quite complicated, and the fastest pointer-machine algorithms are slower by an inverse-Ackermann-function factor.[1] (See Table 2.1.)

TABLE 2.1

Time bounds, where n is the number of vertices, m is either the number of edges/arcs for graph problems or the number of NCA queries for the NCA problem, and $\alpha(m, n)$ is the standard functional inverse of the Ackermann function.

Problem	Previous pointer-machine bound		Previous RAM bound	
Off-line NCAs	$O(m\alpha(m, n) + n)$	[3]	$O(n + m)$	[38, 52]
MST verification	$O(m\alpha(m, n) + n)$	[58]	$O(n + m)$	[22, 40]
MST construction	$O(m\alpha(m, n) + n)$	[18]	$O(n + m)$	[26, 39]
Interval analysis	$O(m\alpha(m, n) + n)$	[57]	$O(n + m)$	[29, 57]
Dominators	$O(m\alpha(m, n) + n)$	[43]	$O(n + m)$	[10, 16]
Component trees	$O(m\alpha(m, n) + n)$		$O(n + m)$	[62]

A pointer machine [59] allows binary comparisons and arithmetic operations on data, dereferencing of pointers, and equality tests on pointers. It does not permit pointer arithmetic or tests on pointers other than testing for equality and is thus less powerful than the RAM model [2]. Pointer machines are powerful enough to simulate

[1] We use Tarjan's definition [56]. Let $A(i, j)$ for $i, j \geq 1$ be defined by $A(1, j) = 2^j$ for $j \geq 1$; $A(i, 1) = A(i-1, 2)$ for $i \geq 2$; and $A(i, j) = A(i-1, A(i, j-1))$ for $i, j \geq 2$. Then $\alpha(m, n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) > \log n\}$.

functional programming languages like LISP and ML. Pointer machine algorithms can be simpler than RAM algorithms in that they avoid the use of table look-up and more complicated bit-manipulation techniques. Thus pointer machine algorithms may have a conceptual, if not a practical, advantage over RAM algorithms.

We develop linear-time algorithms for all six problems that are simpler than the previous algorithms, in some cases substantially. We extract the commonalities among the problems and develop a set of techniques that in various combinations give our efficient algorithms. We do not use table look-up or any bit-manipulation techniques, so all our algorithms can run on a pointer machine.

Our improvements come mainly from three new ideas. The first is a refined analysis of path compression. Path compression is a well-known technique first used to speed up the standard disjoint-set-union (DSU) data structure [56] and later extended to speed up the evaluation of functions defined on paths in trees [58]. Our applications use either the DSU structure or path evaluation for the function *minimum* or *maximum*, or both. We show that, under a certain restriction on the compressions, which is satisfied by our applications, compression takes constant rather than inverse-Ackermann amortized time.

The second new idea is to replace the table-based methods of the RAM algorithms with radix sorting. Each of the RAM algorithms precomputes answers to small subproblems, stores the answers in a table, and looks up the answers by random access. If the size of the subproblems is small enough, the total size of all distinct subproblems and the total time to solve them are linear (or even sublinear) in the size of the original problem. Our alternative approach is to construct an encoding of each subproblem, group isomorphic subproblems together using a radix sort, solve one instance of each group of isomorphic subproblems, and transfer its solution to the isomorphic subproblems.

The third new idea is to change the partitioning strategy. In order to reduce the original problem to a collection of small subproblems, the RAM algorithms partition a tree corresponding to the original problem into small subtrees. For some of the problems, partitioning the entire tree into subtrees produces serious technical complications, especially for computing dominators. Instead, for all but one of the problems we partition only the bottom part of the tree into small subtrees. For NCAs and MSTs, this technique together with our refined analysis of path compression suffices to yield a linear-time algorithm. For interval analysis and finding dominators, we also partition the remainder of the tree into a set of maximal disjoint paths. Only one of our applications, building a component tree, relies on the original idea of partitioning the entire tree into small subtrees.

The remainder of our paper proceeds as follows. Section 3 formally defines the problems we consider and reviews previous work. Section 4 discusses DSU and computing path minima on trees, and gives a refined analysis of path compression. Section 5 discusses the use of radix sorting to solve a graph problem for a collection of many small instances. Sections 6 through 10 discuss our applications: NCAs, MSTs, flowgraph interval analysis, finding dominators, and building a component tree, respectively. Section 11 contains concluding remarks. Our paper is a significantly revised and improved combination of two conference papers [15, 32], including new results in sections 8 and 10.

3. Problem definitions and previous work. Throughout this paper we denote the base-two logarithm by log. We assume $n \geq 2$ throughout.

3.1. Nearest common ancestors.

PROBLEM 3.1 (off-line nearest common ancestors). *Given an n-node tree T rooted at node r and a set P of m node pairs, find, for each pair $\{v, w\}$ in P, the nearest common ancestor of v and w in T, denoted by $nca(v, w)$.*

The fastest previous pointer-machine algorithm is that of Aho, Hopcroft, and Ullman (AHU) [3], which runs in $O(n + m\alpha(m + n, n))$ time. The AHU algorithm uses a DSU data structure; it runs in $O(n + m)$ time on a RAM if this structure is implemented using the DSU algorithm of Gabow and Tarjan [29] for the special case in which the set of unions is known in advance. The first linear-time RAM algorithm was actually given by Harel and Tarjan [38]. Other linear-time RAM algorithms were given by Schieber and Vishkin [52], Bender and Farach-Colton [12], and Alstrup et al. [9].

There are several variants of the NCAs problem of increasing difficulty. For each but the last, there is a nonconstant-factor gap between the running time of the fastest RAM and pointer-machine algorithms.

- *Static on-line.* T is given a priori but P is given on-line: each NCA query must be answered before the next one is known.
- *Linking roots.* T is given dynamically. Specifically, T is initially a forest of singleton nodes. Interspersed with the on-line NCA queries are on-line $link(v, w)$ operations, each of which is given a pair of distinct roots v and w in the current forest and connects them by making v the parent of w.
- *Adding leaves.* Like linking roots, only v is any node other than w and w is a singleton.
- *General linking.* Like linking roots, only v can be any node that is not a descendant of w.
- *Linking and cutting.* Like general linking, but with additional interspersed $cut(v)$ operations, each of which is given a nonroot node and makes it a root by disconnecting it from its parent.

Harel and Tarjan [38] showed that the static on-line problem (and thus each more general variant) takes $\Omega(\log \log n)$ time on a pointer machine for each query, in the worst case. Alstrup and Thorup [7] gave a matching $O(n + m \log \log n)$-time pointer-machine algorithm for general linking, which is also optimal for the static on-line, linking roots, and adding leaves variants. Earlier, Tsakalidis and van Leeuwen [63] gave such an algorithm for the static on-line variant, and a modified version of van Leeuwen's earlier algorithm [64] has the same bound for linking roots. The fastest known pointer-machine algorithm for linking and cutting is the $O(n + m \log n)$-time algorithm of Sleator and Tarjan [53]; Harel and Tarjan [38] conjectured that this is asymptotically optimal, which in the cell-probe model follows from a result of Pătraşcu and Demaine [47]. On a RAM, the fastest known algorithms take $\Theta(n + m)$ time for the static on-line [38, 52] and adding leaves [27] variants, $O(n + m\alpha(m + n, n))$ time for linking roots [38] and general linking [27], and $O(n + m \log n)$ time for linking and cutting [53]. All these algorithms use $O(n + m)$ space. For a more thorough survey of previous work see Alstrup et al. [9].

3.2. Verification and construction of MSTs.

PROBLEM 3.2 (MST construction). *Given an undirected, connected graph $G = (V, E)$ whose edges have real-valued weights, find a spanning tree of minimum total edge weight (an MST) of G.*

PROBLEM 3.3 (MST verification). *Given an undirected, connected graph $G = (V, E)$ whose edges have real-valued weights and a spanning tree T of G, determine whether T is an MST of G.*

In both problems, we denote by n and m the numbers of vertices and edges, respectively. Since G is connected and $n \geq 2$, $m \geq n-1$ implies $n = O(m)$.

MST construction has perhaps the longest and richest history of any network optimization problem; Graham and Hell [35] and Chazelle [18] provide excellent surveys. A sequence of faster and faster algorithms culminated in the randomized linear-time algorithm of Karger, Klein, and Tarjan [39]. This algorithm requires a RAM, but only for a computation equivalent to MST verification. It is also *comparison-based*: the only operations it does on edge weights are binary comparisons. Previously, Fredman and Willard [26] developed a linear-time RAM algorithm that is not comparison-based. Subsequently, Chazelle [18] developed a deterministic, comparison-based $O(m\alpha(m,n))$-time pointer-machine algorithm, and Pettie and Ramachandran [49] developed a deterministic, comparison-based pointer-machine algorithm that runs in minimum time to within a constant factor. Getting an asymptotically tight bound on the running time of this algorithm remains an open problem.

Although it remains open whether there is a comparison-based, deterministic linear-time MST construction algorithm, even for a RAM, such algorithms do exist for MST verification. Tarjan [58] gave a comparison-based, deterministic $O(m\alpha(m,n))$-time pointer-machine algorithm for verification. Komlós [41] showed how to do MST verification in $O(m)$ comparisons, without providing an efficient way to determine which comparisons to do. Dixon, Rauch, and Tarjan [22] combined Tarjan's algorithm, Komlós's bound, and the tree partitioning technique of Gabow and Tarjan [29] to produce a comparison-based, deterministic linear-time RAM algorithm. King later gave a simplified algorithm [40].

3.3. Interval analysis of flowgraphs. A *flowgraph* $G = (V, E, r)$ is a directed graph with a distinguished *root vertex* r such that every vertex is reachable from r. A *depth-first spanning tree* D of G is a spanning tree rooted at r defined by some depth-first search (DFS) of G, with the vertices numbered from 1 to n in preorder with respect to the DFS (the order in which the search first visits them). We identify vertices by their preorder number. We denote by n and m the number of vertices and arcs of G, respectively.

PROBLEM 3.4 (interval analysis). *Given a flowgraph G and a depth-first spanning tree D of G, compute, for each vertex v, its* head $h(v)$*, defined by*

$$h(v) = \max\{u : u \textit{ is a proper ancestor of } v \textit{ in } D \textit{ and there is a path from } v \textit{ to } u \textit{ in } G \textit{ containing only descendants of } u\},$$
or null if this set is empty.

The heads define a forest called the *interval forest* H, in which the parent of a vertex is its head. If v is any vertex, the descendants of v in H induce a strongly connected subgraph of G, which is called an *interval*; these intervals impose a hierarchy on the loop structure of G. Interval analysis has been used in global flow analysis of computer programs [4], in testing flowgraph reducibility [60], and in the construction of two maximally edge-disjoint spanning trees of a flowgraph [57]. Tarjan [57] gave an $O(m\alpha(m,n))$-time pointer-machine algorithm for interval analysis using DSU. The Gabow–Tarjan DSU algorithm [29] reduces the running time of this algorithm to $O(m)$ on a RAM.

3.4. Finding dominators. Let $G = (V, E, r)$ be a flowgraph. We denote by n and m the number of vertices and arcs of G, respectively. Vertex v *dominates* vertex

w if every path from r to w contains v, and v is the *immediate dominator* of w if every vertex that dominates w also dominates v. The dominators define a tree rooted at r, the *dominator tree* T, such that v dominates w if and only if v is an ancestor of w in T: for any vertex $v \neq r$, the immediate dominator of v is its parent in T.

PROBLEM 3.5 (finding dominators). *Given a flowgraph $G = (V, E, r)$, compute the immediate dominator of every vertex other than r.*

Finding dominators in flowgraphs is an elegant problem in graph theory with fundamental applications in global flow analysis and program optimization [1, 19, 24, 45] and additional applications in VLSI design [11], theoretical biology [5, 6], and constraint programming [51]. Lengauer and Tarjan [43] gave a practical $O(m\alpha(m, n))$-time pointer-machine algorithm, capping a sequence of previous improvements [1, 45, 50, 55]. Harel [37] claimed a linear-time RAM algorithm, but Alstrup et al. [10] found problems with some of his arguments and developed a corrected algorithm, which uses powerful bit-manipulation-based data structures. Buchsbaum et al. [16] proposed a simpler algorithm, but Georgiadis and Tarjan [32] gave a counterexample to their linear-time analysis and presented a way to repair and modify the algorithm so that it runs in linear time on a pointer machine; Buchsbaum et al. [16, Corrig.] gave a different resolution that results in a linear-time algorithm for a RAM.

3.5. Building a component tree. Let T be a tree and let L be a list of the edges of T. The *Kruskal tree* of T with respect to L is a tree representing the connected components formed by deleting the edges of T and adding them back one at a time in the order of their occurrence in L. Specifically, K contains $2n-1$ nodes. Its leaves are the nodes of T. Each internal node is a component formed by adding an edge (v, w) back to T; its children are the two components that combine to form it.

PROBLEM 3.6 (component-tree construction). *Given an n-node tree T and a list L of its edges, build the corresponding Kruskal tree.*

Compressed component trees (formed by adding edges a group at a time rather than one at a time) have been used in shortest-path algorithms [48, 62]. It is straightforward to build a component tree or a compressed component tree in $O(n\alpha(n, n))$ time on a pointer machine using DSU. The Gabow–Tarjan DSU algorithm [29] improves this algorithm to $O(n)$ time on a RAM, as described by Thorup [62].

4. Path compression on balanced trees.

4.1. DSU via path compression and balanced unions. The *disjoint set union* (DSU) problem calls for the maintenance of a dynamic partition of a universe U, initially consisting of singleton sets. Each set has a unique *designated element*; the designated element of a singleton set is its only element. Two operations are allowed:

- *unite*(v, w): Form the union of the sets whose designated elements are v and w, with v being the designated element of the new set.
- *find*(v): Return the designated element of the set containing element v.

There are alternative, equivalent formulations of the DSU problem. In one [59, 56], each set is accessed by a label, rather than by a designated element. In another [61], sets have labels but can be accessed by *any* element. In yet another [61], each set is accessed by a *canonical element*, which in the case of a *unite*(v, w) operation can be freely chosen by the implementation to be either v or w. Our formulation more closely matches our uses. We denote by n the total number of elements and by m the total number of finds.

The standard solution to the DSU problem [56, 61] represents the sets by rooted trees in a forest. Each tree represents a set, whose elements are the nodes of the tree.

Each node has a pointer to its parent and a bit indicating whether it is a root; the root points to the designated element of the set. To provide constant-time access to the root from the designated node, the latter is either the root itself or a child of the root. With this representation, to perform *unite*(v, w), find the roots of the trees containing v and w, link them together by making one root the parent of the other, and make v a child of the new root if it is not that root or a child of that root already. To perform *find*(v), follow parent pointers until reaching a root, reach the designated element of the set in at most one more step, and return this element. A unite operation takes $O(1)$ time. A find takes time proportional to the number of nodes on the find path. A sequence of intermixed unite and find operations thus takes $O(n + s)$ time, where s is the total number of nodes on find paths.

One way to reduce s is to use *path compression*: after a find, make the root the parent of every other node on the find path. Another way to reduce s is to do *balanced unions*. There are two well-known balanced-union rules. In the first, *union-by-size*, each root stores the number of its descendants. To perform *unite*(v, w), make the root of the larger tree the parent of the root of the smaller, making either the parent of the other in case of a tie. In the second, *union-by-rank*, each root has a nonnegative integer *rank*, initially zero. To perform *unite*(v, w), make the root of higher rank the parent of the root of lower rank; in case of a tie, make either root the parent of the other and add one to the rank of the remaining root. Both of these union rules produce *balanced* trees. More specifically, let F be the forest built by doing all the unite operations and none of the finds. We call F the *reference forest*. F is *balanced* or, more precisely, *c-balanced* if for a constant $c > 1$ the number of nodes of height h in F is $O(n/c^h)$ for every h. Both union-by-size and union-by-rank produce 2-balanced forests. Furthermore, since only roots must maintain sizes or ranks, these fields obviate the need for separate bits to indicate which nodes are roots.

For any sequence of unions and finds such that the unions build a balanced forest and the finds use path compression, the total running time is $O(n+m\alpha(m+n, n))$: the analysis of path compression by Tarjan and van Leeuwen [61] applies if the reference forest is balanced. We seek a linear time bound, which we can obtain for sequences of finds that are suitably restricted. Before obtaining this bound, we discuss a more general use of path compression and balanced union: to find minima on paths in dynamic trees.

4.2. Finding minima on paths. The *dynamic path-minimum problem* calls for the maintenance of a forest of rooted trees, each initially a one-node tree, whose arcs, which are directed from parent to child, have real values. The trees are subject to three operations:

- *link*(v, w, x): Nodes v and w are the roots of different trees in F, and x is a real number. Make v the parent of w by adding arc (v, w) to F, with value x.
- *findroot*(v): Return the root of the tree in F containing the node v.
- *eval*(v): Return the minimum value of an arc on the path to v from the root of the tree containing it.

We shall denote by n the total number of nodes and by m the total number of *findroot* and *eval* operations. Variants of this problem include omitting the *findroot* operation, replacing minimum by maximum, and requiring the *eval* operation to return an arc of minimum value rather than just the minimum value. The two solutions to be described are easily modified to handle these variants. We call a data structure that solves the dynamic path-minimum problem a *link-eval structure*.

Tarjan [58] considered this problem and developed two data structures to solve it: a simple one [58, sec. 2], which uses path compression on the forest defined by the links, and a sophisticated one [58, sec. 5], which uses path compression on a balanced forest related to the one defined by the links. Tarjan's simple link-eval structure uses a compressed version of F, represented by parent pointers, with the nodes rather than the arcs storing values. Each root has value infinity. Perform *link*(v, w, x) by making v the parent of w and giving w the value x. Perform *findroot*(v) by following parent pointers from v to the root of the tree containing it, compressing this path, and returning the root. Perform *eval*(v) by following parent pointers from v to the root of the tree containing it, compressing this path, and returning the value of v. To compress a path $v_0, v_1, \ldots, v_k$ with v_i the parent of v_{i+1} for $0 \leq i < k$, repeat the following step for each i from 2 through k: replace the parent of v_i by v_0, and replace the value of v_i by the value of v_{i-1} if the latter is smaller. Compression preserves the results of *findroot* and *eval* operations while making tree paths shorter.

If the final forest F is balanced, then this simple link-eval structure takes $O(n + m\alpha(m + n, n))$ time to perform a sequence of operations [58]: the effect of a compression on the structure of a tree is the same whether the compression is due to a *findroot* or an *eval*. In our MST application the final forest is actually balanced. Our application to finding dominators requires Tarjan's sophisticated link-eval structure.

4.3. Delayed linking with balancing. Tarjan's sophisticated structure delays the effect of some of the links so that they can be done in a way that makes the resulting forest balanced. Since our analysis requires some knowledge of the inner workings of this structure, we describe it here. We streamline the structure slightly, and we add to it the ability to do *findroot* operations, which were not supported by the original. We also describe (in section 4.4) a variant that uses linking-by-rank; the original uses linking-by-size.

We represent the forest F defined by the link operations by a *shadow forest* R. Each tree in F corresponds to a tree in R with the same vertices and the same root. Each tree T in R is partitioned into one or more subtrees $S_0, S_1, \ldots, S_k$, such that the root of S_i is the parent of the root of S_{i+1} for $0 \leq i < k$, and the root of S_0 is the root of T. We call the roots of the subtrees $S_0, S_1, \ldots, S_k$ (including the root of S_0) *subroots*. We represent R by a set of parent pointers that are defined for nodes that are not subroots and, for each subroot, a pointer to its child that is a subroot, if any. (Each subroot has a null parent pointer; the deepest subroot has a null child pointer.) Since parents are needed only for nodes that are not subroots and child pointers are required only for subroots, we can use a single pointer per node to store both kinds of pointers, if we mark each node to indicate whether it is a subroot. We shall use *shp*(v) to denote the parent of v in its subtree and *shc*(v) to denote the child of v that is a subroot, if there is one; *shp*$(v) =$ *null* if v is a subroot, and *shc*$(v) =$ *null* if v is a subroot without a child that is a subroot.

With each node v we store a value $b(v)$. We manipulate the trees of R and the node values to preserve two related invariants:

(i) *eval*$(v) = \min\{b(u) : u$ is an ancestor in R of v in the same subtree$\}$;
(ii) $b($*shc*$(v)) \leq b(v)$ if *shc*$(v) \neq$ *null*.

To help keep evaluation paths short, we use both path compression and a variant of union-by-size. We denote by *size*(v) the number of descendants of v in R and by *subsize*(v) the number of descendants of v in the same subtree as v. For convenience, we let *size*(*null*) $= 0$. Then *subsize*$(v) =$ *size*(v) if v is not a subroot, and *subsize*$(v) =$

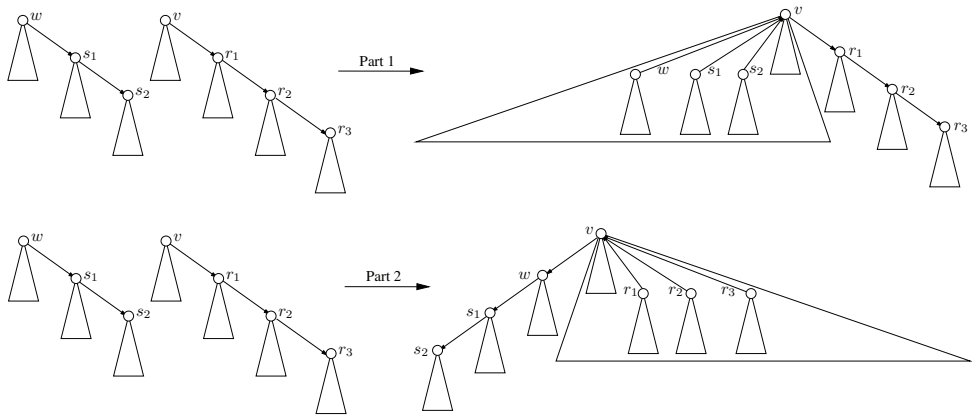


FIG. 4.1. *Linking by size: Part* 1, *size*$(v) \geq$ *size*(w), *and Part* 2, *size*$(v) <$ *size*(w).

size(v) − *size*(*shc*(v)) if v is a subroot. We maintain sizes but only for subroots, which allows us to compute the subsize of a subroot in constant time.

To initialize the structure, make each node v a singleton tree (*shp*(v) = *shc*(v) = *null*), with $b(v) = \infty$ and *size*$(v) = 1$. To perform *eval*(v), return $b(v)$ if *shp*(v) = *null*; otherwise, compress the path to v from the subroot of the subtree containing it (exactly as in the simple link-eval structure of section 4.2), and then return $\min\{b(v), b(shp(v))\}$. Perform *link*(v, w, x) as follows. First, set $b(w)$ (previously infinity) equal to x. Next, if *size*$(v) \geq$ *size*(w), perform Part 1 below; otherwise, perform Part 2 below and, if necessary, Part 3. (See Figures 4.1 and 4.2.)

Part 1 (*size*$(v) \geq$ *size*(w)). Combine the subtree rooted at v with all the subtrees in the tree rooted at w by setting *shp*$(u) = v$ and $b(u) = \min\{b(u), x\}$ for each subroot u of a subtree in the tree rooted at w. Find such subroots by following *shc* pointers from w. (In Figure 4.1 (Part 1), the successive values of u are w, s_1, s_2.) This step effects a compression to v from the deepest subroot descendant of w. The updates to the b-values maintain (i) and (ii).

Part 2 (*size*$(v) <$ *size*(w)). Combine all the subtrees in the tree rooted at v by setting *shp*$(u) = v$ for each subroot $u \neq v$ of a subtree in the tree rooted at v. (In Figure 4.1 (Part 2), the successive values of u are r_1, r_2, r_3.) This step effects a compression to v from the deepest subroot descendant of v. Then set *shc*$(v) = w$. This may cause violations of invariants (i) and (ii).

Part 3. In order to restore (i) and (ii) after Part 2, repeat the following step until it no longer applies. Let s_0 = *shc*(v) and s_1 = *shc*(s_0). (In the first iteration, $s_0 = w$.) If $s_1 \neq$ *null* and $x < b(s_1)$, compare the subsizes of s_0 and s_1. If the former is not smaller, combine the subtrees with subroots s_0 and s_1, making s_0 the new subroot, by simultaneously setting *shp*$(s_1) = s_0$ and *shc*(s_0) = *shc*(s_1). If the former is smaller, combine the subtrees with subroots s_0 and s_1, making s_1 the new subroot, by simultaneously setting *shp*$(s_0) = s_1$, *shc*$(v) = s_1$, $b(s_1) = x$, and *size*(s_1) = *size*(s_0). Once this step no longer applies, (i) and (ii) are restored.

Complete the linking by setting *size*(v) = *size*(v) + *size*(w). We call this linking method *linking-by-size*.

The method must keep track of which nodes are subroots. Nodes that are not subroots can be marked as such by, e.g., setting their sizes to zero, since sizes are maintained only for subroots. We have omitted this updating from Parts 1, 2, and 3.

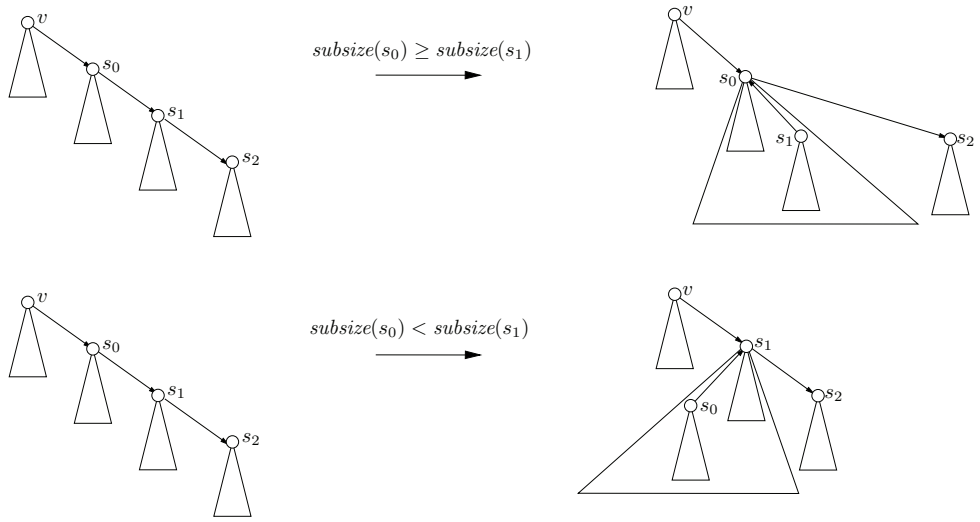


FIG. 4.2. *Linking by size: Part* 3.

This version of the data structure differs from the original [58] only in the placement of Part 3 of the link operation. In the original, Part 3 is done before Parts 1 and 2 to restore (i) and (ii), whether or not $size(v) \geq size(w)$. Delaying Part 3 allows it to be avoided entirely if $size(v) \geq size(w)$; in this case Part 1 alone suffices to restore (i) and (ii).

This structure does not support *findroot* (because an *eval* operation reaches only a subroot, not a root), but we can easily extend it to do so. To each subroot that is not a root, we add a pointer to its deepest subroot descendant; to each deepest subroot, we add a pointer to the root of its tree. Then a root is reachable from any subroot descendant in at most two steps. To perform *findroot*(v), compress the path to v from the subroot of its subtree; then follow at most two pointers to reach a root, and return this root. Operation *findroot* has the same asymptotic complexity as *eval*. The running time of a link operation increases by at most a constant factor because of the extra pointer updates needed.

In the sophisticated link-eval structure, path compression is performed on the subtrees, not on the trees. The next lemma implies that these subtrees are balanced.

LEMMA 4.1. *Consider a shadow forest built using linking-by-size. If u is a tree node such that $shp(u)$ and $shp(shp(u))$ are both nonnull, then $subsize(shp(shp(u))) \geq 2 \cdot subsize(u)$.*

Proof. A node u can be assigned a parent $shp(u)$ in Part 1, 2, or 3 of a link operation. If this occurs in Part 3, $subsize(shp(u)) \geq 2 \cdot subsize(u)$ after u gets its parent. Once this happens, $subsize(u)$ stays the same and $subsize(shp(u))$ can only increase. Thus when $shp(u)$ gets a parent, $subsize(shp(shp(u))) \geq subsize(shp(u)) \geq 2 \cdot subsize(u)$, and this inequality persists. Regardless of when u gets a parent $shp(u)$, if $shp(u)$ gets its parent in Part 3, then $subsize(shp(shp(u))) \geq 2 \cdot subsize(shp(u)) \geq 2 \cdot subsize(u)$ when this happens, and this inequality persists. Suppose then that both u and $shp(u)$ get their parents in Part 1 or 2. When u gets its parent, $size(shp(u)) \geq 2 \cdot subsize(u)$. Subsequently, $size(shp(u))$ cannot decrease until $shp(u)$ gets its parent, at which time $subsize(shp(shp(u))) \geq size(shp(u)) \geq 2 \cdot subsize(u)$. This inequality persists. ∎

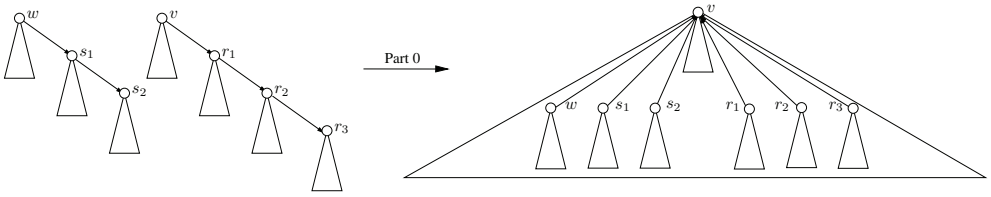


FIG. 4.3. *Linking by rank: Part* 0, *maxrank*(v) = *maxrank*(w).

COROLLARY 4.2. *The subtrees in any shadow forest built using linking-by-size are* $\sqrt{2}$*-balanced.*

4.4. Linking by rank. An alternative to using linking-by-size in the sophisticated link-eval structure is to use linking-by-rank. In place of a size, every node has a nonnegative integer *rank*, initially zero. The ranks satisfy the following invariant:

(iii) *rank*(*shp*(v)) > *rank*(v).

We explicitly maintain ranks only for subroots. If v is a virtual tree root (i.e., in F), we denote by *maxrank*(v) the maximum rank of a subroot descendant. With each virtual tree root v, we store *maxrank*(v) (in addition to *rank*(v)).

Perform *link*(v, w, x) as follows. First, set $b(w) = x$. Then compare *maxrank*(v) to *maxrank*(w). We split the rest of the operation into the following parts.

Part 0. If *maxrank*(v) = *maxrank*(w), set *rank*(v) = *maxrank*(v)+1, *maxrank*(v) = *maxrank*$(v) + 1$, and combine all the subtrees in the trees rooted at v and w into a single subtree rooted at v by setting *shp*$(u) = v$ for each subroot $u \neq v$, setting *shc*(v) = *null*, and setting $b(u) = \min\{b(u), b(w)\}$ if u was a descendant of w. (See Figure 4.3.)

Part 1. If *maxrank*(v) > *maxrank*(w), set *rank*$(v) = \max\{$*rank*(v), *maxrank*$(w) + 1\}$, and combine the subtree rooted at v with all the subtrees in the tree rooted at w by setting *shp*$(u) = v$ and $b(u) = \min\{b(u), b(w)\}$ for each subroot descendant u of w.

Part 2. If *maxrank*(v) < *maxrank*(w), combine all the subtrees in the tree rooted at v into a single subtree, unless *shc*(v) = *null*, by setting *rank*(v) = *maxrank*$(v) + 1$, *maxrank*(v)=*maxrank*(w), and, for each subroot $u \neq v$, *shp*$(u) = v$. Then set *shc*$(v) = w$. This may cause violations of invariants (i) and (ii).

Part 3. To restore (i) and (ii) after Part 2, repeat the following step until it no longer applies. Let s_0 = *shc*(v) and s_1 = *shc*(s_0). If $s_1 \neq$ *null* and $x < b(s_1)$, compare *rank*(s_0) to *rank*(s_1), and if *rank*(s_0) = *rank*(s_1), simultaneously set *shp*$(s_1) = s_0$, *shc*(s_0) = *shc*(s_1), *rank*(s_0) = *rank*$(s_0) + 1$, and *maxrank*$(v) = \max\{$*maxrank*(v), *rank*$(s_0) + 1\}$; if *rank*(s_0) > *rank*(s_1), simultaneously set *shp*$(s_1) = s_0$ and *shc*(s_0) = *shc*(s_1); if *rank*(s_0) < *rank*(s_1), simultaneously set *shp*$(s_0) = s_1$, *shc*$(v) = s_1$, and $b(s_1) = x$.

Parts 1, 2, and 3 of linking-by-rank correspond to Parts 1, 2, and 3 of linking-by-size; Part 0 handles the case of equal *maxranks*, in which all subtrees of both trees are combined. (We could add a corresponding Part 0 to linking-by-size, but this is unnecessary.) As does linking-by-size, linking-by-rank produces balanced forests, as we now show. For a node u, let *subsize*(u) be the number of descendants of u in its subtree.

LEMMA 4.3. *In any shadow forest built using linking-by-rank, any node* u *has* *subsize*$(u) \geq 2^{(rank(u)-1)/2}$.

Proof. To obtain this result we actually need to prove something stronger. Suppose we perform a sequence of link-by-rank operations. We track the states of nodes,

their ranks, and their subsizes as the links take place. Each node is in one of two states: *normal* or *special*. The following invariants will hold:

(a) a *normal node* u has $subsize(u) \geq 2^{rank(u)/2}$;

(b) a *special node* u has $subsize(u) \geq 2^{(rank(u)-1)/2}$;

(c) a *special root* u has a normal subroot descendant of rank at least $rank(u)$.

Initially all nodes are normal; since all initial ranks are zero, (a), (b), and (c) hold initially. We need to determine the effect of each part of an operation $link(v, w, x)$.

If $maxrank(v) = maxrank(w)$, we make v normal after the link; all other nodes retain their states. This preserves (a), (b), and (c); the only question is whether v satisfies (a), since it gains one in rank and can change from special to normal. Before the link, both the tree rooted at v and the tree rooted at w have a subroot of rank $maxrank(v)$. Since each of these nodes has subsize at least $2^{(maxrank(v)-1)/2}$ before the link by (a) and (b), after the link $subsize(v) \geq 2 \cdot 2^{(rank(v)-2)/2} = 2^{rank(v)/2}$. Hence (a) holds for v after the link.

If $maxrank(v) > maxrank(w)$ and $rank(v)$ does not change as a result of the link, all nodes retain their states. The link preserves (a), (b), and (c), because no node increases in rank. If $rank(v)$ does change because of the link (becoming one greater than the old value of $maxrank(w)$), we make v special. Node v now satisfies (b), because before the link w had a normal subroot descendant u of rank $maxrank(w)$, and $subsize(u) \geq 2^{maxrank(w)/2}$ by (a); hence, after the link, $subsize(v) \geq 2^{(rank(v)-1)/2}$. Node v satisfies (c), because before the link it had a normal subroot descendant z of rank $maxrank(v) \geq maxrank(w) + 1$, which it retains after the link.

The last case is $maxrank(v) < maxrank(w)$. In this case we look at the effects of Part 2 and Part 3 separately. If Part 2 does anything, we make v special. Node v satisfies (b), because before the link it had a normal subroot descendant of rank $maxrank(v)$, which satisfied (a); hence, after the link, $subsize(v) \geq 2^{(rank(v)-1)/2}$. Node v satisfies (c) after the link, because before the link w had a normal subroot descendant of rank $maxrank(w) \geq maxrank(v) + 1$ by (a), which becomes a normal subroot descendant of v.

Finally, we must account for the effect of Part 3. Each combination of subtrees done by Part 3 preserves (a), (b), and (c), except possibly for those that combine two subtrees with subroots, say y and z, of equal rank. In this case the rank of the surviving subroot increases by one; and if the ranks of y and z previously equaled $maxrank(v)$, $maxrank(v)$ increases by one. To preserve the invariants in this case, we make the surviving root, say y, normal. Now y satisfies (a), because before the subtrees rooted at y and z were combined, both y and z have subsize at least $2^{(rank(y)-1)/2}$; after the subtrees are combined, $subsize(y) \geq 2 \cdot 2^{(rank(y)-2)/2} = 2^{rank(y)/2}$. Because y satisfies (a), v satisfies (c).

Thus linking preserves the invariants. By induction, they remain true throughout any sequence of links. The lemma follows from (a) and (b). □

COROLLARY 4.4. *The subtrees in any shadow forest built using linking-by-rank are* $\sqrt{2}$*-balanced.*

THEOREM 4.5. *A sequence of operations performed using the sophisticated link-eval structure with either linking-by-size or linking-by-rank takes* $O(n)$ *time for the links and* $O(n + m\alpha(m + n, n))$ *time for the findroot and eval operations.*

Proof. The time for a link is $O(k + 1)$, where k is the decrease in the number of subtrees caused by the link. Thus the total time for all the links is $O(n)$. The total length of compressed paths, and hence the total time for *findroot* and *eval* operations, is $O(n+m\alpha(m+n, n))$ by the Tarjan–van Leeuwen analysis of path compression [61], applying Corollary 4.2 (for linking-by-size) or Corollary 4.4 (for linking-by-rank). □

4.5. Refined analysis of path compression. In order to use path compression on balanced trees as a tool for building linear-time algorithms, we need to show that the total time becomes linear if the compressions are suitably restricted. In order to capture both DSU and link-eval applications, we abstract the situation as follows. An intermixed sequence of the following two kinds of operations is performed on a rooted forest, initially consisting of n single-node trees:

- *assign*(u, v). Given two distinct roots u and v, make u the parent of v.
- *compress*(u). Compress the path to u from the root of the tree containing it by making the root the parent of every other node on the path.

LEMMA 4.6. *Suppose ℓ nodes are* marked *and the remaining $n - \ell$* unmarked. *Suppose the assignments build a balanced forest, and that each node has its parent change at most k times before it is in a tree containing a marked node. If there are m compress operations, then the total number of nodes on compression paths is $O(kn + m\alpha(m + \ell, \ell))$.*

Proof. Let F be the balanced forest built by the entire sequence of assignments, ignoring the compressions; let $c > 1$ be such that F is c-balanced; and let $h(v)$ be the height of a node v in F. Let

$$a = \lceil \log_c (n/\ell) + \log_c (1/(c-1)) + 1 \rceil.$$

Classify each node v into one of three types: *low*, if v has no marked descendant in F; *middle*, if v has a marked descendant in F and $h(v) < a$; and *high* otherwise.

A compression path from a tree root to one of its descendants consists of zero or more high nodes followed by zero or more middle nodes followed by zero or more low nodes. Every node on the path except the first two (totaling at most $2m$ over all compressions) has its parent change to one of greater height as a result of the compression.

Consider a compression path containing only low nodes. Since the root is low, the tree in which the compression takes place contains no marked nodes. All but two nodes on the path change parent but remain in a tree with no marked nodes. The number of times this can happen to a particular node is at most k by the hypothesis of the lemma, totaling at most kn over all compressions.

Consider a compression path containing at least one middle or high node. Every low node on the path except one has its parent change from low to middle or high as a result of the compression. Thus the total number of low nodes on such paths is at most $n + m$. Every middle node on the path whose parent changes obtains a parent of greater height. This can happen to a middle node at most a times before its parent is high. At most one middle node on a compression path has a high parent, totaling at most m over all compression paths. Each middle node has a marked node as a descendant; each marked node has at most $a + 1$ middle nodes as ancestors (at most one per height less than a). The total number of middle nodes is thus at most $\ell(a + 1)$. Combining estimates, we find that the total number of middle nodes on compression paths is at most $\ell \cdot a \cdot (a+1) + m$. Since $\ell \le n$ and a is $O(\log (n/\ell))$, the first term is $O(n)$, implying that the total number of middle nodes on compression paths is $O(n) + m$.

Finally, we need to count the number of high nodes on compression paths. Since F is c-balanced, the total number of high nodes is at most

$$\sum_{i \ge a} \frac{n}{c^i} \le \frac{n}{c^a} \cdot \frac{c}{c-1} = \frac{n}{c^{a-1}(c-1)} \le \ell.$$

Let the *rank* of a node v be $h(v) - a$. Then every high node has nonnegative rank, and the number of high nodes of rank $i \geq 0$ is at most ℓ/c^i. The analysis of Tarjan and van Leeuwen [61, Lem. 6] applied to the high nodes bounds the number of high nodes on compression paths by $O(\ell + m\alpha(m + \ell, \ell))$. Combining all our estimates gives the lemma. □

Lemma 4.6 gives a bound of $O(n+m)$ if, for example, $\ell = O(n/\log\log n)$, by the properties of the inverse-Ackermann function [56]. In our applications $\ell = n/\log^{1/3} n$, which is sufficiently small to give an $O(n + m)$ bound.

We conclude this section by reviewing some previous results on DSU and refined analysis of the DSU structure. The linear-time RAM DSU algorithm of Gabow and Tarjan [29] assumes a priori knowledge of the unordered set of unions. An earlier version of our work [15] contained a result much weaker than Lemma 4.6, restricted to disjoint set union, which required changing the implementation of unite based on the marked nodes. Alstrup et al. [10] also proved a weaker version of Lemma 4.6 in which the $m\alpha(m + \ell, \ell)$ term is replaced by $\ell \log \ell + m$, which sufficed for their purpose. They derived this result for a hybrid algorithm that handles long paths of unary nodes outside the standard DSU structure. Dillencourt, Samet, and Tamminen [20] gave a linear-time result assuming the *stable tree property*: essentially, once a find is performed on any element in a set X, all subsequent finds on elements currently in X must be performed before X can be united with another set. Fiorio and Gustedt [25] exploited the specific order of unions in an image-processing application. Gustedt [36] generalized the previous two works to consider structures imposed on sets of allowable unions by various classes of graphs. This work is orthogonal to that of Gabow and Tarjan [29]. Other improved bounds for path compression [14, 44, 46] restrict the order in which finds are performed, in ways different from our restriction.

5. Topological graph computations. Consider a computation that takes as input a graph G whose vertices and edges (or arcs) have $O(1)$-bit labels and produces some output information (possibly none) associated with the graph itself and with each vertex and edge (or arc). We call such a computation a *topological graph computation*, because it is based only on the graph structure and the $O(1)$-bit labels, in contrast, for example, to a problem in which graph vertices and edges (or arcs) have associated real values. In general the output of a topological graph computation can be arbitrarily complex, even exponential in size, and can contain pointers to elements of the input graph. Our MST verification algorithm will exploit this flexibility; in all our other applications, the size of the output is linear in the size of the input.

Suppose we need to perform a topological graph computation on not just one input graph but on an entire collection of graphs. If the input instances are small and there are many of them, then many of them will be isomorphic. By doing the computation once for each nonisomorphic instance (a *canonical instance*) and copying these solutions to the duplicate instances, we can amortize away the cost of actually doing the computations on the canonical instances; most of the time is spent identifying the isomorphic instances and transferring the solutions from the canonical instances to the duplicate ones. The total time spent is then linear in the total size of all the instances.

Gabow and Tarjan [29] used this idea to solve a special case of DSU in which the unordered set of unions is given in advance; Dixon, Rauch, and Tarjan [21] applied the technique to MST verification and other problems. These applications use table look-up and require a RAM. Here we describe how to accomplish the same thing on a pointer machine. Our approach is as follows. Encode each instance as a list of

pointers. Use a radix sort to sort these lists. Identify the first instance in each group of identically encoded instances as the canonical instance. Solve the problem for each canonical instance. Map the solutions back to the duplicate instances. The details follow.

Let $\mathcal{G}$ be the set of input instances, each of which contains at most g vertices. Let N be the total number of vertices and edges (or arcs) in all the instances. Let k be the maximum number of bits associated with each vertex and edge (or arc) of an instance. Construct a singly linked master list whose nodes, in order, represent the integers from zero through $\max\{g, 2^k + 1\}$ and are so numbered. For each instance G, perform a DFS, numbering the vertices in preorder and adding to each vertex a pointer into the master list corresponding to its preorder number; the preorder numbering allows us to maintain a global pointer into the master list to facilitate this assignment of pointers to vertices. Represent the label of each vertex and edge (or arc) by a pointer into the master list, using a pointer to the zero node to encode the lack of a label. Construct a list L of triples corresponding to the vertices of G, one triple per vertex, consisting of a pointer to the vertex, and its number and label, both represented as pointers into the master list. Construct a list Q of quadruples corresponding to the edges (or arcs) of the graph, one quadruple per edge (or arc), consisting of a pointer to the edge (or arc), and the numbers of its endpoints and its label, represented as pointers into the master list. (For an undirected graph, order the numbers of the edge endpoints in increasing order.) Encode the instance by a list whose first entry is a pair consisting of a pointer to the instance and the number of its vertices, represented as a pointer into the master list, catenated with lists L and Q.

Constructing encodings for all the instances takes $O(N)$ time. Recall that the elements of the encodings are pointers to the master list. Attach a bucket to each element of the master list. Use a radix sort for variable length lists [2], following the encoding pointers to reach the buckets, to arrange the encodings into groups that are identical except for the first components of each list element (pair, triple, or quadruple): instances whose encodings are in the same group are isomorphic. This also takes $O(N)$ time.

Now perform the topological graph computation on any one instance of each group (the canonical instance for that group). Finally, for each duplicate instance, traverse its encoding and the encoding of the corresponding canonical instance concurrently, transferring the solution from the canonical instance to the duplicate instance. The exact form this transfer takes depends upon the form of the output to the topological graph computation. One way to do the transfer is to traverse the encodings of the canonical instance and the duplicate instance in parallel, constructing pointers between corresponding vertices and edges (or arcs) of the two instances. Then visit each vertex and edge (or arc) of the canonical instance, copying the output to the duplicate instance but replacing each pointer to a vertex or edge (or arc) by a pointer to the corresponding vertex or edge (or arc) in the duplicate instance. If the output has size linear in the input, this takes $O(N)$ time. Summarizing, we have the following theorem.

THEOREM 5.1. *If the output of a topological graph computation has size linear in the input size, the computation can be done on a collection of instances of total size N in $O(N)$ time on a pointer machine, plus the time to do the computation on one instance of each group of isomorphic instances.*

This method extends to allow the vertices and edges (or arcs) of the instances to be labeled with integers in the range $[1, g]$ if these labels are represented by pointers to the nodes of a precomputed master list. We shall need this extension in our

applications to finding dominators and computing component trees (sections 9 and 10, respectively). In another of our applications, MST verification, the output of the topological graph computation has exponential size: it is a comparison tree, whose nodes indicate comparisons between the weights of two edges. In this case, we do not construct a new copy of the comparison tree for each duplicate instance. Instead, when we are ready to run the comparison tree for a duplicate instance, we construct pointers from the edges of the canonical instance to the corresponding edges of the duplicate instance and run the comparison tree constructed for the canonical instance, but comparing weights of the corresponding edges in the duplicate instance. The total time is $O(N)$ plus the time to build the comparison trees for the canonical instances plus the time to run the comparison trees for all the instances.

It remains to bound the time required to do the topological graph computation on the canonical instances. The number of canonical instances is $g^{O(g^2)}$. In all but one of our applications, the time to do a topological graph computation on an instance of size g or smaller is $O(g^2)$; for MST verification, it is $g^{O(g^2)}$. Thus the following theorem suffices for us.

THEOREM 5.2. *If a topological graph computation takes $g^{O(g^2)}$ time on a graph with g or fewer vertices, and if $g = \log^{1/3} N$, then the total time on a pointer machine to do the topological graph computation on a collection of graphs of total size N, each having at most g vertices, is $O(N)$.*

Proof. The proof is immediate from Theorem 5.1, since the total time to do the topological graph computation on the canonical instances is $g^{O(g^2)}g^{O(g^2)} = g^{O(g^2)} = O(N)$. □

The ability to recover the answers from the topological graph computations on the instances in $\mathcal{G}$ is subtle but crucial. Alstrup, Secher, and Spork [8] show how to compute connectivity queries on a tree T undergoing edge deletions in linear time. They partition T into bottom-level microtrees (discussed in the next section) and compute, for each vertex v in a microtree, a bit-string that encodes the vertices on the path from v to the root of its microtree. They show how to answer connectivity queries using a constant number of bitwise operations on these bit-strings and applying the Even and Shiloach decremental connectivity algorithm [23] to the upper part of T.

The Alstrup, Secher, and Spork algorithm [8] runs on a pointer machine: since the connectivity queries return yes/no answers, they do not need index tables to recover the answers. In contrast, while their method can be extended to solve the off-line NCAs problem in linear time on a RAM, and even to simplify the Gabow–Tarjan linear-time DSU result [29], both of these extensions require index tables to map the results of the bitwise operations back to vertices in T.

The idea of using pointers to buckets in lieu of indexing an array was described in general by Cai and Paige [17] in the context of multisequence discrimination. Their technique leaves the efficient identification of buckets with specific elements as an application-dependent problem. They solve this problem for several applications, including discriminating trees and DAGs, but their solutions exploit structures specific to their applications and do not extend to general graphs.

6. Nearest common ancestors. We now have the tools to solve our first application, the off-line nearest common ancestors (NCAs) problem: given a rooted n-node tree T and a set P of m queries, each of which is a pair $\{v, w\}$ of nodes in T, compute $nca(v, w)$ for each query $\{v, w\}$. Aho, Hopcroft, and Ullman's algorithm [3] for this problem, as presented by Tarjan [58], solves it using DSU. The algorithm traverses T bottom-up, building a shadow copy as a DSU forest. It maintains, for each subtree

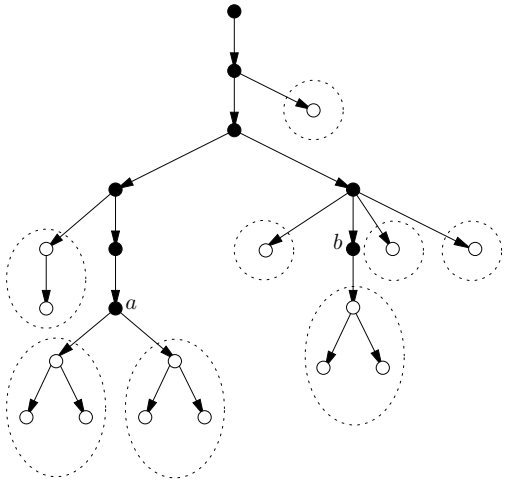


FIG. 6.1. *Partitioning of a tree T with $g = 3$; fringe nodes are open, and core nodes are filled; bottom-level microtrees are encircled. Nodes a and b are the leaves of the core.*

built so far, the set of its nodes, with the root of the subtree as the designated element. Initially, each node is in a singleton set. Each node v also has a set $P(v)$ of queries $\{v, w\}$; each query is in two such lists, one for v and one for w. The algorithm is as follows.

Visit the nodes of T in a postorder [54]. (Any postorder will do.) When visiting a node v, for every pair $\{v, w\}$ in $P(v)$ such that w has already been visited, return *find*(w) as the answer to the query $nca(v, w)$. Finish the visit to v by performing *unite*$(p(v), v)$ if v is not the root of T, where $p(v)$ is the parent of v in T.

The correctness of this algorithm follows from basic properties of postorder. The DSU operations dominate the running time, which is $O(n+m\alpha(m+n, n))$ if the standard DSU structure presented in section 4 is used. In this algorithm, the unordered set of unions is known in advance, since it is given by the input tree T. Thus the use of the Gabow–Tarjan [29] linear-time RAM DSU algorithm results in a linear-time RAM algorithm for NCAs. Knowing the set of unions in advance, however, is not sufficient to solve the DSU problem in linear time on a pointer machine [46]. We exploit a different property of the unions: they occur in a bottom-up order.

We partition T into a set of small bottom-level trees, called *microtrees*, and T', the rest of T. For any node v, let $T(v)$ be the subtree of T induced by the descendants of v, and let $|T(v)|$ be the number of nodes in $T(v)$. Let $g \geq 1$ be a fixed parameter to be chosen later. We define $T(v)$ to be a *microtree* if $|T(v)| \leq g$ but $|T(p(v))| > g$. For a node x in $T(v)$, *micro*$(x) = T(v)$ is the *microtree of* x and *root*(*micro*(x)) is the *root of its microtree*. Let T' be the subtree of T induced by the vertices in T that are not in microtrees. Each leaf in T' has at least g descendants in T, and the descendants of two different leaves of T' form disjoint sets, so T' has at most n/g leaves. We call the microtrees the *fringe* of T and T' the *core* of T. See Figure 6.1. It is straightforward to partition T into its microtrees and core in linear time by visiting the nodes in postorder and computing their numbers of descendants.

We call a query $\{v, w\}$ *small* if v and w are in the same microtree, and *big* otherwise. We can partition the queries into big and small and assign each small query to the microtree containing it in linear time. We answer all the big queries by

using the AHU algorithm. We answer all the small queries by doing a topological graph computation on the set of graphs defined by each microtree and its associated queries. By choosing g appropriately, we get a linear-time bound for both parts of the computation.

Specifically, choose $g = \log^{1/3} n$. Answer all the big queries by running the AHU algorithm, restricted to the big queries. To bound the running time, apply Lemma 4.6 to the tree built by the parent assignments done by the unite operations. Mark every leaf of T'. Each find occurs in a set containing at least one marked node. Therefore, setting $k = 1$, to count the initial parent assignment for each node, satisfies the hypothesis of the lemma. Since the number of marked nodes is at most $n/g = n/\log^{1/3} n$, the lemma implies an $O(n + m)$ bound on the time to answer all the big queries.

Answer all the small queries by constructing, for each microtree, a graph containing the microtree edges and, for each query with both nodes in the microtree, an edge denoted as a query edge by a bit. Then do a topological graph computation on these graphs to answer the small queries, using the method of section 5. With $g = \log^{1/3} n$, this takes $O(n + m)$ time. Thus we obtain the following theorem.

THEOREM 6.1. *The off-line NCAs problem can be solved in $O(n + m)$ time on a pointer machine.*

7. Minimum spanning trees.

7.1. Verification. Our next applications, minimum spanning tree (MST) verification and construction, combine topological graph processing with use of the simple link-eval structure of section 4.2. Let T be a spanning tree of a connected, undirected graph G whose edges have real-valued weights. For any edge $\{v, w\}$, let $c(v, w)$ be the weight of $\{v, w\}$. We denote the set of nontree edges by P. For any pair (v, w) of vertices, we denote by $T(v, w)$ the unique path from v to w in T. The tree T is minimum if and only if, for every edge $\{v, w\}$ in P, $c(v, w) \geq c(x, y)$ for every edge $\{x, y\}$ on $T(v, w)$. Thus to verify that T is minimum it suffices to compute $\max\{c(x, y) : \{x, y\} \text{ on } T(v, w)\}$ for every edge $\{v, w\}$ in P. We assume henceforth that T is rooted at a fixed but arbitrary vertex and that each vertex v has a set $P(v)$ of the pairs $\{v, w\}$ in P.

Tarjan's $O(m\alpha(m, n))$-time MST verification algorithm [58] is like the AHU NCA algorithm, except that it uses a link-eval structure (with max instead of min) in place of a DSU structure to compute the needed path maxima. The algorithm builds the link-eval forest during a bottom-up traversal of T. As part of the process of computing path maxima, the algorithm computes $u = \mathit{nca}(v, w)$ for each pair $\{v, w\}$ in P and stores $\{v, w\}$ in a set $Q(u)$. Initially each node of T is in a single-node tree of the link-eval structure, and $Q(u)$ is empty for each node u. The algorithm follows.

Visit the nodes of T in a postorder. (Any postorder will do.) When visiting a vertex v, for every pair $\{v, w\}$ in $P(v)$ such that w has already been visited, add $\{v, w\}$ to $Q(\mathit{findroot}(w))$. For every pair $\{x, y\}$ in $Q(v)$, return $\max\{\mathit{eval}(x), \mathit{eval}(y)\}$ as the answer to the query $\{x, y\}$. Finish the visit to v by performing $\mathit{link}(p(v), v, c(p(v), v))$ unless v is the root of T.

When the algorithm answers a query $\{x, y\}$ while visiting a vertex v, $v = \mathit{nca}(x, y)$, and $\mathit{eval}(x)$ and $\mathit{eval}(y)$ are the maximum costs of the arcs on $T(v, x)$ and $T(v, y)$, respectively. In Tarjan's original presentation, the NCA calculations are separate from the path evaluations, but combining them gives a more coherent algorithm. Ignoring the arc costs and *eval* operations, the link-eval structure functions exactly like the DSU structure in the AHU NCA algorithm.

If the sophisticated link-eval structure of section 4.3 or section 4.4 is used, this algorithm runs in $O(m\alpha(m,n))$ time. Unfortunately, these structures delay the effect of the links, so parent assignments do not necessarily occur in a bottom-up order, and we cannot immediately apply the approach of section 6 to reduce the running time to linear. This problem was pointed out by Georgiadis and Tarjan [32]. Instead, we use a result of King [40] to transform the original tree into an $O(n)$-node balanced tree on which to compute path maxima. Then we can use the simple link-eval structure of section 4.2 in combination with the approach of section 6 to obtain a linear-time algorithm.

7.2. The Borůvka tree. A *Borůvka step* [13] applied to a weighted, undirected graph G is as follows: Select a least-weight edge incident to each vertex, and contract to a single vertex each connected component formed by the selected edges. Repeating this step until only a single vertex remains produces an MST defined by the original edges corresponding to the edges selected in all the steps if all edge weights are distinct, which we can assume without loss of generality.

This algorithm can be enhanced to produce the *Borůvka tree* B, whose nodes are the connected components that exist during the Borůvka steps, with each node having as children those components from which it is formed during a Borůvka step. If component C is the parent of component D, the weight of arc (C,D) is the weight of the edge selected for the vertex corresponding to D by the Borůvka step in which D is contracted into C. The leaves of B are the vertices of G, each of which is originally a single-vertex component. Each Borůvka step reduces the number of vertices by at least a factor of two; hence, B is 2-balanced. Also, B contains at most $2n-1$ nodes. In general the enhanced Borůvka algorithm runs in $O(m \log n)$ time on a pointer machine. On a tree, however, it runs in $O(n)$ time, because each contracted graph is a tree, and a tree has $O(n)$ edges. We apply the enhanced Borůvka algorithm to the tree T that is to be verified, thereby constructing the Borůvka tree B of T. In addition to being balanced, B has the following key property [40]: for any pair of vertices $\{v,w\}$, $\max\{c(x,y) : (x,y) \text{ on } T(v,w)\} = \max\{c(x,y) : (x,y) \text{ on } B(v,w)\}$. Thus we can compute path maxima on B instead of on T without affecting the answers to the queries.

7.3. Comparison trees for computing path maxima. Now we can apply the approach of section 6. Let $g = \log^{1/3} n$. Partition B into microtrees and a core B' as in section 6. Partition the pairs in P into *big pairs*, those with ends in different microtrees, and *small pairs*, those with ends in the same microtree. Compute path maxima for all the big pairs by running Tarjan's algorithm on B, restricted to the big pairs and using the simple link-eval structure of section 4.2.

To bound the running time of this computation, we apply Lemma 4.6 to B. Mark every leaf of B'. Each *findroot* and *eval* occurs in a subtree of B containing a marked node, so setting $k = 1$ satisfies the hypothesis of the lemma. Since the number of marked nodes is at most $2n/g = 2n/\log^{1/3} n$, the lemma implies an $O(m)$ bound on the time to compute path maxima for all the big pairs.

We would like to compute path maxima for all the small pairs by applying the method of section 5. To this end, construct for each microtree a graph containing the microtree edges and, for each pair with both ends in the microtree, an edge designated as a query edge by a bit. Now a new difficulty arises: since the edge costs are arbitrary real numbers, computing path maxima is not a topological graph computation; we cannot encode the edge costs in $O(1)$ bits, or even in $O(\log g)$ bits.

We overcome this difficulty by following the approach of Dixon, Rauch, and Tar-

jan [22]: Do a topological graph computation that builds, for each distinct marked graph, a comparison tree, whose nodes designate binary comparisons between costs of unmarked edges of the graph (tree edges), such that the output nodes of the comparison tree designate, for each marked edge (query pair), which of the unmarked edges on the path between the ends of the edge has maximum cost. Having built all the comparison trees, run the appropriate comparison tree for each microtree and its associated pairs, using the actual costs of the microtree arcs to determine the outcomes of the comparisons.

With $g = \log^{1/3} n$, the time for this computation is $O(m)$, plus the time to build comparison trees for the topologically distinct instances, plus the time to run the comparison trees for the actual instances. Komlós [41] proved that the path maxima needed for MST verification can be determined in a number of binary comparisons of tree edge costs that is linear in the number of graph edges, which implies for each instance the existence of a comparison tree that has depth linear in the number of edges. Dixon, Rauch, and Tarjan [22] observed that the comparison tree implied by Komlós' result can be built in a time per comparison-tree node that is quadratic in the number of graph vertices. If we use their method to build the comparison trees during the topological graph computation, then $g = \log^{1/3} n$ implies by the results of section 5 that the total time to build the comparison trees is $O(m)$. The total time to run them is linear in the total size of all the actual instances, which is also $O(m)$. Thus we obtain the following theorem.

THEOREM 7.1. *Computing all the path maxima needed for MST verification, and doing the verification itself, takes $O(m)$ time on a pointer machine.*

7.4. Construction of MSTs. The randomized linear-time MST construction algorithm of Karger, Klein, and Tarjan [39] runs on a pointer machine except for the part that computes the path maxima needed for MST verification. Using the algorithm of section 7.3, this part can be done (deterministically) in linear time on a pointer machine, resulting in a randomized linear-time pointer-machine algorithm for constructing an MST.

7.5. Remarks. It is instructive to compare our MST verification algorithm to those of Dixon, Rauch, and Tarjan [22] and of King [40]. Our use of King's Borůvka tree construction as an intermediate step allows us to use only bottom-level microtrees, whereas Dixon et al. partition the original tree entirely into microtrees, with an extra *macrotree* to represent the connections between them. It also allows us to use the simple link-eval structure instead of the sophisticated one. Lemma 4.6 allows us to break big queries into only two parts (having an NCA in common); Dixon et al. break each big query into as many as six parts. King explicitly implements Komlós' comparison algorithm for the Borůvka tree, but her algorithm is heavily table-driven and requires a RAM. She also must compute NCAs separately.

There is an alternative, though more complicated, way to verify an MST in linear time on a pointer machine. This method replaces the use of the Borůvka tree by a partition of the original tree into bottom-level microtrees and a set of maximal paths that partition the core. The method does NCA computations on trees derived from the maximal paths, and it uses a sophisticated link-eval structure instead of the simple one. We discuss this method in more detail in section 9.7. Though the use of the Borůvka tree gives us a simpler algorithm for MST verification, there is no corresponding concept for either of our remaining applications, and we must rely on the alternative of partitioning the core into maximal paths.

8. Interval analysis. We turn now to two problems on flowgraphs. The first is *interval analysis*. Let $G = (V, A, r)$ be a flowgraph, and let D be a given DFS tree rooted at r. Identify vertices by their preorder number with respect to the DFS: $v < w$ means that v was visited before w. *Reverse preorder* of the vertices is decreasing order by (preorder) vertex number. For each vertex v, the *head* of v is

$$h(v) = \max\{u \neq v : \text{there is a path from } v \text{ to } u \text{ containing only descendants of } u\};$$

$h(v) = \mathit{null}$ if this set is empty. The heads define a forest H called the *interval forest*: $h(v)$ is the parent of v in H. Each subtree $H(v)$ of H induces a strongly connected subgraph of G, containing only vertices in $D(v)$ (the descendants of v in D). See Figure 8.1. Tarjan [57] proposed an algorithm that uses an NCA computation, incremental backward search, and a DSU data structure to compute H in $O(m\alpha(m, n))$ time on a pointer machine. We shall add microtrees, a maximal path partition of the core, and a stack to Tarjan's algorithm, thereby improving its running time to $O(m)$ on a pointer machine.

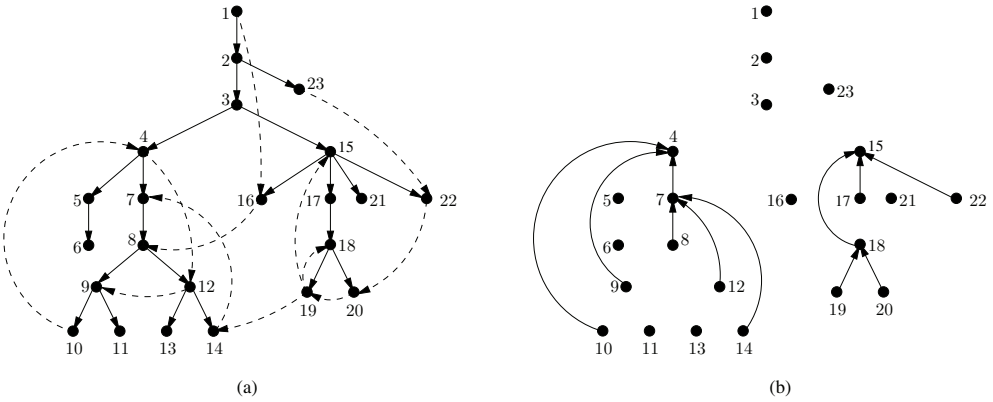


FIG. 8.1. (a) *A DFS tree D of the input flowgraph G; nontree arcs are dashed.* (b) *The interval forest H of G with respect to D; arrows are parent pointers.*

Tarjan's algorithm proceeds as follows: Delete all the arcs from the graph. For each vertex u, form a set of all deleted arcs (x, y) such that $\mathit{nca}(x, y) = u$. Process the vertices in any bottom-up order. (Tarjan uses reverse preorder, but any bottom-up order will do.) To process a vertex u, add back to the graph arcs corresponding to all the deleted arcs (x, y) with $\mathit{nca}(x, y) = u$. Then examine each arc (v, u) entering u. If $v \neq u$, set $h(v) = u$, and contract v into u; for all arcs having v as an end, replace v by u. This may create multiple arcs and loops, which poses no difficulty for the algorithm. Continue until all arcs into u have been examined, including those formed by contraction. When adding arcs back to the graph, the arc corresponding to an original arc is the one formed by doing end replacements corresponding to all the contractions done so far.

To keep track of contractions, Tarjan's algorithm uses a DSU structure whose elements are the graph vertices. The algorithm also uses a reverse adjacency set $R(u)$, initially empty, for each vertex u. A more detailed description of the algorithm is as follows. To process u, for each arc (x, y) such that $\mathit{nca}(x, y) = u$, add x to $R(\mathit{find}(y))$. (The replacement for x is done later.) Then, while $R(u)$ is nonempty, delete a vertex x from $R(u)$; let $v \leftarrow \mathit{find}(x)$; if $v \neq u$, set $h(v) \leftarrow u$, set $R(u) \leftarrow R(u) \cup R(v)$, and do $\mathit{unite}(u, v)$.

With the sets $R(u)$ represented as singly linked circular lists (so that set union takes constant time), the running time of this algorithm on a pointer machine is linear except for the NCA computations and the DSU operations, which take $O(m\alpha(m,n))$ time in Tarjan's original implementation. We shall reduce the running time to linear by using microtrees to eliminate redundant computation and by reordering the unites into a bottom-up order.

As in section 6, partition D into a set of bottom-level microtrees (the fringe), each with fewer than $g = \log^{1/3} n$ vertices, and D', the remainder of D (the core). Use a topological graph computation to compute $h(v)$ for every vertex v such that $h(v)$ is in the fringe. The definition of heads implies that for any such vertex v, $h(v)$ and v are in the same microtree, and furthermore that the only information needed to compute heads in the fringe is, for each microtree, the subgraph induced by its vertices, with nontree edges marked by a bit. With $g = \log^{1/3} n$, this computation takes $O(m)$ time by Theorem 5.2.

It remains to compute heads for vertices whose heads are in the core. Our approach is to run Tarjan's algorithm starting from the state it would have reached after processing the fringe. This amounts to contracting all the strong components in the fringe and then running the algorithm. This approach does not quite work as stated, because the DSU operations are not restricted enough for Lemma 4.6 to apply. To overcome this difficulty, we partition the core into maximal paths. Then we run Tarjan's algorithm path-by-path, keeping track of contractions with a hybrid structure consisting of a DSU structure that maintains contractions outside the path being processed and a stack that maintains contractions inside the path being processed. The latter structure functions in the same way as the one Gabow used in his algorithm [28] for finding strong components. Now we give the complete description of our algorithm.

Partition the vertices in D' into a set of maximal paths by choosing, for each nonleaf vertex v in D', a child $c(v)$ in D'. (Any child will do.) The arcs $(v, c(v))$ form a set of paths that partition the vertices in D'. For such a path P, we denote the smallest and largest vertices on P by $top(P)$ and $bottom(P)$, respectively; $bottom(P)$ is a leaf of D'. Since D' has at most n/g leaves, the number of paths is at most n/g. Partitioning D' into paths takes $O(n)$ time.

After constructing a maximal path partition of the core, initialize a DSU structure containing every vertex (fringe and core) as a singleton set. Visit the fringe vertices in bottom-up order, and, for each fringe vertex v with $h(v)$ also in the fringe, perform $unite(h(v), v)$; for such a vertex, $h(v)$ has already been computed. Initialize $R(u) \leftarrow \emptyset$ for every vertex u. For every arc (x, y) with x and y in the same microtree, add x to $R(find(y))$. For every remaining arc (x, y), compute $u = nca(x, y)$ and add (x, y) to the set of arcs associated with u. These NCA computations take $O(m)$ time using the algorithm of section 6. Indeed, every NCA query is big, so the AHU algorithm answers them in linear time. This completes the initialization.

Now process each path P in the path partition, in bottom-up order with respect to $top(P)$. To process a path P, initialize an empty stack S. Process each vertex u of P in bottom-up order. To process u, for each arc (x, y) such that $nca(x, y) = u$, add x to $R(find(y))$ unless $u = x$ and $p(y) \neq x$. (Heads do not depend on arcs (x, y) such that x is an ancestor of y but not its parent.) Then, while $R(u)$ is nonempty, delete a vertex x from $R(u)$. Let $v \leftarrow find(x)$. If v is not on P, set $h(v) \leftarrow u$, set $R(u) \leftarrow R(u) \cup R(v)$, and do $unite(u, v)$. If, on the other hand, v is on P, $v \neq u$, and v is no less than the top vertex on S, pop from S each vertex w less than or equal to v, set $h(w) \leftarrow u$, and set $R(u) \leftarrow R(u) \cup R(w)$. Once $R(u)$ is empty, push u onto

S. After processing all vertices on P, visit each vertex u on P again, in bottom-up order, and if $h(u)$ is now defined, perform *unite*$(h(u), u)$. See Figure 8.2

This algorithm delays the unites for vertices on a path until the entire path is processed, using the stack to keep track of the corresponding contractions. Specifically, the algorithm maintains the following invariant: if vertex u on path P is currently being processed and x is any original vertex, then the vertex into which x has been contracted is $v =$ *find*(x) if v is not on P, or the largest vertex on S less than or equal to v if v is on P and S is nonempty, or u otherwise. It is straightforward to verify this invariant by induction on time; the correctness of this implementation of Tarjan's algorithm follows.

THEOREM 8.1. *The interval analysis algorithm runs in $O(m)$ time on a pointer machine.*

Proof. The running time is linear except for the find operations: each vertex gets added to S once and has its head set at most once. To bound the time for the find operations, we apply Lemma 4.6 to the tree built by the parent assignments done by the unite operations. Mark the tops of all paths. Since there are at most n/g paths, there are at most $n/g = n/\log^{1/3} n$ marked vertices. We claim that $k = 5$ satisfies the hypothesis of the lemma. We need a property of the interval forest H: if $h(v) = u$, then every vertex $w \neq u$ on the path in D from u to v is a descendant of u in H. This holds because there is a path containing only vertices in $D(u)$ from w to v (via D) to u.

Consider any vertex v. We bound the number of times the parent of v in the DSU structure can change before v is in a set with a marked vertex. One parent change can occur during the initialization. After initialization, the parent of v can change once by a find before v is in a set with a designated vertex on the current path P. The parent of v can change only once more by a find while P is the current path, since its set does not change again until P is no longer the current path. Once P is not the current path, the next parent change of v caused by a find results in v being in a set with a designated vertex on the new current path Q. The parent of v can change only once more while Q is the current path, after which it is in the same set as *top*(P) (by the property above) and thus is in a set with a marked vertex. Therefore, the parent of v can change at most five times before it is in a set with a marked vertex, so the claim is true.

With $k = 5$ and $\ell \leq n/\log^{1/3} n$, Lemma 4.6 gives a bound of $O(m)$ on the time for the *find* operations. □

Interval analysis is an important component of program flow analysis [4]. It also has other applications, including testing flow graph reducibility [60], finding a pair of arc-disjoint spanning trees in a directed graph [57], and verifying a dominator tree [33]. Our interval analysis algorithm gives $O(m)$-time algorithms on a pointer machine for these applications as well.

In the next section we shall need a compressed version of the interval forest H' that is defined with respect to the fringe-core partition: the parent $h'(v)$ of a vertex v is its nearest core ancestor in H if it has one, *null* otherwise. We can easily compute H' from H in linear time, but if we only want H' and not H, we can avoid the topological graph computation on the microtrees: First, find the strong components of the graphs induced by the vertex sets of the microtrees. For each such component, find its smallest vertex u, and perform *unite*(u, v) for every other vertex v in the component. Then run the algorithm above for the core. This computes $h(v) = h'(v)$ for every vertex v with head in the core. Complete the computation by setting $h'(v) = h'(u)$ for each vertex $v \neq u$ in a fringe strong component with smallest vertex u.

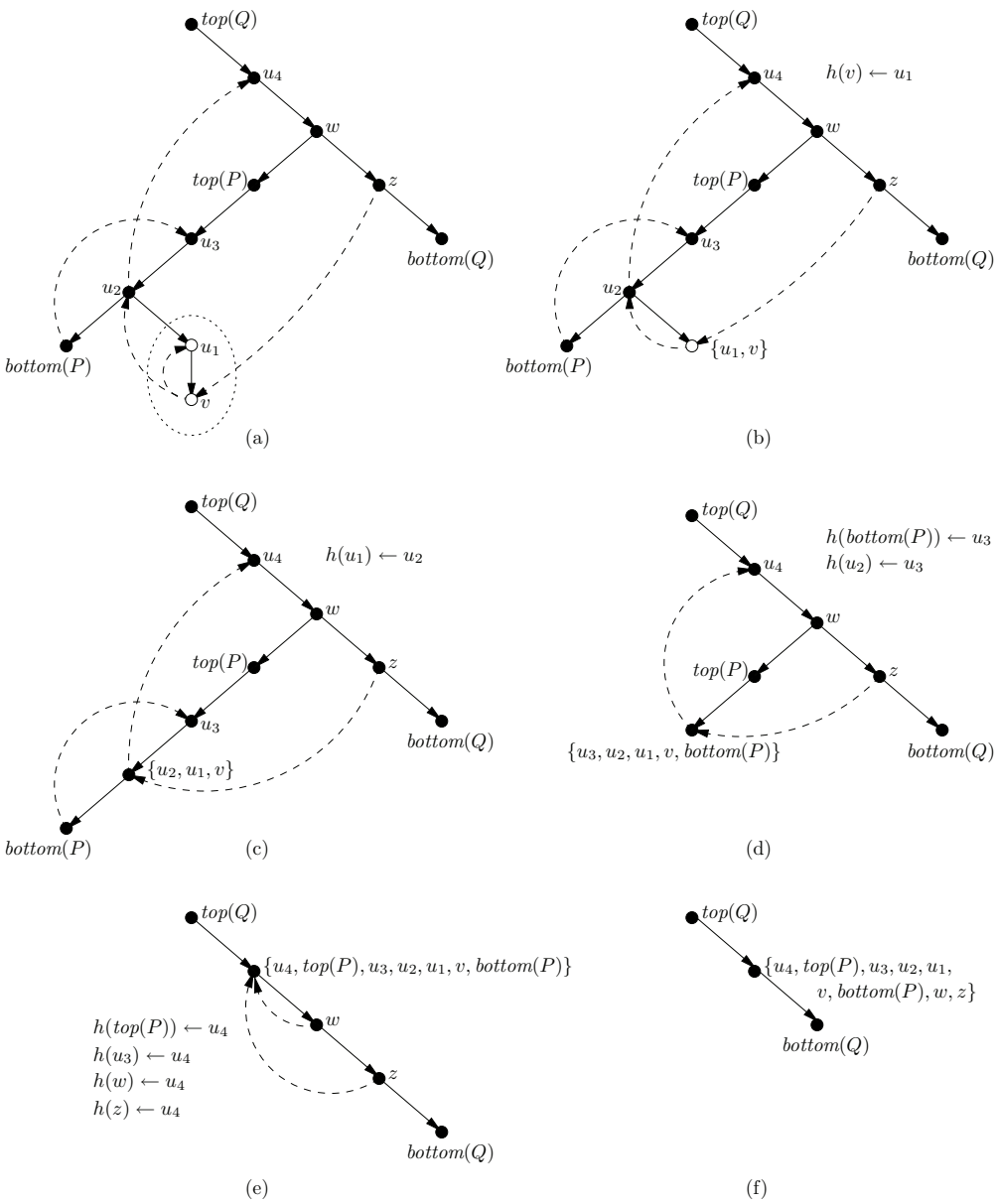


FIG. 8.2. *Idealized execution of the algorithm on the graph in* (a)*, with circled microtree. Arcs depict the effects of contractions: whenever* $x \in R(y)$*,* $(find(x), find(y))$ *is an arc in the corresponding graph. The first vertex in each labeled set is the corresponding original vertex in* (a). (a)→(b) *During preprocessing,* $h(v) \leftarrow u_1$*, and* v *is inserted into the set of* u_1*.* (b)→(c) *When processing* u_2*,* $h(u_1) \leftarrow u_2$ *via the arc* (v, u_2)*.* (c)→(d) *When processing* u_3*, the stack* S *is (top-down)* $(u_2, bottom(P))$*. Hence, when processing the arc* $(bottom(P), u_3)$*,* S *is popped so that* $h(u_2) \leftarrow u_3$ *and* $h(bottom(P)) \leftarrow u_3$*.* (d) *shows the state after doing the unite*$(\cdot)$*'s for path* P*.* (d)→(e) *When processing* u_4*,* S *is* $(w, z, bottom(Q))$*. Arc* (u_2, u_4) *sets* $h(u_3) \leftarrow u_4$ *and adds* $top(P)$ *and* z *to* $R(u_4)$*. Processing* $top(P)$ *causes* $h(top(P)) \leftarrow u_4$*, and processing* z *pops the stack so that* $h(w) \leftarrow u_4$ *and* $h(z) \leftarrow u_4$*.* (f) *After processing path* Q*.*

9. Dominators. Our second flowgraph problem is finding immediate dominators. Let $G = (V, A, r)$ be a flowgraph. We denote the immediate dominator of any vertex v by $idom(v)$. Let D be an arbitrary but fixed DFS tree rooted at r. As in section 8, we identify vertices by their preorder number with respect to the DFS; reverse preorder is decreasing order by vertex number. We use the notation $v \stackrel{*}{\to} w$ to denote that v is an ancestor of w in D, and $v \stackrel{+}{\to} w$ to denote that v is a proper ancestor of w in D. Sometimes we use the same notation to denote the respective paths in D from v to w. We denote by $p(v)$ the parent of v in D. We shall need the following basic property of DFS.

LEMMA 9.1 (see [54]). *Any path from a vertex v to a vertex $w > v$ contains a common ancestor of v and w.*

We shall describe an algorithm to compute immediate dominators in $O(m)$ time on a pointer machine. This is our most complicated application: it uses all the ideas and algorithms we have developed so far. Our algorithm is a reengineering of the algorithms presented by Buchsbaum et al. [16] and Georgiadis and Tarjan [31, 32]. As we proceed with the description, we shall point out the relationships between concepts we introduce here and the corresponding ideas in those previous works.

9.1. Semidominators, relative dominators, tags, and extended tags. Lengauer and Tarjan (LT) [43] devised a three-pass, $O(m\alpha(m, n))$-time algorithm to compute immediate dominators. We shall improve their algorithm by speeding up the first two steps. Central to the LT algorithm is the concept of *semidominators*. A path $x_0, x_1, \ldots, x_k$ in G is a *high path* if $x_i > x_k$ for $i < k$. As a degenerate case, a single vertex is a high path. A high path avoids all proper ancestors of its last vertex. The *semidominator* of a vertex w is

$$sdom(w) = \min(\{w\} \cup \{u : \text{for some } (u, v) \text{ in } A \text{ there is a high path from } v \text{ to } w\}).$$

The *relative dominator* of a vertex w is

$$rdom(w) = \operatorname{argmin}\{sdom(u) : sdom(w) \stackrel{+}{\to} u \stackrel{*}{\to} w\}.$$

With this definition, relative dominators are not unique, but for any vertex any relative dominator will do.

The LT algorithm operates as follows:

Step 1. Compute semidominators.

Step 2. Compute relative dominators from semidominators.

Step 3. Compute immediate dominators from relative dominators.

Step 3 relies on the following lemma.

LEMMA 9.2 (see [43, Cor. 1]). *For any vertex $v \neq r$, if $sdom(rdom(v)) = sdom(v)$, then $idom(v) = sdom(v)$; otherwise, $idom(v) = idom(rdom(v))$.*

Using Lemma 9.2, the LT algorithm performs Step 3 in a straightforward top-down pass over D that takes $O(n)$ time on a pointer machine.

The LT algorithm performs Steps 1 and 2 in a single pass that visits the vertices of D in reverse preorder and uses a link-eval data structure to compute semidominators and relative dominators. We shall present separate algorithms for Steps 1 and 2, although these steps can be partially combined, as we discuss in section 9.7.

Step 2 is almost identical to MST verification. Indeed, suppose we assign a cost $sdom(v)$ to each tree arc $(p(v), v)$ and apply the MST verification algorithm to the tree D (ignoring arc directions) with query set $Q = \{\{sdom(v), v\} : v \neq r\}$, with the modification that the answer to a query is an arc of minimum cost on the query path

rather than the cost of such an arc. Then for $v \neq r$, $rdom(v)$ is the vertex u such that $(p(u), u)$ is the answer to the query $(sdom(v), v)$. Modifying the link-eval structure to replace maximum by minimum and to return arcs (or, better, vertices) rather than costs is straightforward. The algorithm of section 7 thus performs Step 2 in $O(n)$ time on a pointer machine. (The number of queries is $O(n)$.)

It remains to implement Step 1, the computation of semidominators. Lengauer and Tarjan reduce this computation, also, to a problem of finding minima on tree paths, using the following lemma.

LEMMA 9.3 (see [43, Thm. 4]). *For any vertex w,*

$$sdom(w) = \min\big(\{w\} \cup \{nca(u,w) : (u,w) \in A\} \cup \{sdom(v) : \exists (u,w) \in A, nca(u,w) \stackrel{+}{\rightarrow} v \stackrel{*}{\rightarrow} u\}\big).$$

The lemma gives a recurrence for $sdom(w)$ in terms of $sdom(v)$ for $v > w$. The LT algorithm performs Step 1 by visiting the vertices in reverse preorder and using a link-eval structure to perform the computations needed to evaluate the recurrence.

Even though Step 1 is now reduced to computing minima on tree paths, we cannot use the MST verification algorithm directly for this purpose, because that algorithm answers the queries in an order incompatible with the requirements of the recurrence. Instead we develop an alternative strategy. For convenience we restate the problem, which allows us to simplify slightly the recurrence in Lemma 9.3. Suppose each vertex w has an integer *tag* $t(w)$ in the range $[1, n]$. The *extended tag* of a vertex w is defined to be

$$et(w) = \min\{t(v) : \text{there is a high path from } v \text{ to } w\}.$$

LEMMA 9.4. *If $t(w) = \min(\{w\} \cup \{v : (v,w) \in A\})$ for every vertex, then $sdom(w) = et(w)$ for every vertex.*

Proof. The proof is immediate from the definitions of semidominators and extended tags. □

We can easily compute the tag specified in Lemma 9.4 for every vertex in $O(m)$ time. Thus the problem of computing semidominators becomes that of computing extended tags.

Lemma 9.3 extends to give the following recurrence for extended tags.

LEMMA 9.5. *For any vertex w,*

$$et(w) = \min(\{t(w)\} \cup \{et(v) : \exists (u,w) \in A,\ nca(u,w) \stackrel{+}{\rightarrow} v \stackrel{*}{\rightarrow} u\}).$$

Proof. The proof is analogous to that of Lemma 9.3. Let x be the right side of the equation in the statement of the lemma. First we prove $et(w) \leq x$. If $x = t(w)$, $et(w) \leq x$ is immediate from the definition of $et(w)$. Suppose $x = et(v)$ for v such that $nca(u,w) \stackrel{+}{\rightarrow} v \stackrel{*}{\rightarrow} u$ and (u,w) in A. By the definition of $et(v)$, $et(v) = t(z)$ for some vertex z such that there is a high path from z to v. Extending this path by the tree path from v to u followed by the arc (u,w) gives a high path from z to w. Hence $et(w) \leq et(v) = x$.

Next we prove $x \leq et(w)$. Let z be a vertex such that $et(w) = t(z)$ and there is a high path from z to w (by the definition of the extended tags). If $z = w$, then $x \leq et(w)$ from the definition of x. If not, let (u,w) be the last arc on the high path from z to w. Let v be the first vertex along the high path such that $nca(u,w) \stackrel{+}{\rightarrow} v \stackrel{*}{\rightarrow} u$. Such a v exists since u is a candidate ($nca(u,w) \leq w < u$). We claim that the part of

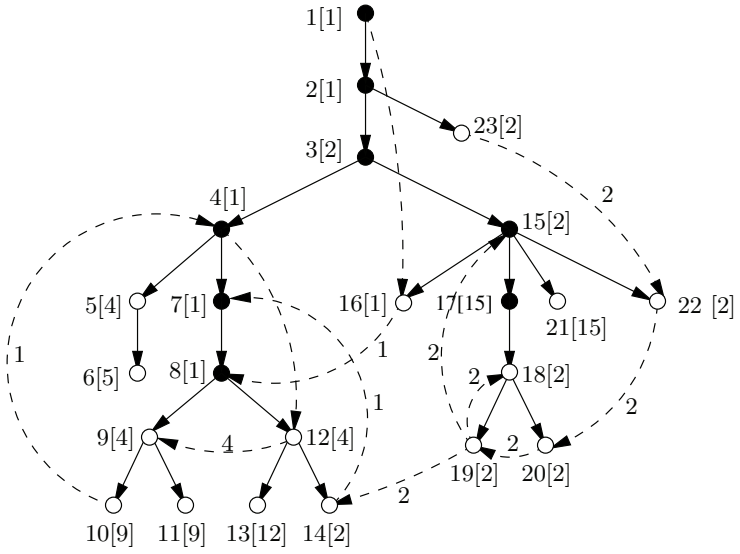


FIG. 9.1. *Extended tags and arc tags. The number inside each bracket is the extended tag of the corresponding vertex. The number on each arc is its tag; the arc tag of a tree or forward arc is infinite and not shown in the figure.*

the high path from z to v is itself a high path. Suppose to the contrary that this part contains a vertex less than v, and let y be the last such vertex. Then y must be an ancestor of v by Lemma 9.1, and since y is on a high path for w, $nca(u,w) \xrightarrow{+} y \xrightarrow{+} v$. This contradicts the choice of v. It follows that $et(v) \le t(z)$; that is, $x \le et(w)$. □

We introduce one more definition that simplifies some of our formulas and discussion. For an arc (u,w), the *arc tag* of (u,w) is

$$at(u,w) = \min\{et(v) : nca(u,w) \xrightarrow{+} v \xrightarrow{*} u\}$$

if this minimum is over a nonempty set, and infinity otherwise (when $nca(u,w) = u$). An example is shown in Figure 9.1. Using arc tags, the recurrence in Lemma 9.5 becomes

$$et(w) = \min(\{t(w)\} \cup \{at(u,w) : (u,w) \in A\}). \tag{9.1}$$

9.2. The interval forest. We could use (9.1) to compute extended tags just as the LT algorithm uses Lemma 9.3 to compute semidominators, but we seek a faster method. Note that there are two kinds of arcs (u,w) that must be handled: those such that u and w are unrelated (*cross arcs*), and those such that $w \xrightarrow{+} u$ (*back arcs*). (Arcs such that $u \xrightarrow{+} w$ do not contribute to the recurrence.) We apply different techniques to the cross arcs and the back arcs, which allows us to tease apart the intertwined computations implied by (9.1) and reorder them to apply our techniques.

To handle the back arcs, we use the interval forest discussed in section 8. Recall the following definitions. For each vertex w, the head $h(w)$ of w is the maximum vertex $u \ne w$ such that there is a path from w to u containing only descendants of u, if this maximum is over a nonempty set, and *null* otherwise. Lemma 9.1 implies that the constraint on u in the definition of $h(w)$ is equivalent to $u \xrightarrow{+} w$ and there is a high path from w to u. The heads define a forest H called the *interval forest*: $h(w)$

is the parent of w in H. The following lemma allows us to compute extended tags by computing arc tags only for the cross arcs and propagating minima up the interval forest.

LEMMA 9.6. *For any vertex w,*

$$et(w) = \min(\{t(v) : v \in H(w)\} \cup \{at(u,v) : (u,v) \in A,\ v \in H(w),\ u \notin D(w)\}).$$

Proof. Let x be the right side of the equation in the statement of the lemma. First we prove $et(w) \le x$. Let v be in $H(w)$. Since there is a high path from v to w, $et(w) \le t(v)$. Let (u,v) be in A such that v is in $H(w)$ but u is not in $D(w)$. Let y be a vertex of minimum $et(y)$ such that $nca(u,v) \xrightarrow{+} y \xrightarrow{*} u$, and let z be a vertex of minimum $t(z)$ such that there is a high path from z to y. Then there is a high path from z to y to u to v to w, which implies $et(w) \le t(z) = et(y) = at(u,v)$. We conclude that $et(w) \le x$.

Next we prove $x \le et(w)$. Let z be a vertex such that $et(w) = t(z)$ and there is a high path from z to w. If z is in $H(w)$, then $x \le t(z) = et(w)$. Suppose z is not in $H(w)$. Let (u,v) be the first arc along the high path from z to w such that v is in $H(w)$. Then u cannot be in $D(w)$, or it would be in $H(w)$, contradicting the choice of (u,v). Thus $nca(u,v) \xrightarrow{+} u$. Let y be the first vertex along the high path such that $nca(u,v) \xrightarrow{+} y \xrightarrow{*} u$. By Lemma 9.1, the part of the high path from z to y is itself a high path. Thus $x \le at(u,v) \le et(y) \le t(z) = et(w)$. □

COROLLARY 9.7. *For any vertex w,*

$$et(w) = \min(\{t(w)\} \cup \{et(v) : h(v) = w\} \cup \{at(v,w) : (v,w) \textit{ is a cross arc}\}).$$

Corollary 9.7 gives an alternative recursion for computing extended tags by processing the vertices in reverse preorder. Lemma 9.6 also allows us to compute extended tags for all the vertices on a tree path, given only arc tags for arcs starting to the right of the path.

9.3. Microtrees and left paths. As in section 6, we partition D into a set of bottom-level microtrees (the fringe), each containing fewer than $g = \log^{1/3} n$ vertices, and D' (the core), the remainder of D. We call a cross arc *small* if both its ends are in the same microtree, and *big* otherwise. We also partition D' into maximal paths as in section 8, but a particular set of maximal paths. Specifically, we partition D' into *left paths*, as follows: An arc $(p(v), v)$ of D' is a *left arc* if v is the smallest child of $p(v)$ in D'. A *left path* is a maximal sequence of left arcs. We can partition D into microtrees and left paths in $O(m)$ time during the DFS that defines D. If P is a left path, as in section 8 we denote by $top(P)$ and $bottom(P)$ the smallest and largest vertices on P, respectively. The importance of left paths is twofold. First, there are at most n/g of them. Second, if $(p(v), v)$ is a left arc, any child of $p(v)$ smaller than v must be in the fringe, not the core. That is, left paths have only microtrees descending on their left. Left paths serve in place of the *lines* of Georgiadis and Tarjan [31, 32]; left paths are catenations of those lines.

Our hypothetical plan for computing extended tags in linear time is to use a topological graph computation to handle the microtrees and a link-eval structure to compute arc tags for the big cross arcs. This plan does not quite work: computing extended tags is unlike the previous problems we have considered in that there is an interaction between the fringe and the core. In particular, we need at least some information about the small cross arcs in order to compute extended tags in the core, and information about the big cross arcs to compute extended tags in the fringe. For

the former computation we do not, however, need to compute arc tags for the small cross arcs: the recurrence in Lemma 9.6 expresses the extended tags of vertices in the core in terms only of tags of vertices and arc tags of big cross arcs. To handle the limited interaction between fringe and core, we use a two-pass strategy. During the first pass, we compute arc tags of big cross arcs and extended tags in the core while computing limited information in the fringe. In the second pass, we use the information computed in the first pass in a topological graph computation to compute extended tags in the fringe.

The information we need in the fringe is a set of values defined as follows. For a vertex w in a microtree $D(s)$, the *microtag* of w is

$$\begin{aligned} mt(w) = \min\big(&\{t(v) : \text{there is a path from } v \text{ to } w \text{ in } D(s)\} \cup \\ &\{at(u,v) : (u,v) \text{ is a cross arc}, v \in D(s), u \notin D(s), \\ &\qquad \text{and there is a path in } D(s) \text{ from } v \text{ to } w\}\big). \end{aligned}$$

Our microtags correspond to the *pushed external dominators* of Buchsbaum et al. [16] (also used by Georgiadis and Tarjan [31, 32]). The next lemma shows that when computing the arc tags of big cross arcs, we can use microtags in place of extended tags for fringe vertices; that is, we shall use microtag values in the link-eval structure when linking fringe vertices.

LEMMA 9.8. *Let w be a vertex in a microtree $D(s)$. Then*

$$\min\{et(v) : s \stackrel{*}{\to} v \stackrel{*}{\to} w\} = \min\{mt(v) : s \stackrel{*}{\to} v \stackrel{*}{\to} w\}.$$

Proof. Let x and y be the values of the left and right sides of the equation in the statement of the lemma, respectively. First we prove that $x \geq y$. Let v be a vertex such that $x = et(v)$ and $s \stackrel{*}{\to} v \stackrel{*}{\to} w$. Let z be a vertex such that $t(z) = et(v)$ and there is a high path from z to v. If z is in $D(s)$, then this high path is in $D(s)$, which implies that $x = t(z) \geq mt(v) \geq y$. Suppose on the other hand that z is not in $D(s)$. Let (p,q) be the last arc along the high path such that p is not in $D(s)$, and let z' be the first vertex along the high path such that $nca(p,q) \stackrel{+}{\to} z' \stackrel{*}{\to} p$. Note that (p,q) must be a cross arc, since p is not in $D(s)$ and is on a high path to v in $D(s)$. See Figure 9.2. As in the proof of Lemma 9.5, the part of the high path from z to z' is itself a high path, which implies $x = t(z) \geq et(z') \geq at(p,q) \geq mt(v) \geq y$.

Next we prove that $x \leq y$. Let v be a vertex such that $y = mt(v)$ and $s \stackrel{*}{\to} v \stackrel{*}{\to} w$. Suppose $mt(v) = t(z)$ for some z in $D(s)$ from which there is a path to v in $D(s)$. Let u be the first vertex on this path that is an ancestor of w. Then the path from z to u is a high path by Lemma 9.1 and the choice of u. Thus $x \leq et(u) \leq t(z) = y$. Suppose on the other hand that $mt(v) = at(p,q)$ for an arc (p,q) such that q but not p is in $D(s)$ and there is a path from q to v. Let u be the first vertex on this path that is an ancestor of w. By Lemma 9.1, the part of the path from q to u is a high path. Let z be a vertex such that $t(z) = at(p,q)$ and there is a high path from z to a vertex z' such that $nca(p,q) \stackrel{+}{\to} z' \stackrel{*}{\to} p$. See Figure 9.2. This path, together with the path $z' \stackrel{*}{\to} p$, the arc (p,q), and the high path from q to u, is a high path. Thus $x \leq et(u) \leq t(z) = at(p,q) = mt(v) = y$. □

To help compute extended tags during the first pass, we use a compressed interval forest H' in place of the interval forest H. Recall that in H', the parent $h'(v)$ of a vertex v is the nearest ancestor of v in H that is a core vertex. Forests H and H' are identical on the core; each subtree of H consisting of fringe vertices with a core root is compressed in H' to the root with all the fringe vertices as children. The use

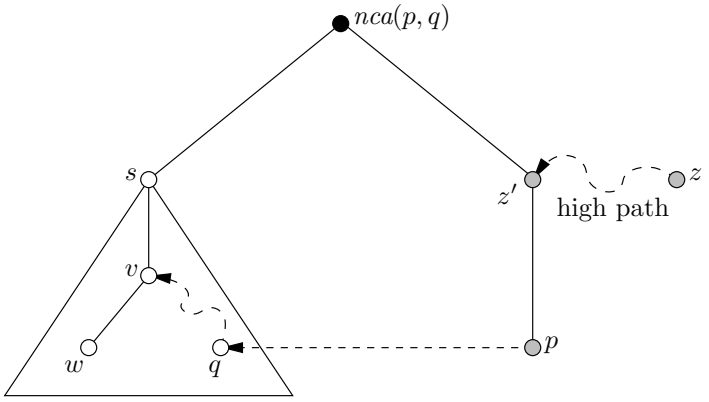


FIG. 9.2. *Proof of Lemma* 9.8. *Dashed curves represent graph paths. Solid edges represent tree paths. Each gray vertex can be in the core or in the fringe.*

of H' in place of H is an optimization only: we can build either H or H' in linear time using the algorithm of section 8, but, as noted in section 8, building H' instead of H avoids the use of topological graph computations on the microtrees and thus is simpler. The algorithm of section 8 builds H' by partitioning D into microtrees and maximal paths. We can use the set of left paths as the maximal paths, avoiding the need for two different partitions.

To compute extended tags in the core, we use the following corollary of Lemma 9.6.

COROLLARY 9.9. *If w is a core vertex*

$$\begin{aligned} et(w) = \min\big(&\{t(v) : v = w \text{ or } v \text{ is fringe with } h'(v) = w\} \cup \\ &\{et(v) : v \text{ is core with } h'(v) = w\} \cup \\ &\{at(u,v) : (u,v) \text{ is a big cross arc such that} \\ &\qquad v = w \text{ or } v \text{ is fringe with } h'(v) = w\}\big). \end{aligned}$$

The algorithm of Georgiadis and Tarjan [32] for computing dominators does not use H' explicitly, but it does do an incremental backward search using a stack to maintain strongly connected parts of lines, in effect doing a just-in-time computation of (part of) H'. Making this computation separate, as we have done, breaks the overall algorithm into smaller, easier-to-understand parts, which could be combined if desired.

9.4. Computation of arc tags. The heart of the algorithm is the computation of arc tags. We split each such computation into two parts, either of which can be void: a *top part*, which computes a minimum of extended tags over part or all of a left path, and a *bottom part*, which computes a minimum of extended tags of core vertices and microtags of fringe vertices using a sophisticated link-eval structure. Specifically, let (u, v) be a big cross arc. Let P be the left path containing $nca(u, v)$, and let Q be the intersection of P and the path $nca(u, v) \xrightarrow{+} u$. We denote the last vertex on Q by $mid(u, v)$. Note that Q can be nonempty (contain arcs) only if v is a fringe vertex. See Figure 9.3.

For a given left path P, we compute minima of extended tags for all such nonempty paths Q at the same time. We do not need to know any of these minima until all

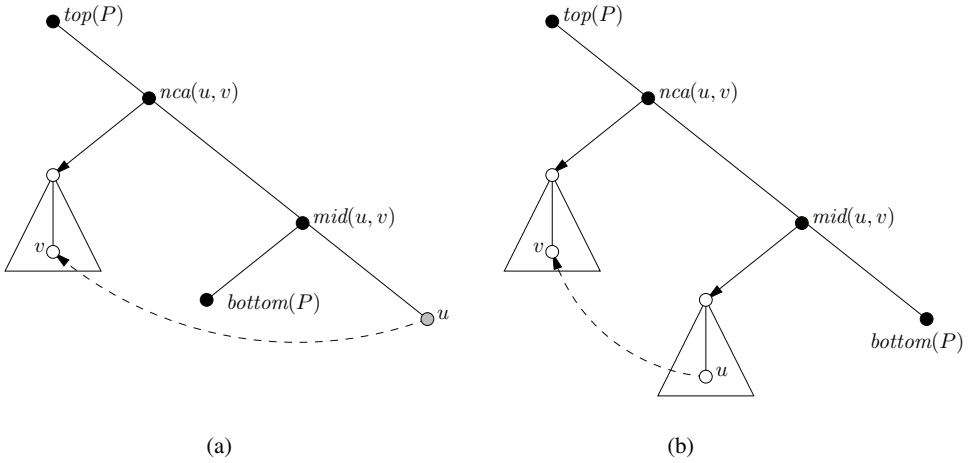


FIG. 9.3. *Examples of nonempty* $nca(u,v) \xrightarrow{+} mid(u,v)$ *paths.* (a) *Case* $u > bottom(P)$. (b) *Case* $u < bottom(P)$.

the extended tags for vertices on P have been computed. This allows us to compute the minima for such paths Q in arbitrary order. One way to compute these minima is to use the MST verification algorithm, as suggested above for doing Step 2 of the LT algorithm. In this application, however, the tree being verified is actually a path, and we can use an algorithm that is at least conceptually simpler, if not asymptotically faster. The problem we need to solve is that of computing minima for given subsequences of a sequence of numbers. This is the *range minimum query* (RMQ) problem [30]. This problem has a linear-time reduction [30] to an NCA problem on a Cartesian tree [65]. We can thus compute minima for paths Q by constructing the Cartesian tree and applying our NCA algorithm. Either method allows us to compute the top parts of arc tags in $O(m)$ time on a pointer machine.

To compute the bottom parts of arc tags, we use a sophisticated link-eval structure. We delay the links for arcs on a left path until the top of the left path is reached, and for arcs in a microtree until its root is reached. This allows us to establish a linear-time bound for all the link-eval operations using Lemma 4.6.

9.5. The first pass. We now have all the pieces necessary to describe the first pass of our algorithm for computing extended tags. Before the first pass, build the compressed interval forest H', compute $nca(u,v)$ for each big cross arc (u,v), and construct, for each core vertex w, the set of big cross arcs (u,v) with $nca(u,v) = w$. This takes $O(m)$ time on a pointer machine using the method of section 8: the NCAs are computed as part of the algorithm that builds H'. Each vertex v has a *computed tag* $ct(v)$ that is initialized to $t(v)$ and that decreases as the first pass proceeds, until $ct(v) = mt(v)$ if v is fringe, or $ct(v) = et(v)$ if v is core. Each fringe vertex v also has an associated set of cross arcs, initially empty. For each fringe vertex v, if v has a parent in H' and $ct(h'(v)) > ct(v)$, replace $ct(h'(v))$ by $ct(v)$. Finally, initialize a sophisticated link-eval data structure with no edges and each vertex of G as a node.

The first pass visits each microtree once and each left path twice. The visits are in reverse preorder with respect to the roots of the microtrees and the top and bottom vertices of the left paths; the first visit to a left path corresponds to its bottom (largest) vertex, and the second visit to its top (smallest) vertex. Conceptually, envision a

reverse preorder traversal of D, with actions taken as described below whenever a microtree root or bottom or top vertex of a left path is visited.

When visiting a microtree $D(s)$, it will be true that, for each vertex v in $D(s)$,

$$et(v) \le ct(v) \le \min(\{t(v)\} \cup \{at(u,v) : (u,v) \in A,\ u \notin D(s)\}). \tag{9.2}$$

Compute microtags for all vertices in $D(s)$ by finding the strong components of the subgraph induced by the vertices in $D(s)$ and processing the strong components in topological order. To process a component, compute a microtag for the component, equal to the minimum of the $ct(\cdot)$ values for all vertices in the component and the microtags for all preceding components (those with an arc leading to the component). Then set $ct(v)$ for every vertex in the component equal to the computed microtag. The assigned value of $ct(v)$ must be $mt(v)$, assuming (9.2) holds. The time required for this computation is linear in the size of the subgraph induced by $D(s)$ [54]. Having computed microtags for $D(s)$, perform $link(p(v), v, ct(v))$ for every vertex in $D(s)$, in bottom-up order. Finally, for each cross arc (u, v) in the set of cross arcs of a vertex u in $D(s)$, set $ct(v) \leftarrow \min\{ct(v), eval(u)\}$, and then set $ct(h'(v)) \leftarrow \min\{ct(h'(v)), ct(v)\}$ if v has a parent in H'. Such computations happen here only for arcs (u, v) such that u is in a microtree hanging on the left of some left path. It will become clear later that, for such an arc, the top part of the evaluation of $at(u, v)$ gets done first, when the left path is processed. The $eval(u)$ operation does the bottom part of the evaluation, finishing the job. We describe below when these arcs are entered in the set associated with u.

When visiting a left path P for the first time, begin by visiting the vertices w of P in bottom-up order and setting $ct(h'(w)) \leftarrow \min\{ct(h'(w)), ct(w)\}$ if w has a parent in H'. Once these updates are completed, $ct(w) = et(w)$ for every vertex w on P. Then collect all the arcs (u, v) in the sets associated with the vertices on P, i.e., the arcs (u, v) such that $nca(u, v) \in P$. For each such arc (u, v), set $mid(u, v) \leftarrow p(root(micro(u)))$ if $u < bottom(P)$, and $mid(u, v) \leftarrow findroot(u)$ otherwise. The *findroot* operation in the latter case is an operation on the link-eval structure. Having computed all the *mid* values for all the cross arcs, evaluate the top parts of their arc tags, using either of the methods discussed in section 9.4. For each such arc (u, v) with computed arc tag top part x, do the following. If $u > bottom(v)$ (see Figure 9.3(a)), set $x \leftarrow \min\{x, eval(u)\}$; otherwise (see Figure 9.3(b)), add (u, v) to the set of cross arcs of u. In the former case, the $eval(u)$ operation computes the bottom part of the arc tag; in the latter case, the computation of the bottom part is done when the microtree containing u (which hangs to the left of P) is visited. In either case, set $ct(v) \leftarrow \min\{ct(v), x\}$, and then set $ct(h'(v)) \leftarrow \min\{ct(h'(v)), ct(v)\}$ if v is a fringe vertex with a parent in H'.

When visiting a left path P for the second time, perform $link(p(w), w, ct(w))$ for each vertex on P in bottom-up order, unless P is the last path, in which case the first pass is done.

Based on the results of the previous sections, it is straightforward (but tedious) to prove that this algorithm correctly computes extended tags. Note that the algorithm eagerly pushes $ct(\cdot)$ values up H', rather than lazily pulling them; the latter would require computing sets of children for H', whereas the former can be done using just parent pointers.

LEMMA 9.10. *The first pass takes $O(m)$ time on a pointer machine.*

Proof. The running time of all parts of the algorithm is linear based on previous results, except for the *findroot* and *eval* operations. To bound the time for these, we

apply Lemma 4.6 to the shadow subtrees built by the link operations. These subtrees are $\sqrt{2}$-balanced by Corollary 4.2 for linking-by-size and Corollary 4.4 for linking-by-rank. Mark the parents (in D) of the tops of all the left paths. This marks at most $n/g = n/\log^{1/3} n$ vertices. We claim that $k = 5$ satisfies the hypothesis of the lemma.

We need to use details of the link implementation, for which we refer the reader to section 4.3 for linking-by-size and section 4.4 for linking-by-rank. The links occur in batches with no intermixed *findroot* or *eval* operations, one batch per microtree and one batch per left path. Let v be any vertex. We count the number of times the subroot of the shadow subtree containing v can change, as the result of a batch of links, before v is in a subtree containing a marked node. Let $v_0 = v, v_1, v_2, \ldots$ be the successive roots of the shadow trees containing v. The subroot of the shadow subtree containing v can change only as the result of a batch of links that include the current v_i as one of the vertices being linked. Suppose v is fringe. The first batch of links to include v_0 is the one for $\mathit{micro}(v)$. This batch of links makes $p(\mathit{root}(\mathit{micro}(v)))$ the root of the tree containing v; that is, $v_1 = p(\mathit{root}(\mathit{micro}(v)))$. The next batches of links that include v_1 are those for other microtrees whose roots are children of v_1 in D. Such a batch does not change the root of the tree containing v but can change the subroot of the subtree containing v, making it equal to v_1. Once such links are done, the only remaining batch of links that includes v_1 is the one for the left path P_1 containing v_1. This batch makes $v_2 = p(\mathit{top}(P_1))$, which means that the shadow tree containing v (but not necessarily the shadow subtree containing v) has a marked vertex. The next batches of links that include v_2 are those for microtrees whose roots are children of v_2 in D. Such a batch cannot change the root of the tree containing v, but it can change the subroot of the subtree containing v, making it equal to v_2, which is marked. Otherwise, the next (and last) batch of links that includes v_2 is the one for the left path P_2 containing v_2. This batch makes $v_3 = p(\mathit{top}(P_2))$.

Now v is either in the subtree rooted at v_3, and hence in a subtree with a marked vertex, or it is a shadow descendant of v_2, which is no longer the root of the shadow tree containing v. No subsequent link can change the root of the subtree containing v without putting v and v_2, which is marked, in the same subtree. Tracing through the analysis above, we see that the subroot of the shadow subtree containing a fringe vertex v can change at most four times before v is in a subtree with a marked vertex. If v is a core vertex, the last part of the same analysis applies: the first batch of links that can change either the root of the tree containing v or the subroot of the subtree containing v is the one for the left path containing v; the subroot of the subtree containing v can change at most twice before v is in a subtree with a marked vertex. The shadow parent of vertex v can change at most once before the root of the shadow subtree containing v changes. Thus the shadow parent of v can change at most five times before v is in a shadow subtree with a marked vertex. This verifies the claim. With $k = 5$ and $\ell \le n/\log^{1/3} n$, Lemma 4.6 gives a bound of $O(m)$ on the time for the *findroot* and *eval* operations. □

9.6. The second pass. Having computed extended tags for all core vertices, we compute extended tags for all fringe vertices by using a topological graph computation on the microtrees. In the first pass, just before a microtree $D(s)$ is processed, each vertex v in $D(s)$ has $ct(v) = \min(\{t(v)\} \cup \{at(u,v) : (u,v) \in A,\ u \notin D(s)\})$. It follows that if we compute extended tags within the subgraph induced by the vertices of $D(s)$, using these $ct(\cdot)$ values as the initial tags, we will obtain the correct extended tags for the vertices in $D(s)$ with respect to the original tags in the entire graph. The $ct(\cdot)$ values are in the range $[1, n]$, but we can map them to the range $[1, g]$ by sorting all the

$ct(\cdot)$ values using a pointer-based radix sort, extracting a sorted list of $ct(\cdot)$ values for each subproblem, and mapping each such sorted list to $[1, g]$. To do this on a pointer machine, we need to maintain a singly linked master list of length n, whose nodes correspond to the integers 1 through n, and store with each integer a pointer to its corresponding position in the master list, and we need to track such pointers through the entire running of the algorithm. We assume that each input tag is given along with a corresponding pointer into the master list. For the special case of computing semidominators, we construct the master list and the corresponding pointers as we perform the DFS and number the vertices. The only manipulations of vertex numbers are comparisons, so it is easy to track these pointers through the entire computation.

Once the tags are mapped to $[1, g]$, the computation of extended tags on the microtrees is a topological graph computation, which we perform using the method described in section 6. With the choice $g = \log^{1/3} n$, the second pass requires $O(m)$ time on a pointer machine.

Combining all the parts of the algorithm, we obtain the following theorem.

THEOREM 9.11. *Finding immediate dominators takes $O(m)$ time on a pointer machine.*

9.7. An alternative method for Step 2. We conclude our discussion of dominators by sketching an alternative method for performing Step 2 (computing relative dominators) that does some of the work in the second pass of Step 1 and then uses a simplification of the algorithm for the first pass of Step 1 to do the rest.

For a microtree $D(s)$, the $ct(\cdot)$ values of its vertices just before $D(s)$ is processed provide enough information not only to compute the semidominators of each of its vertices but also to compute the relative dominator of each vertex v such that $sdom(v)$ is in $D(s)$. This we can do as part of the topological graph computation that forms the second pass of Step 1. The remaining part of Step 2 is to compute $rdom(v) = \text{argmin}\{sdom(u) : sdom(v) \stackrel{+}{\rightarrow} u \stackrel{*}{\rightarrow} v\}$ for each vertex v with $sdom(v)$ in the core. We can do this by running a simplified version of the first pass of Step 1. We modify the link-eval structure so that an *eval* returns a vertex of minimum value, rather than the value itself. We compute the relative dominators in the same way that pass 1 of Step 1 computes the arc tags of big cross arcs, but without using the interval tree H' and without using NCAs. We begin by storing each pair $(sdom(v), v)$ with $sdom(v)$. Then we perform $link(p(v), v, sdom(v))$ for every fringe vertex v, in reverse preorder. Finally, we process each left path P, in reverse preorder with respect to $bottom(P)$. To process a left path P, we collect all the pairs (u, v) stored with its vertices. For each such pair, we set $mid(u, v) \leftarrow findroot(v)$. We evaluate each top part from u to $mid(u, v)$ using an NCA computation on a derived Cartesian tree as discussed in section 9.4, modified to return a candidate relative dominator $rd(u, v)$ for each pair. For each pair we set $rdom(v) \leftarrow \text{argmin}\{sdom(eval(v)), sdom(rd(u, v))\}$. Finally, we perform $link(p(v), v, sdom(v))$ for every vertex on P in reverse preorder, unless P is the last path, in which case we are done. This method for doing Step 2 takes $O(n)$ time.

This approach also leads to an alternative algorithm for MST verification, as mentioned in section 7.5, which avoids the use of the Borůvka tree as an intermediate step, replacing it with NCA computations on Cartesian trees derived from the paths of a partition of the core of the original tree T into maximal paths. We must still do verification within microtrees, but these are microtrees of the original tree rather than of the Borůvka tree.

9.8. Remarks. From the definition of microtags we have that for any w in a microtree $D(s)$, $mt(w) \leq mt(v)$ for any $s \stackrel{*}{\rightarrow} v \stackrel{*}{\rightarrow} w$. This inequality implies that the

eval function need only operate on the core tree. The algorithms of Buchsbaum et al. [16] and Georgiadis and Tarjan [31, 32] rely on this fact but also require a hybrid link-eval structure for the evaluation of path minima on the core. Lemma 4.6 allows us to use a standard (simpler) link-eval structure that can include the fringe, which also yields a more uniform treatment of the core and fringe vertices.

Our dominators algorithm uses the linear-time offline NCA algorithm for two subproblems: interval analysis and range minimum queries. Georgiadis [31] observed that a refined partition of the core tree into unary paths of size $O(g)$ enables us to use trivial algorithms to compute NCAs; topological graph computations are still required, but they are performed on Cartesian trees corresponding to each unary path.

10. Component trees. Our final application is a tree problem, unusual in that it seems to require partitioning all of the given tree, rather than just the bottom part, into microtrees.

10.1. Kruskal trees. The Borůvka tree discussed in section 7 represents the connected components that are formed as Borůvka's MST algorithm is run. We can define the analogous concept for other MST algorithms. For example, the *Kruskal tree* is the tree whose nodes are the connected components formed as Kruskal's MST algorithm [42] is run. Kruskal's algorithm starts with all vertices in singleton components and examines the edges in increasing order by weight, adding an edge to the MST being built, and combining the two corresponding components when the edge has ends in two different components. The Kruskal tree K is binary, with one node per component, whose children are the components combined to form the given component. Each leaf of K is a vertex of the original graph; each nonleaf node is a nonsingleton component. See Figure 10.1.

Even if the given graph is a tree, constructing the Kruskal tree is equivalent to sorting the edges by weight, because the Kruskal tree for a star (a tree of diameter two) contains enough information to sort the edges. If we are given the edges in order by weight, however, the problem of constructing the Kruskal tree becomes more interesting. We shall develop an $O(n)$-time, pointer-machine algorithm to build the Kruskal tree K of a tree T, given a list of the edges of T in order by weight.

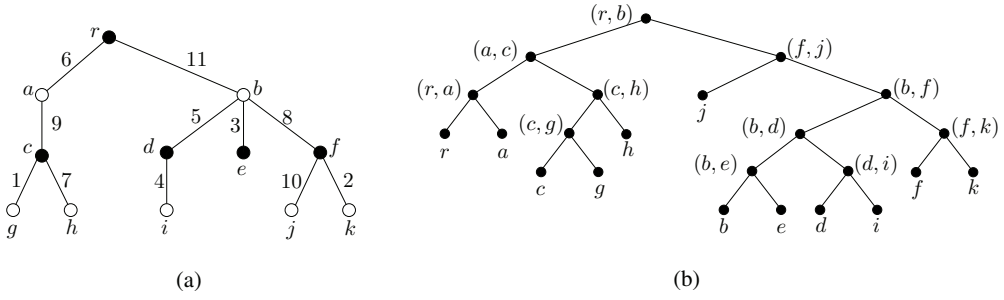


FIG. 10.1. (a) *The input weighted tree T; the filled nodes are subtree roots when T is partitioned with $g = 3$.* (b) *The Kruskal tree K of T. Leaves correspond to the nodes of T; internal nodes correspond to edges of T.*

10.2. Bottom-up construction of a Kruskal tree. It is straightforward to build K bottom-up using a DSU structure whose nodes are the nodes of T and whose sets are the node sets of the current components. As the algorithm proceeds, each designated node of a set stores the node of K corresponding to the set. Root T at an

arbitrary vertex; let $p(v)$ denote the parent of v in the rooted tree. Initialize a DSU structure with each node in a singleton set, storing itself (a leaf of K). Process the edges (now arcs) in the given order. To process an arc $(p(v), v)$, let $u = \mathit{find}(p(v))$. Add a new node x to K, whose two children are the nodes stored at u and v. Store x at u, and perform $\mathit{unite}(u, v)$. (For example, in Figure 10.1, the node corresponding to (f, j) is stored at b.)

This algorithm runs in $O(n\alpha(n, n))$ time on a pointer machine; only the finds take nonlinear time. Although it builds K bottom-up, it does not process T bottom-up but in the given arc order. As in sections 7–9, we thus cannot directly apply the method of section 6 to reduce the running time to linear. On the other hand, if we generalize the DSU structure to allow *unite* operations to have arbitrary nodes, rather than just designated nodes, as parameters, and we replace each $\mathit{unite}(u, v)$ operation in the algorithm by $\mathit{unite}(p(v), v)$, then the (unordered) set of unions is known in advance, because the unions correspond to the arcs of T. As Thorup [62] observed in the context of solving an equivalent problem (see section 10.4), this means that the algorithm runs in linear time on a RAM if the linear-time DSU algorithm of Gabow and Tarjan [29] is used.

Not only are the unions not bottom-up on T, but also there is no obvious way to transform the problem into one on a balanced tree as in section 7. Instead, we partition all of T into microtrees and do a topological graph computation to precompute the answers to finds within the microtrees. Once these answers are known, running the algorithm to build K takes $O(n)$ time. Number the arcs of T from 1 through $n - 1$ in the given order. For any nonroot vertex v, let $\mathit{num}(v)$ be the number of $(p(v), v)$; let $\mathit{num}(v) = \infty$ if v is the root. For any nonroot vertex v, let $f(v)$ be the node returned by $\mathit{find}(p(v))$ in the algorithm that builds K. (For example, in Figure 10.1, $f(j) = b$.) Then $f(v)$ is the nearest ancestor u of v that has $\mathit{num}(u) > \mathit{num}(v)$. We will precompute $f(v)$ if v and $f(v)$ are in the same microtree.

10.3. Linear-time construction. Let $g = n/\log^{1/3} n$. Partition all of T into microtrees, each of size at most g, using the method of Dixon, Rauch, and Tarjan [22], slightly modified. Visit the nodes of T in a bottom-up order, computing, for each node v, a size $s(v)$ and possibly marking v as a subtree root. The value of $s(v)$ is the number of descendants w of v such that no node on the path from v to w is marked. When visiting v, set $s(v) \leftarrow 1 + \sum\{s(w) : w \text{ is a child of } v\}$. If $s(v) > g$, mark every child of v and set $s(v)$ to 1. Every marked node v determines a microtree whose nodes are the descendants w of v such that v is the only marked node on the path from v to w. The construction guarantees that every microtree contains at most g nodes. It also guarantees that there are at most n/g parents of marked nodes, since, for each such parent, the set of microtrees rooted at its children contains at least g nodes. Partitioning T into microtrees takes $O(n)$ time.

To precompute the answers to finds in the microtrees, begin by initializing $f(v) \leftarrow \mathit{null}$ for every nonroot node v. Then use a pointer-based radix sort to renumber the nodes in each microtree consecutively from 1 up to at most g in an order consistent with their original numbers (given by *num*). This does not affect the answers to the finds for any vertex whose answer is in the same microtree. To do the pointer-based radix sort, build a master list of nodes representing the numbers 1 through n, and use pointers to these nodes in lieu of the actual numbers. For each microtree, build a similar master list of nodes representing the numbers 1 through the number of nodes in the microtree, and use pointers to these nodes in lieu of numbers. Now the problem of answering the finds within microtrees is actually a topological graph computation

as defined in section 5, and with $g = n/\log^{1/3} n$ it can be done in $O(n)$ time by Theorem 5.2. This computation gives a nonnull value $f(v)$ for every vertex v such that v and $f(v)$ are in the same microtree.

Having precomputed the answers to some of the finds, we run the algorithm that builds K, but using the precomputed answers. Specifically, to process an arc $(p(v), v)$, let $u = f(v)$ if $f(v) \neq null$, $u = find(p(v))$ otherwise. Then proceed as in section 10.2.

THEOREM 10.1. *Suppose that the edges of a weighted tree T are given in order by weight. Then the Kruskal tree of T can be built in $O(n)$ time on a pointer machine.*

Proof. The algorithm runs on a pointer machine; the running time is $O(n)$ except for the time to do the finds. We bound the time for the finds by applying Lemma 4.6 to the tree built by the parent assignments done by the unite operations. Mark every parent of a microtree root. This marks at most n/g nodes. If an operation $find(p(v))$ is actually done, because its answer is not precomputed, $f(v)$ and v are in different microtrees. The union operations are such that if x and y are in the same set and x is an ancestor of y, every vertex on the tree path from x to y is also in the same set. Thus when $find(p(v))$ is done, $f(v)$, $p(v)$, and $p(root(micro(v)))$ are all in the same set. Since $p(root(micro(v)))$ is marked, this find occurs in a set with a marked node. We conclude that Lemma 4.6 applies with $k = 1$, giving an $O(n)$ time bound for the finds that are not precomputed. □

We do not know whether there is a way to build K in linear time using only bottom-level microtrees. If there is, it is likely to be considerably more complicated than the algorithm we have proposed.

10.4. Compressed Kruskal trees. We can generalize the Kruskal tree to allow equal-weight edges: when adding edges, we add all edges of the same weight at the same time and add a node to the Kruskal tree for every new component so formed, whose children are the components connected together to form it. The resulting component tree is not necessarily binary. Thorup [62] and Pettie and Ramachandran [48] have used such a compressed Kruskal tree in shortest path algorithms. Given a tree and a partition of its edges into equal-weight groups, ordered by weight, we can construct the generalized Kruskal tree in linear time on a pointer machine as follows: Break ties in weight arbitrarily. Build the Kruskal tree, labeling each component node with the group of the edge that formed it. Contract into a single node each connected set of nodes labeled with the same group. The last step is easy to do in $O(n)$ time.

11. Concluding remarks. We have presented linear-time pointer-machine algorithms for six tree and graph problems, all of which have in common the need to evaluate a function defined on paths in a tree. Linear time is optimal and matches the previous bound for RAM algorithms for these problems; our algorithms improve previous pointer-machine algorithms by an inverse-Ackermann-function factor. Our improvements rely mainly on three new ideas: refined analysis of path compression when the compressions favor certain nodes; radix sorting to group isomorphic small subproblems; and careful partitioning of the tree corresponding to the original problem into a collection of microtrees and maximal paths, as appropriate to the particular application.

Our algorithms are simpler than the previous linear-time algorithms. Indeed, our approach provides the first linear-time dominators algorithm that could feasibly be implemented at all: the linear-time algorithm of Alstrup et al. [10] requires Q-heaps [26], implying an impossibly-large constant factor. Buchsbaum et al. implemented their original dominators algorithm [16], of which our algorithm is an improvement,

and presented experimental results demonstrating low constant factors, though the simpler LT algorithm was faster. Georgiadis, Tarjan, and Werneck [34] report more recent experiments with algorithms for finding dominators, with results that vary depending on input size and complexity.

Our methods are sufficiently simple and general that we expect them to have additional applications, which remain to be discovered.

Note Added in Proof. We recently discovered how to avoid the need for NCAs in Tarjan's algorithm for interval analysis described in section 8. The resulting simplified algorithm builds the sets $R(u)$ and finds heads during a single DFS of the flowgraph. It maintains the same DSU structure as in section 8. Initially $R(u)$ is empty for every vertex u. The algorithm is as follows. When the DFS retreats along an arc (x, y), add x to $R(\mathit{find}(y))$. When the DFS visits a vertex u in postorder, while $R(u)$ is nonempty delete a vertex x from $R(u)$, let $v \leftarrow \mathit{find}(x)$, and if $v \neq u$, set $h(v) \leftarrow u$, set $R(u) \leftarrow R(u) \cup R(v)$, and do $\mathit{unite}(u, v)$. This is the computation done by the original algorithm after it does insertions into R-sets for arcs (x, y) such that $\mathit{nca}(x, y) = u$. If the original algorithm processes the vertices in postorder with respect to the DFS, then the R-sets of the descendants of a vertex u just after the insertions for arcs (x, y) such that $\mathit{nca}(x, y) = u$ are exactly the same as they are in the simplified algorithm just before it processes u; the simplified algorithm has done these insertions earlier. It follows that the simplified algorithm is correct.

This idea extends to the linear-time algorithm for interval analysis in section 8, which can be simplified further by making the maximal path partition of the core rightmost instead of arbitrary. Then the computation of heads for vertices whose heads are in the core can be done in a single DFS that also generates the path partition. Here are the details of this computation. Do set unions to combine the vertices in each strong component of the fringe into a single set. Redo the DFS that generated the tree D used to define the fringe and the core, but do the following. Initialize the current path P, the stack S, and all sets $R(u)$ to be empty. When retreating along an arc (x, y), if $\mathit{find}(y)$ is not on P add x to $R(\mathit{find}(y))$; if (x, y) is a tree arc and y is in the core add y to P and push y onto S. (If $\mathit{find}(y)$ is on P when retreating along (x, y), (x, y) is a forward arc; as noted in section 8, heads do not depend on such arcs.) When visiting a core vertex u in postorder, while $R(u)$ is nonempty do the following. Delete a vertex x from $R(u)$. Let $v \leftarrow \mathit{find}(x)$. If v is not on P, set $h(v) \leftarrow u$, set $R(u) \leftarrow R(u) \cup R(v)$, and do $\mathit{unite}(u, v)$. If v is on P and v is no less than the top vertex on S, pop from S each vertex w less than or equal to v, set $h(w) \leftarrow u$, and set $R(u) \leftarrow R(u) \cup R(w)$. When advancing along a tree arc (x, y), do $\mathit{unite}(h(v), v)$ for each vertex on P but not on S, and then empty P and S. It is straightforward to show that this algorithm is correct. An analysis like that in section 8 shows that the algorithm runs in linear time.

Acknowledgment. We thank Stephen Alstrup and Amos Fiat for some pointers to previous work.

REFERENCES

[1] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation, and Compiling, Vol.* II: *Compiling*, Prentice–Hall, Englewood Cliffs, NJ, 1972.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *On finding lowest common ancestors in trees*, SIAM J. Comput., 5 (1976), pp. 115–132.

[4] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison–Wesley, Reading, MA, 1986.
[5] S. Allesina and A. Bodini, *Who dominates whom in the ecosystem? Energy flow bottlenecks and cascading extinctions*, J. Theoret. Biol., 230 (2004), pp. 351–358.
[6] S. Allesina, A. Bodini, and C. Bondavalli, *Secondary extinctions in ecological networks: Bottlenecks unveiled*, Ecological Modelling, 194 (2006), pp. 150–161.
[7] S. Alstrup and M. Thorup, *Optimal algorithms for finding nearest common ancestors in dynamic trees*, J. Algorithms, 35 (2000), pp. 169–188.
[8] S. Alstrup, J. P. Secher, and M. Spork, *Optimal on-line decremental connectivity in trees*, Inform. Process. Lett., 64 (1997), pp. 161–164.
[9] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, *Nearest common ancestors: A survey and a new distributed algorithm*, in Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architecture, 2002, pp. 258–264.
[10] S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup, *Dominators in linear time*, SIAM J. Comput., 28 (1999), pp. 2117–2132.
[11] M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana, *Fault equivalence identification using redundancy information and static and dynamic extraction*, in Proceedings of the 19th IEEE VLSI Test Symposium, 2001, pp. 124–130.
[12] M. A. Bender and M. Farach-Colton, *The LCA problem revisited*, in Proceedings of the 4th Latin American Symposium on Theoretical Informatics, Lecture Notes in Comput. Sci. 1776, Springer-Verlag, New York, 2000, pp. 88–94.
[13] O. Borůvka, *O jistém problému minimálním*, Práce Moravské Přírodovědecké Společnosti v Brně (Acta Societ. Science. Natur. Moravicae), 3 (1926), pp. 37–58.
[14] A. L. Buchsbaum, R. Sundar, and R. E. Tarjan, *Data-structural bootstrapping, linear path compression, and catenable heap-ordered double-ended queues*, SIAM J. Comput., 24 (1995), pp. 1190–1206.
[15] A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook, *Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators*, in Proceedings of the 30th ACM Symposium on Theory of Computing, 1998, pp. 279–288.
[16] A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook, *A new, simpler linear-time dominators algorithm*, ACM Trans. Programming Languages and Systems, 20 (1998), pp. 1265–1296; *Corrigendum*, 27 (2005), pp. 383–387.
[17] J. Cai and R. Paige, *Using multiset discrimination to solve language processing problems without hashing*, Theoret. Comput. Sci., 145 (1995), pp. 189–228.
[18] B. Chazelle, *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), pp. 1028–1047.
[19] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, *Efficiently computing static single assignment form and the control dependence graph*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 451–490.
[20] M. B. Dillencourt, H. Samet, and M. Tamminen, *A general approach to connected-component labeling for arbitrary image representations*, J. ACM, 39 (1992), pp. 253–280.
[21] B. Dixon, M. Rauch, and R. E. Tarjan, *Verification and Sensitivity Analysis of Minimum Spanning Trees in Linear Time*, Tech. report CS-TR-289-90, Department of Computer Science, Princeton University, Princeton, NJ, 1990.
[22] B. Dixon, M. Rauch, and R. E. Tarjan, *Verification and sensitivity analysis of minimum spanning trees in linear time*, SIAM J. Comput., 21 (1992), pp. 1184–1192.
[23] S. Even and Y. Shiloach, *An on-line edge deletion problem*, J. ACM, 28 (1981), pp. 1–4.
[24] J. Ferrante, K. Ottenstein, and J. Warren, *The program dependency graph and its uses in optimization*, ACM Trans. Programming Languages and Systems, 9 (1987), pp. 319–349.
[25] C. Fiorio and J. Gustedt, *Two linear time union-find strategies for image processing*, Theoret. Comput. Sci., 154 (1996), pp. 165–181.
[26] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.
[27] H. N. Gabow, *Data structures for weighted matching and nearest common ancestors with linking*, in Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1990, pp. 434–443.
[28] H. N. Gabow, *Path-based depth-first search for strong and biconnected components*, Inform. Process. Lett., 74 (2000), pp. 107–114.
[29] H. N. Gabow and R. E. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci., 30 (1985), pp. 209–221.
[30] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, *Scaling and related techniques for geometry problems*, in Proceedings of the 16th ACM Symposium on Theory of Computing, 1984, pp. 135–143.

[31] L. Georgiadis, *Linear-Time Algorithms for Dominators and Related Problems*, Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, 2005.
[32] L. Georgiadis and R. E. Tarjan, *Finding dominators revisited*, in Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2004, pp. 862–871.
[33] L. Georgiadis and R. E. Tarjan, *Dominator tree verification and vertex-disjoint paths*, in Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 433–442.
[34] L. Georgiadis, R. E. Tarjan, and R. F. Werneck, *Finding dominators in practice*, J. Graph Algorithms Appl., 10 (2006), pp. 69–94.
[35] R. L. Graham and P. Hell, *On the history of the minimum spanning tree problem*, Ann. History Comput., 7 (1985), pp. 43–57.
[36] J. Gustedt, *Efficient union-find for planar graphs and other sparse graph classes*, Theoret. Comput. Sci., 203 (1998), pp. 123–141.
[37] D. Harel, *A linear time algorithm for finding dominators in flow graphs and related problems*, in Proceedings of the 17th ACM Symposium on Theory of Computing, 1985, pp. 185–194.
[38] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
[39] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees*, J. ACM, 42 (1995), pp. 321–328.
[40] V. King, *A simpler minimum spanning tree verification algorithm*, Algorithmica, 18 (1997), pp. 263–270.
[41] J. Komlós, *Linear verification for spanning trees*, Combinatorica, 5 (1985), pp. 57–65.
[42] J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 7 (1956), pp. 53–57.
[43] T. Lengauer and R. E. Tarjan, *A fast algorithm for finding dominators in a flowgraph*, ACM Trans. Programming Languages and Systems, 1 (1979), pp. 121–141.
[44] M. Loebl and J. Nešetřil, *Linearity and unprovability of set union problem strategies* I. *Linearity of strong postorder*, J. Algorithms, 23 (1997), pp. 207–220.
[45] E. S. Lorry and V. W. Medlock, *Object code optimization*, Comm. ACM, 12 (1969), pp. 13–22.
[46] J. M. Lucas, *Postorder disjoint set union is linear*, SIAM J. Comput., 19 (1990), pp. 868–882.
[47] M. Pătraşcu and E. D. Demaine, *Logarithmic lower bounds in the cell-probe model*, SIAM J. Comput., 35 (2006), pp. 932–963.
[48] S. Pettie and V. Ramachandran, *Computing shortest paths with comparisons and additions*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 267–276.
[49] S. Pettie and V. Ramachandran, *An optimal minimum spanning tree algorithm*, J. ACM, 49 (2002), pp. 16–34.
[50] P. W. Purdom and E. F. Moore, *Algorithm* 430*: Immediate predominators in a directed graph*, Comm. ACM, 15 (1972), pp. 777–778.
[51] L. Quesada, P. Van Roy, Y. Deville, and R. Collet, *Using dominators for solving constrained path problems*, in Proceedings of the 8th International Symposium on Practical Aspects of Declarative Languages, Lecture Notes in Comput. Sci. 3819, Springer-Verlag, New York, 2006, pp. 73–87.
[52] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.
[53] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362–391.
[54] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
[55] R. E. Tarjan, *Finding dominators in directed graphs*, SIAM J. Comput., 3 (1974), pp. 62–89.
[56] R. E. Tarjan, *Efficiency of a good but not linear set union algorithm*, J. ACM, 22 (1975), pp. 215–225.
[57] R. E. Tarjan, *Edge-disjoint spanning trees and depth-first search*, Acta Inform., 6 (1976), pp. 171–185.
[58] R. E. Tarjan, *Applications of path compression on balanced trees*, J. ACM, 26 (1979), pp. 690–715.
[59] R. E. Tarjan, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110–127.
[60] R. E. Tarjan, *Testing flow graph reducibility*, J. Comput. System Sci., 9 (1994), pp. 355–365.
[61] R. E. Tarjan and J. van Leeuwen, *Worst-case analysis of set union algorithms*, J. ACM, 31 (1984), pp. 245–281.

[62] M. Thorup, *Undirected single-source shortest paths with positive integer weights in linear time*, J. ACM, 46 (1999), pp. 362–394.
[63] A. K. Tsakalides and J. van Leeuwen, *An Optimal Pointer Machine Algorithm for Finding Nearest Common Ancestors*, Tech. report RUU-CS-88-17, Department of Computer Science, University of Utrecht, Utrecht, Germany, 1988.
[64] J. van Leeuwen, *Finding Lowest Common Ancestors in Less Than Logarithmic Time*, unpublished report, 1976.
[65] J. Vuillemin, *A unifying look at data structures*, Comm. ACM, 23 (1980), pp. 229–239.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1574–1601

THE COMBINED POWER OF CONDITIONS AND INFORMATION ON FAILURES TO SOLVE ASYNCHRONOUS SET AGREEMENT*

ACHOUR MOSTEFAOUI†, SERGIO RAJSBAUM‡, MICHEL RAYNAL†, AND CORENTIN TRAVERS†

Abstract. To cope with the impossibility of solving agreement problems in asynchronous systems made up of n processes and prone to t process crashes, system designers tailor their algorithms to run fast in "normal" circumstances. Two orthogonal notions of "normality" have been studied in the past through *failure detectors* that give processes information about process crashes, and through *conditions* that restrict the inputs to an agreement problem. This paper investigates how the two approaches can benefit from each other to solve the *k-set agreement* problem, where processes must agree on at most k of their input values (when $k = 1$ we have the famous consensus problem). It proposes novel failure detectors for solving k-set agreement and a protocol that combines them with conditions, establishing a new bridge among asynchronous, synchronous, and partially synchronous systems with respect to agreement problems. The paper also proves a lower bound when solving the k-set agreement problem with a condition.

Key words. asynchronous system, condition, consensus, failure detection, input vector, legal condition, set agreement, process crash, shared memory, snapshot

AMS subject classifications. 68Q10, 68Q25, 68Q85

DOI. 10.1137/050645580

1. Introduction.

Distributed services have to run efficiently and reliably in complex environments with unpredictable processing and communication delays, where components can fail in various ways. It is unavoidable to encounter scenarios where system performance will degrade, or even manual intervention will be required. Therefore, system designers tailor their applications to run fast in "normal" circumstances while having expensive recovery procedures in the rare cases of "abnormal" circumstances. Two complementary notions of "normality" have been considered, mirroring the traditional computer science duality of *control* and *data*. On the control side we have the *failure detector* approach [8], which abstracts away useful failure pattern information, available in common operating scenarios. On the data side, we have the *condition-based* approach [36], which looks at common input data patterns of a certain distributed problem we are interested in. The aim of this paper is to study how the two approaches interact and can benefit from each other, with respect to solving *agreement problems*.

1.1. Context of the paper.

Distributed services often rely on an underlying agreement protocol. The most popular and fundamental of the agreement problems is consensus, which is actually indispensable for a lot of services. This paper investigates the possibilities and limitations of solving consensus, and other weaker agreement problems, in a system with failure detectors and conditions.

*Received by the editors November 21, 2005; accepted for publication (in revised form) July 7, 2008; published electronically November 21, 2008. An extended abstract of this paper appeared in the proceedings of PODC 2005. This work was supported by grants from LAFMI (Franco-Mexican Lab in Computer Science), DGAPA-UNAM, and the European Network of Excellence *ReSIST*.
http://www.siam.org/journals/sicomp/38-4/64558.html

†IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France (achour@irisa.fr, raynal@irisa.fr, ctravers@irisa.fr).

‡Instituto de Matemáticas, Universidad Nacional Autónoma de México, D. F. 04510, Mexico (rajsbaum@math.unam.mx).

Agreement problems. The *consensus* problem can informally be stated as follows. Each process proposes a value, and the processes that are not faulty have to decide the same value such that the value decided is one of the proposed values. As a familiar example, it is easy to see that the *atomic broadcast* problem relies on consensus as it requires that all of the processes deliver the messages they broadcast in the same order: They have, consequently, to agree in one way or another on the same message delivery order. However, it is well known that the consensus problem has no solution in message-passing asynchronous systems made up of n processes that need to tolerate a single process crash failure [16].

The *k-set agreement* problem [9] relaxes the consensus requirement to allow up to k different values, out of the proposed values, to be decided; when $k = 1$, we have the consensus problem. Set agreement is an abstraction of problems that are weaker than consensus. Its discovery was motivated by the search for a problem that is solvable when $k-1$ processes can crash, but not when k can crash, in an asynchronous system. Since then, it has been very valuable in the development of the foundations of distributed computing. The proofs in [6, 29, 50] showing that k-set agreement is not solvable in a system of $k+1$ processes where k can crash uncovered a deep connection between distributed computing and topology and motivated a significant amount of subsequent research.

The situation is totally different in synchronous systems, where both consensus and k-set agreement can be solved for any value of t (the maximum number of process crashes) [10, 49]. However, there are limitations on how fast these problems can be solved in a synchronous system, as a function of the number of failures t to be tolerated. It has been shown that consensus requires $t + 1$ rounds in the worst case [15, 33], and there are protocols that meet this lower bound. These results have been generalized for the set agreement problem in [10, 20, 48].

The following two complementary notions of "normality" have been considered[1] to cope with the consensus and set agreement asynchronous impossibility results and synchronous lower bounds.

Control: Enriching the underlying system. The first approach focuses on the behavior of the underlying system. In this case "normal circumstance" means a period during which the system behaves in a relatively synchronous way. Namely, periods during which upper bounds on process execution speeds and on message transmission delays hold (various such *partially synchronous* models have been considered, e.g., [14]), or periods during which message exchange patterns satisfy some properties (e.g., the notion of winning/losing responses introduced in [35]) that allow solving consensus. A *failure detector* abstracts away such low-level assumptions by providing processes with a primitive they can invoke that returns information on process failures. One of the noteworthy features of failure detectors is the modular approach they favor: One can independently, on one side, solve a problem with the help of a particular class of failure detector, and, on another side, implement the assumed failure detector with the help of the underlying timing or order assumptions. The design, transportability, and proof of protocols then become modular and easier to achieve.

Chandra and Toueg introduced the *failure detector* notion [8] and defined eight classes that can be used to solve asynchronous consensus. Together with Hadzilacos, they later showed that one of these classes is the weakest class of failure detectors to solve consensus when $t < n/2$ [7] (n being the total number of processes, and t an upper bound on the number of faulty processes). The weakest failure detector

[1] There are other approaches, like randomization or stronger shared memory primitives.

for consensus and any value of t was identified in [12]. These results are of great theoretical interest because they identify the minimum knowledge about failures that needs to be abstracted to solve consensus. Failure detectors to solve set agreement have also been proposed [27, 44, 52], but we do not know yet what is the minimum knowledge about failures that needs to be abstracted to solve this problem.

The failure detector approach has favored the design of indulgent protocols [21]. A protocol is *indulgent* with respect to its failure detector if it never violates its safety property. This means that, when the underlying failure detector satisfies its specification (normal circumstances) the protocol terminates correctly; and, when the underlying failure detector does not satisfy its specification (abnormal circumstances), it is possible that the protocol does not terminate, but if it terminates, it does so correctly. Various indulgent failure detector-based consensus protocols have been proposed (e.g., [8, 23, 31, 43, 46, 51]).

Data: Restricting the inputs. The *condition-based* approach [36] consists in looking at certain combinations of input values of a given distributed problem. It is often the case in practice that some combinations of the input values of processes occur more frequently than others. For example, in an election, it is often the case that the difference in the number of votes that candidates receive is significant. More precisely, an *input vector* contains the values proposed by the processes in an execution. A *condition* C is a set of input vectors, each representing a common combination of inputs to the problem. If a protocol solves k-set agreement for C, then whenever the input vector belongs to C, all of the correct processes decide. The solution should be indulgent in the sense that if correct processes decide while the input vector does not belong to the condition, they do not decide more than k values.

It was discovered in [36] that there is a family of conditions, called x*-legal*, that tie together asynchronous and synchronous systems with respect to consensus solvability. Informally, in an x-legal condition any two input vectors $I1, I2$ that force different decisions have $d(I1, I2) > x$ (Hamming distance), assuming $n > x$. Thus, in a sense, x is the "power" of the condition; larger values of x make it easier to solve consensus. Assuming up to t process crashes and $d \leq t$ (d can have a negative value), let $\mathcal{S}_t^{[d]}$ be the set of all x-legal conditions, $x = t-d$ (e.g., $\mathcal{S}_t^{[0]}$ consists of the t-legal conditions). Then

$$\mathcal{S}_t^{[-t]} \subset \cdots \subset \mathcal{S}_t^{[-1]} \subset \mathcal{S}_t^{[0]} \subset \mathcal{S}_t^{[1]} \subset \cdots \subset \mathcal{S}_t^{[t]},$$

where $\mathcal{S}_t^{[t]}$ includes the condition made up of all of the possible input vectors. For a condition $C \in \mathcal{S}_t^{[d]}$, $-t \leq d \leq t$, and a system prone to t process crashes, we have the following:

- For values of $d \leq 0$, for inputs in C, consensus is solvable by more and more efficient protocols in a shared memory asynchronous system as we go from $d = 0$ to $d = -t$ [40].
- For values of $d > 0$, consensus is not solvable in an asynchronous system, but, for inputs in C, it is solvable in a message-passing synchronous system with more and more rounds, as we go from $d = 1$ (two rounds) to $d = t$ ($t + 1$ rounds), and this is tight [37] (namely, when $C \in \mathcal{S}_t^{[d]}$ and $C \notin \mathcal{S}_t^{[d-1]}$, $(d+1)$ rounds are sufficient and necessary in worst case scenarios).
- $d = 0$ is the borderline case. On one hand, asynchronous consensus can be solved (despite up to t faulty processes) for a condition C if and only if C is t-legal [36]. On the other hand, consensus can be solved optimally (2 rounds) in a message-passing synchronous system [37] for any t-legal condition.

The condition-based approach has been considered also for set agreement (and for other problems), but a characterization of the conditions that allow solving set agreement was not known (see section 1.3 for more about this and other related works).

1.2. Motivation and results. As we have seen, failure detectors and conditions are two orthogonal approaches to cope with the impossibility of solving agreement problems in asynchronous systems prone to t process crashes. So, the natural question that comes to mind is,

> *What is the relation between the condition-based approach and the failure detection-based approach when solving asynchronous agreement problems?*

More specifically, we are interested in studying how the two approaches can cooperate to solve set agreement problems. We would like to understand which combinations of failure detectors and conditions can be used to solve k-set agreement for a given value of k.

> *For a given condition C, what is a failure detector that abstracts away the synchrony needed to solve k-set agreement?*

These and similar questions are the topic addressed in the paper.

The paper contains three main contributions. While trying to answer the previous questions, we discovered a new class of failure detectors. We present an asynchronous condition-based set agreement protocol based on this kind of failure detector. We present a lower bound showing that our protocol is optimal. The next three sections describe these results in more detail.

1.2.1. A new class of failure detectors. The first contribution of the paper is the definition of a new class of failure detectors that we denote ϕ_t^y, $0 \leq y \leq t$. A failure detector of ϕ_t^y provides a primitive, denoted $\text{QUERY}_y(S)$, that can be invoked by a process with a set of process identities S to be informed whether they have crashed or not. Roughly speaking, $\text{QUERY}_y(S)$ returns *true* only when all of the processes in S have crashed. If at least one process in S is alive, the output should be *false*. If $|S|$ is outside the range $t - y < |S| \leq t$, the query returns no useful information.

Notice that the nature of our failure detectors is different from the standard failure detectors of [8], that return a set of processes suspected to have crashed, and accept no input parameter. The motivation is that often a process p_i is interested in the failures of only a specific part of the network, namely S, while the standard failure detectors must find out the failure status of all processes in the network, even if p_i cares only about the state of a single process p_j.

For each value of y between 0 and t, there is a class of failure detectors, ϕ_t^y. The class ϕ_t^y provides more information on failures than the class ϕ_t^{y-1}. So, the class ϕ_t^t is the most powerful, while ϕ_t^0 is the weakest (it actually provides no dependable information on failures). Indeed, as shown in the paper, ϕ_t^y can be used to solve k-set agreement for smaller values of k than ϕ_t^{y-1}.

The paper also compares the power of the ϕ_t^y failure detectors and the power of the classic failure detector classes introduced by Chandra and Toueg [8]. It is shown that it is possible to build any class ϕ_t^y, $0 \leq y \leq t$, from a perfect failure detector as defined in [8]. (A perfect failure detector eventually detects all crashed processes and never suspects erroneously a noncrashed process.) In contrast, none of the other classes of classic failure detectors can be used to build a failure detector of a class ϕ_t^y, $1 \leq y \leq t$. When we consider the construction in the other direction, we show that no class ϕ_t^y, $0 \leq y < t$, can be used to build any of the classic failure detector classes. When $y = t$, ϕ_t^t can be used to build a failure detector of the class $\mathcal{P}$.

1.2.2. A condition-based set agreement algorithm using failure detectors. A second contribution of the paper is the design of a condition-based set agreement protocol with access to a failure detector of the class ϕ_t^y, $0 \leq y \leq t$. The considered model is the classical asynchronous read/write shared memory distributed system prone to at most t process crashes. The protocol can be instantiated with any condition $C \in \mathcal{S}_t^{[d]}$, $0 \leq d \leq t$. As we have seen, $x = t - d$ represents the power of the condition. That is, once n and t are fixed, the protocol is parameterized by the power of the failure detector (captured by y) and the power of the condition (captured by $x = t - d$).

We use the following terminology. We say that "a protocol solves the k-set agreement problem" if the correct processes always decide; we say that "*a protocol solves the condition-based k-set agreement* problem" if the correct processes decide at least "in normal circumstances," where "*normal circumstances*" means when (1) the input vector belongs to the condition C; or when (2) a process decides or less than k processes crash; or when (3) at least $t - d$ processes crash initially.

The proposed protocol solves the condition-based k-set agreement for $k = 1 + \max(0, d - y)$. Making more explicit the power y of the failure detector and the power $x = t - d$ of the condition, we have $k = 1 + \max(0, t - (x + y))$. This shows how, by adding the power of the condition and the power of the failure detector, we can counterbalance the power t of the "adversary," in order to reduce the value of k. When we consider the boundary values of y and d, the protocol solves the following problems:

- $d = t$ corresponds to the case where there is no additional power provided by the condition, as then condition C may contain all possible input vectors. But, as any input vector belongs to this trivial condition, all correct processes always decide, and, consequently, the protocol solves the k-set agreement problem. More precisely,
  - If $y = t$ (strongest failure detector), the protocol solves the consensus problem, $k = 1$.
  - If $y = 0$ (no failure detector), the protocol solves the trivial $(t + 1)$-set agreement problem.
  - If $0 < y < t$, the protocol solves k-set agreement, with $k = t+1-y$. When we compare to the previous case, this shows the benefit provided by a failure detector of the class ϕ_t^y. The number of decided values linearly decreases according to the power of the failure detector, as measured by y.
- $d = 0$ means that the condition C is t-legal, which means that condition-based consensus can be solved despite asynchrony and up to t crashes, with no failure detector. So, at most one value is decided, and all of the correct processes terminate in normal circumstances. So, the protocol then solves condition-based consensus. (Let us notice that this is independent of the value of y.)
- if $y = 0$ (no failure detector), the protocol then relies only on the condition and solves the condition-based k-set agreement problem for $k = d + 1$. No more than $(d + 1)$ values are decided, and the termination of the correct processes is guaranteed at least in normal circumstances: The number of decided values decreases linearly according to the parameter d defining the condition.

  This case is particularly interesting as it exhibits a new link relating synchronous and asynchronous systems. More precisely, when the condition C belongs to $\mathcal{S}_t^{[d]}$ and the input vector belongs to C, (1) it is possible to solve

consensus in at most $(d+1)$ rounds in a synchronous system [37]; and (2) it is possible to solve $(d+1)$-set agreement in an asynchronous system, both systems being prone to t crashes. The optimal time bound for synchronous condition-based consensus is equal to the number of decided values in asynchronous condition-based set agreement. This time (in synchronous systems) versus the number of decided values (in asynchronous systems) relation sheds a new light on the global picture concerning the relations between synchronous and asynchronous systems.

When we look at the general case, where the condition-based k-set agreement problem is solved with $k = 1 + \max(0, d-y)$, we see that when $y \geq d$, condition-based consensus is solved. This means that, if d is fixed, we need only to take a failure detector of the class ϕ_t^d to solve condition-based consensus (failure detectors of any class ϕ_t^y, with $y > d$ are stronger than necessary). A similar reasoning can be done when y is fixed, and we have the choice of the condition class.

The proposed protocol is indulgent [21, 23]: It never violates its safety requirement (no more than $k = 1 + \max(0, d-y)$ values are decided), and the correct processes always terminate when the input vector belongs to the condition ("normal circumstances"). Interestingly, a simple modification provides a protocol version in which all of the correct processes always terminate. This is obtained at the price of an increase in the number of values that can be decided when the input vector does not belong to the $(t-d)$-legal condition C, namely, up to $k' = t+1-y$ different values can then be decided. When the system is equipped with a failure detector of the class ϕ_t^t, this protocol variant solves the consensus problem whatever the condition it is instantiated with.

1.2.3. A lower bound. A third contribution of the paper is a lower bound result showing that no protocol with access to a failure detector of the class ϕ_t^y can solve k-set condition-based agreement for $k \leq \max(0, d-y)$, if the condition is in $\mathcal{S}_t^{[d]}$. The proof is by reduction to the standard t-resilient k-set agreement problem, that is known to be impossible if $t \geq k$ [6, 28, 29, 50]. This lower bound result has two nice corollaries. One states that (in the absence of a failure detector) there is no condition-based k-set agreement protocol such that $k \leq d$ for any $(t-d)$-legal condition (a previously open problem). The second one states that, among all of the failure detector classes of the family $(\phi_t^y)_{0 \leq y \leq t}$, the class ϕ_t^y is the weakest that allows solving the k-set agreement problem for $k > t - y$.

1.3. Related work.

The condition-based approach for consensus and set agreement. The condition-based approach has been applied to problems other than consensus like interactive consistency [17] and, more related to our work, set agreement [3, 39]. The paper [3] characterizes the set of input vectors that allow us to solve $(n-1)$-set agreement, wait-free, namely, when $t = n-1$. Their notion of solvability is different from ours, since they assume that a protocol never receives input vectors outside of the condition. In [39], another family of conditions for set agreement is defined, but no general lower bounds were proved. Randomization as a means of circumventing the set agreement asynchronous impossibility result has been considered in [45].

Failure detectors. Most of the research about failure detectors has been directed at solving consensus, but there have also been proposals of failure detectors for solving other problems. Failure detectors for implementing various objects and for solving nonblocking atomic commit have been studied (e.g., [12, 22, 47]). The

weakest class of failure detectors to solve consensus was identified in [7, 12]. For our work, weaker classes of failure detectors are especially relevant, since set agreement is an easier problem than consensus (and if conditions are considered, it becomes even easier). Weaker classes of failure detectors were considered in [18, 44, 47, 52].

Failure detectors for set agreement. Among the failure detectors which are not strong enough to solve consensus, the *limited scope accuracy* failure detectors [25, 44, 52] have been studied with respect to set agreement. To illustrate this notion, let us consider the class denoted $\mathcal{S}_x$. A failure detector of that class satisfies the following two properties. The completeness property states that the processes that crash are eventually suspected in a permanent way. The limited scope accuracy property states that there is a correct process that is not suspected by a set—cluster—of x processes (some of these x processes may be correct, while others may be faulty). An $\mathcal{S}_x$-based k-set agreement protocol is presented in [44]. This protocol assumes that $t < k + x - 1$ (which means that $(t+1) - (x-1)$ is the smallest value of k that it can tolerate). Using topological methods, it has been shown in [27] that this is actually a lower bound for any $\mathcal{S}_x$-based k-set agreement protocol (from which it follows that the previous protocol is optimal with respect to the number of faulty processes that can be tolerated). When the limited scope accuracy property has to hold only after some unknown but finite time, we get the class denoted $\diamond\mathcal{S}_x$. It is shown in [27] that any $\diamond\mathcal{S}_x$-based k-set agreement protocol requires $t < \min(n/2, k + x - 1)$. A $\diamond\mathcal{S}_x$-based protocol meeting this lower bound is also presented in [27]. It is shown in [2] that $t < x$ is a necessary and sufficient requirement to transform any failure detector of the class $\diamond\mathcal{S}_x$ into a failure detector of the class $\diamond\mathcal{S}_y$ for $y > x$.

The class of anonymously perfect failure detectors. A failure detector of our class ϕ_t^y returns a binary output and can be invoked with a parameter S that contains a set of process identities. In contrast, the classic failure detectors of [8] return a set of identities, and are invoked with no parameter. A failure detector class whose output is binary has been introduced by Guerraoui to solve the nonblocking atomic commit problem [22], but, differently from ours, a failure detector of this class does not accept a parameter to invoke it. This class, called *anonymously perfect* failure detectors and denoted $?\mathcal{P}$, is defined as follows. Each process has a flag (initially equal to *false*) that is eventually set to *true* if and only if a process has crashed (the identity of the crashed process is not necessarily known, hence the name "anonymous").

The definition of $?\mathcal{P}$ has been extended in [18] to take into account the fact that ℓ processes have crashed (instead of a single one). This class, denoted $?\mathcal{P}^\ell$, provides each process with a flag that is eventually set to *true* if and only if at least ℓ processes have crashed (observe that $?\mathcal{P}$ is $?\mathcal{P}^1$).

So, a failure detector of the class $?\mathcal{P}^\ell$ answers *true* only if there is a set S of ℓ processes that have crashed. The set S is not known to the processes. Differently, when we consider ϕ_t^y, the set S is user-defined and specific to each invocation.

A variant of Ω. A generalization of the class of leader failure detectors, denoted Ω, has been introduced in [47]. More explicitly, Ω_z is the class of all failure detectors that provide the processes with a primitive LEADER() satisfying the following properties. First, LEADER() always returns a set of at most z process identities. Second, there is a time τ such that, after τ, all of the invocations of LEADER() by the correct processes return the same set of processes, and this set includes at least one correct process. It is easy to see that Ω_1 is Ω, and Ω_n provides no information on failures. That is, in general, Ω_z is weaker than the weakest failure detector for

consensus. However, Neiger introduced them to study questions about augmenting the synchronization power of types in the wait-free hierarchy [26], and their relation to set agreement was not studied.

In a follow-up paper [38], we study the relation of Ω_z to set agreement. Moreover, we study our new failure detectors, with respect to $\diamond\mathcal{S}_x$, Ω_z, and show which reductions among these classes are possible and which are not.

1.4. Organization of the paper. The paper is made up of nine sections. After this introduction and a short section introducing the computation model considered in the paper, section 3 presents the new failure detector classes ϕ_t^y. (Section 8 compares them to classic failure detectors by Chandra and Toueg.) Section 4 provides a quick overview of the most relevant notions (for this paper) of the condition-based approach, including a definition of the condition-based k-set agreement problem. Section 5 presents a generic k-set agreement protocol that is based on the combined power of a failure detector of the class ϕ_t^y and a condition of the class $\mathcal{S}_t^{[d]}$. The protocol is discussed in section 6, where, at the price of an increase of the number of decided values, an always terminating version is presented. Section 7 focuses on the lower bound result. Finally, section 9 summarizes the content of the paper.

2. About the model of computation. This paper considers the usual *asynchronous model* with n processes $p_1, \ldots, p_n$, where at most t can crash $1 \leq t < n$. The processes communicate through a shared memory made up of single-writer, multireaders atomic registers.

We assume that processes have access to an oracle that provides possibly unreliable information on process failures. A *failure detector* provides processes with a primitive they can invoke to get information from the oracle on process failures.

3. The failure detector classes $\{\phi_t^y\}_{0\leq y\leq t}$.

3.1. Definition. This section introduces a new class of failure detectors, parameterized by an integer y, $0 \leq y \leq t$, denoted ϕ_t^y. (A comparison to classic failure detectors is done in section 8.) The power of a such a failure detector depends on the value of y. As we are about to see, a failure detector is more powerful for larger values of y, because it can return information about more specific regions of the network, namely, about smaller sets S of processes, with $|S| > t - y$.

More precisely, a failure detector of the class ϕ_t^y provides a primitive $\text{QUERY}_y(S)$ that returns a boolean answer. A process invokes it with the parameter S, a set of processes specific to each invocation. Intuitively, if p_i invokes $\text{QUERY}_y(S)$, the answer will be *true* only when all processes in S have crashed. In that sense, these failure detectors are different from the standard failure detectors, introduced by Chandra and Toueg [8], that return a set of processes suspected to have crashed, and do not accept an input parameter.[2] The motivation is that often a process p_i is interested in the failures of only a specific sector of the network, namely S, while the Chandra–Toueg failure detectors must find out the failure state of all processes in the network, even if p_i cares only about the state of only one process p_j.

A query $\text{QUERY}_y(S)$ such that $t-y < |S| \leq t$ is *relevant*, otherwise it is *irrelevant*. Intuitively, "relevant" means that it provides dependable information on failures. The class ϕ_t^y is defined by the following properties:

[2]We have shown in [38] that there are transformations between the ϕ failure detectors and a version with no input parameter. It is consequently possible to define them according to the failure pattern only.

- *Triviality property.* If $|S| \leq t - y$, then $\text{QUERY}_y(S)$ returns *true*. If $|S| > t$, then $\text{QUERY}_y(S)$ returns *false*.
- *Safety property.* If $\text{QUERY}_y(S)$ is relevant, then if at least one process in S has not crashed when $\text{QUERY}_y(S)$ is invoked, the invocation returns *false*.
- *Liveness property.* Let $\text{QUERY}_y(S)$ be a relevant query. Let τ be a time such that, at time τ, all of the processes in S have crashed. There is a time $\tau' \geq \tau$ such that all of the invocations of $\text{QUERY}_y(S)$ after τ' return *true*.

The triviality property says that the invoking process gets back a *true* output when the set S is too small, because, in this case, the failure detector is not powerful enough to answer reliably on a region of the network that is too focused. If the set S is too big, the output is *false*, because, by definition, no more than t processes can fail. The safety property states that if the output of a relevant query is *true*, then all of the processes in S have crashed. The liveness property states that $\text{QUERY}_y(S)$ eventually outputs *true* when all of the processes in S have crashed (and the query is relevant).

3.2. Ranking the classes $\{\phi_t^y\}_{0 \leq y \leq t}$. A failure detector of the class ϕ_t^0 provides no information related to failures as the invocation $\text{QUERY}_y(S)$ answers always *true* if $|S| \leq t$, and *false* if $|S| > t$. At the other extreme, with a failure detector of the class ϕ_t^t, a process can query about the failure status of a single specific process, since $\text{QUERY}_y(S)$ may return significant failure information about sets S of any size, $1 \leq |S| \leq t$. That is, ϕ_t^0 and ϕ_t^t are two extreme classes. This section compares the power of distinct classes of failure detectors denoted ϕ_t^{y1} and ϕ_t^{y2}.

DEFINITION 1. *For two classes of failure detectors A and B, we denote $A \leq B$, and say that B is* at least as strong *as A if any failure detector in B can be used to build a failure detector in A. We also say that B is* stronger *than A (denoted $A < B$) if $A \leq B$ and $B \not\leq A$. The classes A and B are* equivalent, *(denoted $A \equiv B$) if $A \leq B$ and $B \leq A$.*

We shall see that ϕ_t^{y1} is stronger than ϕ_t^{y2} if $y1 > y2$, since ϕ_t^{y1} provides more information about failures than ϕ_t^{y2}. Given a run of the processes, let $\text{QUERY}_y(S)$ be a failure detector query invocation that, from some time on, is indefinitely repeated. Let us examine the outputs returned by the infinite sequence of queries when the failure detector belongs to ϕ_t^{y1} and ϕ_t^{y2}, respectively. Notice that $t - y2 > t - y1$ (since $y1 > y2$).

- Case 1: $|S| > t$. Both outputs are systematically equal to *false*.
- Case 2: $|S| \leq t - y1$. Both outputs are systematically equal to *true*.
- Case 3: $t - y2 < |S| \leq t$ (so, we also have $t - y1 < |S| \leq t$) for a relevant query. If at least one process of S never crashes, both outputs are always equal to *false*. If all of the processes of S crash, eventually both outputs are permanently equal to *true*.
- Case 4: $t - y1 < |S| \leq t - y2$. In this case, the output is always *true* if the failure detector belongs to the class Φ_t^{y2}. If it belongs to Φ_t^{y1}, the output is as in Case 3 (it depends on the failures).

The last case, namely, when $t - y1 < |S| \leq t - y2$, exhibits a noteworthy difference between ϕ_t^{y1} and ϕ_t^{y2}: ϕ_t^{y1} provides information on failures while ϕ_t^{y2} does not. Indeed, for $y1 > y2$, it is impossible to build a failure detector in ϕ_t^{y1} from one in ϕ_t^{y2}. On the other hand, any failure detector in ϕ_t^{y1} can be used to build a failure detector in ϕ_t^{y2} by returning *true* if $|S| \leq t - y2$, returning *false* if $|S| > t$, and returning the output of ϕ_t^{y1} if $t - y2 < |S| \leq t$. (Formally, the next theorem is a consequence of Corollary 2 of section 7.)

THEOREM 1. $(y1 > y2) \Rightarrow (\phi_t^{y2} < \phi_t^{y1})$.

4. The condition-based approach. The condition-based approach was introduced in [36] to study conditions restricting the inputs to consensus that make the problem solvable in an asynchronous system where t processes can crash. This line of research has been extended to study conditions for other problems and in other distributed computing models [3, 17, 32, 36, 37, 39, 53]. In this paper, we are interested in conditions for the set agreement problem in an asynchronous system.

4.1. Conditions. Let $\mathcal{V}$ be the set of values that can be proposed by the processes. Moreover, let $\perp \notin \mathcal{V}$ be a default value. An *input vector* is a size n vector over $\mathcal{V} \cup \{\perp\}$. The input vector J *proposed* in an execution has in its ith entry $J[i]$ the value of $\mathcal{V}$ proposed by p_i, or $\perp$ if p_i did not take any step in the execution. We usually denote by I an input vector with all entries in $\mathcal{V}$, and with J an input vector that may have some entries equal to $\perp$; such a vector J is called a *view*. The set $\mathcal{V}_x^n$ consists of all of the input vectors, with at most x entries equal to $\perp$, and $\mathcal{V}^n = \mathcal{V}_0^n$.

DEFINITION 2. *A* condition C *is a subset of* $\mathcal{V}^n$.

Notation. For any pair of vectors $J1, J2 \in \mathcal{V}_x^n$, $J1$ is *contained* in $J2$, denoted $J1 \leq J2$, if for all $k : J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$. Moreover, $J1 < J2$ if $J1 \leq J2$ and $J1 \neq J2$, which means that J_2 has at least one non-$\perp$ value that $J1$ does not have. Also, $\#_a(J)$ denotes the number of occurrences of a value a in the vector J, with $a \in \mathcal{V} \cup \{\perp\}$. For a set of input vectors $C \subseteq \mathcal{V}^n$, $\mathcal{C}_x$ is the set of all vectors J, with at most x entries equal to $\perp$ and such that $J \leq I$ for some $I \in C$. Finally, $dist(J, J')$ is the Hamming distance separating J and J', where J and J' are two vectors of $\mathcal{V}_x^n$.

4.2. Legality of a condition. The main result of the condition-based approach to solve asynchronous consensus is based on the following definition as formulated in [17, 53].

DEFINITION 3. *A condition* C *is* x-legal *if there exists a function* $h : C \mapsto \mathcal{V}$ *with the following properties:*

- *for all* $I \in C$: $h(I) = a \Rightarrow \#_a(I) > x$, *and*
- *for all* $I1, I2 \in C$: $h(I1) \neq h(I2) \Rightarrow dist(I1, I2) > x$.

A fundamental result of the condition-based approach is a characterization of the conditions C for which consensus can be solved (for a precise definition of solving consensus for C, see Definition 5, with $k = 1$).

THEOREM 2 (see [36]). *There is a* t*-fault tolerant protocol solving consensus for* C *if and only if* C *is* t*-legal.*

A general method to define t-legal conditions is described in [40], and two natural t-legal conditions are described in [36].

It is convenient to extend h to vectors J with $\perp$ values. The lemma that follows shows that this is easy, provided $J \in \mathcal{C}_x$.

LEMMA 1. *Let* C *be an* x*-legal condition, and* $I1, I2 \in C$, $J \in \mathcal{C}_x$ *such that* $J \leq I1$ *and* $J \leq I2$. *Then* $h(I1) = h(I2)$.

Proof. Assume for contradiction that $h(I1) \neq h(I2)$. We have $dist(I1, I2) > x$ because C is x-legal. From the fact that J has at most x entries equal to $\perp$ and $J \leq I1$, we have $dist(J, I1) \leq x$ (similarly, we also have $dist(J, I2) \leq x$). From these inequalities, the fact that the entries of J that differ in $I1$ and $I2$ are only its $\perp$ entries, and again the fact that J has at most x entries equal to $\perp$, we conclude that $dist(I1, I2) \leq x$. A contradiction. □

Using this lemma we have a consistent definition.

DEFINITION 4. *Let* C *be an* x*-legal condition and* J *be any vector in* $\mathcal{C}_x$. *The function* h *is extended to* J *by taking any* $I \in C$, *with* $J \leq I$ *and letting* $h(J) = h(I)$.

Assuming up to t process crashes and $-t \leq d \leq t$, let $\mathcal{S}_t^{[d]}$ be the set of all $(t-d)$-legal conditions (thus $\mathcal{S}_t^{[0]}$ consists of the t-legal conditions). It is easy to check that

$$\mathcal{S}_t^{[-t]} \subset \cdots \subset \mathcal{S}_t^{[-1]} \subset \mathcal{S}_t^{[0]} \subset \mathcal{S}_t^{[1]} \subset \cdots \subset \mathcal{S}_t^{[t]},$$

where $\mathcal{S}_t^{[t]}$ includes the condition made up of all of the possible input vectors.

Notation. In the rest of the paper, C_t^d denotes a condition that belongs to $\mathcal{S}_t^{[d]}$.

4.3. The k-set agreement problem.

***k*-set agreement.** Consensus is a fundamental problem in distributed computing that is impossible in an asynchronous system even with a single crash failure. While consensus requires all processes to decide on the same value, *k-set agreement* [9] permits the processes to choose up to k different values. The problem is solvable when $k-1$ processes can crash, but not when k can crash. The proofs [6, 29, 50] of this result uncovered a deep connection between distributed computing and topology and motivated a significant amount of subsequent research.

The set of values $\mathcal{V}$ that can be proposed is assumed to be such that $|\mathcal{V}| > k$. Each process starts an execution with an arbitrary input value from $\mathcal{V}$, the value it *proposes*, and all correct processes have to decide on a value such that (1) any decided value has been proposed, and (2) no more than k different values are decided. The *consensus* problem is k-set agreement for $k = 1$.

Condition-based set agreement. We are interested in conditions C that, when satisfied, make k-set agreement solvable in an asynchronous system where at most t process can crash. As we shall see, k-set agreement is solvable for $C \in \mathcal{S}_t^{[d]}$ if $k \geq d+1$.

Notice that, if an input vector $J \in \mathcal{C}_t$ occurs in an execution of a protocol, then as far as the processes with non-$\perp$ values in J can tell, the input vector could belong to $I \in C$, because they cannot distinguish from another execution where the other processes wake up and propose their values after the former processes have made their decision. Given $C \in \mathcal{S}_t^{[d]}$, we say that C is *d-satisfied* for input vector J if $J \in \mathcal{C}_t$ or $\#_\perp(J) \geq t - d$.

Definition 5. *A t-fault tolerant protocol solves the k-set agreement problem for a condition $C \in \mathcal{S}_t^{[d]}$ if in every execution with input vector J, the protocol satisfies the following properties:*

- Validity. *Every decided value is a proposed value.*[3]
- Agreement. *No more than k different values are decided.*
- Termination. *Every correct process must decide if* (1) *C is d-satisfied for J and no more than t processes crash, or* (2.a) *a process decides, or* (2.b) *fewer than k processes crash.*

The first two are the safety requirements of the standard set agreement problem, and they should hold even if the input pattern does not belong to C. The third item requires termination under "normal" operating scenarios: (1) inputs that could belong to C or at least $t - d$ processes crash, and (2.a) executions where a process decides, or (2.b) fewer than k processes crash (a situation where k-set agreement is solvable without conditions).

[3] It is shown in [13] that the solvability of k-set agreement is highly sensitive to the validity property adopted.

Notice that, if set agreement is solvable for a condition C, then it is solvable for any C' contained in C: the same protocol works. As mentioned above, when $t < k$, k-set agreement is solvable for $\mathcal{V}^n$, hence for any condition C.

5. Combining conditions with ϕ_t^y to solve set agreement. This section presents a set agreement protocol with access to a failure detector of the class ϕ_t^y, $0 \leq y \leq t$, and instantiated with the function h of a $(t-d)$-legal condition $C \in \mathcal{S}_t^{[d]}$, $0 \leq d \leq t$. It is a t-fault tolerant protocol that solves k-set agreement for C_t^d, where $k = 1 + \max(0, d-y)$ (recall Definition 5). Thus, all of the pairs (d, y) such that $d \leq y$ allow solving condition-based consensus.

5.1. Base objects. In order to make the protocol simpler to understand, it is presented in a modular way. More specifically, it relies on the following base objects: Three arrays of atomic registers, a consensus object, an adopt–commit-abort object, and a condition-set agreement object. An adopt–commit-abort object and a condition-set agreement object can always be implemented on top of base read/write registers. As far as the consensus object is concerned, as we will see, it can always be implemented in the particular context in which it is used by the processes.

The shared memory. The shared memory is made up of three arrays (denoted $V[1..n]$, $W[1..n]$, and $DEC[1..n]$) of single-writer multireader atomic registers. All are initialized to $[\perp, \ldots, \perp]$. The jth entry of an array $X[1..n]$ can be read by any process, but only p_i can write to the ith component $X[i]$. To simplify the presentation we assume that, in addition to these atomic read and write operations, a process p_i can also invoke the nonprimitive operation snapshot(X) that allows it to read the content of all of the registers of the array X as if this reading was done instantaneously. (Such an operation can be implemented in shared memory systems made up of single-writer, multireader atomic registers despite any number of process crashes ($1 \leq t < n$) [1, 4].) In accordance with the terminology defined in [30], the read, write, and snapshot() operations are linearizable (i.e., they appear as if they had been executed one after the other, in agreement with their real-time occurrence order).

The underlying consensus object. A consensus object is accessed by a process p_i when p_i invokes the operation alg_cons(v_i), where v_i is the value proposed by p_i. Such an object allows any subset of processes to invoke alg_cons(). Its properties are the following:

- *Termination.* Any correct participating process decides a value.
- *Validity.* A decided value is a proposed value.
- *Agreement.* No two different values are decided.

As we will see, the underlying consensus object is used when more than $t - y$ process crash. It is shown in Theorem 8 that, in this case, a failure detector of the class $\mathcal{P}$ (the class of perfect failure detectors [8]; see section 8) can be built from a failure detector of the class ϕ_t^y (such a construction is described in the proof of Theorem 8), and consensus can be solved in a single-writer multireader atomic register asynchronous system enriched with such a failure detector.[4]

The underlying adopt–commit-abort object. The adopt–commit-abort object we use here is a simple variant of the adopt–commit-abort object introduced in [19, 52] in the context of shared memory systems, and an object introduced in [43]

[4] A $\diamond\mathcal{P}$-based alg_cons() protocol is described in [42]. That protocol uses an underlying adopt–commit-abort object. (Trivially, any failure detector in $\mathcal{P}$ is also in $\diamond\mathcal{P}$.) Other shared memory consensus algorithms based on failure detectors can be found in [5, 24, 34].

in the context of message-passing systems.[5] Such an object has a single operation, denoted adopt_commit(). A process p_i invokes adopt_commit(v_i), where v_i is the value it proposes to the adopt–commit-abort object and obtains a pair (d, v) as a result, where d is control tag and v a value. The object is defined by the following properties.

- *Termination.* Any correct participating process decides a pair (d, v).
- *Validity.* If a process decides (d, v), then $d \in \{commit, adopt, abort\}$, and v is a proposed value.
- *Agreement.* If a process decides $(commit, v)$, then any other process that decides, decides (d, v), with $d \in \{commit, adopt\}$.
- *Obligation.* If all of the participating processes propose the same value v, then only the pair $(commit, v)$ can be decided.

Intuitively, the adopt–commit-abort object is an abortable variant of consensus. Let us observe that a process that decides $(abort, -)$, can conclude that no process decides $(commit, -)$. However, when a process decides $(adopt, -)$, it cannot conclude which control tag has been decided by the other processes.

The underlying condition-set agreement object. A condition-set agreement object has a single operation, denoted cond_algo(). This object is designed to solve a set agreement problem with the help of a $(t - d)$-legal condition C. A process uses this object only in the particular context where the input vector J is such $\#_\perp(J) \leq t - y$.

A process p_i invokes cond_algo(V_i), where V_i is its local view of the input vector J (we have then $V_i \leq J$ and $\#_\perp(V_i) \leq t-y$), and only when the views can be ordered by containment, $V_i \leq V_j$ or $V_j \leq V_i$ for all i, j. If it returns from that invocation, p_i obtains a value v. The object is defined by the following properties.

- *Termination.* Every correct process decides if (1) $J \in \mathcal{C}_t$ or $\#_\perp(J) \geq t - d$ (C is d-satisfied for J), or (2.a) a process decides, or (2.b) more than $(n - k)$ correct processes invoke cond_algo().
- *Validity.* A decided value is a value that has been proposed by a process in its input view.
- *Agreement.* At most, $k = 1 + \max(0, d - y)$ values are decided.

5.2. The set agreement protocol.

Description of the protocol. The k-set agreement protocol based on a condition in $C \in \mathcal{S}_t^{[d]}$ and a failure detector of the class ϕ_t^y is described in Figure 1. The variables subscripted with i are local variables of p_i. A process is made up of two tasks: a main task $T1$ and a background task $T2$. The behavior of the task $T1$ can be decomposed into four parts.

- A process first writes the value v_i it proposes into $V[i]$ (line 1). Then, using the snapshot() operation, it reads the array of proposed values until that array contains "enough" values (line 2). "Enough" means here that there are no more than $(t-y)$ missing values, or there are more than $(t-y)$ processes that have crashed; this is known from the invocation QUERY$_y(S_i)$. (Let us recall that, when $|S_i| \leq t - y$, QUERY$_y(S_i)$ answers always *true*).
- Then, the behavior of p_i depends on the number of values it knows. If there are too many crashes (line 4), p_i sets a local variable $prop_i$ to the value

[5] A wait-free implementation of an adopt–commit-abort object from single-writer multireader atomic registers can be found in [52]. An implementation in message-passing systems, where a majority of processes is correct, is presented in [43]. For completeness, an implementation of an adopt–commit-abort object is described in Appendix A.

```
Function k-set_agreement^{[d,y]}_{n,t} (v_i)

task T1:
(1)  V[i] ← v_i;
(2)  repeat V_i ← snapshot (V); S_i ← {j | V_i[j] = ⊥}
(3)                                  until QUERY_y (S_i) end repeat;
(4)  case (#_⊥(V_i) > t − y) then prop_i ← CONS; w_i ← ⊥
(5)       (#_⊥(V_i) ≤ t − y) then prop_i ← COND; w_i ← cond_algo(V_i)
(6)  end case;
(7)  W[i] ← w_i; (d_i, res_i) ← adopt_commit(prop_i); W_i ← snapshot(W);
(8)  case (res_i = CONS) ∨ (d_i = abort)
(9)                  then DEC[i] ← cond_algo(v_i); return (DEC[i])
(10)      (res_i = COND) ∧ (d_i = commit)
(11)                 then DEC[i] ← W_i[j] such that W_i[j] ≠ ⊥; return(DEC[i])
(12)      (res_i = COND) ∧ (d_i = adopt)
(13)                 then est_i ← W_i[j] such that W_i[j] ≠ ⊥;
(14)                      DEC[i] ← cond_algo(est_i); return(DEC[i])
(15) end case

task T2:
(16) j ← 0;
(17) repeat forever j ← (j mod n) + 1;
(18)                 if (DEC[j] ≠ ⊥) then return(DEC[j]) end if
(19) end repeat
```

FIG. 1. *A k-set agreement protocol with* $k = 1 + \max(0, d - y)$.

CONS to try to decide a value from the underlying consensus algorithm (let us remind that, when there are more than $(t - y)$ crashes, it is possible to solve consensus from ϕ_t^y). In the other case, p_i knows enough proposed values to decide from the condition (line 5); p_i computes, consequently, a value w_i that could be decided from the condition and sets $prop_i$ to *COND*.

- The process then uses the underlying adopt–commit-abort object (line 7) in order to try agreeing on the same tag, namely, *CONS* or *COND*. Moreover, each p_i deposits in the array $W[1..n]$ the value it has computed at line 4 or line 5 and reads that array with the snapshot() operation.
- The last part depends on the result returned by the adopt–commit-abort object.
  - If p_i obtains $d_i = abort$ or $res_i = CONS$, it concludes that no value can be decided from the condition. It consequently uses the consensus object to decide a value (lines 8–9).
  - If p_i obtains $res_i = COND$, at least one entry of W_i is not equal to $\perp$. Then, if, additionally, $d_i = commit$, p_i concludes that any value deposited in W can be decided from the condition, and it decides it (lines 10–11).
  - If p_i obtains $res_i = COND$ together with $d_i = adopt$, it does not know if the other processes p_j have obtained $d_j = commit$ or $d_j = abort$. So, to be consistent, p_i participates in the underlying consensus to which it proposes a value that could be decided from the condition (lines 12–14). It then decides the value returned by the consensus object.

The aim of task $T2$ is to guarantee that a correct process always decides as soon as a process decides. To that end, when a process p_j is about to decide in task

$T1$ (execution of the return(v) statement), it first writes v in $DEC[j]$.[6] Task $T2$ of a process p_i is then a simple loop statement that terminates when the predicate $(\exists j : DEC[j] \neq \perp)$ becomes true. The execution of the return() statement by a process p_i terminates its execution of k-set_agreement$^{[d,y]}_{n,t}$ (v_i).

Proof of correctness.

THEOREM 3. *When instantiated with a failure detector of the class ϕ^y_t and a condition C in $S^{[d]}_t$, the protocol described in Figure* 1 *solves the condition-based k-set agreement problem where $k = 1 + \max(0, d - y)$.*

Proof. Validity property (a decided value has been proposed by a process). Let us observe that a decided value dec_i is either an initial value v_j proposed to the consensus by a process p_j (line 9) or a value w_i obtained by a process p_i from the condition-set agreement object (lines 11 and 14). The validity property follows immediately from the corresponding validity property of the consensus object and the condition-set agreement object, and hence line 18 preserves validity.

Agreement property (at most k different values are decided). Let us observe that, due to the agreement property of the adopt–commit-abort object, it is not possible for two processes p_i and p_j that have invoked adopt_commit() at line 7 to be such that both the predicate $(res_i = CONS) \vee (d_i = abort)$ and the predicate $(res_j = COND) \wedge (d_j = commit)$ are true. It follows from that observation that it is not possible for a process p_i to execute line 9 while another process p_j executes line 11. So, there are only two cases to consider (in addition to the trivial case of line 18).

- No process begins executing line 9 or 14. In that case, these processes decide the value returned by the consensus object. Due to the consensus agreement property, there is a single such value.
- No process begins executing line 11 or 14. In that case, these processes decide a value returned from the condition-set agreement object . Due to the condition agreement property, there are at most $k = 1 + \max(0, d - y)$ such values, which proves the case.

Termination property. Let J be the input vector. We have to show that every correct process decides if (1) the condition C is d-satisfied for J, or (2.a) a process decides, or (2.b) fewer than k processes crash.

Let us notice that, as there are at most t process crashes (model assumption), the repeat loop of lines 2–3 always terminates. Moreover, let us also observe that, due to the termination property of the adopt–commit-abort object, any invocation of adopt_commit() issued by a correct process terminates (observation $O1$).

Let us also observe that the underlying consensus protocol is used only when the number of crashes is greater than $t - y$ (line 4), i.e., when a failure detector of the class $\mathcal{P}$ can be built from a failure detector of the class ϕ^y_t (item (3) of Theorem 8). The termination property of the consensus object ensures that all of the correct processes that invoke cons_alg() terminate their operation (observation $O2$). Let us now proceed by a case analysis.

- Case (1): We have to show that any correct process decides when the condition C is d-satisfied for J, where J is the input vector.
  In that case, it follows from item (1) of the termination property of the condition-set agreement object that any invocation cond_algo() issued by a

[6] This write plays the same role as the reliable broadcast of the decided value in message-passing systems (e.g., see the consensus protocols in [8, 23, 43, 51]). Their aim is to prevent a process from deadlocking.

correct process terminates. This, combined with the observations $O1$ and $O2$, allows us to conclude that any correct process terminates when C is d-satisfied for J.

- Case (2.a): We have to show that any correct process decides, as soon as a process decides.
  This property is trivially guaranteed by the management of the array $DEC[1..n]$ and task $T2$.
- Case (2.b): We have to show that any correct process decides when fewer than k processes crash.
  If no correct process accesses the condition-set agreement object, the fact that any correct process decides follows from the observations $O1$ and $O2$. So, let us consider the case where correct processes access the condition object. As $k = 1 + \max(0, d - y)$ and fewer than k processes crash, this means that at most $\max(0, d-y)$ processes crash. Moreover, as $t \geq d$, we have $t-y \geq d-y$. It follows (from the properties of the failure detector ϕ_t^y) that all of the processes p_i that execute the repeat loop (lines 2–3) and do not crash while executing that loop, eventually exit it, and we have then $\#_\perp(V_i) \leq t - y$. They consequently all of access the condition-set agreement object at line 5. It follows that all of the correct processes (there are more than $(n - k)$ of them) invoke the condition-set agreement object. Due to item (2.b) of the termination property of the condition-set agreement object, it follows that any correct process decides in the k-set agreement protocol. □

5.3. Implementation of a condition-set agreement object.

Description of the protocol. A t-fault tolerant protocol implementing a condition-set agreement object is described in Figure 2. This protocol is instantiated with a function h associated with a $(t - d)$-legal condition C. It uses a deterministic function $F()$ and a predicate $P()$. The function $F()$ takes a view J as a parameter and returns a non-$\perp$ value of the vector J. The value $\top$ is a default value not in $\mathcal{V}$ and different from $\perp$. It is assumed that the function h is extended to all views J of C, with at most $t - d$ entries equal to $\perp$ as in Definition 4. The predicate $P()$ is true on all such views:

$$P(V_i) \equiv \big(\exists I \in C \text{ such that } V_i \leq I\big).$$

Thus, $P()$ is used to test if p_i's current view V_i of the input vector could originate from a vector of the condition.[7]

The protocol can be seen as a case analysis. The first step is for p_i to check whether $\#_\perp(V_i) \leq t - d$ in order to benefit from the condition. If $\#_\perp(V_i) > t - d$, p_i cannot benefit from it and consequently decides a value from its local view V_i at line 18 (the processes executing that line decide at most $\max(0, d - y)$ different values).

Otherwise, we have $\#_\perp(V_i) \leq t - d$, and then p_i has enough non-$\perp$ entries in its view V_i to test if the condition can help it decide. So, p_i enters the lines 2–17. There are three cases. If $P(V_i)$ is satisfied (first case), p_i decides the value from the condition and writes it in the shared array D to help other processes decide (line 4).

If $P(V_i)$ is not satisfied (second case), p_i first checks if $\#_\perp(V_i) = t-d$. If so (second case), it knows that no other process will evaluate P to true in the previous line, and

[7] It is shown in [36] that, for some conditions, there are very efficient ways to compute the predicate $P()$. As an example, for the $(t - d)$-legal condition $C1$ (defined in section 4.2), we have $P(V_i) \equiv \#_{\max(J)}(V_i) > (t - d) - \#_\perp(V_i)$.

```
Function cond_algo (V_i)     % We have #_⊥(V_i) ≤ t − y, and V_i ≤ V_j or V_j ≤ V_i for all i, j %

(1)  if (#_⊥(V_i) ≤ t − d)
(2)    then if P(V_i)
(3)           then % The processes executing this line decide the same value %
(4)                w_i ← h(V_i); D[i] ← w_i; return (w_i)
(5)           else  if (#_⊥(V_i) = t − d)
(6)                   then % The processes executing this line decide the same value %
(7)                        w_i ← F(V_i); D[i] ← w_i; return (w_i)
(8)                   else % If processes execute these lines, at most k value can be decided %
(9)                     D[i] ← ⊤;
(10)                    repeat D_i ←snapshot (D) until ( (∃j : D_i[j] ≠ ⊥, ⊤) ∨ (#_⊥(D_i) < k) );
(11)                    if (∃j :  D_i[j] ≠ ⊥, ⊤) then return (D_i[j] such that D_i[j] ≠ ⊥, ⊤)
(12)                                          else  ∀j : if (D_i[j] = ⊤) then Y_i[j] ← V[j]
(13)                                                                     else  Y_i[j] ← ⊥ end_if;
(14)                                                 w_i ← F(Y_i); D[i] ← w_i; return (w_i)
(15)                    end_if
(16)                end_if
(17)         end_if
(18)   else w_i ← F(V_i); D[i] ← w_i; return (w_i) % Here (t − d) < (#_⊥(V_i) ≤ t − y) %
(19)        % The processes executing that line decide at most max(0, d − y)) values %
     end_if
```

FIG. 2. *A condition protocol.*

that any other process p_j, with $\#_\perp(V_j) = t - d$ has $V_i = V_j$, so it deterministically decides $F(V_i)$ (line 7).

In the third case, $\#_\perp(V_i) < t - d$, and p_i writes $\top$ in $D[i]$ to indicate it cannot decide from its local view V_i (so, $D[j] = \perp$ means that p_j has not yet finished executing its protocol or has crashed). Then, as it cannot decide by itself, p_i starts the "best effort termination" part of the protocol (lines 9–15). It enters a loop (line 10), during which it looks for a decided value ($\exists j : D_i[j] \neq \perp, \top$) and decides if there is one (line 11) or a configuration where $\#_\perp(D_i) < k$ (this is the only place where k is used in the protocol). If the condition $(\nexists j : D_i[j] \neq \perp, \top) \wedge (\#_\perp(D_i) < k)$ is satisfied, p_i builds a local view of the input vector corresponding to the processes that have executed at least until line 9. As we will see in the proof, if several such views (Y_i, Y_j, etc.) are computed, due to the invocations of snapshot(D) at line 10 that precede their construction, the associated containment property implies that these views (Y_i, Y_j, etc.) are also ordered by containment. The process p_i then decides the value $F(Y_i)$. Let us notice that, as $\#_\perp(D_i) < k$, the vector Y_i has at most $k - 1$ entries equal to $\perp$. It follows that at most k different values can be decided at line 14. Let α be the number of values decided at line 4, 7, 11, and 18, and let β be the number of values decided at lines 14. The proof will show that $\alpha + \beta \leq k$.

Correctness proof.

THEOREM 4. *When instantiated with a $(t - d)$-legal condition C, the protocol described in Figure* 2 *implements a condition-set agreement object with $k = 1 + \max(0, d - y)$.*

Proof. Validity property (a decided value is a value proposed in the input view of a process). This property follows directly from the fact that both the function $h()$ and the function $F()$ extract a non-$\perp$ value from the vector they are applied to.

Agreement property (at most $k = 1 + \max(0, d - y)$ different values are decided). The processes that decide, do it at line 4, 7, 11, 14, or 18. We determine the maxi-

mum number of values that are decided by the processes at each of these lines of the protocol.

- Consider the processes that decide at line 4.
  These processes p_i are such that $(\#_\perp(V_i) \leq t-d)$ and $P(V_i)$ is satisfied. We show that a single value can be decided at line 4.
  Let p_i and p_j be two processes that decide at line 4. Due to the use of a snapshot operation, we have either $V_i \leq V_j$ or $V_j \leq V_i$. Let us consider that $V_i \leq V_j$.
  We then have (1) $\#_\perp(V_i) \leq t-d$ and $\#_\perp(V_j) \leq t-d$, (2) both $P(V_i)$ and $P(V_j)$ are satisfied, i.e., $\exists I1 \in C$ such that $V_i \leq I1$ and $\exists I2 \in C$ such that $V_i \leq V_j \leq I2$, (3) and the condition is $(t-d)$-legal. It follows from these three items and Lemma 1 that $h(V_i) = h(V_j) = h(I_1) = h(I_2)$. Consequently, no more than one value can be decided by the processes executing line 4.
- Consider the processes that decide at line 7.
  These processes p_i are such that $(\#_\perp(V_i) = t-d)$ and $P(V_i)$ is not satisfied. In this case, at most one value is decided at line 7, because, due to the snapshot containment property, all processes that execute this line have exactly the same view V_i. Moreover, if a process executes this line, no process executes line 4. This is because any process p_j that executes line 4 has a view V_j such that $(\#_\perp(V_j) \leq t-d = \#_\perp(V_i))$, and as we have either $V_i < V_j$ or $V_j < V_i$, we conclude that $V_i < V_j$. Consequently, if p_i executes line 7, $P(V_i)$ is false, and hence $P(V_j)$ is also false as $V_i \leq V_j$, by definition of the predicate $P()$.
- Consider the processes that decide at line 18.
  These processes p_i are such that $(t-y \geq \#_\perp(V_i) > t-d)$. We show that these processes decide at most $\max(0, d-y)$ different values.
  Due to the containment property on the vectors provided by the snapshot operation, any pair of processes p_i and p_j that execute line 18 are such that $V_i \leq V_j$ (or $V_j \leq V_i$). We conclude from that observation that the processes that execute line 18 have at most $\max(0,(t-y)-(t-d))$=$\max(0,d-y)$ different vectors. As F is deterministic, at most $\max(0,d-y)$ different values can be decided by the processes that decide at line 18.
  It follows that, when we consider the processes that decide at line 4, 7, or 18, at most $k = 1+\max(0,t-d)$ different values can be decided.
- Consider the processes that decide at line 11.
  A process p_i that decides at line 11 decides a value (that it retrieves in $D[j]$) that has been decided by another process p_j (p_j has deposited that value in $D[j]$ at line 4, 7, 14, or 18). Consequently, no additional value can be decided at line 11.
- Finally, consider the processes that decide at line 14.
  Let β be the number of different values decided by the processes that execute line 14. Let α be the number of values decided by the processes that execute line 4, 7, or 18. We claim that $\alpha + \beta \leq k = 1+\max(0,t-d)$.

It follows from this case analysis that at most $k = 1+\max(0,t-d)$ different values can be decided, which proves the theorem.

Proof of the claim. Let us consider two time instants t_0 and t_1 defined as follows:
- t_0 = first time instant where $\#_\perp(D) = k-1$ (or $+\infty$ if it never happens),
- t_1 = first time instant where $\exists\ D[j] \notin \{\top, \perp\}$ (or $+\infty$ if it never happens).[8]

[8]If both t_0 and t_1 are equal to $+\infty$, no process decides, and the claim is trivially true.

Let us first consider $t_1 \leq t_0$. Let us notice that a process p_i stops the repeat loop of line 10 as soon as $(\exists\, D_i[j] \notin \{\top, \bot\} \ \vee\ \#_\bot(D_i) < k)$. As the test that (at line 11) immediately follows the exit of the repeat loop privileges the case $(\exists\, D_i[j] \notin \{\top, \bot\})$ with respect to the case $(\#_\bot(D_i) < k)$ when both are satisfied, it follows that p_i immediately executes the return () statement at line 11. Consequently, when $t_1 \leq t_0$, any process p_i that enters the loop of line 10 and then decides, decides at line 11. We then have $\beta = 0$ (no process decides at line 14).

Let us now consider the case $t_0 < t_1$. Let us first observe that, since the function $F()$ is deterministic and each Y_i computed at lines 12–13 contains at most $(k-1)$ entries equal to $\bot$, it follows that the β values decided at line 14 correspond to (at least) β different Y_i vectors, which means (due to line 12) at least β different D_i vectors.

Due to the containment property of the invocations of the snapshot (D) invocations at line 10, the previous β D_i vectors are totally ordered (see the definition of "$<$" in section 4.1), e.g., $D_{i1} < D_{i2} < \cdots < D_{i\beta} < \cdots$ and contain only $\bot$ and $\top$ entries. Moreover, for any pair of such vectors, there is at least one entry which is equal to $\bot$ in one vector and to $\top$ in the other. As D is initialized to $[\bot, \ldots, \bot]$ and there are at least β different D_{ix}, we conclude that at least $(\beta - 1)$ values $\top$ have been written into D after t_0 (because, due to the snapshot (D) operations, we have $\#_\bot(D_{i1}) \leq k-1$ at time t_0, $\#_\bot(D_{i2}) \leq k-2$ at time t'_0, $t'_0 > t_0$, etc.).

Before being decided, the α different values decided at lines 4, 7, and 18 have been written into the array D (they are decided after t_1). Due to the definition of t_1, they have been written into D at or after t_1, i.e. (from the case assumption), after t_0.

Hence, after t_0, α entries of D have been set to proposed values by lines 4, 7, and 18, and $(\beta - 1)$ entries have been set to $\top$. As, at t_0, the number of entries of D that were equal to $\bot$ was equal to $(k-1)$, it follows that $\alpha + (\beta - 1) \leq (k-1)$, i.e., $\alpha + \beta \leq k$, which proves the claim when $t_0 < t_1$. *End of the proof of the claim.*

Termination property (let J be the actual input vector). Every correct process decides if (1) $J \in \mathcal{C}_t$ or $\#_\bot(J) \geq t - d$ -C is d-satisfied for J-, or (2.a) a process decides, or (2.b) more than $(n-k)$ correct processes invoke cond_algo().

If the input vector V_i, $V_i \leq J$, is such that $\#_\bot(V_i) > t - d$, the process p_i trivially decides at line 18. When $\#_\bot(V_i) = t - d$, the test on line 5 leads to termination. On another side, if $\#_\bot(V_i) \leq t - d$ and $J \in \mathcal{C}_t$, then $P(V_i)$ is satisfied, and p_i decides at line 4. So, the case (1) is done.

Let us consider case (2.a). Before deciding a value at line 4, 7, 14, or 18, a process deposits that value in the array D. It follows that, after a process has decided, the repeat loop of line 10 always terminates, and any process that executes line 11 decides, which proves the case.

Let us finally consider case (2.b). Let us assume that more than $(n-k)$ correct processes invoke the object and no one decides. This means that none of them executes line 4, 7, or 18. They all, consequently, enter the repeat loop at line 10, from which we conclude that eventually the predicate $\#_\bot(D_i) < k$ becomes true. It follows that the correct processes exit the repeat loop and decide. □

6. Discussion.

6.1. Initial crashes: No condition is needed. The theorem that follows considers a particular case, namely, the case where the faulty processes crash before the protocol starts its execution.

THEOREM 5. *Consider an execution of the protocol described in Figure* 1 *instantiated with a failure detector of the class* ϕ_t^y. *Let us assume that more than* $(t - y)$

processes have crashed before the protocol starts. The protocol then solves the consensus problem (whatever the $(t-d)$-legal condition it is instantiated with).

Proof. If more than $(t-y)$ processes have initially crashed, due to the property of the QUERY$_y$ () invocations at line 3, we have $(\#_\perp(V_i) > t-y)$ for any process p_i. It follows that, for any process p_i, we have $prop_i = CONS$ (line 4). Due to the obligation property of the adopt–commit-abort object, every process p_i obtains $(commit, CONS)$. Consequently, all of the processes invoke the underlying consensus object, which proves the theorem. □

Remark 1. The previous theorem considers the case where the faulty processes have crashed before the protocol starts. It is interesting to observe that a similar result appears in [16], where a consensus protocol is presented for asynchronous systems where a majority of processes are correct, and the faulty processes crash before the protocol starts its execution.

6.2. An always terminating version of the protocol. It is possible to trade safety for liveness by providing a version of the protocol where every correct process always decides. This can be obtained at the price of an enlarged set of possibly decided values. More precisely, let I be an input vector, and let C be the $(t-d)$-legal condition the protocol is instantiated with. When $I \in C$, at most $k = 1 + \max(0, d-y)$ values are decided; when $I \notin C$, up to $k' = t+1-y$ values can be decided. Interestingly, this always terminating version of the protocol provides a new insight into the way the parameters t, y (power of the failure detection) and d (power of the condition) are related.

In the protocol described in Figure 2, the statement that can prevent a correct process p_i from terminating is the repeat loop at line 10. This occurs when p_i enters lines 9–15, V_i being such that $\#_\perp(V_i) \leq \min(t-y, t-d)$ (assumption on the input parameter and line 1), while $P(V_i)$ is equal to *false* (line 2). The modification to get an always terminating cond_algo() protocol is very simple: It consists in replacing the lines 9–15 in Figure 2 by a weakened statement that always terminates, namely,

```
[9-15]'   if (∃j : D[j] ≠ ⊥) then return (D[j] such that D[j] ≠ ⊥)
                              else w_i ← F(V_i); D[i] ← w_i; return (w_i)
          end if
```

THEOREM 6. *Let us consider the protocol depicted in Figure* 2 *instantiated with a $(t-d)$-legal condition C, where lines* 9–15 *are replaced by the statement* [9–15]'. *Every correct process decides. Let I be an input vector. If $I \in C$, at most $k = 1+\max(0, d-y)$ values are decided. If $I \notin C$, at most $k' = t+1-y$ values can be decided.*

Proof. Every correct process trivially terminates, and a decided value comes from a proposed vector (same proof as in Theorem 4).

As far as the number of values that are decided is concerned, let us first consider the case where the input vector belongs to the condition. In that case, when a process p_i executes line 2, $P(V_i)$ is trivially satisfied. It follows that the new line [9-15]' is never executed. Consequently, Theorem 4 remains valid when $I \in C$, and at most $k = 1 + \max(0, d-y)$ values are then decided.

Considering now the case where the input vector does not belong to the condition, let us first observe that if a process p_i decides at line [9-15]' a value $D[j]$ such that $D[j] \neq \perp$, it does not decide a new value as $D[j]$ is counted as a decided value at line 4, 7, 18 or in the **else** part of the new **if** statement. So, let us count the number of values that can be decided by the processes executing line 4 or the **else** part of the new line [9-15]'. For each such process p_i, we have $\#_\perp(V_i) \leq t - \max(y, d+1)$. Moreover (due to the containment property on the vectors V_i provided by the cond_algo() invocations), we have $V_i \leq V_j$ (or $V_j \leq V_i$) for two processes executing line 4 or the **else** part of

line [9-15]'. It then follows that there are at most $k1 = t - \max(y, d+1) + 1$ different vectors V_i for the processes that execute line 4 or the **else** part of line [9-15]'. Let us observe that if a process p_j decides at line 4, the same vector V_j will not be used to decide another value at line [9-15]'. Finally, due to that observation and the fact that $F()$ is deterministic, at most $k1$ different values can be decided by the processes executing line 4 or the **else** part of line [9-15]'. On the other side, the processes that execute line 18 decide at most $k2 = \max(0, d-y)$ different values (the proof is the same as the corresponding proof in Theorem 4). Recall that all of the processes that execute line 7 decide the same value. Finally, summing up, we get $k' = k1 + k2 + 1$, i.e., $k' = (t - \max(y, d+1) + 1) + (\max(0, d-y) + 1)$, which can be simplified to provide $k' = t + 1 - y$. □

Let us notice that when the input vector does not belong to the condition, the maximal number of values that can be decided, namely, $k' = t + 1 - y$, does not depend on d. If the information on failure is maximal ($y = t$), the protocol solves consensus. At the other extreme, if there is no information on failures ($y = 0$) and there is no power provided by the condition, the protocol solves the trivial version of the set agreement problem, namely, $k' = t + 1$.

7. A lower bound. This section presents a lower bound matching Theorem 3.

THEOREM 7. *When instantiated with a failure detector of the class ϕ_t^y and a $(t-d)$-legal condition, no protocol solves the condition-based k-set agreement problem for $k \leq \max(0, d-y)$.*

Proof. Assume for contradiction that a protocol solves the k-set agreement problem for $k \leq \max(0, d-y)$. Hence, $d > y$ and $\max(0, d-y) = d - y$. Partition the processes in two groups: the *main* processes $p_1, \ldots, p_{n-t+d}$ and the *secondary* processes, $p_{n-t+d+1}, \ldots, p_n$. Consider the executions where the secondary processes crash before taking any steps. These are executions with at least $t - d$ failures. By Definition 5, all correct process must decide whatever the input vector. Now, consider the subset of these executions with at most $d-y$ additional failures. The total number of failures is at most $t - y$ failures. Recall that any relevant query is invoked with a set the size of which is greater than $(t - y)$. So, all relevant invocations QUERY$_y()$ issued by the main processes will include at least one correct process and thus will return *false*, and all other invocations return the trivial output. Thus, in these executions, the failure detector gives no information, and therefore the main processes have to solve the standard set agreement problem (i.e., terminate for every input vector), tolerating $d-y$ failures. The results of [6, 28, 29, 50] (more specifically, Corollary 5.5 in [28]) imply that, in one of these executions, at least $d - y + 1$ different values are decided, a contradiction. □

The following corollaries are direct consequences of the previous theorem. They consider the extreme cases where there is either no failure detector (i.e., $y = 0$) or no condition (i.e., $d = t$). The first corollary answers an open problem stated in [3, 39]. The second corollary shows the optimality[9] of ϕ_t^y.

COROLLARY 1. *Let C be a $(t-d)$-legal condition. There is no condition-based k-set agreement protocol for C when $k \leq d$.*

COROLLARY 2. *When considering the family $(\phi_t^y)_{0 \leq y \leq t}$ of failure detector classes, ϕ_t^y, with $y = t-k+1$, is the weakest that allows solving the k-set agreement problem.*

[9] This result complements another k-set agreement minimality result [27], which shows that, among the family $(\mathcal{S}_x)_{1 \leq x \leq t+1}$ of perpetual failure detectors (introduced in [44, 52]), $\mathcal{S}_x$ is the weakest to solve the k-set agreement problem for $k > t - x + 1$.

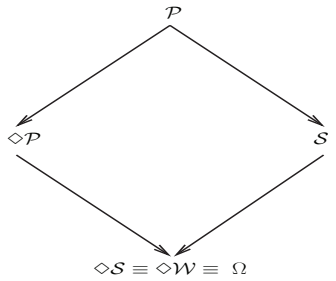


FIG. 3. *Relations among Chandra–Toueg's failure detector classes.*

8. Comparing ϕ_t^y with Chandra and Toueg's failure detector classes.

8.1. Chandra and Toueg's failure detector classes. This section presents the failure detectors introduced by Chandra and Toueg, used in this paper. These classes are defined from the following completeness and accuracy properties [8]:
- *Strong (Weak) completeness.* Eventually, every process that crashes is permanently suspected by every (some) correct process.
- *Perpetual strong accuracy.* No process is suspected before it crashes.
- *Eventual strong accuracy.* There is a time after which no correct process is suspected.
- *Perpetual weak accuracy.* Some correct process is never suspected.
- *Eventual weak accuracy.* There is a time after which some correct process is never suspected.

The classes we are interested in are the following [8]. They are collectively called "Chandra and Toueg's failure detector classes" in the rest of the paper.

- $\mathcal{P}$: The class of *perfect* failure detectors. It includes all of the failure detectors satisfying strong completeness and perpetual strong accuracy.
- $\mathcal{S}$: The class of *strong* failure detectors. It includes all of the failure detectors satisfying strong completeness and perpetual weak accuracy. We have $\mathcal{P} \subseteq \mathcal{S}$.
- $\diamond\mathcal{P}$: The class of *eventually perfect* failure detectors. It includes all of the failure detectors satisfying strong completeness and eventual strong accuracy. We have $\mathcal{P} \subseteq \diamond\mathcal{P}$.
- $\diamond\mathcal{S}$: The class of *eventually strong* failure detectors. It includes all of the failure detectors satisfying strong completeness and eventual weak accuracy. We have $\diamond\mathcal{P} \subseteq \diamond\mathcal{S}$, and $\mathcal{S} \subseteq \diamond\mathcal{S}$.

The class $\diamond\mathcal{S}$ is the weakest that allows solving the consensus problem and is equivalent to the class $\diamond\mathcal{W}$ in shared memory systems and in message-passing systems with reliable channels [8, 7]. It has also been shown that $\diamond\mathcal{S}$ and the class of *leader* failure detectors, denoted Ω, are equivalent in systems where each process initially knows all of the process identities [7, 11, 41].

Figure 3 summarizes Chandra and Toueg's failure detector classes. Following Definition 1, an arrow from A to B means that $A \geq B$ (any failure detector of the class A can be used to build a failure detector of the class B). The absence of a path from A to B means that it is not the case $A \geq B$ (given any failure detector of the class A, it is not possible to build a failure detector of the class B). Finally, $A \equiv B$ if $A \leq B$ and $B \leq A$. The figure follows from [7, 8].

8.2. ϕ_t^y with respect to Chandra and Toueg's failure detector classes. This section studies the relation between ϕ_t^y and the classic failure detectors introduced by Chandra and Toueg. We show that ϕ_t^t allows building a perfect failure

init: $suspected_i \leftarrow \emptyset$

repeat forever for all j **such that** $p_j \in (\{p_1, \ldots, p_n\} \setminus suspected_i)$ **do**
  if QUERY$_t(\{p_j\})$ **then** $suspected_i \leftarrow suspected_i \cup \{p_j\}$ **end if**
end repeat

FIG. 4. *From ϕ_t^t to $\mathcal{P}$ (algorithm for p_i).*

detector, namely, $\mathcal{P} \leq \phi_t^t$ (Theorem 8). Therefore, $\mathcal{P}$ is equivalent to ϕ_t^t as $\phi_t^t \leq \mathcal{P}$ (from their definitions).

THEOREM 8. (1) $\mathcal{P} \equiv \phi_t^t$. *Let f denote the actual number of process crashes in a run.* (2) *If $f \leq t-y$, ϕ_t^y, $0 \leq y \leq t-1$ does not allow building a failure detector of any of Chandra and Toueg's failure detector classes (e.g., $\diamond\mathcal{S}, \diamond\mathcal{P}, \Omega$).* (3) *If $f > t-y$, ϕ_t^y, $0 \leq y \leq t-1$ allows building a failure detector of the class $\mathcal{P}$.*

Proof. Let us first consider item (1). The construction described in Figure 4 constructs a perfect failure detector from a failure detector of the class ϕ_t^t. This construction works as follows. A process that queries the perfect failure detector obtains the current value of the set $suspected_i$. As $y = t$, QUERY$_t(S)$—where S is made up of a single process p—eventually returns *true* if and only if p has crashed. The strong completeness and strong accuracy properties defining the class $\mathcal{P}$ follow. Moreover, $\phi_t^t \leq \mathcal{P}$ follows directly from their definitions. Therefore, $\mathcal{P}$ is equivalent to ϕ_t^t.

For proving item (2), let us first observe that, an implementation that systematically suspects all of the processes trivially satisfies the completeness property of any of Chandra and Toueg's failure detector classes but prevents its accuracy property from being satisfied. So, assuming that ϕ_t^y $(0 \leq y \leq t-1)$ allows implementing the accuracy property of any of Chandra and Toueg's failure detector classes, we show that it does not allow implementing the associated (weak or strong) completeness property.

Let us consider any run during which no more than $x = t-y$ $(1 \leq x \leq t)$ processes crash. Due to the definition of ϕ_t^y, we have the following:

- Any QUERY$_t(S)$, where $|S| \leq t-y = x$ always returns *true* whatever the x (≥ 1) processes composing S. This follows from the triviality property of ϕ_t^y, $|S| \leq t-y = x$.
- Any QUERY$_t(S)$, where $|S| > x$ always returns *false* whatever the processes composing S. This follows from the safety property of ϕ_t^y, as at least one process among these processes has not crashed.

These observations show that, when no more than $x = t-y$ $(1 \leq x \leq t)$ processes crash, the boolean value returned by a query depends only on the number of processes defining S (it depends neither on which processes are in S, nor on the failure pattern). It follows that, when no more than $x = t-y$ $(1 \leq x \leq t)$ processes crash, there is no way for a process to know if a given process has crashed or not, thereby making impossible to implement the (weak or strong) completeness property of any of Chandra and Toueg's failure detector classes.

The proof of item (3) consists in designing an algorithm that, in runs where $f > t-y$, builds a failure detector of the class $\mathcal{P}$ from a failure detector of the class ϕ_t^y. Let us first observe that, as $f > t-y$, there is a set S such as $|S| = t-y+1$, and, after some finite time, QUERY$_y(S)$ returns *true* forever. The algorithm is the following.

- Each set $suspected_i$ is initialized to $\emptyset$. Initially, each process p_i issues QUERY$_y$ (X) for all of the possible sets X of size $|X| = t-y+1$ until such a query

```
when QUERY_y (S) is invoked by p_i:
    case (|S| ≤ t − y)        then return (true)
         (|S| > t)            then return (false)
         (t − y < |S| ≤ t)    then return (S ⊆ suspected_i)
    end case
```

FIG. 5. *From $\mathcal{P}$ to ϕ_t^y (algorithm for p_i).*

returns *true*. Due to the fact that all of the queries are relevant ($t-y < |X| \leq t$), and the previous observation, this eventually happens. When it occurs, p_i considers the corresponding set (say S) and executes $suspected_i \leftarrow S$.

- Then, for each $p_j \notin suspected_i$, p_i regularly executes $\text{QUERY}_y(S \cup \{p_j\})$. If the query returns *true*, p_i can conclude from the property of ϕ_t^y that p_j has crashed. It consequently adds p_j to $suspected_i$. Otherwise, p_i keeps on issuing $\text{QUERY}_y(S \cup \{p_j\})$.

It follows from the definition of S and the safety and liveness properties of ϕ_t^y that the sets $suspected_i$ of the correct processes eventually include all of the crashed processes and never includes a "not yet" crashed process, i.e., they satisfy the properties that define the class $\mathcal{P}$ of perfect failure detectors [8]. □

8.3. From Chandra and Toueg's failure detectors to ϕ_t^y. Figure 5 presents a simple protocol transforming any failure detector of the class $\mathcal{P}$ into a failure detector of the class ϕ_t^y. The underlying set $suspected_i$ satisfies (by assumption) the properties defining the class $\mathcal{P}$. In contrast, we show that there is no protocol transforming any failure detector of the class ϕ_t^y, for $y < t$, into a failure detector of the class $\mathcal{P}$.

THEOREM 9. *The protocol of Figure* 5 *transforms any failure detector of the class $\mathcal{P}$ into a failure detector of the class ϕ_t^y for $0 \leq y \leq t$.*

Proof. The triviality property of ϕ_t^y is ensured by the first two case statements. The safety property follows from the fact that, due to the perpetual strong accuracy of the underlying failure detector, $suspected_i$ contains only crashed processes. Finally, the liveness property of ϕ_t^y follows from the fact that, due to the completeness of the underlying failure detector, the set $suspected_i$ eventually contains all crashed processes. □

The next theorem states that there is no protocol transforming a failure detector of the class $\mathcal{S}$, $\diamond\mathcal{P}$, $\diamond\mathcal{S}$, or $\diamond\mathcal{W}$ into a failure detector of the class ϕ_t^y for $0 < y$. It is surprising that these failure detectors are not strong enough to implement ϕ_t^y, even when $y < t$, as in this case ϕ_t^y cannot solve consensus (Corollary 2), while these failure detectors can solve consensus. (In the case of $y = t$, both ϕ_t^t and those failure detectors can solve consensus.)

THEOREM 10. *For $1 \leq y \leq t$, $\phi_t^y \not\leq \mathcal{S}$, $\phi_t^y \not\leq \diamond\mathcal{P}$, $\phi_t^y \not\leq \diamond\mathcal{S}$, and $\phi_t^y \not\leq \diamond\mathcal{W}$.*

Proof. The impossibility comes from the fact that nothing prevents the sets $suspected_i$ from containing correct processes for an unbounded amount of time. As $\diamond\mathcal{S} < \diamond\mathcal{P}$ and $\diamond\mathcal{W} < \diamond\mathcal{P}$, it is sufficient to prove it for $\diamond\mathcal{P}$, as far as $\diamond\mathcal{S}$, $\diamond\mathcal{W}$, and $\diamond\mathcal{P}$ are concerned. The proof for $\mathcal{S}$ is verbatim the same as the one for $\diamond\mathcal{P}$ (replacing only $\diamond\mathcal{P}$ by $\mathcal{S}$).

The proof consists in assuming (for contradiction) that there is a protocol transforming a failure detector of the class $\diamond\mathcal{P}$ into a failure detector of the class ϕ_t^y. Let us consider a run where an infinite sequence of relevant queries is issued, all of the form $\text{QUERY}_y(S)$, for the same S, $t - y < |S| \leq t$, and suppose that all processes in S are initially crashed. The answers returned by the protocol define then a sequence consisting of a finite prefix of *false* answers followed by an infinite suffix of *true* an-

swers (by the safety and liveness property of ϕ_t^y). Let τ be a time instant after which all of the invocations of QUERY$_y(S)$ return *true*.

However, it could be that no process ever crashes, and no process in S takes a step until after $\tau + \delta$ (where $\delta > 0$ is an arbitrary finite period), with $\diamond\mathcal{P}$ suspecting each process exactly as in the previous fault-prone run from the very beginning until $\tau + \delta$.

As $\diamond\mathcal{P}$ provides each process with the same outputs in both runs until time $\tau + \delta$, it follows that the queries QUERY$_y(S)$ issued between τ and $\tau + \delta$ returns *true* in both runs. This contradicts the safety property of ϕ_t^y in the failure-free run. □

9. Conclusion. This paper focused on the combination of two approaches to solve the *k-set agreement* problem, namely, failure detectors and conditions. It has proposed novel failure detectors for solving the k-set agreement problem, that, when combined with a condition, establish a new bridge among asynchronous, synchronous, and partially synchronous systems with respect to agreement problems.

The paper has presented three main contributions. The first is the new class of failure detectors denoted ϕ_t^y, $0 \leq y \leq t$. The processes can invoke a primitive QUERY$_y(S)$ with any set S of process identities. Roughly speaking, QUERY$_y(S)$ returns *true* only when all processes in S have crashed, provided $t - y < |S| \leq t$. These failure detectors seem interesting in their own right. They have been thoroughly investigated and compared to the classical failure detectors introduced by Chandra and Toueg.

The second contribution of the paper is a condition-based protocol that solves the k-set agreement problem, with $k = 1 + \max(0, t - (x + y))$, for a condition C of power x and a failure detector of power y, with termination guaranteed for inputs in C. By "power" we mean the following: C is *x-legal* if and only if it can be used to solve x-fault tolerant asynchronous consensus and the failure detector is in the class ϕ_t^y, $0 \leq y \leq t$. Several noteworthy properties and variants of this protocol (that provides a new way to solve asynchronous set agreement and, in particular, consensus) have been studied.

The third contribution is a corresponding lower bound, showing that there is no ϕ_t^y-based k-set agreement protocol for $(t - d)$-legal conditions with $k \leq \max(0, d - y)$. It follows from this lower bound that there is no condition-based k-set agreement protocol such that $k \leq d$ for any $(t - d)$-legal condition.

Appendix A. An adopt–commit-abort object implementation.

As announced in the paper, this appendix describes an implementation of an adopt–commit-abort protocol. The implementation described in Figure 6 is a merge of the one described in [52] (designed for an asynchronous shared memory system) and the one described in [43] designed for an asynchronous message-passing system). It uses two arrays of one-writer multireader atomic registers denoted $PHASE1[1..n]$ and $PHASE2[1..n]$, both initialized to $[\perp, \ldots, \perp]$. Then, an entry of such an array contains a pair or remains equal to $\perp$.

The behavior of a process p_i can be decomposed into three phases.

- Phase 1 (lines 1–2). A process p_i first deposits its input value v_i in $PHASE1[i]$ to make public the fact that v_i has been proposed to the adopt–commit-abort object. Then, it reads (asynchronously) the whole array $PHASE1[1..n]$ to know if other values have been proposed. The local set $set1_i$ is used to keep these values.
- Phase 2 (lines 3–6). During the second phase, if (from its point of view) no value different from its value v_i has been proposed, p_i sets $PHASE2[i]$ to the pair $(single, v_i)$, otherwise it sets $PHASE2[i]$ to the pair $(several, v_i)$. Then, p_i determines how many pairs (x, v) have been deposited in $PHASE2[1..n]$.

Function adopt_commit (v_i)

(1) $PHASE1[i] \leftarrow v_i$;
(2) $set1_i \leftarrow \{v \mid PHASE1[j] = v \ \wedge \ v \neq \perp \ \wedge \ 1 \leq j \leq n\}$;
(3) **if** $(set1_i = \{v_i\})$ **then** $PHASE2[i] \leftarrow (single, \ v_i)$
(4) **else** $PHASE2[i] \leftarrow (several, v_i)$
(5) **end if**;
(6) $set2_i \leftarrow \{(x, v) \mid PHASE2[j] \neq \perp \ \wedge \ PHASE2[j] = (x, v) \ \wedge \ 1 \leq j \leq n\}$;
(7) **case** $set2_i = \{(single, v)\}$ **then return** $(commit, v)$
(8) $set2_i = \{(single, v), (several, v'), \dots\}$ **then return** $(adopt, \ v)$
(9) $(single, v) \notin set2_i$ **then return** $(abort, \ v_i)$
(10) **end case**.

FIG. 6. *A shared memory adopt-commit protocol.*

(Let us recall that we have $PHASE2[k] = \perp$ until p_k deposits a pair in $PHASE2[k]$.) These non-$\perp$ values (pairs) are collected in the set $set2_i$.

- Phase 3 (lines 7–10). Finally, p_i computes the final value it will return as the result of its invocation.
  - If $set2_i$ contains only the pair $(single, v)$, p_i returns $(commit, v)$: it "commits" the value v.
  - If $set2_i$ contains several pairs and one of them is $(single, v)$, then p_i "adopts" that value v by returning $(adopt, v)$.
  - Finally, when $set2_i$ does not contain $(single, v)$, p_i has seen no value to be adopted or committed. It consequently "aborts," returning the value v_i it has initially proposed.

The proof of the termination, validity, and obligation properties of the adopt–commit-abort object are trivial. A proof of the agreement property for the shared memory model can be found in [52]. A proof for a message-passing model can be found in [43] (that proof assumes a majority of correct processes). That proof consists in showing that, for any pair of processes p_i and p_j that execute line 6, we have $set2_i = \{(single, v)\} \Rightarrow (single, v) \in set2_j$ (i.e., line 7 and line 9 are "mutually exclusive").

Acknowledgments, We would like to thank Rachid Guerraoui for interesting questions during PODC 2005 that helped us refine our approach, and Matthieu Roy and Xavier Defago for discussions on the set agreement problem and the implementation of failure detectors. Finally, we want to thank the anonymous referees for their very careful reading and valuable comments.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[2] E. Anceaume, A. Fernandez, A. Mostefaoui, G. Neiger, and M. Raynal, *Necessary and sufficient conditions for transforming limited accuracy failure detectors*, J. Comput. System Sci., 68 (2004), pp. 123–133.

[3] H. Attiya and Z. Avidor, *Wait-free n-set consensus when inputs are restricted*, in Proceedings of the 16th International Symposium on Distributed Computing (DISC'02), Lecture Notes Comput. Sci. 2508, D. Malkhai, ed., Springer-Verlag, New York, 2002, pp. 326–338.

[4] H. Attiya and O. Rachman, *Atomic snapshots in O(n log n) operations*, SIAM J. Comput., 27 (1998), pp. 319–340.

[5] H. Attiya and J. Welch, *Distributed Computing, Fundamentals, Simulation and Advanced Topics*, 2nd ed., Wiley Ser. Parallel Distrib. Comput., Wiley, New York, 2004.

[6] E. Borowsky and E. Gafni, *Generalized FLP impossibility results for t-resilient asynchronous computations*, in Proceedings of the 25th ACM Symposium on the Theory of Computing (STOC'93), ACM Press, 1993, pp. 91–100.

[7] T.D. Chandra, V. Hadzilacos, and S. Toueg, *The weakest failure detector for solving consensus*, J. ACM, 43 (1996), pp. 685–722.

[8] T.D. Chandra and S. Toueg, *Unreliable failure detectors for reliable distributed systems*, J. ACM, 43 (1996), pp. 225–267.

[9] S. Chaudhuri, *More* choices *allow more faults: Set consensus problems in totally asynchronous systems*, Inform. and Comput., 105 (1993), pp. 132–158.

[10] S. Chaudhuri, M. Herlihy, N. Lynch, and M. Tuttle, *Tight bounds for k-set agreement*, J. ACM, 47 (2000), pp. 912–943.

[11] F. Chu, *Reducing Ω to $\diamond\mathcal{W}$*, Inform. Process. Lett., 76 (1998), pp. 293–298.

[12] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg, *The weakest failure detectors to solve certain fundamental problems in distributed computing*, in Proceedings of the 23rd International ACM Symposium on Principles of Distributed Computing (PODC'04), ACM Press, 2004, pp. 338–346.

[13] R. De Prisco, D. Malkahi, and M. Reiter, *On k-set consensus problems in asynchronous systems*, IEEE Trans. Parallel Distrib. Syst., 12 (2001), pp. 7–21.

[14] C. Dwork, N. Lynch, and L. Stockmeyer, *Consensus in the presence of partial synchrony*, J. ACM, 35 (1988), pp. 288–323.

[15] M.J. Fischer and N. Lynch, *A lower bound for the time to assure interactive consistency*, Inform. Process. Lett., 71 (1982), pp. 183–186.

[16] M.J. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[17] R. Friedman, A. Mostefaoui, S. Rajsbaum, and M. Raynal, *Distributed agreement problems and their connection with error-correcting codes*, IEEE Trans. Comput., 56 (2007), pp. 865–875.

[18] R. Friedman, A. Mostefaoui, and M. Raynal, *The notion of veto number for distributed agreement problems*, in Proceedings of the 6th International Workshop on Distributed Computing (IWDC'04), Lecture Notes Comput. Sci. 3326, N. Das et al., eds., Springer-Verlag, New York, 2004, pp. 315–325.

[19] E. Gafni, *Round-by-round fault detectors: Unifying synchrony and asynchrony*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC'00), ACM Press, 1998, pp. 143–152.

[20] E. Gafni, R. Guerraoui, and B. Pochon, *From a static impossibility to an adaptive lower bound: The complexity of early deciding set agreement*, in Proceedings of the 37th ACM Symposium on Theory of Computing (STOC'05), ACM Press, 2005.

[21] R. Guerraoui, *Indulgent algorithms*, in Proceedings of the 19th International ACM Symposium on Principles of Distributed Computing (PODC'00), ACM Press, 2000, pp. 289–297.

[22] R. Guerraoui, *Non-blocking atomic commit in asynchronous systems with failure detectors*, Distrib. Comput., 15 (2002), pp. 17–25.

[23] R. Guerraoui and M. Raynal, *The information structure of indulgent consensus*, IEEE Trans. Comput., 53 (2004), pp. 453–466.

[24] R. Guerraoui and M. Raynal, *The alpha of asynchronous consensus*, Comput. J., 50 (2007), pp. 53–67.

[25] R. Guerraoui and A. Schiper, *Gamma-accurate failure detectors*, in Proceedings of the 10th Workshop on Distributed Algorithms (WDAG'96), Lect. Notes Comput. Sci. 1151, O. Babaoglu and K. Marzullo, eds., Springer-Verlag, New York, 1996, pp. 269–286.

[26] M.P. Herlihy, *Wait-free synchronization*, ACM Trans. Program. Lang. Syst., 11 (1991), pp. 124–149.

[27] M.P. Herlihy and L.D. Penso, *Tight bounds for k-set agreement with limited scope accuracy failure detectors*, Distrib. Comput., 18 (2005), pp. 157–166.

[28] M.P. Herlihy and S. Rajsbaum, *Algebraic spans*, Math. Structures Comput. Sci., 10 (2000), pp. 549–573.

[29] M.P. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J. ACM, 46 (1999), pp. 858–923.

[30] M.P. Herlihy and J.L. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Program. Lang. Syst., 12 (1990), pp. 463–492.

[31] M. Hurfin, A. Mostefaoui, and M. Raynal, *A versatile family of consensus protocols based on Chandra and Toueg's unreliable failure detectors*, IEEE Trans. Comput., 51 (2002), pp. 395–408.

[32] T. Izumi and T. Masuzawa, *Condition adaptation in synchronous consensus*, IEEE Trans. Comput., 55 (2006), pp. 843–853.

[33] L. Lamport and M. Fischer, *Byzantine Generals and Transaction Commit Protocols*, manuscript, 1982.

[34] W.-K. Lo and V. Hadzilacos, *Using failure detectors to solve consensus in asynchronous shared memory systems*, in Proceedings of the 8th International Workshop on Distributed Computing (WDAG'94), Lect. Notes Comput. Sci. 857, G. Tel and P. Vitányi, eds., Springer-Verlag, New York, 1994, pp. 280–295.

[35] A. Mostefaoui, E. Mourgaya, and M. Raynal, *Asynchronous implementation of failure detectors*, in Proceedings of the International IEEE Conference on Dependable Systems and Networks (DSN'03), IEEE Computer Press, 2003, pp. 351–360.

[36] A. Mostefaoui, S. Rajsbaum, and M. Raynal, *Conditions on input vectors for consensus solvability in asynchronous distributed systems*, J. ACM, 50 (2003), pp. 922–954.

[37] A. Mostefaoui, S. Rajsbaum, and M. Raynal, *Synchronous condition-based consensus*, Distrib. Comput., 18 (2006), pp. 325–343.

[38] A. Mostefaoui, S. Rajsbaum, M. Raynal, and C. Travers, *On the computability power and the robustness of set agreement-oriented failure detector classes*, Distrib. Comput., to appear: DOI 10.1007/s00446-008-0064-2, 2008.

[39] A. Mostefaoui, S. Rajsbaum, M. Raynal, and M. Roy, *Condition-based protocols for set agreement problems*, in Proceedings of the 16th International Symposium on Distributed Computing (DISC'02), Lect. Notes Comput. Sci. 2508, D. Malkhai, ed., Springer-Verlag, New York, 2002, pp. 48–62.

[40] A. Mostefaoui, S. Rajsbaum, M. Raynal, and M. Roy, *Condition-based consensus solvability: A hierarchy of conditions and efficient protocols*, Distrib. Comput., 17 (2004), pp. 1–20.

[41] A. Mostefaoui, S. Rajsbaum, M. Raynal, and C. Travers, *From $\diamond\mathcal{W}$ to Ω: A simple bounded quiescent reliable broadcast-based transformation*, J. Parallel Distrib. Comput., 67 (2007), pp. 125–129.

[42] A. Mostefaoui, S. Rajsbaum, M. Raynal, and C. Travers, *The Combined Power of Conditions and Information on Failures to Solve Asynchronous Set Agreement*, Technical report 1897, IRISA, Université de Rennes, Rennes, France, 2008.

[43] A. Mostéfaoui and M. Raynal, *Solving consensus using Chandra and Toueg's unreliable failure detectors: A general quorum-based approach*, in Proceedings of the 13th International Symposium on Distributed Computing (DISC'99), Lect. Notes Comput. Sci. 1693, P. Jayanti, ed., Springer-Verlag, New York, 1999, pp. 49–63.

[44] A. Mostefaoui and M. Raynal, *k-set agreement with limited accuracy failure detectors*, in Proceedings of the 19th International ACM Symposium on Principles of Distributed Computing (PODC'00), ACM Press, 2000, pp. 143–152.

[45] A. Mostefaoui and M. Raynal, *Randomized k-set agreement*, in Proceedings of the 13th International ACM Symposium on Parallel Algorithms and Architectures (SPAA'01), ACM Press, 2001, pp. 291–297.

[46] A. Mostefaoui and M. Raynal, *Leader-based consensus*, Parallel Process. Lett., 11 (2001), pp. 95–107.

[47] G. Neiger, *Failure detectors and the wait-free hierarchy*, in Proceedings of the 14th International ACM Symposium on Principles of Distributed Computing (PODC'95), ACM Press, 1995, pp. 100–109.

[48] Ph. Raïpin Parvédy, M. Raynal, and C. Travers, *Strongly-terminating early-stopping k-set agreement in synchronous systems with general omission failures*, in Proceedings of the 13th Colloquium on Structural Information and Communication Complexity (SIROCCO'06), Lect. Notes Comput. Sci. 4056, P. Flocchini and L. Gasieniec, eds., Springer-Verlag, New York, 2006, pp. 182–196.

[49] M. Raynal, *Consensus in synchronous systems: A concise guided tour*, in Proceedings of the 9th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'02), IEEE Computer Press, 2002, pp. 221–228.

[50] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, SIAM J. Comput., 29 (2000), pp. 1449–1483.

[51] A. Schiper, *Early consensus in an asynchronous system with a weak failure detector*, Distrib. Comput., 10 (1997), pp. 149–157.

[52] J. Yang, G. Neiger, and E. Gafni, *Structured derivations of consensus algorithms for failure detectors*, in Proceedings of the 17th International ACM Symposium on Principles of Distributed Computing (PODC'98), ACM Press, 1998, pp. 297–308.

[53] Y. Zibin, *Condition-based consensus in synchronous systems*, in Proceedings of the 17th International Symposium on Distributed Computing, Lect. Notes Comput. Sci. 2848, F. Fich., ed., Springer-Verlag, New York, 2003, pp. 239–248.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1602–1623

THE PRICE OF STABILITY FOR NETWORK DESIGN WITH FAIR COST ALLOCATION*

ELLIOT ANSHELEVICH†, ANIRBAN DASGUPTA‡, JON KLEINBERG‡, ÉVA TARDOS‡, TOM WEXLER‡, AND TIM ROUGHGARDEN§

Abstract. Network design is a fundamental problem for which it is important to understand the effects of strategic behavior. Given a collection of self-interested agents who want to form a network connecting certain endpoints, the set of *stable* solutions—the Nash equilibria—may look quite different from the centrally enforced optimum. We study the quality of the *best* Nash equilibrium, and refer to the ratio of its cost to the optimum network cost as the *price of stability*. The best Nash equilibrium solution has a natural meaning of stability in this context—it is the optimal solution that can be proposed from which no user will defect. We consider the price of stability for network design with respect to one of the most widely studied protocols for network cost allocation, in which the cost of each edge is divided equally between users whose connections make use of it; this fair-division scheme can be derived from the Shapley value and has a number of basic economic motivations. We show that the price of stability for network design with respect to this fair cost allocation is $O(\log k)$, where k is the number of users, and that a good Nash equilibrium can be achieved via *best-response dynamics* in which users iteratively defect from a starting solution. This establishes that the fair cost allocation protocol is in fact a useful mechanism for inducing strategic behavior to form near-optimal equilibria. We discuss connections to the class of *potential games* defined by Monderer and Shapley, and extend our results to cases in which users are seeking to balance network design costs with latencies in the constructed network, with stronger results when the network has only delays and no construction costs. We also present bounds on the convergence time of best-response dynamics, and discuss extensions to a weighted game.

Key words. network design, price of stability, Shapley cost-sharing

AMS subject classifications. 68Q99, 90B18, 91A43

DOI. 10.1137/070680096

1. Introduction.

In many network settings, the system behavior arises from the actions of a large number of independent agents, each motivated by self-interest and optimizing an individual objective function. As a result, the global performance of the system may not be as good as in a case where a central authority can simply dictate a solution; rather, we need to understand the quality of solutions that are consistent with self-interested behavior. Recent theoretical work has framed this type of question in the following general form: How much worse is the solution quality of a Nash

*Received by the editors January 16, 2007; accepted for publication (in revised form) July 8, 2008; published electronically November 21, 2008. A preliminary version of this paper appeared in the Proceedings of the 45th Annual Symposium on Foundations of Computer Science, 2004.
http://www.siam.org/journals/sicomp/38-4/68009.html

†Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12810 (eanshel@cs.rpi.edu). This author's research was supported by ITR grant 0311333.

‡Department of Computer Science, Cornell University, Upson Hall, Ithaca, NY 14853 (adg@cs.cornell.edu, kleinber@cs.cornell.edu, eva@cs.cornell.edu, wexler@cs.cornell.edu). The second author's research was supported by the Department of Computer Science. The third author's research was supported in part by a David and Lucile Packard Foundation Fellowship and NSF grants 0081334 and 0311333. The fourth author's research was supported in part by NSF grant CCR-032553, ITR grant 0311333, and ONR grant N00014-98-1-0589. The fifth author's research was supported by ITR grant 0311333.

§Computer Science Department, Stanford University, Gates Building, Stanford, CA 94305 (tim@cs.stanford.edu). This author's research was supported in part by ONR grant N00014-04-1-0725, an NSF CAREER Award, and an Alfred P. Sloan Fellowship.

equilibrium,[1] compared to the quality of a centrally enforced optimum? Questions of this genre have received considerable attention in recent years, for problems including routing [37, 39, 13], load balancing [14, 15, 27, 38], and facility location [41]; see [34, Chapters 17–21] for an overview of this literature.

An important issue to explore in this area is the middle ground between centrally enforced solutions and completely unregulated anarchy. In most networking applications, it is not the case that agents are completely unrestricted; rather, they interact with an underlying protocol that essentially proposes a collective solution to all participants, each of which can either accept it or defect from it. As a result, it is in the interest of the protocol designer to seek the *best* Nash equilibrium; this can naturally be viewed as the optimum subject to the constraint that the solution is *stable*, with no agent having an incentive to unilaterally defect from it once it is offered. Hence, one can view the ratio of the solution quality at the best Nash equilibrium relative to the global optimum as a *price of stability*, since it captures the problem of optimization subject to this constraint. Some recent work [3, 13] has considered this definition (termed the "optimistic price of anarchy" in [3]); it stands in contrast to the larger line of work in algorithmic game theory on the *price of anarchy* [35]—the ratio of the *worst* Nash equilibrium to the optimum—which is more suited to worst-case analysis of situations with essentially no protocol mediating interactions among the agents. Indeed, one can view the activity of a protocol designer seeking a good Nash equilibrium as being aligned with the general goals of mechanism design [33]—producing a game that yields good outcomes when players act in their own self-interest.

Network design games. Network design is a natural area in which to explore the price of stability, given the large body of work in the networking literature on methods for sharing the cost of a designed network—often a virtual overlay, multicast tree, or other subnetwork of the Internet—among a collection of participants. (See, e.g. [19, 22] for overviews of work in this area.)

A cost-sharing mechanism can be viewed as the underlying protocol that determines how much a network serving several participants will cost to each of them. Specifically, say that each user i has a pair of nodes (s_i, t_i) that it wishes to connect; it chooses an s_i-t_i path S_i, and the cost-sharing mechanism then charges user i a cost of $C_i(S_1, \ldots, S_k)$. (Note that this cost can depend on the choices of the other users as well.) Although there are in principle many possible cost-sharing mechanisms, research in this area has converged on a few mechanisms with good theoretical and empirical behavior; here we focus on the following particularly natural one: the cost of each edge is shared equally by the set of all users whose paths contain it, so that

$$C_i(S_1, S_2, \ldots, S_k) = \sum_{e \in S_i} \frac{c_e}{|\{j : e \in S_j\}|}.$$

This equal-division mechanism has a number of basic economic motivations; it can be derived from the Shapley value [32], and it can be shown to be the unique cost-sharing scheme satisfying a number of different sets of axioms [19, 22, 32]. For the former reason, we will refer to it as the *Shapley cost-sharing mechanism*. Note that the total edge cost of the designed network is equal to the sum of the costs in the union of all S_i, and the costs allocated to users in the Shapley mechanism completely pay for this total edge cost: $\sum_{i=1}^{k} C_i(S_1, S_2, \ldots, S_k) = \sum_{e \in \cup_i S_i} c_e$.

[1] Recall that a Nash equilibrium is a state of the system in which no agent has an interest in unilaterally changing its own behavior.

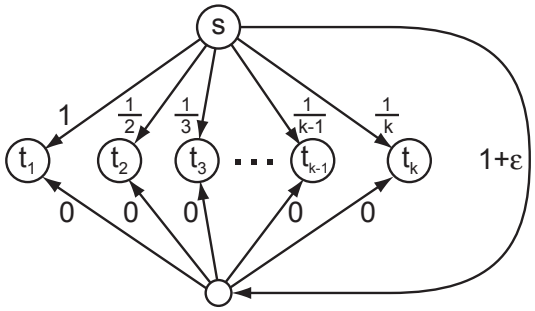


FIG. 1.1. *An instance in which the price of stability converges to* $H(k) = \Theta(\log k)$ *as* $\varepsilon \to 0$.

Now, the general question is to determine how this basic cost-sharing mechanism serves to influence the strategic behavior of the users, and what effect this has on the structure and overall cost of the network one obtains. Given a solution to the network design problem consisting of a vector of paths $(S_1, \ldots, S_k)$ for the k users, user i would be interested in deviating from this solution if there were an alternate s_i-t_i path S_i' such that changing to S_i' would lower its cost under the resulting allocation: $C_i(S_1, \ldots, S_{i-1}, S_i', S_{i+1}, \ldots, S_k) < C_i(S_1, \ldots, S_{i-1}, S_i, S_{i+1}, \ldots, S_k)$. We say that a set of paths is a *Nash equilibrium* if no user has an interest in deviating. As we will see below, there exists a set of paths in Nash equilibrium for every instance of this network design game. (In this paper, we will be concerned only with *pure* Nash equilibria, i.e., with equilibria where each user deterministically chooses a single path.)

The goal of a network design protocol is to suggest for each user i a path S_i so that the resulting set of paths is in Nash equilibrium and its total cost exceeds that of an optimal set of paths by as small a factor as possible; this factor is the *price of stability* of the instance. It is useful at this point to consider a simple example that illustrates how the price of stability can grow to a super-constant value (with k). Suppose k players wish to connect from the common source s to their respective terminals t_i, and assume player i has its own path of cost $1/i$, and all players can share a common path of cost $1 + \varepsilon$ for some small $\varepsilon > 0$ (see Figure 1.1). The optimal solution would connect all agents through the common path for a total cost of $1 + \varepsilon$. However, if this solution were offered to the users, they would defect from it one by one to their alternative paths. The unique Nash equilibrium has a cost of $\sum_{i=1}^{k} 1/i = H(k)$.

While the price of stability in this instance grows with k, it only does so logarithmically. It is thus natural to ask how large the price of stability can be for this network design problem. If we think about the example in Figure 1.1 further, it is also interesting to note that a Nash equilibrium is reached by players taking turns updating their paths (in other words, best-response dynamics) starting from an optimal solution; it is natural to ask to what extent this holds in general.

Our results. Our first main result is that in every instance of the network design problem with Shapley cost-sharing, there always exists a Nash equilibrium of total cost at most $H(k)$ times optimal. In other words, the simple example in Figure 1.1 is in fact the worst possible case.

We prove this result using a *potential function* method due to Rosenthal [36] (based on [6]) and later generalized by Monderer and Shapley [30]: One defines a potential function Φ on possible solutions and shows that every improving move of one of the users (to lower its own cost) reduces the value of Φ. Since the set of possible solutions is finite, it follows that every sequence of improving moves leads to a Nash

equilibrium. The goal of Monderer's and Shapley's and Rosenthal's work was to prove existence statements of this sort; for our purposes, we make further use of the potential function to prove a bound on the price of stability. Specifically, we give bounds relating the value of the potential for a given solution to the overall cost of that solution; if we then iterate using best-response dynamics starting from an optimal solution, the potential does not increase, and hence we can bound the cost of any solution that we reach. Thus, for this network design game, best-response dynamics starting from the optimum do in fact always lead to a good Nash equilibrium.

We can extend our basic result to a number of more general settings. To begin with, the $H(k)$ bound on the price of stability extends directly to the case in which users are selecting arbitrary subsets of a ground set (with elements' costs shared according to the Shapley value), rather than paths in a graph; it also extends to the case in which the cost of each edge is a nondecreasing concave function of the number of users on it. In addition, our results also hold if we introduce capacities into our model; each edge e may be used by at most u_e players, where u_e is the capacity of e.

We arrive at a more technically involved set of extensions if we wish to add latencies to the network design problem. Here each edge has a concave *construction cost* $c_e(x)$ when there are x users on the edge, and a *latency cost* $d_e(x)$; the cost experienced by a user is the full latency plus a fair share of the construction cost, $d_e(x) + c_e(x)/x$. We give general conditions on the latency functions that allow us to bound the price of stability in this case by $d \cdot H(k)$, where d depends on the delay functions used. Moreover, we obtain stronger bounds in the case where users experience only delays, not construction costs; this includes a result that relates the cost of a best Nash equilibrium to that of an optimum with twice as many players, and a result that improves the potential-based bound on the price of stability for the single-source, delay-only case.

Since a number of our proofs are obtained by following the results of best-response dynamics via a potential function, it is natural to investigate the speed of convergence of best-response dynamics for this game. We show that with k players, it can run for a time exponential in k. Whether there is a way to schedule players' moves to make best-response dynamics converge in a polynomial number of steps for this game in general is an interesting open question.

Finally, we consider a natural generalization of the cost-sharing model that carries us beyond the potential-function framework. Specifically, suppose each user has a *weight* (perhaps corresponding to the amount of traffic it plans to send), and we change the cost-allocation so that user i's payment for edge e is equal to the ratio of its weight to the total weight of all users on e. In addition to being intuitively natural, this definition is analogous to certain natural generalizations of the Shapley value [29]. The weighted model, however, is significantly more complicated: There is no longer a potential function whose value tracks improvements in users' costs when they greedily update their solutions. We also show, using a construction involving user weights that grow exponentially in k, that the price of stability can be as high as $\Omega(k)$. We have obtained some initial positive results here, including the convergence of best-response dynamics when all users seek to construct a path from a node s to a node t (the price of stability here is 1), and in the general model of users selecting sets from a ground set, where each element appears in the sets of at most two users.

Related work. Network design games under a different model were considered by a subset of the authors in [3]; there the setting was much more "unregulated" in that users could offer to pay for an arbitrary fraction of any edge in the network. This model resulted in instances where no pure Nash equilibrium existed, and in many cases in

[3] when pure Nash equilibria did exist, certain users were able to act as "free riders," paying very little or nothing at all. The present model, on the other hand, ensures that there is always a pure Nash equilibrium within a logarithmic factor of optimal, in which users pay for a fair portion of the resources they use. Network creation games of a fairly different flavor—in which users correspond to nodes, and can build subsets of the edges incident to them—have been considered in [2, 12, 5, 16, 21, 31]. The model in this paper associates users instead with connection requests, and allows them to contribute to the cost of any edge that helps them to form a path that they need.

The bulk of the work on cost-sharing (see, e.g., [19, 22] and the references there) tends to assume a fixed underlying set of edges. Jain and Vazirani [23] and Kent and Skorin-Kapov [26] consider cost-sharing for a single source network design game. Cost-sharing games assume that there is a central authority that designs and maintains the network, and decides appropriate cost-shares for each agent, depending on the graph and all other agents, via a complex algorithm. The agents' only role is to report their utility for being included in the network.

Here, on the other hand, we consider a simple cost-sharing mechanism, the Shapley value, and ask what the strategic implications of a given cost-sharing mechanism are for the way in which a network will be designed. This question explores the feedback between the protocol that governs network construction and the behavior of self-interested agents that interact with this protocol. An approach of a similar style, though in a different setting, was pursued by Johari and Tsitsiklis [24]; there, they assumed a network protocol that priced traffic according to a scheme due to Kelly [25], and asked how this protocol would affect the strategic decisions of self-interested agents routing connections in the network.

The special case of our game with only delays is closely related to the congestion games of [39, 37]. They consider a game where the amount of flow carried by an individual user is infinitesimally small (a *nonatomic game*), while in this paper we assume that each user has a unit of flow, which it needs to route on a single path. In the nonatomic game of [39, 37] the Nash equilibrium is essentially unique (hence there is no distinction between the price of anarchy and stability), while in our atomic game there can be many equilibria. Fabrikant, Papadimitriou, and Talwar [17] consider our atomic game with delays only. They give a polynomial time algorithm to minimize the potential function Φ in the case that all users share a common source, and show that finding any equilibrium solution is PLS-complete for multiple source-sink pairs. Our results extend the price of anarchy results of [39, 37] about nonatomic games to results on the price of stability for the case of single source atomic games. Subsequent to our work, further results on the price of anarchy and stability in atomic games with delays were obtained in [4, 7, 11, 10, 40]. For games without delays, Agarwal and Charikar [1] give improved bounds on the price of stability in single-source undirected networks, and Fiat et al. [20] give bounds with the additional assumption that each vertex is the destination of some player. Other aspects of these and closely related games were recently explored in [9, 18].

A weighted game similar to ours is presented by Libman and Orda [28], with a different mechanism for distributing costs among users. They do not consider the price of stability and instead focus on convergence in parallel networks. Recently, Chen and Roughgarden [8] proved general results on the price of stability and the existence of approximate Nash equilibria in the weighted version of our game.

2. Nash equilibria of network design with Shapley cost-sharing. In this section we consider the *Fair Connection Game* for k players as defined in the Introduc-

tion. Let a directed graph $G = (V, E)$ be given, with each edge having a nonnegative cost c_e. Each player i has a set of terminal nodes T_i that he wants to connect. A strategy of player i is a set of edges $S_i \subset E$ such that S_i connects all nodes in T_i. We assume that we use the Shapley value to share the cost of the edges; i.e., all players using an edge split up the cost of the edge equally. Given a vector of players' strategies $S = (S_1, \ldots, S_k)$, let x_e be the number of agents whose strategy contains edge e. Then the cost to agent i is $C_i(S) = \sum_{e \in S_i}(c_e/x_e)$, and the goal of each agent is to connect its terminals with minimum total cost.

In the worst case, Nash equilibria can be very expensive in this game, so that the price of anarchy becomes as large as k. To see this, consider k players with common source s and sink t, and two parallel edges of cost 1 and k. The worst equilibrium has all players selecting the more expensive edge, thereby paying k times the cost of the optimal network. However, we can bound the price of stability by $H(k)$, which is the harmonic sum $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$, as follows.

THEOREM 2.1. *The price of stability of the fair connection game is at most* $H(k)$.

Proof. The fair connection game that we have defined falls into the class of congestion games as defined by Rosenthal [36], as the cost of edge e to a user i is $f_e(x) = c_e/x$, which depends only on edge e and the number of users x whose strategy contains e. Rosenthal [36] shows that all congestion games have deterministic Nash equilibria. He proves this using a potential function Φ, defined as follows.

$$\Phi(S) = \sum_{e \in E} \sum_{x=1}^{x_e} f_e(x) \tag{2.1}$$

Rosenthal [36] shows that for any strategy $S = (S_1, \ldots, S_k)$ if a single player i deviates to strategy S_i', then the change in the potential value $\Phi(S) - \Phi(S')$ of the new strategy set $S' = (S_1, \ldots, S_i', \ldots, S_k)$ is exactly the change in the cost to player i. Note that the change of player i's strategy affects the cost of many other players $j \neq i$, but the value of Φ is not affected by the change in the cost of these players, it simply tracks the cost of the player who changes its strategy. Monderer and Shapley [30] call a game in which such a function Φ exists a *potential game*. To show that such a potential game has a Nash equilibrium, start from any state $S = (S_1, \ldots, S_k)$ and consider a sequence of selfish moves (allowing players to change strategies to improve their costs). In a congestion game any sequence of such improving moves leads to a Nash equilibrium as each move decreases the potential function Φ, and hence must lead to a stable state.

Neither Rosenthal nor Monderer and Shapley say anything about the quality of Nash equilibria with respect to the centralized optimum, but we can use their potential function to establish our bound. Let x_e be defined as above with respect to S. Now the potential function of (2.1) in our case is $\Phi(S) = \sum_{e \in E} c_e H(x_e)$. Consider the strategy $S^* = (S_1^*, \ldots, S_k^*)$ defining the optimal centralized solution. Let $OPT = \sum_{e \in S^*} c_e$ be the cost of this solution. Then, $\Phi(S^*) \leq \sum_{e \in S^*}(c_e \cdot H(k))$, which is exactly $H(k) \cdot OPT$. Now we start from strategy S^* and follow a sequence of improving self-interested moves. We know that this will result in a Nash equilibrium S with $\Phi(S) \leq \Phi(S^*)$.

Note that the potential value of any solution S is at least the total cost: $\Phi(S) \geq \sum_{e \in S} c_e = cost(S)$. Therefore, there exists a Nash equilibrium with cost at most $H(k) \cdot OPT$, as desired. ∎

Recall from the example in Figure 1.1 that the upper bound of Theorem 2.1 is tight.

Unfortunately, even though Theorem 2.1 says that cheap Nash equilibria exist, finding them is NP-complete.

THEOREM 2.2. *Given an instance of a fair connection game and a value C, it is NP-hard to determine if the game has a Nash equilibrium of cost at most C.*

Proof. The reduction is from 3D matching. Given an instance of 3D matching with node sets X, Y, Z, form a graph $G = (V, E)$ as follows. Form a node for each node in X, Y, and Z, and also a node $v_{i,j,k}$ for each 3D edge (x_i, y_j, z_k). Also add an additional node t. Form a directed edge from each node $v_{i,j,k}$ to t with cost function $c_e = 3$. Form a directed edge from each node v in X, Y, Z to all nodes representing 3D edges that contain v. Make these edges have a cost $c_e = 0$. Let $C = |X| + |Y| + |Z|$, and form a player for each node v in $X \cup Y \cup Z$. This player has two terminals: v and t.

If there exists a 3D matching in the 3D-matching instance, then there exists a Nash equilibrium in the above fair connection game of cost C: Take the 3D matching M, and let S_i for the player whose terminals are v and t be the edge from v to the unique node $v_{i,j,k}$ corresponding to the 3D edge in M, and the edge from this node to t. Since M is a matching, the cost of S is exactly $3C/3 = C$. S is a Nash equilibrium, since any deviation for a player involves paying for some edge of cost 3 by himself, while the current amount he is paying is 1.

If no 3D matching exists, then any solution to the fair connection game must cost more than C. Therefore, no Nash equilibrium can exist of cost at most C. This finishes the proof. □

Notice that the same proof shows that determining if there exists a Nash equilibrium that costs as little as OPT is NP-complete.

We can extend the results of Theorem 2.1 to concave cost functions. Consider the extended fair connection game where instead of a constant cost c_e, each edge has a cost which depends on the number of players using that edge, $c_e(x)$. We assume that $c_e(x)$ is a nondecreasing, concave function, modeling the buy-at-bulk economies of scale of buying edges that can be used by more players. Notice that the cost of an edge $c_e(x)$ might increase with the number of players using it, but the cost per player $f_e(x) = c_e(x)/x$ decreases if $c_e(x)$ is concave.

THEOREM 2.3. *Take a fair connection game with each edge having a nondecreasing concave cost function $c_e(x)$, where x is the number of players using edge e. Then the price of stability is at most $H(k)$.*

Proof. The proof is analogous to the proof of Theorem 2.1. We use the potential function $\Phi(S)$ defined by (2.1). As before, the change in potential if a player i deviates equals exactly the change of that player's payments. We start with the strategy S^* with minimum total cost, and perform a series of improving deviations until we reach a Nash equilibrium S with $\Phi(S) \leq \Phi(S^*)$. To finish the proof all we need to show is that $cost(S) \leq \Phi(S) \leq H(k) \cdot cost(S)$ for all strategies S. The second inequality follows since $c_e(x)$ is nondecreasing, and therefore $\sum_{x=1}^{x_e}(c_e(x)/x) \leq H(x_e) \cdot c_e(x_e)$. To see that $cost(S) \leq \Phi(S)$ notice that since $c_e(x)$ is concave, the cost per player must decrease with x; i.e., $c_e(x)/x$ is a nonincreasing function. Therefore, $cost(S) = \sum_{e \in S} c_e(x_e) = \sum_{e \in S} x_e \cdot (c_e(x_e)/x_e) \leq \Phi(S)$, which finishes the proof. □

Notice that the requirement of cost functions being concave is general enough to encompass the utility function of a player being a combination of the cost he has to pay for his edges and the distance between his terminals in the network of bought edges. If c_e is the cost function of an edge, we simply set $c'_e(x) = c_e(x) + x$. The

payment of each player i now becomes $|S_i| + \sum_{e \in S_i}(c_e(x_e)/x_e)$, and c'_e is still concave if c_e is concave.

Extensions. The proof of Theorem 2.3 extends to a general congestion game, where players attempt to share a set of resources R that they need. Instead of having an underlying graph structure, we now think of each $s \in R$ as a resource with a concave cost function $c_s(x)$ of the number of users selecting sets containing s. The possible strategies of each player i is a set $\mathcal{S}_i$ of subsets of R. Each player seeks to select a set $S_i \in \mathcal{S}_i$ so as to minimize his cost. Since the proofs above did not rely on the graph structure, they translate directly to this extension.

We can further extend the results to the case where the cost of a player is a combination of the cost $c_e(x)/x$, and a function of the selected set, such as the distance between terminals in the network design case. More precisely, the price of stability is still at most $H(k)$ if each player is trying to minimize the cost $\sum_{e \in S_i}(c_e(x_e)/x_e) + d_i(S_i)$, where c_e is monotone increasing and concave, and d_i is an arbitrary function specific to player i (e.g., a distance function, or diameter of S_i, etc.). The proof is analogous to Theorem 2.3, except with a new potential $\Phi(S) = \sum_i d_i(S_i) + \sum_{e \in S} \sum_{x=1}^{x=x_e}(c_e(x)/x)$. Notice that this is technically not a congestion game on the given graph G. Finally we note that all these results (as well as those subsequent) hold in the presence of capacities. Adding capacities u_e to each edge e and disallowing more than u_e players to use e at any time does not substantially alter any of our proofs.

The case of undirected graphs. While the bound of $H(k)$ is tight for general directed graphs, it is not tight for undirected graphs. Finding the correct bound is an interesting open problem; see [20] for some recent progress. In the case of two players, our bound on the price of stability is $H(2) = 3/2$. In section 4 we show that this bound can be improved to $4/3$ in the case of two players and a single source. We also give an example to show that this bound is tight.

3. Dealing with delays. In most of the previous section, we assumed that the utility of a player depends only on the cost of the edges he uses. What changes if we introduce latency into the picture? We have extended this to the case when the players' cost is a combination of "design" cost and the length of the path selected. More generally, delay on an edge does not have to be simply the "hop-count," but can also depend on congestion, i.e., on the number of players using the edge. In this section we will consider such a model.

Assume that each edge has both a cost function $c_e(x)$ and a latency function $d_e(x)$, where $c_e(x)$ is the cost of building the edge e for x users and the users will share this cost equally, while $d_e(x)$ is the delay suffered by users on edge e if x users are sharing the edge. The goal of each user will be to minimize the sum of his cost and his latency. If we assume that both the cost and latency for each edge depend only on the number of players using that edge, then this fits directly into our model of a congestion game above: the total cost felt by each user on the edge is $f_e(x) = c_e(x)/x + d_e(x)$. If the function $xf_e(x)$ is concave, then Theorem 2.3 applies. But while concave functions are natural for modeling cost, latency tends to be convex.

3.1. Combining costs and delays. First, we extend the argument in the proof of Theorem 2.3 to general functions f_e. The most general version of this argument is expressed in the following theorem.

THEOREM 3.1. *Consider a fair connection game with arbitrary edge-cost functions f_e. Suppose that $\Phi(S)$ is as in (2.1), with $cost(S) \leq A \cdot \Phi(S)$, and $\Phi(S) \leq B \cdot cost(S)$ for all S. Then, the price of stability is at most $A \cdot B$.*

Proof. Let S^* be a strategy such that S_i^* is the set of edges i used in the centralized optimal solution. We know from above that if we perform a series of improving deviations on it, we must converge to a Nash equilibrium S' with potential value at most $\Phi(S^*)$. By our assumptions, $cost(S') \leq A \cdot \Phi(S') \leq A \cdot \Phi(S^*) \leq AB \cdot cost(S^*) = AB \cdot OPT$. □

Our main interest in this section are functions $f_e(x)$ that are the sums of the fair share of a cost and a delay, i.e., $f_e(x) = c_e(x)/x + d_e(x)$. We will assume that $d_e(x)$ is monotone increasing, while $c_e(x)$ is monotone increasing and concave.

COROLLARY 3.2. *If $c_e(x)$ is concave and nondecreasing, $d_e(x)$ is nondecreasing for all e, and $x_e d_e(x_e) \leq A \sum_{x=1}^{x_e} d_e(x)$ for all e and x_e, then the price of stability is at most $A \cdot H(k)$. In particular, if $d_e(x)$ is a polynomial with degree at most l and nonnegative coefficients, then the price of stability is at most $(l+1) \cdot H(k)$.*

Proof. For functions $f_e(x) = c_e(x)/x + d_e(x)$, both the cost and potential of a solution come in two parts corresponding to cost c and delay d.

For the part corresponding to the cost, the potential overestimates the cost by at most a factor of $H(k)$ as proved in Theorem 2.3. If on the delay, the potential underestimates the cost by at most a factor of A, then we get the bound of $A \cdot H(k)$ for the price of stability by Theorem 3.1. □

Therefore, for reasonable delay functions, the price of stability cannot be too large. In particular, if the utility function of each player depends on a concave cost and delay that is independent of the number of users on the edge, then we get that the price of stability is at most $H(k)$ as we have shown at the end of the previous section. If the delay grows linearly with the number of users, then the price of stability is at most $2H(k)$.

3.2. Games with only delays. In this subsection we consider games with only delay. We assume that the cost of a player for using an edge e used by x players is $f_e(x) = d_e(x)$, and d_e is a monotone increasing function of x. This cost function models delays that are increasing with congestion.

We will mostly consider the special case when there is a common source s. Each player i has one additional terminal t_i, and the player wants to connect s to t_i via a directed path. Fabrikant, Papadimitriou, and Talwar [17] showed that in this case, one can compute the Nash equilibrium minimizing the potential function Φ via a minimum cost flow computation. For each edge e they introduce many parallel copies, each with capacity 1, and cost $d_e(x)$ for integers $x > 0$. We will use properties of a minimum cost flow for establishing our results.

3.2.1. A bicriteria result. First we show a bicriteria bound, and compare the cost of the cheapest Nash equilibrium to that of the optimum solution with twice as many players.

THEOREM 3.3. *Consider the single source case of a congestion game with only delays. Let S be the minimum cost Nash equilibrium and S^* be the minimum cost solution for the problem where each player i is replaced by two players. Then $cost(S) \leq cost(S^*)$.*

Proof. Consider the Nash equilibrium obtained by Fabrikant [17] via a minimum cost flow computation. Assume that x_e is the number of users using edge e at this equilibrium. By assumption, all users share a common source s. Let $D(v)$ denote the cost of the minimum cost path in the residual graph from s to v. The length of the path of user i is at most $D(t_i)$ (as otherwise the residual graph would have a negative cycle), and hence we get that $cost(S) \leq \sum_i D(t_i)$.

Now consider modified delay function $\hat{d}_e$ for each edge $e = (u, v)$. Define $\hat{d}_e(x) = d_e(x)$ if $x > x_e$, and $\hat{d}_e(x) = D(v) - D(u)$ if $x \leq x_e$. Note that for any edge e we have $D(v) - D(u) \leq d_e(x_e + 1)$ as edge $e = (u, v)$ is in the residual graph with cost $d_e(x_e + 1)$. This implies that the modified delay $\hat{d}$ is monotone. For edges with $x_e \neq 0$ we also have that $d_e(x_e) \leq D(v) - D(u)$ as the reverse edge (v, u) is in the residual graph with cost $-d_e(x_e)$, so the delay of an edge is not decreased.

Now observe that, subject to new delay $\hat{d}$, the shortest path from s to t_i has length $D(t_i)$. The minimum possible cost of two paths from s to t_i for the two users corresponding to user i is then at least $2D(t_i)$ for each player i. Therefore the minimum cost of a solution with delays $\hat{d}$ is at least $2\sum_i D(t_i)$.

To bound $cost(S^*)$ we need to bound the difference in cost of a solution when measured with delays $\hat{d}$ and d. Note that for any edge $e = (u, v)$ and any number x we have that $x\hat{d}_e(x) - xd_e(x) \leq x_e(D(v) - D(u))$, and hence the difference in total cost is at most $\sum_{e=(u,v)} x_e(D(v) - D(u)) = \sum_i D(t_i)$. Using this, we get that $cost(S^*) \geq \sum_i D(t_i) \geq cost(S)$. □

Note that a similar bound is not possible for a model with both costs and delays, when additional users compensate to some extent for the price of stability. Consider a problem with two parallel links e and e' and k users. Assume on link e the cost function is $c_e(x) = 1 + \varepsilon$ for a small $\varepsilon > 0$, and the latency function is $d_e(x) = 0$. On the other link e' the cost is $c_{e'}(x) = 0$, and the delay with x users is $d_{e'}(x) = 1/(k - x + 1)$. The optimum solution is to use the first edge e, and it costs $1 + \varepsilon$. Note that the optimum with any number of extra users costs the same. On the other hand, the only Nash equilibrium is to have all users on e', incurring delay 1, for a total cost of k.

3.2.2. Bounding the price of stability with only delays. Note that the $H(k)$ term in Corollary 3.2 comes from the concave cost c, and so the bound obtained there improves by an $H(k)$ factor when the cost consists only of the delay. The results from Corollary 3.2 already tell us that if the delay functions are such that $x_e d_e(x_e) \leq A\sum_{x=1}^{x_e} d_e(x)$, the the price of stability is at most A. Specifically, we know that if the delays are polynomial of degree l, then the price of stability is at most $l + 1$, and therefore with linear delays the price of stability is at most 2.

Roughgarden [37] showed a tighter bound for nonatomic games. He assumed that the delay is monotone increasing, and the total cost of an edge $xd_e(x)$ is a convex function of traffic x. He showed that for any class of such functions $\mathcal{D}$ containing all constant functions, the price of anarchy is always obtained on a two-node, two-link network. Let us call $\alpha(\mathcal{D})$ the price of anarchy for nonatomic games with delays from the class $\mathcal{D}$ (which is also the price of stability, since the Nash equilibrium is unique in that context). For example, Roughgarden [37] showed that for polynomials of degree at most l this bound is $O(l/\log l)$, and for linear delays it is 4/3. Here we extend this result to a single source atomic game, and thereby show tighter bounds than in Corollary 3.2 for the single source case.

THEOREM 3.4. *If in a single source fair connection game all costs are delays, and all delays are from a set $\mathcal{D}$ satisfying the above condition, then the price of stability is at most $\alpha(D)$.*

Proof. As in the proof of Theorem 3.3 consider the Nash equilibrium obtained via a minimum cost flow computation, and let $D(v)$ be the length of the shortest path from s to v in the residual graph. As before we have that $cost(S) \leq \sum_i D(t_i)$. Further, for each edge $e = (u, v)$ we have that $D(v) - D(u) \leq d_e(x_e + 1)$, and for edges with $x_e \neq 0$, we also have that $d_e(x_e) \leq D(v) - D(u)$.

Give each edge $e = (u, v)$ a capacity of x_e, and augment our network by adding a parallel edge e' with constant delay $D(v) - D(u)$. Let $\hat{G}$ denote the resulting network flow problem. Note that the new capacity and the added links do not affect the equilibrium, as $d_e(x_e) \leq D(v) - D(u)$. For each edge e, the two parallel copies: Edge e with new capacity x_e and edge e' can carry any number of paths at least as cheaply as the original edge e could since $D(v) - D(u) \leq d_e(x_e + 1)$; hence this change in the network can only improve the minimum possible cost. We will prove the bound in this new network by comparing the cost of the Nash equilibrium with the minimum possible cost of a (possibly fractional) flow carrying one unit of flow from s to each of the terminals t_i.

The nice property of $\hat{G}$ is that the optimum fractional flow $\hat{x}$ in $\hat{G}$ is easy to determine. Consider an edge $e = (u, v)$ that is used by $x_e \neq 0$ paths in the equilibrium. We will obtain a fractional flow $\hat{x}_e$ by splitting the corresponding x_e amount of flow between the two edges e and e'. For an edge e let $\ell_e(x) = d_e(x) + xd'_e(x)$. By assumption, $d_e(x) \leq \ell_e(x)$ for all x. For an edge e such that $\ell_e(x_e) \leq D(v) - D(u)$, we set $\hat{x}_e = x_e$, and $\hat{x}_{e'} = 0$. Otherwise, let $\hat{x}_e$ be such that $\ell_e(\hat{x}_e) = D(v) - D(u)$, and let $\hat{x}_{e'} = x_e - \hat{x}_e$.

First, we claim that $\hat{x}$ is the minimum cost fractional solution in $\hat{G}$. For all edges $e = (u, v)$ such that $\hat{x}_e \neq x_e$, we have that $\ell_e(\hat{x}_e) = D(v) - D(u)$. When $\hat{x}_e = x_e$, then we have that flow $\hat{x}_e$ is equal to the capacity of the edge, and $\ell_e(\hat{x}_e) \leq D(v) - D(u)$. Therefore, if there is a negative cycle in the residual graph of $\hat{x}_e$ with constant edge costs $\ell_e(x_e)$ for e and costs $D(v) - D(u)$ for e', then this is also a negative cost cycle in G with constant edge costs $D(v) - D(u)$. This is impossible, however, since x_e is a min-cost flow with those costs. We can now use Lemma 3.5 to see that $\hat{x}_e$ is also a min-cost flow for edge costs $xd_e(x)$.

The theorem then follows, as on each original edge $e \in E$ the cost $x_e d_e(x_e)$ is at most $\alpha(\mathcal{D})$ times the cost of the corresponding two edges e and e' in $\hat{G}$ by Lemma 3.6. □

To finish the proof of the theorem, we require the following lemmas.

LEMMA 3.5. *Let G be a network, and x_e be a fractional flow sending one unit of flow from the source s to each sink t_i. Let ℓ denote the gradient of the total cost $xd_e(x)$, that is, let $\ell_e(x) = d_e(x) + xd'_e(x)$ for each edge e. The flow x_e is minimum cost subject to the cost $\sum_e xd_e(x)$ if and only if it is a minimum cost flow subject to the constant cost function $c_e = \ell_e(x_e)$.*

Proof. If the flow x_e is not of minimum cost subject to costs c_e, then the residual graph has a negative cycle, and moving a small amount of flow along the cycle decreases the cost $\sum_e xd_e(x)$, as the cost c_e is exactly the gradient of this objective function. To see the other direction, we use the fact that the cost function is convex by assumption, and hence all local optima are also global optima. □

Next, it is useful to recall from [37] what is $\alpha(\mathcal{D})$. Consider edge e, with delay $d(x)$ from class $\mathcal{D}$. Now consider a graph with two parallel links: Edge e, which has delay $d(x)$, that will carry some r units of flow, and a parallel link e' with constant delay $d(r)$ independent of the traffic. Now the unique Nash equilibrium is to route all r units of flow on e, while we get the optimum by setting x such that the gradient $c(x) = d(x) + xd'(x)$ is equal to $d(r)$, and sending x units of flow along e, and the remainder $r - x$ along edge e'. This is because of the following lemma from [37].

LEMMA 3.6 ([37]). *If a set $\mathcal{D}$ of delay functions satisfies the above condition, then the price of stability is at most $\alpha(D) = \max_{r,x,d\in\mathcal{D}} rd(r)/(xd(x) + (r - x)d(r))$, and the maximum is achieved by setting x such that $d(x) + xd'(x) = d(r)$.*

4. The undirected case. While the bound of $H(k)$ on the price of stability is tight for general directed graphs with costs, it is not tight for undirected graphs. Finding the correct bound is an interesting open problem. In the case of two players, our bound on the price of stability is $H(2) = 3/2$. We now show that that this bound can be improved to 4/3 in the case of two players and a single source.

Here is an example of an undirected two-player game with the price of stability approaching 4/3. Let G have 3 nodes: s, t_1, and t_2. Player 1 wants to connect t_1 with s, and player 2 wants to connect t_2 with s. There are edges (s, t_1) and (s, t_2) with cost 2. There is an edge (t_1, t_2) with cost $1 + \varepsilon$. The optimal centralized solution has cost $3 + \varepsilon$. However, the cheapest Nash has cost 4. This example implies that the following claim is tight.

CLAIM 4.1. *The price of stability is at most* 4/3 *in a fair connection game with two players in an undirected graph, each having two terminals with one terminal in common.*

Proof. Let s be the common terminal, and let t_1 and t_2 be the personal terminals. Consider the optimal centralized solution (S_1, S_2). Let $X_1 = S_1 \backslash S_2$ be the edges only being used by player 1, $X_2 = S_2 \backslash S_1$ be the edges only used by player 2, and $X_3 = S_1 \cap S_2$ be the edges shared by the two players. Let (S_1', S_2') be a Nash equilibrium that a series of improving responses converges to starting with (S_1, S_2). Similarly, let $Y_1 = S_1' \backslash S_2'$, $Y_2 = S_2' \backslash S_1'$, and $Y_3 = S_1' \cap S_2'$. Finally, set $x_i = cost(X_i)$ and $y_i = cost(Y_i)$ for $1 \le i \le 3$. By the properties of $\Phi(S_1, S_2)$ from the proof of Theorem 2.1, we know that $\Phi(S_1', S_2') \le \Phi(S_1, S_2)$. Substituting in the definition of Φ, we obtain that

$$y_1 + y_2 + \frac{3}{2} y_3 \le x_1 + x_2 + \frac{3}{2} x_3. \tag{4.1}$$

Look at S_1' and S_2' as paths instead of sets of edges (there will be no cycles since then this would not be a Nash). We now show that in (S_1', S_2'), as in any Nash equilibrium, once the paths of the two players merge, they do not separate again. Suppose to the contrary that this happens. Let v be the first node that S_1' and S_2' have in common, and set P_1 and P_2 be the subpaths of S_1' and S_2' after v, respectively. We know that $cost(P_1 \backslash P_2) = cost(P_2 \backslash P_1)$, since if they were not equal, say $cost(P_1 \backslash P_2) > cost(P_2 \backslash P_1)$, then player 1 could deviate to P_2 instead and pay strictly less. However, even if they are equal, player 1 could deviate to use P_2 instead of P_1, and pay strictly less, since he will pay the same as before on edges in $P_1 \cap P_2$, and pay only $cost(P_1 \backslash P_2)/2$ in total on the other edges. Therefore, the only way this could be a Nash equilibrium is if $P_1 \cap P_2 = P_1 = P_2$, as desired.

Consider a deviation from (S_1', S_2') that player 1 could make. He could decide to use $X_1 \cup X_2 \cup Y_2 \cup Y_3$ instead of $S_1' = Y_1 \cup Y_3$. This is a valid deviation because player 1 still connects his terminals by following X_1 until X_1 meets with X_2, then following X_2 back to t_2, and then following S_2' to s. Since (S_1', S_2') is a Nash equilibrium, this deviation must cost more to player 1 than his current payments, and so $x_1 + x_2 + y_2/2 + y_3/2 \ge y_1 + y_3/2$. By symmetric reasoning, $x_1 + x_2 + y_1/2 + y_3/2 \ge y_2 + y_3/2$. If we add these inequalities together, we obtain that

$$y_1/2 + y_2/2 \le 2x_1 + 2x_2. \tag{4.2}$$

To show that the price of stability is at most 4/3, it is enough to show that $cost(S_1', S_2') \le \frac{4}{3} cost(S_1, S_2)$. Using the above notation, this is the same as showing $3y_1 + 3y_2 + 3y_3 \le 4x_1 + 4x_2 + 4x_3$. We do this by using Inequalities 4.1 and 4.2 as

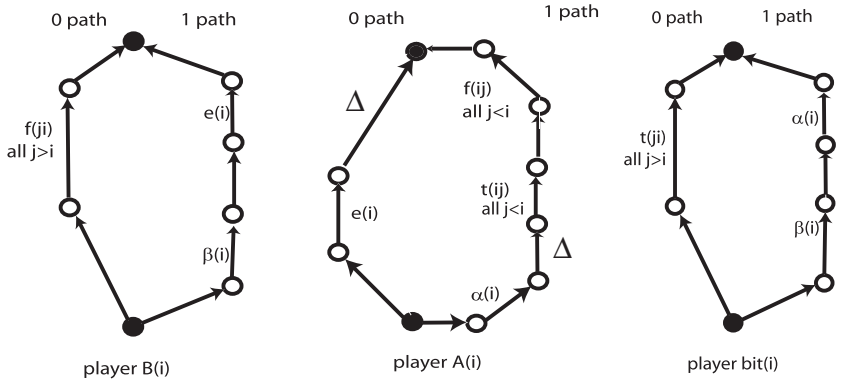


FIG. 5.1. *The construction of an exponential best-response run. The filled-in nodes are the sources and sinks of the players.*

follows:

$$\begin{aligned} 3y_1 + 3y_2 + 3y_3 &\leq 3y_1 + 3y_2 + 4y_3 \\ &= \frac{1}{3}(y_1 + y_2) + \frac{8}{3}\left(y_1 + y_2 + \frac{3}{2}y_3\right) \\ &\leq \frac{4}{3}(x_1 + x_2) + \frac{8}{3}\left(x_1 + x_2 + \frac{3}{2}x_3\right) \\ &= 4x_1 + 4x_2 + 4x_3. \qquad \square \end{aligned}$$

5. Convergence of best response. In this section, we show that, in general, best response dynamics in our game can take a long time to converge to an equilibrium. Specifically, we construct a sequence of best responses that takes exponential time to converge to a Nash equilibrium.

THEOREM 5.1. *Best response dynamics for k players may run in time exponential in k.*

To prove this, we now construct an example (shown in Figure 5.1) in which by appropriate ordering of the best response of players, we can simulate a $\Omega(k)$-bit binary counter. The idea is that we have a set of players corresponding to each bit of the counter. Then we describe a sequence of best response moves that lead to the counter incrementing from the "all zeros" state onwards. Since with n bits, we can implement a counter that counts up to 2^n, and then as long as we can show that each increment of the counter corresponds to a sequence of best response moves from the current configuration, we would have shown an exponentially long best response sequence. In what follows, we first describe the set of gadgets to construct the counter, and then show the set of inequalities that result from the increments of the counter.

The graph has $3n$ players: n "bit" players, each denoted by bit(i), and for each bit player we also have two "auxiliary" players. The auxiliary players of the bit(i) player are denoted by $A(i)$ and $B(i)$. We construct the graph as follows. For each player we form a gadget as shown in Figure 5.1. The gadget for each bit player and each auxiliary player has only two path options, we call these the 0 path and the 1 path. Figure 5.1 shows the set of edges that belong to each gadget. The gadget for the ith bit player, for instance, consists of edges $\alpha(i)$, $\beta(i)$, a set of edges $t(j,i)$, one for each $j > i$, and a few unnamed edges as shown in the figure. To construct the graph, we simply take the union of these gadgets. The labeled edges are shared with the other gadgets, whereas the unlabeled edges are not. Furthermore, the label Δ

TABLE 5.1

Table showing the first 15 *steps of the scheduled best response run for two players. The number "*1*" represents that this player is taking the* 1*-path in the current configuration, and "*0*" represents that it takes the* 0*-path. The first, fifth, eleventh, and last configuration correspond to the counter values of* 00*,* 01*,* 10*, and* 11*, respectively.*

bit_2	bit_1	B_2	A_2	B_1	A_1
0	0	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	0	1	0
1	1	0	0	1	0
1	1	0	1	1	0
1	1	0	1	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	0	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	0	1	0

actually denotes the cost of the corresponding edges; these edges too are individual to each gadget.

Thus, each of the $\alpha(i)$ edges are in the gadgets of players $A(i)$ and $\text{bit}(i)$, and so appear in the 1 path of both $A(i)$ and $\text{bit}(i)$. Each edge $\beta(i)$ is in the 1 path of two players $B(i)$ and $\text{bit}(i)$, and each edge $e(i)$ is in the 0 path of $A(i)$ and the 1 path of $B(i)$. Each edge $f(i,j)$ and $t(i,j)$ is actually shared by two gadgets. For a specific i and j with $i > j$, the same edge $f(i,j)$, for instance, is present in the 0-path of $B(j)$, and also in the 1 path of $A(i)$. In addition to the unnamed edges drawn in the gadget, assume there are unnamed edges before and after every edge type of $f(i,j)$ or $t(i,j)$ and these unnamed edges are directed and specific to a particular gadget. In the above figure, we have not specified the order in which the edges of, for example, $f(j,i)$ for all $j > i$ in the gadget of $bit(i)$ proceed. In all of these cases, there is one index that is fixed, and one that is variable, so we assume that these edges appear in increasing order of the variable index.

We will later prove that *no player will ever choose a path in this graph other than the* 0 *or the* 1 *path corresponding to its gadget.* This significantly simplifies the strategy sets of the players, so now we can simply think of each player as choosing either strategy 0 or 1. Below we describe an exponential-length best-response run that simulates a binary counter. A sample run of this type is shown in Table 5.1. The first, fifth, and eleventh steps shown correspond to the values 00, 01, and 10 of the counter, and indeed these are the values of the strategies of the $\text{bit}(i)$ players. The rest of the steps pictured are intermediate steps, and only exist to get the $\text{bit}(i)$ players into the correct configuration.

In what follows, we abuse notation and represent the cost of an edge by the label itself, e.g., $\alpha(i)$, $f(i,j)$, and $t(i,j)$ will also be used to denote the cost of the corresponding edges. All the unnamed edges are zero cost, except for the two sets of edges, one in each path of $A(i)$ that each have cost Δ, which is large, say $3n$ times bigger than the sum of the cost of all the named edges. Thus player $A(i)$ always pays at least Δ in order to get to the sink, but never should agree to pay more than $\Delta + \Delta/3n$.

We also refer to the player going on the "one" path as the *player being set* and going on the "zero" path as the *player being reset*. Each player has one source and one sink and the paths of each player are as shown in the gadgets above. The costs of the paths of the ith bit player are referred to as $x_i^{(0)}$ and $x_i^{(1)}$, and those of the player $A(i)$ and $B(i)$ as $a_i^{(0)}$, $a_i^{(1)}$ and $b_i^{(0)}$, $b_i^{(1)}$, respectively. Note that these denote the actual costs of these paths, not the cost shares in any configuration. Now we describe the sequence of best-response moves that lead to the counter incrementing. For the moves that we will describe to actually be best-response moves, particular inequalities will have to hold on the edge costs. Below we give a sequence of best-response moves that is exponential length, together with the inequalities that must hold for these to be valid best-response moves. We will later show that all of these inequalities can be satisfied.

Start Step: All the players are reset.

General Step: The bits from 1 to $\ell - 1$ are all set. The bits from $\ell + 1$ to n may be at 0 or 1. The ℓth bit is currently at 0 and has to be set at 1. Also, all the $A(j)$ players are reset. The $B(j)$ players are set if and only if the jth players are set.

Notice that the first, fifth, and last step in Table 5.1 are of this form. We now show a sequence of best responses which will set the ℓ'th bit, set all the bits 1 to $\ell - 1$ to 0 (which is what should happen in a counter when the ℓ'th bit is set), and return all the auxiliary players to the form described in the "General Step" above.

- First, the ℓth bit sets. At this point, there are no other players using either of bit ℓth player's paths, so for this to happen, we need only the following to be true.

$$x_\ell^{(1)} < x_\ell^{(0)}$$
$$\text{implying, } \alpha(\ell) + \beta(\ell) < \sum_{j>\ell} t(j,\ell)$$

The setting of the bit(ℓ) player will now first trigger $A(\ell)$ and then $B(\ell)$, as follows.

- The cost of the 1 path of $A(\ell)$ has now decreased by $\alpha(\ell)/2$ because of the player bit(ℓ) using it. $A(\ell)$ is thus triggered and is allowed to set. Since both the 0 and 1 path have the Δ cost edges that are to be paid by $A(i)$ alone, they do not matter in the best response calculations. Thus, for this to be a valid best response move, it is enough that

$$a_\ell^{(1)} - \alpha(\ell)/2 < a_\ell^{(0)}$$
$$\text{implying that, } \alpha(\ell)/2 + \sum_{j<\ell} f(\ell,j) + \sum_{j<\ell} t(\ell,j) < e(\ell)$$

- The setting of $A(\ell)$ triggers all the $B(i)$ for $i < \ell$ to be reset. Recall that the corresponding $A(i)$ are already reset. We allow these $B(i)$ to reset. Due to $A(\ell)$ being on its 1 path, the cost of 0 path has changed by $f(\ell,i)/2$. Since only the player bit(i) is using the edge $\beta(i)$, then for $B(i)$ to want to reset, we need that

$$b_i^{(0)} - f(\ell,i)/2 < b_i^{(1)} - e(i)/2 - \beta(i)/2$$
$$\text{that is, } \sum_{j>i} f(j,i) - f(\ell,i)/2 < e(i)/2 + \beta(i)/2$$

- $A(\ell)$ now also triggers all the bit players bit(i) for all $i < l$ to reset by reducing the cost of the 0 path of bit(i) by $t(\ell, i)/2$. For each such $i < \ell$, the bit $B(i)$ has also just been reset. Thus for this to be a best response move, it should be the case that

$$x_i^{(0)} - t(\ell, i)/2 < x_i^{(1)}$$
$$\sum_{j>i} t(j, i) - t(\ell, i)/2 < \alpha(i) + \beta(i)$$

- Now because of the setting of the bit(ℓ), the 1 path of $B(\ell)$ became cheaper by $\beta(\ell)/2$. $B(\ell)$ wants to set and is allowed to do so. For this, we need that

$$b_\ell^{(1)} - \beta(\ell)/2 < b_\ell^{(0)}$$
$$e(\ell) + \beta(\ell)/2 < \sum_{j>\ell} f(j, \ell)$$

- Lastly, we now want $A(\ell)$ to reset. As a result of the setting of $B(\ell)$, the 0 path of $A(\ell)$ became cheaper by $e(\ell)/2$. Edges on the 1-path of $A(\ell)$ are shared by players bit(x) and $B(x)$ for $x < i$, and by the player bit(ℓ). So we need that the 0-path of $A(\ell)$ be less expensive to it even if all these players pay off their corresponding shares on the 1-path. Again since the Δ-cost edges are present on both sides, they do not matter in the computation of the best response. That is, we need that

$$a_\ell^{(0)} - e(\ell)/2 < a_\ell^{(1)} - \sum_{j<\ell} (f(\ell, j)/2 + t(\ell, j)/2) - \alpha(\ell)/2$$
$$e(\ell)/2 < \alpha(\ell)/2 + \sum_{j<\ell} (f(\ell, j)/2 + t(\ell, j)/2)$$

- Now we have the subgame of bit and auxiliary players from 1 to $\ell - 1$ being completely reset, and no other player corresponding to the higher-numbered bits influencing any of their paths. This corresponds to the first $\ell - 1$ bits of the counter becoming 0 because the ℓ'th bit just became 1. We can now use the best response run corresponding to incrementing the first $\ell - 1$ bits of the counter from all 0's to all 1's again, without any interference from the players corresponding to the higher-numbered bits. This gets us back to the configuration in the start of the recursion (the "General Step" above), except now we need to deal with the $(\ell + 1)^{st}$ bit.

Proof of Theorem 5.1. We now prove that the above game has an exponential best response run under the above best response scheduling.

All we need to show is that the moves described in the scheduling are best responses. We first argue that each player has only two cheap paths available to him, which we have described as the zero path and the one path. To complete the construction we next need to come up with a set of values for the links that satisfy the set of best response inequalities above. Taken together, it follows that the moves are all best responses and simulate a $\Omega(k)$-size counter.

First note that the 0 path and 1 path of any one particular player are vertex disjoint. Recall that the unnamed edges, both in the figure and the ones in between the $f(i, j)$ edges and the $t(i, j)$ edges are all exclusive to each gadget. By our construction, the unnamed edges also impose the following property on the 0/1-paths of all players: The 0/1-path of any one player must either have at least one edge in common or must

be vertex disjoint from the 0/1-path of any other player. We do not have to worry about the case of just having vertices in common.

We need to argue that each player has only two cheap paths available to him. Intuitively the argument is as follows. If one player deviates out of his gadget (i.e., takes a path other than its 0 or 1 path), it will never be able to come back to his own sink, or would have to pay an exorbitant amount (Δ) in order to reach its sink. Thus, the only available cheap strategies to a player are the 0 and 1 paths.

Before going into the details, we first define a function to make the notation simpler. For any edge e, define the function $sink(e)$ to be the set of all possible sinks (terminal nodes of players) to which paths from this edge can lead. Note that the graph only has edges $f(x, y)$ and $t(x, y)$ where $x > y$; there are no corresponding edges for $x \leq y$. For notational simplicity, for a general tuple (x, y), we define $sink(f(x, y))$ as follows. Note that $f(x, y)$ can lead to the sink of $A(x)$ and $B(y)$ if $x > y$, as well as to $sink(f(x + 1, y))$ and $sink(f(x, y + 1))$. Thus, inductively, if $x > y$

$$sink(f(x, y)) = \{A(x), \ldots, A(n), B(y), \ldots, B(n)\}$$

and $\emptyset$ otherwise (since the corresponding $f(x, y)$ edges do not appear in the gadget at all).

Similarly, since for $x \leq y$, the edges $t(x, y)$ are not present in the construction, define $sink(t(x, y)) = \emptyset$ for $x \leq y$. In order to compute $sink(t(x, y))$ for $x > y$, note that $t(x, y)$ can lead to $A(x)$, $bit(y)$, as well as $sink(t(x+1, y))$, $sink(t(x, y+1))$, and $sink(f(x, 1))$ (since in the 1 path of $A(x)$, edge $t(x, y)$ can be followed by $f(x, 1)$). Thus, if $x > y$

$$sink(t(x, y)) = \{bit(y), \ldots, bit(n), A(x), \ldots, A(n), B(1), \ldots, B(n)\}$$

and $\emptyset$ else.

We now flesh out the argument for each player for each of the two cases, the set-case and the reset-case. Consider player $B(i)$. If a strategy follows the first unnamed edge of 1-path, then it reaches $\beta(i)$. From the end vertex of $\beta(i)$ it has the option of choosing the next unnamed edge in 1-path of gadget $B(i)$, or in the 1-path of $bit(i)$. In the first case the strategy has to go through $e(i)$. But after it crosses $e(i)$ there is only a zero-cost edge left to be covered to the sink of $B(i)$, and so other deviations are useless. If the path instead had chosen to enter the gadget of $bit(i)$ after the edges $\beta(i)$, then it would have to travel through $\alpha(i)$ and then either get stuck at the sink of $bit(i)$ or travel through one or more edges of the form $\{t(i, y), y < i\}$. But in order to enter an edge of the form $t(i, y)$ this strategy would have to pay at least $\Delta/3n$ to share the edge priced Δ, which is more than the total cost of $B(i)$. So any strategy starting on the 1-path of $B(i)$ does not enter the $A(i)$ gadget.

Next consider a strategy starting out on the 0-path of $B(i)$ through the first unnamed edge. The edges then appearing on this path are of the form $\{f(x, i), x > i\}$. These edges also appear in the gadgets $A(x)$. If this strategy chooses to deviate to exit the $B(i)$ gadget after the edge $f(x, i)$ and enter the $A(x)$ one, the labeled edge that it meets next is $f(x, i + 1)$. But no path from $f(x, i + 1)$ leads to the sink of $B(i)$, as is verified from the sink-function above. So this deviation cannot happen. We have now shown that the best strategies of $B(i)$ must be the 0 and 1 paths only.

Now, take the $A(i)$ player and a strategy which starts out on the first unnamed edge of the 1-path. This strategy then has to go through $\alpha(i)$. From the endpoint of $\alpha(i)$ the only edges are to the sink of $bit(i)$, or continue along the $A(i)$ gadget. The edges that appear next are of the form $\{t(i, x), x < i\}$. If the path does not continue

along the gadget of $A(i)$, and instead switches to the 0-path of $bit(x)$, then the next edge it encounters is $t(i+1,x)$. But according to the sink function, there is no way to reach the sink of $A(i)$ from $t(i+1,x)$, so this is not a valid deviation. Next, the path cannot deviate out of any the $\{f(i,x), x < i\}$ edges since $A(i)$ does not lie in any $sink(f(i+1,x))$. Thus strategies of $A(i)$ starting out on the 1-path of $A(i)$ stay on this path.

Next consider the strategy of $A(i)$ that starts out through the unnamed edge in the 0-path of $A(i)$. This path cannot deviate after $e(i)$ because it would just get stuck in the sink of $B(i)$. Thus, we have shown that the best strategies of $A(i)$ must be the 0 and 1 paths only.

Lastly consider the ith bit player. Along 0-path of bit-i, the only shared nodes and edges of this player are the $t(x,i)$ edges that are each shared with $A(x)$ for all $x > i$. Suppose this bit player follows the $t(x,i)$ edge to the gadget of $A(x)$ with $x > i$. The next edge is then $t(x,i+1)$ which does not lead to $bit(i)$-sink. Thus the 0-path strategies do not allow any deviations. A path starting with the first edge of the 1-path of this bit-player can deviate after $\beta(i)$ and enter edge $e(i)$ of the $B(i)$ gadget. But from there all paths get stuck at sinks. Thus the 1-path deviations cannot happen after $\beta(i)$. But the deviations cannot happen after $\alpha(i)$ either, as the remaining edge is zero cost.

This concludes the proof that the best strategies of the players always correspond to the 0 and 1 paths in the gadgets.

For the last part of the construction, we show that it is possible to come up with a set of values for the links such that the best response inequalities are satisfied. Let the edge costs be as follows. In all the remaining formulae let c be any constant greater than 10. Let $\alpha(i) = 1$, and $\beta(i) = 2c^i(n-i) - c^i/2 - 3e(i)/2$. Also, $e(i) = \sum_{j<i} f(i,j) + \sum_{j<i} t(i,j) + 3/4$ for all i. Finally, for all pairs i, j, $f(i,j) = c^j$ and $t(i,j) = 2c^j$. Given these values, we can check that the inequalities above are satisfied for all $i < n$, and thereby we can have a run of best responses of length exponential in the number of players. □

6. Weighted players. So far we have assumed that players sharing an edge e pay equal fractions of e's cost. We now consider a game with fixed edge costs where players have weights $w_i \geq 1$, and players' payments are proportional to their weight. More precisely, given a strategy $S = (S_1, \ldots, S_k)$, define W to be the total weight of all players, and let W_e be the sum of the weights of players using e. Then player i's payment for edge e will be $\frac{w_i}{W_e} c_e$.

Note that the potential function $\Phi(S)$ used for the unweighted version of the game is not a potential function once weights are added. In particular, in a weighted game, improving moves can increase the value of $\Phi(S)$, as this is no longer a congestion game. The following theorem uses a new potential function for a special class of weighted games.

THEOREM 6.1. *In a weighted game where each edge e is in the strategy spaces of at most two players, there exists a potential function for this game, and hence a Nash equilibrium exists.*

Proof. Consider the following potential function. For each edge e used by players i and j, define

$$\Phi_e(S) = \begin{cases} c_e w_i & \text{if player } i \text{ uses } e \text{ in } S \\ c_e w_j & \text{if player } j \text{ uses } e \text{ in } S \\ c_e \theta_{ij} & \text{if both players } i \text{ and } j \text{ use } e \text{ in } S \\ 0 & \text{otherwise,} \end{cases}$$

where $\theta_{ij} = (w_i + w_j - \frac{w_i w_j}{w_i + w_j})$. For any edge e with only one player i, simply set $\Phi_e(S) = w_i c_e$ if i uses e and 0 otherwise. Define $\Phi(S) = \sum_e \Phi_e(S)$. We now simply need to argue that if a player makes an improving move, then $\Phi(S)$ decreases. Consider a player i and an edge e that player i joins. If the edge already supported another player j, then i's cost for using e is $c_e \frac{w_i}{w_i + w_j}$, while the change in $\Phi_e(S)$ is

$$c_e(w_i - \frac{w_i w_j}{w_i + w_j}) = c_e \frac{{w_i}^2}{w_i + w_j}.$$

Thus the change in potential when i joins e equals the cost i incurs, scaled up by a factor of w_i. In fact, it is easy to show the more general fact that when player i moves, the change in $\Phi(S)$ is equal to the change in player i's payments scaled up by w_i. This means that improving moves always decrease $\Phi(S)$, thus proving the theorem. □

Note that this applies not only to paths, but also to the generalized model in which players select subsets from some ground set. The analogous condition is that no ground element appears in the strategy spaces of more than two players.

COROLLARY 6.2. *Any two-player weighted game has a Nash equilibrium.*

While the above potential function also implies a bound on the price of stability, even with only two players this bound is very weak. However, if there are only two players with weights 1 and $w \geq 1$, then we can show that the price of stability is at most $1 + \frac{1}{1+w}$, and this is tight for all w.

The following result shows the existence of Nash equilibria in weighted single commodity games.

THEOREM 6.3. *For any weighted game in which all players have the same source s and sink t, best response dynamics converge to a Nash equilibrium, and hence Nash equilibria exist.*

Proof. Start with any initial set of strategies S. For every $s - t$ path P define the marginal cost of P to be $c(P) = \sum_{e \in P} \frac{c_e}{W_e}$, where W_e is the sum of the weights of players using e in the state S. Observe that if player i currently uses path P, then i's payment is $w_i c(P)$. Define $P(S)$ to be a tuple of the values $c(P)$ over all paths P, sorted in increasing order. We want to show that the cheapest improving deviation of any player causes $P(S)$ to strictly decrease lexicographically.

Suppose that one of the best moves for player i is to switch paths from P_1 to P_2. Let $\mathcal{P}$ denote the set of paths that intersect $P_1 \cup P_2$. For any pair of paths P and Q, let $c_P(Q)$ denote the new value of $c(Q)$ after player i has switched to path P. To show that $P(S)$ strictly decreases lexicographically, it suffices to show that

$$\min_{P \in \mathcal{P}} c_{P_2}(P) < \min_{P \in \mathcal{P}} c(P). \tag{6.1}$$

Define $P' = \arg\min_{P \in \mathcal{P}} c(P)$. Since P_2 was i's best response, $c_{P_2}(P_2) \leq c_P(P)$ for all paths P. In particular, $c_{P_2}(P_2) \leq c_{P'}(P')$. We also know that $c_{P'}(P') \leq c(P')$, since in deviating to P', player i adds itself to some edges of P'. In fact, $c_{P'}(P') < c(P')$ unless $P' = P_1$. Assuming $P' \neq P_1$, we now have that $c_{P_2}(P_2) < c(P')$, which proves inequality (6.1). If $P' = P_1$, then since player i decided to deviate, $c_{P_2}(P_2) < c(P_1)$. Therefore, we once again have that $c_{P_2}(P_2) < c(P')$, as desired. □

In the case where the graph consists of only 2 nodes s and t joined by parallel links, we can similarly show that any sequence of improving responses converge to a Nash equilibrium.

Weighted games with three or more players need not have a pure Nash equilibrium [8]. The following claim shows that, even when Nash equilibria do exist, the prices of stability bounds from the unweighted case do not carry over.

THEOREM 6.4. *There are weighted games for which the price of stability is* $\Theta(\log W)$ *and* $\Theta(k)$.

An example exhibiting this is a modified version of the graph in Figure 1.1. Change the edge with cost $1+\varepsilon$ to cost 1, and for all other edges with positive cost, set the new cost to be $\frac{1}{2}$. For $1 \le i \le k$ let player i have weight $w_i = 2^{i-1}$. Since each player has a greater weight than all smaller weight players combined, the only Nash equilibrium has cost $\frac{k}{2} = \Theta(\log W)$, while the optimal solution has cost 1.

7. Conclusions and open questions. For the Fair Connection Game, we showed that the price of stability is always at most $H(k)$, and that this is tight for directed graphs. However, the case for undirected graphs remains largely unresolved. The results of section 4 show that, at least for two players, the price of stability for undirected graphs can be strictly better than that for directed. The largest lower bound that we know of for the price of stability in undirected networks is 12/7. The worst-case price of stability in undirected games could be constant, showing that fair sharing works very well in such games. See [1] for improved price of stability bounds in the undirected case, as well as Fiat et al. [20] for recent progress on a special case of this question.

Another open question is whether or not a good Nash equilibrium can be computed in polynomial time for a large number of players. One approach to solving this problem is to first compute an approximation to the centralized optimum, and then simulate best-response dynamics to reach a Nash equilibrium with cost at most $O(\log k)$ times that of the initial state. Unfortunately, Theorem 5.1 shows that arbitrary best-response dynamics can require exponential time to converge. On the other hand, it is possible that best responses can always be scheduled in a way that guarantees convergence in polynomial time. A weaker goal is to show how to compute a $(1+\varepsilon)$-approximate Nash equilibrium, where no player can decrease its cost by more than a $1+\varepsilon$ factor, in polynomial time.

For the version of the Fair Connection Game with latencies instead of edge costs, the main challenge is to provide price of stability results for the general scenario, rather that just the single-source special case. See [7, 10] for recent results for the special case of linear latency functions.

Finally, for weighted games (section 6) we have given only preliminary results. Recently, Chen and Roughgarden [8] proved that pure Nash equilibria need not exist in such games. They also showed that every weighted game has an $O(\log w_{\max})$-approximate Nash equilibrium that costs at most $O(\log W)$ times the centralized optimum (where $w_{\max}$ is the maximum weight and W is the sum of the weights, assuming that the minimum weight is 1). These bounds are nearly tight. But the following natural question is still open: What is the price of stability of approximate Nash equilibria in Weighted Fair Connection Games if all players share a common terminal?

REFERENCES

[1] A. AGARWAL AND M. CHARIKAR, *On the price of stability in undirected networks*, unpublished manuscript.

[2] S. ALBERS, S. EILTS, E. EVEN-DAR, Y. MANSOUR, AND L. RODITTY, *On Nash equilibria for a network creation game*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006, pp. 89–98.

[3] E. ANSHELEVICH, A. DASGUPTA, É. TARDOS, AND T. WEXLER, *Near-optimal network design with selfish agents*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 511–520.

[4] B. Awerbuch, Y. Azar, and L. Epstein, *The price of routing unsplittable flow*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), 2005, pp. 57–66.

[5] V. Bala and S. Goyal, *A non-cooperative model of network formation*, Econometrica, 68 (2000), pp. 1181–1229.

[6] M. Beckmann, C. B. McGuire, and C. B. Winsten, *Studies in the Economics of Transportation*, Yale University Press, New Haven, CT, 1956.

[7] I. Caragiannis, M. Flammini, C. Kaklamanis, P. Kanellopoulos, and L. Moscardelli, *Tight bounds for selfish and greedy load balancing*, in Proceedings of the 33rd Annual International Colloquium in Automata, Languages, and Programming (ICALP), Lecture Notes in Computer Science 4051, 2006, pp. 311–322.

[8] H. Chen and T. Roughgarden, *Network design with weighted players*, in Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architextures (SPAA), 2006, pp. 28–37.

[9] C. Chekuri, J. Chuzhoy, L. Lewin-Eytan, J. Naor, and A. Orda, *Non-cooperative multicast and facility location games*, in Proceedings of the 7th ACM Conference on Electronic Commerce (EC), 2006, pp. 72–81.

[10] G. Christodoulou and E. Koutsoupias, *On the price of anarchy and stability of correlated equilibria of linear congestion games*, in Proceedings of the 13th Annual Eurpoean Symposium on Algorithms (ESA), 2005, pp. 59–70.

[11] G. Christodoulou and E. Koutsoupias, *The price of anarchy of finite congestion games*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), 2005, pp. 67–73.

[12] J. Corbo and D. C. Parkes, *The price of selfish behavior in bilateral network formation games*, in Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (PODC), 2005, pp. 99–107.

[13] J. R. Correa, A. S. Schulz, and N. E. Stier Moses, *Selfish routing in capacitated networks*, Math. Oper. Res., 29 (2004), pp. 961–976.

[14] A. Czumaj, P. Krysta, and B. Vöcking, *Selfish traffic allocation for server farms*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), 2002, pp. 287–296.

[15] A. Czumaj and B. Vöcking, *Tight bounds for worst-case equilibria*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 413–420.

[16] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. J. Shenker, *On a network creation game*, in Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC), 2003, pp. 347–351.

[17] A. Fabrikant, C. H. Papadimitriou, and K. Talwar, *The complexity of pure Nash equilibria*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 604–612.

[18] A. Fanelli, M. Flammini, G. Melideo, and L. Moscardelli, *Multicast transmissions in non-cooperative networks with a limited number of selfish moves*, in Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 4162, 2006, pp. 363–374.

[19] J. Feigenbaum, C. Papadimitriou, and S. Shenker, *Sharing the cost of multicast transmissions*, J. Comput. System Sci., 63 (2001), pp. 21–41.

[20] A. Fiat, H. Kaplan, M. Levy, S. Olonetsky, and R. Shabo, *On the price of stability for designing undirected networks with fair cost allocations*, in Proceedings of the 33rd Annual International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Computer Science 4061, 2006, pp. 608–618.

[21] H. Heller and S. Sarangi, *Nash Networks with Heterogeneous Agents*, Working Paper Series 2001, E-2001-1, Virginia Tech.

[22] S. Herzog, S. Shenker, and D. Estrin, *Sharing the "cost" of multicast trees: An axiomatic analysis*, IEEE/ACM Transactions on Networking, Dec. 1997.

[23] K. Jain and V. Vazirani, *Applications of approximation algorithms to cooperative games*, in Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC), 2001, pp. 364–372.

[24] R. Johari and J. N. Tsitsiklis, *Efficiency loss in a network resource allocation game*, Math. Oper. Res., 29 (2004), pp. 407–435.

[25] F. P. Kelly, *Charging and rate control for elastic traffic*, Eur. Trans. Telecommunications, 8 (1997), pp. 33–37.

[26] K. Kent and D. Skorin-Kapov, *Population monoton cost allocation on mst's*, in Operations Research Proceedings KOI, 1996, pp. 43–48.

[27] E. Koutsoupias and C. H. Papadimitriou, *Worst-case equilibria*, in Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 1563, 1999, pp. 404–413.
[28] L. Libman and A. Orda, *Atomic resource sharing in noncooperative networks*, Telecommun. Systems, 17 (2001), pp. 385–409.
[29] D. Monderer and D. Samet, *Variations of the Shapley Value*, in Handbook of Game Theory Vol. III, R.J. Aumann and S. Hart, eds., Elsevier Science, New York, 2002.
[30] D. Monderer and L. S. Shapley, *Potential games*, Games and Economic Behavior, 14 (1996), pp. 124–143.
[31] T. Moscibroda, S. Schmid, and R. Wattenhofer, *On the topologies formed by selfish peers*, in Electronic Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS), 2006.
[32] H. Moulin and S. Shenker, *Strategyproof sharing of submodular costs: Budget balance versus efficiency*, Economic Theory, 18 (2001), pp. 511–533.
[33] N. Nisan and A. Ronen, *Algorithmic mechanism design*, Games and Economic Behavior, 35 (2001), pp. 166–196.
[34] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, eds., *Algorithmic Game Theory*, Cambridge, 2007.
[35] C. H. Papadimitriou, *Algorithms, games, and the Internet*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), 2001, pp. 749–753.
[36] R. W. Rosenthal, *The network equilibrium problem in integers*, Networks, 3 (1973), pp. 53–59.
[37] T. Roughgarden, *The price of anarchy is independent of the network topology*, J. Comput. Syst. Sci., 67 (2003), pp. 341–364.
[38] T. Roughgarden, *Stackelberg scheduling strategies*, SIAM J. Comput., 33 (2004), pp. 332–350.
[39] T. Roughgarden and É. Tardos, *How bad is selfish routing?*, J. ACM, 49 (2002), pp. 236–259.
[40] S. Suri, C. Tóth, and Y. Zhou, *Selfish load balancing and atomic congestion games*, Algorithmica, 2007, Algorithmica, 47 (2007), pp. 79–96.
[41] A. Vetta, *Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions*, in Proceedings of the 43rd Annual Symposium on Foundations of Computer Science (FOCS), 2002, pp. 416–425.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1624–1647

A LOWER BOUND FOR THE SIZE OF SYNTACTICALLY MULTILINEAR ARITHMETIC CIRCUITS*

RAN RAZ†, AMIR SHPILKA‡, AND AMIR YEHUDAYOFF§

Abstract. We construct an explicit polynomial $f(x_1, \ldots, x_n)$, with coefficients in $\{0, 1\}$, such that the size of any syntactically multilinear arithmetic circuit computing f is at least $\Omega(n^{4/3}/\log^2 n)$. The lower bound holds over any field.

Key words. lower bounds, arithmetic circuits, explicit constructions

AMS subject classification. 68Q17

DOI. 10.1137/070707932

1. Introduction. Arithmetic circuits are the standard model for computing polynomials (see section 1.1 for a definition). Roughly speaking, given a set of variables $X = \{x_1, \ldots, x_n\}$, an arithmetic circuit uses additions and multiplications to compute a polynomial f in the set of variables X. Given a polynomial f, we are interested in the number of operations needed to compute f.

The best lower bound known for the size of arithmetic circuits is the classical $\Omega(n \log n)$ of Strassen [8] and of Baur and Strassen [1]. Proving better lower bounds is an outstanding open problem. In this paper, we focus on a restricted class of arithmetic circuits, the class of syntactically multilinear arithmetic circuits. We prove an $\Omega(n^{4/3}/\log^2 n)$ lower bound for the size of syntactically multilinear arithmetic circuits computing an explicit polynomial.

1.1. Syntactically multilinear arithmetic circuits. An *arithmetic circuit* Φ over the field $\mathbb{F}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ is a directed acyclic graph as follows: Every vertex v in Φ is either of in-degree 0 or of in-degree 2. Every vertex v in Φ of in-degree 0 is labeled by either a variable in X or a field element in $\mathbb{F}$. Every vertex v in Φ of in-degree 2 is labeled by either $\times$ or $+$. An arithmetic circuit Φ is called an *arithmetic formula* if Φ is a directed binary tree (the edges of an arithmetic formula are directed from the leaves to the root).

Let Φ be an arithmetic circuit over the field $\mathbb{F}$ and the set of variables X. The vertices of Φ are also called *gates*. Every gate of in-degree 0 is called an *input gate*. Every gate of in-degree 2 labeled by $\times$ is called a *product gate*. Every gate of in-degree 2 labeled by $+$ is called an *addition gate*. Every gate of out-degree 0 is called an *output gate*. For two gates u and v in Φ, if (u, v) is an edge in Φ, then u is called a *son* of v, and v is called a *father* of u. The *size* of Φ, denoted $|\Phi|$, is the number of edges in

*Received by the editors November 12, 2007; accepted for publication (in revised form) July 30, 2008; published electronically December 10, 2008. Appeared in the Proceedings of the 48th FOCS, 2007, pp. 438–448.

http://www.siam.org/journals/sicomp/38-4/70793.html

†Faculty of Mathematics and Computer Science, Weizmann Institute, Rehovot 76100, Israel, and Microsoft Research, Redmond, WA 98052 (ran.raz@weizmann.ac.il). This author's research was supported by grants from the Binational Science Foundation (BSF), the Israel Science Foundation (ISF), and the Minerva Foundation.

‡Faculty of Computer Science, Technion, Haifa 32000, Israel (shpilka@cs.technion.ac.il).

§Faculty of Mathematics and Computer Science, Weizmann Institute, Rehovot 76100, Israel (amir.yehudayoff@weizmann.ac.il). This author's research was supported by grants from the Binational Science Foundation (BSF), the Israel Science Foundation (ISF), and the Minerva Foundation.

Φ. Since the in-degree of Φ is at most 2, the size of Φ is at most twice the number of gates in Φ.

For a gate v in Φ, define Φ_v to be the subcircuit of Φ rooted at v as follows: The gates of Φ_v are all the gates u in Φ such that there exists a directed path from u to v in Φ. The edges and labels of Φ_v are the same edges and labels of Φ (restricted to the set of gates of Φ_v).

An arithmetic circuit computes a polynomial in a natural way. For a gate v in Φ, define $\widehat{\Phi}_v \in \mathbb{F}[X]$ to be the polynomial computed by Φ_v as follows: If v is an input gate labeled by $\alpha \in \mathbb{F} \cup X$, then $\widehat{\Phi}_v = \alpha$. If v is a product gate with sons v_1 and v_2, then $\widehat{\Phi}_v = \widehat{\Phi}_{v_1} \cdot \widehat{\Phi}_{v_2}$. If v is an addition gate with sons v_1 and v_2, then $\widehat{\Phi}_v = \widehat{\Phi}_{v_1} + \widehat{\Phi}_{v_2}$. For a polynomial $f \in \mathbb{F}[X]$, and a gate v in Φ, we say that v *computes* f if $f = \widehat{\Phi}_v$. For a polynomial $f \in \mathbb{F}[X]$, we say that Φ *computes* f if there exists a gate u in Φ that computes f.

A polynomial $f \in \mathbb{F}[X]$ is called *multilinear* if the degree of each variable in f is at most one. An arithmetic circuit Φ is called *(semantically) multilinear* if every gate in Φ computes a multilinear polynomial. An arithmetic circuit Φ is called *syntactically multilinear* if for every product gate v in Φ with sons v_1 and v_2, the set of variables that occur in Φ_{v_1} and the set of variables that occur in Φ_{v_2} are disjoint.

1.2. Background and motivation. There are two ways to define multilinear arithmetic circuits—a syntactic definition and a semantic definition—as described above. The semantic definition is a natural one, but the syntactic definition is more convenient to work with. Note, for example, that given an arithmetic circuit Φ, deciding whether Φ is syntactically multilinear is straightforward, whereas it is not clear if one can decide whether Φ is semantically multilinear in deterministic polynomial time. We note also that similar distinctions between semantic and syntactic definitions occur in other places in computer science (e.g., read k-times branching programs).

Multilinear arithmetic circuits were defined by Nisan and Wigderson in [4]. The model of syntactically multilinear arithmetic formulas was defined in [5]. In [5], it is shown that any multilinear arithmetic *formula* computing the determinant (or the permanent) of an $n \times n$ matrix must be of size $n^{\Omega(\log n)}$. Prior to our work, no lower bounds (better than the $\Omega(n \log n)$ lower bound of Strassen and of Baur and Strassen) for the size of syntactically multilinear arithmetic *circuits* were known.

The techniques of [5] for proving superpolynomial lower bounds for the size of multilinear arithmetic formulas fail for circuits. In fact, [6] used these techniques to prove that syntactically multilinear arithmetic circuits are superpolynomially more powerful than multilinear arithmetic formulas. More specifically, there exists a polynomial f such that every multilinear arithmetic formula computing f is of size $n^{\Omega(\log n)}$, and, on the other hand, there exists a polynomial-size syntactically multilinear arithmetic circuit computing f.

Every multilinear polynomial f can be computed by a syntactically multilinear arithmetic circuit Φ, but Φ might not be the smallest arithmetic circuit computing f. However, computing a multilinear polynomial by an arithmetic circuit that is *not* syntactically multilinear is usually less intuitive, as cancellations of monomials are needed.

A syntactically multilinear arithmetic circuit is semantically multilinear as well. However, it is still not known whether there is an efficient way to transform a semantically multilinear arithmetic circuit into a syntactically multilinear circuit. We note that a semantically multilinear arithmetic *formula* can be transformed *without changing its size* into a syntactically multilinear arithmetic formula that computes

the same polynomial (see section 2 in [5]). We do not know of any significant example of a semantically multilinear arithmetic circuit that is *not* syntactically multilinear.

Finally, we note that two known classes of arithmetic circuits are contained in the class of syntactically multilinear arithmetic circuits: Pure arithmetic circuits (as defined by Nisan and Wigderson in [4]; see also [7]) are a restricted type of syntactically multilinear arithmetic circuits. Monotone arithmetic circuits computing a multilinear polynomial are also syntactically multilinear (see [4]).

1.3. Results and methods. Our main result is a construction of an explicit polynomial f such that any syntactically multilinear arithmetic circuit computing f is of size $\Omega(n^{4/3}/\log^2 n)$. Formally, we have the following theorem.

THEOREM 1.1. *Let $f \in \mathbb{F}[X, \Omega]$ be the polynomial defined in section 6, where $\mathbb{F}$ is any field, and X and Ω are two sets of variables of size n each. Let Φ be a syntactically multilinear arithmetic circuit over the field $\mathbb{F}$ and the sets of variables X and Ω computing f. Then,*

$$|\Phi| = \Omega\left(\frac{n^{4/3}}{\log^2 n}\right).$$

The paper has three main parts: section 3 investigates the method of Baur and Strassen for computing all partial derivatives of a polynomial. Section 5, which is the heart of our proof, gives a simple characterization of a polynomial for which our lower bound applies. Section 6 constructs a polynomial for which the lower bound applies.

In [1], Baur and Strassen showed that given an arithmetic circuit Ψ computing a polynomial $f \in \mathbb{F}[X]$, there exists an arithmetic circuit Ψ' computing all the partial derivatives of f, such that $|\Psi'| = O(|\Psi|)$. In section 3, we apply the method of Baur and Strassen for syntactically multilinear arithmetic circuits. We show that if Ψ is syntactically multilinear, then Ψ' is syntactically multilinear as well. Furthermore, every variable $x \in X$ does not occur in the computation of $\frac{\partial f}{\partial x}$ in Ψ'.

In section 5, we use the results of section 3 to show that the rank of the partial derivative matrix of a polynomial computed by a "small" syntactically multilinear arithmetic circuit is not full (see Theorem 5.1). We use techniques that were previously used in [5] and [6] together with some new ideas. In particular, we use the partial derivative method of Nisan and Wigderson and the partial derivative matrix of Nisan. We mainly study the rank of the partial derivative matrix. We also use the notion of unbalanced gates and the notion of partitions of the variables.

In section 6, we construct a multilinear polynomial f such that the rank of the partial derivative matrix of f is full. As in [6], to show that the partial derivative matrix of f has full rank, we think of f as a polynomial over some extension field. We also show that f is explicit in the sense that f is in the class VNP, which is Valiant's algebraic analogue of NP (see section 6.3 for more details).

Our lower bound follows from sections 5 and 6 since the rank of the partial derivative matrix of a polynomial computed by a "small" syntactically multilinear arithmetic circuit is not full, and since the rank of the partial derivative matrix of f, the polynomial defined in section 6, is full, it follows that any syntactically multilinear arithmetic circuit computing f is "large."

2. Preliminaries.

2.1. Notation. For an integer $n \in \mathbb{N}$, denote $[n] = \{1, \ldots, n\}$. For a set $B \subseteq [n]$, denote $\overline{B} = [n] \setminus B$, the complement set of B. Similarly, for a set of variables X, and a set $X' \subseteq X$, denote $\overline{X'} = X \setminus X'$, the complement set of X'. For a polynomial f in

the set of variables X, and a variable $x \in X$, denote by $d_x(f)$ the degree of x in f. We say that x *occurs* in f if $d_x(f) > 0$.

2.2. Arithmetic circuits—some more definitions. Let Φ be an arithmetic circuit over a field $\mathbb{F}$ and a set of variables X. For a variable $x \in X$, we say that x *occurs* in Φ if x labels one of the input gates of Φ. Recall that, for a gate v in Φ, we defined Φ_v to be the subcircuit of Φ rooted at v. For a gate v in Φ, define X_v to be the set of variables that occur in Φ_v. That is,

$$X_v = \begin{cases} \emptyset, & v \text{ is an input gate labeled by a field element,} \\ \{x\}, & v \text{ is an input gate labeled by a variable } x \in X, \\ X_{v_1} \cup X_{v_2}, & v \text{ has sons } v_1 \text{ and } v_2 \end{cases}$$

(if u is a son of v, then $X_u \subseteq X_v$). For a variable $x \in X$ and a gate v in Φ, define $d_x(v)$, the *algebraic degree* of x in v, to be the degree of x in the polynomial $\widehat{\Phi}_v$. For a variable $x \in X$ and a gate v in Φ, define $sd_x(v)$, the *syntactic degree* of x in v, to be the degree of x in v when one ignores cancellations of monomials (in other words, if all the constants in Φ are replaced by 1's, and the field $\mathbb{F}$ is replaced by $\mathbb{R}$, then the syntactic degree of x in v is the algebraic degree of x in v). More precisely, define $sd_x(v)$ inductively as follows: If v is an input gate labeled by $\alpha \in \mathbb{F} \cup X$, then

$$sd_x(v) = \begin{cases} 1, & \alpha = x, \\ 0, & \alpha \neq x. \end{cases}$$

If v is a product gate with sons v_1 and v_2, then $sd_x(v) = sd_x(v_1) + sd_x(v_2)$. If v is an addition gate with sons v_1 and v_2, then $sd_x(v) = \max(sd_x(v_1), sd_x(v_2))$. The syntactic degree is a nondecreasing function, while the algebraic degree can decrease (in addition gates). That is, for every variable $x \in X$ and gates u and v in Φ, if there exists a directed path from u to v in Φ, then $sd_x(u) \leq sd_x(v)$. On the other hand, if v is an addition gate, and u is a son of v, it might be the case that $d_x(u) > d_x(v)$.

An arithmetic circuit Φ is called a *multilinear* arithmetic circuit if the polynomial computed at each gate in Φ is multilinear; that is, if for all $x \in X$ and v in Φ, it holds that $d_x(v) \leq 1$. An arithmetic circuit Φ is called a *syntactically multilinear* arithmetic circuit if, for all $x \in X$ and v in Φ, it holds that $sd_x(v) \leq 1$. By induction, for all $x \in X$ and v in Φ, it holds that $d_x(v) \leq sd_x(v)$. Hence, indeed, every syntactically multilinear arithmetic circuit is a multilinear arithmetic circuit as well.

2.3. Partial derivatives. Let f be a polynomial over the field $\mathbb{F}$ and the set of variables $X = \{x_1, \ldots, x_n\}$. For $i \in [n]$, define $\frac{\partial f}{\partial x_i}$, the *partial derivative of f with respect to x_i*, as follows: If f is a monomial in the variables $X \setminus \{x_i\}$, then $\frac{\partial f}{\partial x_i} = 0$. If f is a monomial of the form $f = x_i^d g$, where d is a positive integer, and g is a monomial in the variables $X \setminus \{x_i\}$, then

$$\frac{\partial f}{\partial x_i} = \frac{\partial (x_i^d g)}{\partial x_i} = \underbrace{(1 + 1 + \cdots + 1)}_{d \text{ times}} x_i^{d-1} g.$$

If f is a sum of monomials $f = \sum_j m_j$, where m_j is a monomial in $\mathbb{F}[X]$, then $\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial m_j}{\partial x_i}$.

The following lemma is known as the *chain rule of partial derivatives* (we state the lemma without giving a proof).

LEMMA 2.1. *Let g be a polynomial over the field $\mathbb{F}$ and the set of variables $X = \{x_1, \ldots, x_n\}$. Let h be a polynomial over the field $\mathbb{F}$ and the set of variables $X \cup \{x_0\}$. Let $f \in \mathbb{F}[X]$ be the polynomial h after substituting x_0 by g; that is, $f = h|_{x_0=g}$. Then, for all $i \in [n]$,*

$$\frac{\partial f}{\partial x_i} = \left.\frac{\partial h}{\partial x_i}\right|_{x_0=g} + \left.\frac{\partial h}{\partial x_0}\right|_{x_0=g} \frac{\partial g}{\partial x_i}.$$

2.4. Multiplication of variables in an arithmetic circuit. Let Ψ be an arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$. For a variable $x_i \in X$ and a product gate v in Ψ with sons v_1 and v_2, define $\mathcal{M}_v(x_i)$, the *set of variables multiplying x_i in v*, by

$$\mathcal{M}_v(x_i) = \begin{cases} \emptyset, & x_i \notin X_{v_1},\ x_i \notin X_{v_2}, \\ X_{v_2}, & x_i \in X_{v_1},\ x_i \notin X_{v_2}, \\ X_{v_1}, & x_i \notin X_{v_1},\ x_i \in X_{v_2}, \\ X_{v_1} \cup X_{v_2}, & x_i \in X_{v_1},\ x_i \in X_{v_2}. \end{cases}$$

For a variable $x_i \in X$, define $\mathcal{M}_\Psi(x_i)$, the *set of variables multiplying x_i in Ψ*, by

$$\mathcal{M}_\Psi(x_i) = \bigcup_v \mathcal{M}_v(x_i),$$

where the union is over all product gates v in Ψ. For two variables $x_i, x_j \in X$, if x_i multiplies x_j in Ψ, then x_j multiplies x_i in Ψ, and vice versa; that is,

$$x_i \in \mathcal{M}_\Psi(x_j) \Leftrightarrow x_j \in \mathcal{M}_\Psi(x_i).$$

Thus, for two variables $x_i, x_j \in X$, we say that x_i and x_j are multiplied in Ψ if $x_i \in \mathcal{M}_\Psi(x_j)$. Note that the following are equivalent:

- Ψ is a syntactically multilinear arithmetic circuit over the set of variables X.
- Ψ is an arithmetic circuit over the set of variables X such that every $x_i \in X$ admits $x_i \notin \mathcal{M}_\Psi(x_i)$.

2.5. The partial derivative matrix. Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$ be two sets of variables. Let $f \in \mathbb{G}[Y, Z]$ be a multilinear polynomial over the field $\mathbb{G}$ and the variables Y and Z. Define L_f to be the $2^m \times 2^m$ *partial derivative matrix* of f as follows: for $p \in \mathbb{G}(Y)$, a monic[1] multilinear monomial in Y, and $q \in \mathbb{G}(Z)$, a monic multilinear monomial in Z, define $L_f(p, q)$ to be the coefficient of the monomial $p \cdot q$ in f. Thus, the rows of L_f correspond to monic multilinear monomials in Y, and the columns of L_f correspond to monic multilinear monomials in Z. We are mainly interested in the rank of the partial derivative matrix.

The following propositions bound the rank of the partial derivative matrix in different cases.

PROPOSITION 2.2. *Let $f \in \mathbb{G}[Y, Z]$ be a multilinear polynomial over the field $\mathbb{G}$ and the sets of variables $Y' \subseteq Y$ and $Z' \subseteq Z$. Let $a = \min(|Y'|, |Z'|)$. Then,*

$$\mathrm{Rank}(L_f) \leq 2^a.$$

Proof. There are two cases:

[1] A monic monomial is a monomial whose coefficient is 1.

1. If $|Y'| \leq |Z'|$, then $a = |Y'|$. Thus, L_f has at most 2^a nonzero rows.
2. If $|Y'| > |Z'|$, then $a = |Z'|$. Thus, L_f has at most 2^a nonzero columns. ∎

PROPOSITION 2.3. *Let $f_1, f_2 \in \mathbb{G}[Y, Z]$ be two multilinear polynomials over the field $\mathbb{G}$ and the sets of variables Y and Z. Then,*

$$\mathrm{Rank}(L_{f_1+f_2}) \leq \mathrm{Rank}(L_{f_1}) + \mathrm{Rank}(L_{f_2}).$$

Proof. Note that $L_{f_1+f_2} = L_{f_1} + L_{f_2}$. For any two matrices A and B, it holds that $\mathrm{Rank}(A + B) \leq \mathrm{Rank}(A) + \mathrm{Rank}(B)$. Thus,

$$\mathrm{Rank}(L_{f_1+f_2}) \leq \mathrm{Rank}(L_{f_1}) + \mathrm{Rank}(L_{f_2}). \quad ∎$$

PROPOSITION 2.4. *Let $f_1 \in \mathbb{G}[Y, Z]$ be a multilinear polynomial over the field $\mathbb{G}$ and the sets of variables $Y_1 \subseteq Y$ and $Z_1 \subseteq Z$. Let $f_2 \in \mathbb{G}[Y, Z]$ be a multilinear polynomial over the field $\mathbb{G}$ and the sets of variables $Y_2 \subseteq Y$ and $Z_2 \subseteq Z$. Assume that $Y_1 \cap Y_2 = \emptyset$ and $Z_1 \cap Z_2 = \emptyset$. Then,*

$$\mathrm{Rank}(L_{f_1 \cdot f_2}) = \mathrm{Rank}(L_{f_1}) \cdot \mathrm{Rank}(L_{f_2}).$$

Proof. We think of L_{f_1} as a matrix of size $2^{|Y_1|} \times 2^{|Z_1|}$ and not of size $2^{|Y|} \times 2^{|Z|}$ (an entry in L_{f_1} that corresponds to a monomial that is not in the variables Y_1 and Z_1 is zero). Similarly, we think of L_{f_2} as a matrix of size $2^{|Y_2|} \times 2^{|Z_2|}$, and we think of $L_{f_1 \cdot f_2}$ as a matrix of size $2^{|Y_1 \cup Y_2|} \times 2^{|Z_1 \cup Z_2|}$. Since $Y_1 \cap Y_2 = \emptyset$ and $Z_1 \cap Z_2 = \emptyset$,

$$L_{f_1 \cdot f_2} = L_{f_1} \otimes L_{f_2},$$

where $\otimes$ denotes a tensor product of matrices. For any two matrices A and B, it holds that $\mathrm{Rank}(A \otimes B) = \mathrm{Rank}(A) \cdot \mathrm{Rank}(B)$. Thus,

$$\mathrm{Rank}(L_{f_1 \cdot f_2}) = \mathrm{Rank}(L_{f_1}) \cdot \mathrm{Rank}(L_{f_2}). \quad ∎$$

PROPOSITION 2.5. *Let $f \in \mathbb{G}[Y, Z]$ be a multilinear polynomial over the field $\mathbb{G}$ and the sets of variables Y and Z, where $|Y| = |Z| = m$. Let $t \in Y \cup Z$ be a variable, and let $g = \frac{\partial f}{\partial t}$. Assume that $\mathrm{Rank}(L_f) = 2^m$. Then,*

$$\mathrm{Rank}(L_g) = 2^{m-1}.$$

Proof. Assume without loss of generality that $t \in Z$. Assume without loss of generality that the columns of L_f are ordered such that $L_f = (A\ B)$, where A is a $2^m \times 2^{m-1}$ matrix whose columns correspond to all monomials q in which t does not occur, and B is a $2^m \times 2^{m-1}$ matrix whose columns correspond to all monomials q in which t occurs. Since $g = \frac{\partial f}{\partial t}$, we have $L_g = (B\ 0)$ (where 0 is a $2^m \times 2^{m-1}$ matrix of zeros). Since L_f is of full rank, the rank of B is 2^{m-1}. Hence, $\mathrm{Rank}(L_g) = 2^{m-1}$. ∎

PROPOSITION 2.6. *Let $f \in \mathbb{G}[Y, Z]$ be a multilinear polynomial over the field $\mathbb{G}$ and the sets of variables Y and Z, where $|Y| = |Z| = m$. Assume that the total degree of f is at most $T \in \mathbb{N}$. Then,*

$$\mathrm{Rank}(L_f) \leq 2^{(T+1)\log m}.$$

Proof. Since the total degree of f is at most T, there are at most $\sum_{i=0}^{T} \binom{m}{i} \leq 2^{(T+1)\log m}$ nonzero rows in L_f. ∎

2.6. Unbalanced gates. Let Ψ be an arithmetic circuit over the field $\mathbb{G}$ and the variables $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$. Let v be a gate in Ψ. Define Y_v to be the set of Y variables that occur in Ψ_v and Z_v to be the set of Z variables that occur in Ψ_v. Define $b(v)$ to be the *average* of $|Y_v|$ and $|Z_v|$; that is, $b(v) = (|Y_v| + |Z_v|)/2$. Define $a(v)$ to be the *minimum* of $|Y_v|$ and $|Z_v|$; that is, $a(v) = \min(|Y_v|, |Z_v|)$. Define $d(v)$, the *balance gauge* of v, by $d(v) = b(v) - a(v)$. For an integer $k \in \mathbb{N}$, the gate v is called *k-unbalanced* if $d(v) \geq k$. Note that if Ψ is a syntactically multilinear arithmetic circuit, and u is a product gate in Ψ with sons u_1 and u_2, then $b(u) = b(u_1) + b(u_2)$.

3. Computing partial derivatives of syntactically multilinear arithmetic circuits. Let f be a polynomial over the field $\mathbb{G}$ and the variables $X = \{x_1, \ldots, x_n\}$. In [1], Baur and Strassen showed that the complexity of computing all n partial derivatives of f is (up to a constant factor) not more than computing f. More precisely, given an arithmetic circuit Ψ computing f, one can construct an arithmetic circuit Ψ' computing $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$ such that $|\Psi'| = O(|\Psi|)$ (moreover, the depth[2] of Ψ' is up to a constant factor the same as the depth of Ψ). Later, Morgenstern [3] simplified the construction of such a Ψ'.

Let Ψ be a *syntactically multilinear* arithmetic circuit over the field $\mathbb{G}$ and the set of variables X computing f. Let Ψ' be the arithmetic circuit computing $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$ (as constructed by Baur and Strassen and Morgenstern). For every $i \in [n]$, denote by v_i the gate computing $\frac{\partial f}{\partial x_i}$ in Ψ'. Since Ψ is a syntactically multilinear arithmetic circuit, f is a multilinear polynomial. Hence, for all $i \in [n]$ the degree of x_i in $\frac{\partial f}{\partial x_i}$ is 0; that is, $d_{x_i}(v_i) = 0$. The next theorem shows the following (in addition to what Baur and Strassen showed):

- Ψ' is a *syntactically* multilinear arithmetic circuit.
- For all $i \in [n]$, the *syntactic* degree of x_i in v_i is 0; that is, $sd_{x_i}(v_i) = 0$. In other words, for all $i \in [n]$ the variable x_i does not occur in Ψ'_{v_i} (recall that Ψ'_{v_i} is the subcircuit of Ψ' rooted at v_i); that is, $x_i \notin X_{v_i}$.

THEOREM 3.1. *Let Ψ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ computing f. Then, there exists an arithmetic circuit Ψ' over the field $\mathbb{G}$ and the set of variables X such that the following hold:*

1. *Ψ' computes all n partial derivatives $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$.*
2. *$|\Psi'| \leq 5\,|\Psi|$.*
3. *Ψ' is a syntactically multilinear arithmetic circuit.*
4. *For every $i \in [n]$, it holds that $x_i \notin X_{v_i}$, where v_i is the gate computing $\frac{\partial f}{\partial x_i}$ in Ψ'.*

We defer the proof of Theorem 3.1 to section 3.1. We do not know if Theorem 3.1 holds for multilinear arithmetic circuits (that are not *syntactically* multilinear).

For the proof of our lower bound (Theorem 1.1) we need the following corollary of Theorem 3.1. The corollary shows that a gate v in Ψ' such that X_v is large (that is, many variables occur in Ψ'_v) is connected by directed paths to few output gates in Ψ' (the output gates of Ψ' are the gates computing $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$).

COROLLARY 3.2. *Let Ψ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ computing f. Let Ψ' be the arithmetic circuit computing $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$, as described in Theorem* 3.1. *For $i \in [n]$,*

[2]The depth of an arithmetic circuit Ψ is the length of the longest directed path in Ψ.

denote by v_i the gate computing $\frac{\partial f}{\partial x_i}$ in Ψ'. Then, for every gate v in Ψ',

$$|X_v| \leq n - \left|\left\{i \in [n] \ : \ v \text{ is a gate in } \Psi'_{v_i}\right\}\right|.$$

Note that $|\{i \in [n] : v \text{ is a gate in } \Psi'_{v_i}\}|$ is the number of output gates that v is connected to by directed paths in Ψ'.

Proof. Let v be a gate in Ψ'. Let $i \in [n]$ be such that there exists a directed path from v to v_i in Ψ'; that is, v is a gate in Ψ'_{v_i}. Thus, $X_v \subseteq X_{v_i}$. By property 4 of Theorem 3.1, $x_i \notin X_{v_i}$. Hence, $x_i \notin X_v$.

Therefore, $X_v \cap \{x_i \in Xv \text{ is a gate in } \Psi'_{v_i}\} = \emptyset$. Thus,

$$\begin{aligned} |X_v| &\leq |X| - \left|\left\{x_i \in X \ : \ v \text{ is a gate in } \Psi'_{v_i}\right\}\right| \\ &= n - \left|\left\{i \in [n] \ : \ v \text{ is a gate in } \Psi'_{v_i}\right\}\right|. \end{aligned}$$ □

3.1. Proof of Theorem 3.1. The following lemma is a generalization of Theorem 3.1.

LEMMA 3.3. *Let Ψ be an arithmetic circuit over the field $\mathbb{G}$ and the set of variables X computing f. Then, there exists an arithmetic circuit Ψ' over the field $\mathbb{G}$ and the set of variables X such that the following hold:*

1. *Ψ' computes all n partial derivatives $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$.*
2. *$|\Psi'| \leq 5\,|\Psi|$.*
3. *For every $i \in [n]$, it holds that $\mathcal{M}_{\Psi'}(x_i) \subseteq \mathcal{M}_{\Psi}(x_i)$.*
4. *For every $i \in [n]$, it holds that $X_{v_i} \subseteq \mathcal{M}_{\Psi}(x_i)$, where v_i is the gate computing $\frac{\partial f}{\partial x_i}$ in Ψ'.*

We defer the proof of Lemma 3.3 to section 3.2. First we give some intuition for Lemma 3.3. Let Ψ be an arithmetic circuit computing f. Let Ψ' be the arithmetic circuit computing all n partial derivatives of f (as constructed by Baur and Strassen, and Morgenstern).

- Properties 1 and 2 of Lemma 3.3 were shown by Baur and Strassen.
- Property 3 of Lemma 3.3 states that if two variables x_i and x_j in X are not multiplied in Ψ, then x_i and x_j are not multiplied in Ψ' either.
- Let $i \in [n]$. Denote by v_i the gate computing $\frac{\partial f}{\partial x_i}$ in Ψ'. Property 4 of Lemma 3.3 states that Ψ'_{v_i} depends only on variables that multiply x_i in Ψ.

Theorem 3.1 follows from Lemma 3.3.

3.2. Proof of Lemma 3.3.

Proof. Let Ψ be an arithmetic circuit over the field $\mathbb{G}$ and the set of variables X computing f. The proof of the lemma is by induction on the size of Ψ. The proof has four parts:

1. Induction base.
2. Using Ψ to define a *smaller* arithmetic circuit Φ. Using induction to conclude the existence of an arithmetic circuit Φ' that has properties 1, 2, 3, and 4.
3. Using Φ' to construct Ψ'.
4. Proving that Ψ' has all the needed properties.

Induction base. Assume that Ψ has no edges; that is, $|\Psi| = 0$. Thus, $f = \alpha$ for some $\alpha \in \mathbb{G} \cup X$. Hence, for all $i \in [n]$, it holds that $\frac{\partial f}{\partial x_i} \in \{0, 1\}$. Define Ψ' to be an arithmetic circuit with two input gates: an input gate labeled 1 and an input gate labeled 0. Then, Ψ' has properties 1, 2, 3, and 4.

Defining a smaller arithmetic circuit Φ. Let r be the gate computing f in Ψ. Assume that r is the unique output gate in Ψ (otherwise, consider Ψ_r). Assume

that Ψ has at least two edges. Let v^* in Ψ be a gate with sons v_1^* and v_2^* such that v_1^* and v_2^* are input gates. Let $\alpha_1 \in \mathbb{G} \cup X$ be the label of v_1^* in Ψ. Let $\alpha_2 \in \mathbb{G} \cup X$ be the label of v_2^* in Ψ. Let x_0 be a new variable associated with the gate v^*, and denote $X^0 = X \cup \{x_0\}$. Let X^* be the set of X variables labeling v_1^* and v_2^* in Ψ; that is, $X^* = \{\alpha_1, \alpha_2\} \cap X = X_{v^*}$. Denote by g the polynomial computed by v^* in Ψ; that is, if v^* is an addition gate in Ψ, then $g = \alpha_1 + \alpha_2$, and if v^* is a product gate in Ψ, then $g = \alpha_1 \cdot \alpha_2$.

If $X^* = \emptyset$, then define a *smaller* arithmetic circuit Φ *computing* f as follows: Φ is obtained from Ψ by deleting the edges (v_1^*, v^*) and (v_2^*, v^*), and labeling v^* (which is an input gate) by $g(\alpha_1, \alpha_2) \in \mathbb{G}$. Thus, Φ computes f, and $|\Phi| < |\Psi|$. By induction, there exists an arithmetic circuit Φ' with properties $1, 2, 3$, and 4. Set Ψ' to be Φ'. Then, Ψ' has properties $1, 2, 3$, and 4, and the proof is complete.

Hence, we can assume that $X^* \neq \emptyset$.

Let Φ be the arithmetic circuit over the field $\mathbb{G}$ and the set of variables X^0 obtained from Ψ by deleting the edges (v_1^*, v^*) and (v_2^*, v^*) and labeling v^* (which is an input gate) by x_0. For every gate v in Ψ there is a corresponding gate $u = u(v)$ in Φ, and vice versa. For a gate v in Ψ, we think of the gate $u = u(v)$ in Φ as the same gate as v.

Let $u(r)$ be the gate corresponding to r in Φ. Set h to be the polynomial computed by $u(r)$ in Φ. Thus, h is a polynomial in the variables X^0. By the construction of Φ, it follows that upon substituting x_0 by g in h we obtain f; that is, $f(X) = h(X^0)|_{x_0=g}$.

Since $|\Phi| = |\Psi| - 2$, it follows by induction that there exists an arithmetic circuit Φ' over the field $\mathbb{G}$ and the set of variables X^0 such that

1. Φ' computes all $n+1$ partial derivatives $\frac{\partial h}{\partial x_0}, \frac{\partial h}{\partial x_1}, \ldots, \frac{\partial h}{\partial x_n}$,
2. $|\Phi'| \leq 5|\Phi|$,
3. for every $j \in \{0, \ldots, n\}$, it holds that $\mathcal{M}_{\Phi'}(x_j) \subseteq \mathcal{M}_{\Phi}(x_j)$, and
4. for every $j \in \{0, \ldots, n\}$, it holds that $X^0_{u_j} \subseteq \mathcal{M}_{\Phi}(x_j)$, where u_j is the gate computing $\frac{\partial h}{\partial x_j}$ in Φ'.

Using Φ' to construct Ψ'. We construct Ψ' by adding a *few* gates and edges to Φ'.

Step 1*: Gates and edges added to* Φ' *to substitute* x_0 *by* g*, constructing* Ψ_1. Assume without loss of generality that in Φ' there is a unique input gate v' labeled x_0 (otherwise, join all input gates labeled x_0 to a single input gate labeled x_0). Denote by Ψ_1 the arithmetic circuit obtained by the following changes to Φ':

- Add two input gates to Φ': an input gate v_1' labeled α_1 and an input gate v_2' labeled α_2.
- Add two edges to Φ': the edge (v_1', v') and the edge (v_2', v').
- Label v' by the same label of v^*.

Thus, Ψ_1 is an arithmetic circuit over the field $\mathbb{G}$ and the set of variables X. Moreover, Ψ_1 is Φ' after substituting x_0 by g. We note that every gate in Φ' can also be thought of as a gate in Ψ_1.

Step 2*: Gates and edges added to* Ψ_1 *to compute* $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$. Fix $i \in [n]$. We describe what gates and edges are added to Ψ_1 in order for Ψ' to compute $\frac{\partial f}{\partial x_i}$. By Lemma 2.1 (the chain rule of partial derivatives), since $f = h|_{x_0=g}$,

$$\frac{\partial f}{\partial x_i} = \left.\frac{\partial h}{\partial x_i}\right|_{x_0=g} + \left.\frac{\partial h}{\partial x_0}\right|_{x_0=g} \frac{\partial g}{\partial x_i}. \tag{3.1}$$

As Ψ_1 is Φ' after substituting x_0 by g, it follows that $\frac{\partial h}{\partial x_i}|_{x_0=g}$ is computed by u_i (as a gate in Ψ_1), and $\frac{\partial h}{\partial x_0}|_{x_0=g}$ is computed by u_0 (as a gate in Ψ_1). Consider the following four cases.

Case 1: $\alpha_1 = x_i$ *and* $\alpha_2 \neq x_i$. Consider the following two cases:

1. If v^* is an addition gate, then $\frac{\partial g}{\partial x_i} = 1$. Hence, by (3.1),

$$\frac{\partial f}{\partial x_i} = \left.\frac{\partial h}{\partial x_i}\right|_{x_0=g} + \left.\frac{\partial h}{\partial x_0}\right|_{x_0=g}.$$

   To construct Ψ' add one gate and two edges as follows: Add an addition gate v_i, and add the edges (u_0, v_i) and (u_i, v_i). Thus, v_i computes $\frac{\partial f}{\partial x_i}$.

2. If v^* is a product gate, then $\frac{\partial g}{\partial x_i} = \alpha_2$. Hence, by (3.1),

$$\frac{\partial f}{\partial x_i} = \left.\frac{\partial h}{\partial x_i}\right|_{x_0=g} + \left.\frac{\partial h}{\partial x_0}\right|_{x_0=g} \alpha_2.$$

   Recall that, v_2' is an input gate labeled α_2 in Ψ_1. To construct Ψ' add two gates and four edges as follows:
   - A product gate w_1 and the edges (u_0, w_1) and (v_2', w_1). Thus, w_1 computes $\frac{\partial h}{\partial x_0}|_{x_0=g}\alpha_2$.
   - An addition gate v_i and the edges (u_i, v_i) and (w_1, v_i).

   Thus, v_i computes $\frac{\partial f}{\partial x_i}$.

Case 2: $\alpha_1 \neq x_i$ *and* $\alpha_2 = x_i$. We do the same as in case 1 (replacing 1 and 2). Note that in this case, when v^* is a product gate, we add a product gate w_2 to Ψ_1.

Case 3: $\alpha_1 = \alpha_2 = x_i$. Consider the following two cases:

1. If v^* is an addition gate, then $\frac{\partial g}{\partial x_i} = 1 + 1$. Hence, by (3.1),

$$\frac{\partial f}{\partial x_i} = \left.\frac{\partial h}{\partial x_i}\right|_{x_0=g} + \left.\frac{\partial h}{\partial x_0}\right|_{x_0=g} (1+1).$$

   To construct Ψ' add three gates and four edges as follows:
   - An input gate w_3 labeled by $1 + 1$.
   - A product gate w_4 and the edges (w_3, w_4) and (u_0, w_4). Thus, w_4 computes $\frac{\partial h}{\partial x_0}|_{x_0=g}(1+1)$.
   - An addition gate v_i and the edges (u_i, v_i) and (w_4, v_i).

   Thus, v_i computes $\frac{\partial f}{\partial x_i}$.

2. If v^* is a product gate, then $\frac{\partial g}{\partial x_i} = (1+1)x_i$. Hence, by (3.1),

$$\frac{\partial f}{\partial x_i} = \left.\frac{\partial h}{\partial x_i}\right|_{x_0=g} + \left.\frac{\partial h}{\partial x_0}\right|_{x_0=g} (1+1)x_i.$$

   Recall that v_2' is an input gate labeled $x_i = \alpha_2$ in Ψ_1. To construct Ψ' add four gates and six edges as follows:
   - An input gate w_3 labeled by $1 + 1$.
   - A product gate w_4 and the edges (w_3, w_4) and (v_2', w_4). Thus, w_4 computes $(1+1)x_i$.
   - A product gate w_5 and the edges (u_0, w_5) and (w_4, w_5). Thus, w_5 computes $\frac{\partial h}{\partial x_0}|_{x_0=g}(1+1)x_i$.
   - An addition gate v_i and the edges (u_i, v_i) and (w_5, v_i).

Thus, v_i computes $\frac{\partial f}{\partial x_i}$.

Case 4: $\alpha_1 \neq x_i$ *and* $\alpha_2 \neq x_i$. In this case no gates or edges are added. As g is a function of α_1 and α_2, it follows that $\frac{\partial g}{\partial x_i} = 0$. Hence, by (3.1), $\frac{\partial f}{\partial x_i} = \frac{\partial h}{\partial x_i}|_{x_0=g}$. Denote by v_i the gate u_i in Ψ'. Thus, v_i computes $\frac{\partial f}{\partial x_i}$.

Ψ' has the needed properties. Let Ψ' be the arithmetic circuit over the field $\mathbb{G}$ and the set of variables X as constructed above. The following claims show that Ψ' has the needed properties. To prove the claims we make use of the following:

- The construction of Φ from Ψ.
- The properties of Φ' (which we know by induction).
- The construction of Ψ' from Φ'.

In some of the proofs there are several cases to consider. Some of the cases are similar, but we consider all the cases for completeness.

The following claim shows that Ψ' satisfies property 1 of Theorem 3.1.

CLAIM 3.4. Ψ' *computes all* n *partial derivatives* $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$.

Proof. Let $i \in [n]$. By the construction of Ψ' there exists a gate v_i in Ψ' computing $\frac{\partial f}{\partial x_i}$. □

The following claim shows that Ψ' satisfies property 2 of Theorem 3.1.

CLAIM 3.5. $|\Psi'| \leq 5|\Psi|$.

Proof. By the construction of Φ from Ψ, it follows that $|\Phi| = |\Psi| - 2$. By induction, $|\Phi'| \leq 5|\Phi|$. By the construction of Ψ' from Φ', there are at most ten more edges in Ψ' than in Φ'; that is, $|\Psi'| \leq |\Phi'| + 10$. Hence,

$$|\Psi'| \leq 5(|\Psi| - 2) + 10 = 5|\Psi|. \qquad \square$$

Let $j \in \{0, \ldots, n\}$. Think of u_j both as the gate computing $\frac{\partial h}{\partial x_j}|_{x_0=g}$ in Ψ' and as the gate computing $\frac{\partial h}{\partial x_j}$ in Φ'. Thus, X_{u_j} is the set of X variables that occur in Ψ'_{u_j}, and $X^0_{u_j}$ is the set of X^0 variables that occur in Φ'_{u_j}. We use the following claim.

CLAIM 3.6.

1. *For every* $i \in [n]$,

$$X_{u_i} \subseteq \mathcal{M}_\Psi(x_i).$$

2. *For every variable* α *in* X^*,

$$X_{u_0} \subseteq \mathcal{M}_\Psi(\alpha).$$

Proof. For every $j \in \{0, \ldots, n\}$, by property 4 of Theorem 3.1 of Φ', $X^0_{u_j} \subseteq \mathcal{M}_\Phi(x_j)$, and by the construction of Ψ',

$$X_{u_j} = \begin{cases} X^0_{u_j}, & x_0 \notin X^0_{u_j}, \\ (X^0_{u_j} \setminus \{x_0\}) \cup X^*, & x_0 \in X^0_{u_j}. \end{cases}$$

Proof of 1: Fix $i \in [n]$. Since every variable that multiplies x_i in Φ except (possibly) x_0 also multiplies x_i in Ψ, it follows that $\mathcal{M}_\Phi(x_i) \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(x_i)$. Thus, since $X^0_{u_i} \subseteq \mathcal{M}_\Phi(x_i)$, it follows that $X^0_{u_i} \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(x_i)$. Consider the following two cases:

1. $x_0 \notin X^0_{u_i}$. Then, $X_{u_i} = X^0_{u_i} = X^0_{u_i} \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(x_i)$.
2. $x_0 \in X^0_{u_i}$. Since $X^0_{u_i} \subseteq \mathcal{M}_\Phi(x_i)$, it follows that x_0 multiplies x_i in Φ; that is, $x_0 \in \mathcal{M}_\Phi(x_i)$. Thus, $X^* \subseteq \mathcal{M}_\Psi(x_i)$. Hence, $X_{u_i} = (X^0_{u_i} \setminus \{x_0\}) \cup X^* \subseteq \mathcal{M}_\Psi(x_i)$.

Proof of 2: Fix $\alpha \in X^*$. Since every variable that multiplies x_0 in Φ except (possibly) x_0 also multiplies α in Ψ, it follows that $\mathcal{M}_\Phi(x_0) \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(\alpha)$. Thus, as $X^0_{u_0} \subseteq \mathcal{M}_\Phi(x_0)$, we have $X^0_{u_0} \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(\alpha)$. Consider the following two cases:

1. $x_0 \notin X^0_{u_0}$. Then, $X_{u_0} = X^0_{u_0} = X^0_{u_0} \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(\alpha)$.
2. $x_0 \in X^0_{u_0}$. Since $X^0_{u_0} \subseteq \mathcal{M}_\Phi(x_0)$, it follows that x_0 multiplies x_0 in Φ; that is, $x_0 \in \mathcal{M}_\Phi(x_0)$. Thus, as $\alpha \in X^*$, we have $X^* \subseteq \mathcal{M}_\Psi(\alpha)$. Hence, $X_{u_0} = (X^0_{u_0} \setminus \{x_0\}) \cup X^* \subseteq \mathcal{M}_\Psi(\alpha)$. □

The following claim shows that Ψ' satisfies property 3 of Theorem 3.1.

CLAIM 3.7. *For all $i \in [n]$, it holds that $\mathcal{M}_{\Psi'}(x_i) \subseteq \mathcal{M}_\Psi(x_i)$.*

Proof. Let $i, j \in [n]$ be such that x_i and x_j are multiplied in Ψ'; that is, $x_i \in \mathcal{M}_{\Psi'}(x_j)$. To prove the claim it is enough to show that $x_i \in \mathcal{M}_\Psi(x_j)$. By property 3 of Theorem 3.1 of Φ', for all $\ell \in \{0, \ldots, n\}$,

$$\mathcal{M}_{\Phi'}(x_\ell) \subseteq \mathcal{M}_\Phi(x_\ell).$$

Recall that X^* is the set of X variables that occur in Ψ_{v^*}. Consider two cases.

x_i and x_j are not in X^*. Assume that $x_i \notin X^*$ and $x_j \notin X^*$. Thus, as $x_i \in \mathcal{M}_{\Psi'}(x_j)$ and by the construction of Ψ', it follows that x_i and x_j are multiplied in Φ'; that is, $x_i \in \mathcal{M}_{\Phi'}(x_j)$. Thus, since $\mathcal{M}_{\Phi'}(x_j) \subseteq \mathcal{M}_\Phi(x_j)$, it follows that x_i and x_j are multiplied in Φ; that is, $x_i \in \mathcal{M}_\Phi(x_j)$. Hence, by the construction of Φ (as $x_i \neq x_0$ and $x_j \neq x_0$), it follows that x_i and x_j are multiplied in Ψ; that is, $x_i \in \mathcal{M}_\Psi(x_j)$.

At least one of x_i and x_j is in X^*. Assume without loss of generality that $x_j = \alpha_1$ (recall that $x_i \in \mathcal{M}_\Psi(x_j) \Leftrightarrow x_j \in \mathcal{M}_\Psi(x_i)$. Similar arguments hold for $x_j = \alpha_2$). Let v be a gate in Ψ' in which x_i and $x_j = \alpha_1$ are multiplied; that is, $x_i \in \mathcal{M}_v(\alpha_1)$. In the following we think of u_0 as the gate computing $\frac{\partial h}{\partial x_0}|_{x_0=g}$ in Ψ'. Consider the following cases.

Case 1. v is v'. Since $v = v'$ is a product gate in Ψ', we have that v^* is a product gate in Ψ. Since v' and v^* are the "same" gate, $\mathcal{M}_{v'}(\alpha_1) = \mathcal{M}_{v^*}(\alpha_1)$. Thus, $x_i \in \mathcal{M}_{v'}(\alpha_1) = \mathcal{M}_{v^*}(\alpha_1) \subseteq \mathcal{M}_\Psi(\alpha_1)$.

Case 2. v is w_1. Recall that w_1 is the product gate added to Ψ_1 in order for Ψ' to compute $\frac{\partial f}{\partial \alpha_1}$. The two sons of v in Ψ' are u_0 and v'_2. Since w_1 is added only if $\alpha_1 \neq \alpha_2$, it follows that $\mathcal{M}_v(\alpha_1) \subseteq X_{v'_2} \subseteq \{\alpha_2\}$. Thus, $x_i = \alpha_2$. Hence, since w_1 is added only if v^* is a product gate (that multiplies α_1 and α_2) in Ψ, $x_i = \alpha_2 \in \mathcal{M}_{v^*}(\alpha_1) \subseteq \mathcal{M}_\Psi(\alpha_1)$.

Case 3. v is w_2. Recall that w_2 is the product gate added to Ψ_1 in order for Ψ' to compute $\frac{\partial f}{\partial \alpha_2}$. The two sons of v in Ψ' are u_0 and v'_1, and v computes $\frac{\partial h}{\partial x_0}|_{x_0=g}\alpha_1$. By definition of $\mathcal{M}_v(\alpha_1)$,

$$\mathcal{M}_v(\alpha_1) = \begin{cases} X_{u_0}, & \alpha_1 \notin X_{u_0}, \\ \{\alpha_1\} \cup X_{u_0}, & \alpha_1 \in X_{u_0}. \end{cases}$$

Thus, $\mathcal{M}_v(\alpha_1) = X_{u_0}$. Hence, by Claim 3.6,

$$x_i \in X_{u_0} \subseteq \mathcal{M}_\Psi(\alpha_1).$$

Case 4. v is w_4. Recall that w_4 is added to Ψ_1 in the case that $\alpha_1 = \alpha_2$. Since one of v's sons is an input gate labeled by a field element, $\mathcal{M}_v(\alpha_1) = \emptyset$. Hence, this case cannot happen.

Case 5. v is w_5. Recall that w_5 is the product gate added to Ψ_1 in order for Ψ' to compute $\frac{\partial f}{\partial \alpha_1}$ (when $\alpha_1 = \alpha_2$). Thus, v computes $\frac{\partial h}{\partial x_0}|_{x_0=g}(1+1)\alpha_1$. By definition

of $\mathcal{M}_v(\alpha_1)$,

$$\mathcal{M}_v(\alpha_1) = \begin{cases} X_{u_0}, & \alpha_1 \notin X_{u_0}, \\ \{\alpha_1\} \cup X_{u_0}, & \alpha_1 \in X_{u_0}. \end{cases}$$

Thus, $\mathcal{M}_v(\alpha_1) = X_{u_0}$. Hence, by Claim 3.6,

$$x_i \in X_{u_0} \subseteq \mathcal{M}_\Psi(\alpha_1).$$

Case 6. v is also a product gate in Φ'. There are two cases to consider:

1. If $x_i \notin \mathcal{M}_{\Phi'}(\alpha_1)$, then as $x_i \in \mathcal{M}_{\Psi'}(\alpha_1)$, it holds that $x_i \in \mathcal{M}_{\Phi'}(x_0)$. Hence, as $\mathcal{M}_{\Phi'}(x_0) \subseteq \mathcal{M}_\Phi(x_0)$, it follows that $x_i \in \mathcal{M}_\Phi(x_0)$. Since every variable that multiplies x_0 except (possibly) x_0 in Φ also multiplies α_1 in Ψ, we have $\mathcal{M}_\Phi(x_0) \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(\alpha_1)$. Hence, as $x_i \neq x_0$,
$$x_i \in \mathcal{M}_\Psi(\alpha_1).$$
2. If $x_i \in \mathcal{M}_{\Phi'}(\alpha_1)$, then $x_i \in \mathcal{M}_{\Phi'}(\alpha_1) \subseteq \mathcal{M}_\Phi(\alpha_1)$. Since every variable that multiplies α_1 in Φ except (possibly) x_0 also multiplies α_1 in Ψ, it follows that $\mathcal{M}_\Phi(\alpha_1) \setminus \{x_0\} \subseteq \mathcal{M}_\Psi(\alpha_1)$. Thus, as $x_i \neq x_0$,
$$x_i \in \mathcal{M}_\Psi(\alpha_1). \qquad \square$$

The following claim shows that Ψ' satisfies property 4 of Theorem 3.1.

CLAIM 3.8. *For all $i \in [n]$, it holds that $X_{v_i} \subseteq \mathcal{M}_\Psi(x_i)$.*

Proof. Fix $i \in [n]$. For simplicity of notation, denote $v = v_i, u = u_i$, and $u' = u_0$. We think of u and u' both as gates in Φ' and as gates in Ψ'. By the construction of Ψ', we have to consider the following cases.

Case 1: $\alpha_1 = x_i$ *or* $\alpha_2 = x_i$. Note that this case applies both for $\alpha_1 = \alpha_2$ and $\alpha_1 \neq \alpha_2$. Assume without loss of generality that $\alpha_1 = x_i$. Consider the following two cases.

v^* is an addition gate in Ψ. Since u and u' are the sons of v in Ψ', it follows that $X_v = X_u \cup X_{u'}$. By Claim 3.6, it follows that $X_u \subseteq \mathcal{M}_\Psi(x_i)$ and $X_{u'} \subseteq \mathcal{M}_\Psi(\alpha_1) = \mathcal{M}_\Psi(x_i)$. Hence,

$$X_v \subseteq \mathcal{M}_\Psi(x_i).$$

v^* is a product gate. By the construction of Ψ' (loosely speaking, the sons of v in Ψ' are u, u', and v'_2), it follows that $X_v = X_u \cup X_{u'} \cup X_{v'_2}$. As v^* is a product gate in Ψ (v^* multiplies $x_i = \alpha_1$ and α_2), it follows that $X_{v'_2} \subseteq \mathcal{M}_\Psi(x_i)$. By Claim 3.6, it follows that $X_u \subseteq \mathcal{M}_\Psi(x_i)$ and $X_{u'} \subseteq \mathcal{M}_\Psi(\alpha_1) = \mathcal{M}_\Psi(x_i)$. Hence,

$$X_v \subseteq \mathcal{M}_\Psi(x_i).$$

Case 2: $\alpha_1 \neq x_i$ *and* $\alpha_2 \neq x_i$. By the construction of Ψ', it follows that $X_v = X_u$. By Claim 3.6, it follows that $X_u \subseteq \mathcal{M}_\Psi(x_i)$. Hence,

$$X_v \subseteq \mathcal{M}_\Psi(x_i). \qquad \square$$

Thus, Ψ' is an arithmetic circuit over the field $\mathbb{G}$ and the set of variables X such that the following hold:

1. By Claim 3.4, Ψ' computes all n partial derivatives $\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$.
2. By Claim 3.5, $|\Psi'| \leq 5 \cdot |\Psi|$.
3. By Claim 3.7, for every $i \in [n]$, it holds that $\mathcal{M}_{\Psi'}(x_i) \subseteq \mathcal{M}_\Psi(x_i)$.
4. By Claim 3.8, for every $i \in [n]$, it holds that $X_{v_i} \subseteq \mathcal{M}_\Psi(x_i)$. $\square$

4. Partitions of the variables of an arithmetic circuit. In this section we define a distribution $\mathcal{D}$ on partitions of the variables of an arithmetic circuit. We show that by the distribution $\mathcal{D}$, a specific gate is unbalanced with high probability.

4.1. Definitions. Let $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$, and $Z = \{z_1, \ldots, z_m\}$ be three sets of variables (where $n = 2m$). A one-to-one function $A : X \to Y \cup Z$ is called a *partition* of X to Y and Z. For a partition A of X to Y and Z and $X' \subseteq X$ a subset of X, denote $A(X') = \{A(x) \ : \ x \in X'\}$.

Let $X_1 \subset X$ be a subset of X of size $n/4$. The distribution $\mathcal{D}(X_1)$ on partitions A of X to Y and Z is the uniform distribution on all partitions A such that $A(X_1) \subset Y$. We write $A \sim \mathcal{D}(X_1)$ if A is a partition of X to Y and Z chosen by the distribution $\mathcal{D}(X_1)$.

Let Ψ be an arithmetic circuit over the field $\mathbb{G}$ and the set of variables X computing a polynomial f. Let A be a partition of X to Y and Z. Denote by Ψ_A the arithmetic circuit Ψ after substituting every $x \in X$ by $A(x) \in Y \cup Z$. Denote by f^A the polynomial f after substituting every $x \in X$ by $A(x) \in Y \cup Z$. Note that Ψ_A is an arithmetic circuit over the field $\mathbb{G}$ and the sets of variables Y and Z computing f^A. For every gate in Ψ there is a corresponding gate in Ψ_A, and vice versa. We think of a gate v in Ψ and v's corresponding gate in Ψ_A as the same gate, and we denote both of them by v.

4.2. The probability that a gate is unbalanced. The following proposition bounds the probability that a gate (with a certain number of variables) is unbalanced.

PROPOSITION 4.1. *Let Ψ be an arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$. Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$ be two sets of variables (where $n = 2m$ and m is even). Let $X_1 \subset X$ be a subset of X of size $n/4$. Let $A \sim \mathcal{D}(X_1)$ be a random partition of X to Y and Z such that $A(X_1) \subset Y$. Let β be such that $0 < \beta < 1$, and let v be a gate in Ψ such that $n^\beta < |X_v| < n - n^\beta$. Then, for any integer $k \in \mathbb{N}$,*

$$\Pr_{A \sim \mathcal{D}(X_1)}[v \text{ is not } k\text{-unbalanced in } \Psi_A] = O\left(kn^{-\beta/2}\right).$$

To prove Proposition 4.1 we need some property of the hypergeometric distribution. We defer the proof of Proposition 4.1 to section 4.4.

4.3. The hypergeometric distribution. Let $N, M_1, M_2 \in \mathbb{N}$ be three integers such that $M_1 \leq N$ and $M_2 \leq N$. Denote by $\mathcal{H}(N, M_1, M_2)$ the hypergeometric distribution with parameters M_1, M_2, and N; that is, $\mathcal{H}(N, M_1, M_2)$ is the distribution of $|S_1 \cap S_2|$, where S_1 is a random subset of $[N]$ of size M_1 (chosen uniformly at random from all subsets of $[N]$ of size M_1), and S_2 is a fixed subset of $[N]$ of size M_2.

The following proposition shows that a hypergeometric random variable does not take any specific value with high probability (for a certain range of the parameters).

PROPOSITION 4.2. *Let $n \in \mathbb{N}$ be an integer such that $n/4$ is an integer as well. Let β be such that $0 < \beta < 1$, and let χ be a random variable that has the hypergeometric distribution $\mathcal{H}(3n/4, n/4, M)$, where*

$$n^\beta/4 < M < 3n/4 - n^\beta/4. \tag{4.1}$$

Then, every $j \in \mathbb{N}$ admits $\Pr[\chi = j] = O(n^{-\beta/2})$.

Proof. Denote $j_{\max} = \min(M, n/4)$ the maximal value that χ takes. For every

$j \in \mathbb{N}$, denote $P(j) = \Pr[\chi = j]$. Thus, for every $j \in \{0, \ldots, j_{\max}\}$, we have

$$P(j) = \frac{\binom{M}{j}\binom{3n/4-M}{n/4-j}}{\binom{3n/4}{n/4}}, \tag{4.2}$$

and for every $j \notin \{0, \ldots, j_{\max}\}$, we have $P(j) = 0$. Set $j^* \in \{0, \ldots, j_{\max}\}$ to be the integer that maximizes $P(j)$; that is, every $j \in \mathbb{N}$ admits $P(j) \leq P(j^*)$. To find j^* consider $P(j+1)/P(j)$. By (4.2), for all $j \in \{0, \ldots, j_{\max} - 1\}$,

$$\frac{P(j+1)}{P(j)} = \frac{(M-j)(n/4-j)}{(j+1)(n/2-M+j+1)},$$

which implies that

$$P(j) \leq P(j+1) \Leftrightarrow j \leq \frac{Mn/4 + M - n/2 - 1}{3n/4+2} = \frac{M}{3} + \frac{M - 3n/2 - 3}{3(3n/4+2)}.$$

Since $0 < M < 3n/4$, we have $-1 < (M - 3n/2 - 3)/(3(3n/4+2)) < 0$. Thus, $j^* \in \{\lfloor M/3 \rfloor, \lceil M/3 \rceil\}$.

Using Stirling's formula, we have

$$\binom{N}{\lfloor N/3 \rfloor} = \Theta\left(\frac{1}{\sqrt{N}} 2^{N \cdot H(1/3)}\right) \text{ and } \binom{N}{\lceil N/3 \rceil} = \Theta\left(\frac{1}{\sqrt{N}} 2^{N \cdot H(1/3)}\right), \tag{4.3}$$

where $H(1/3) = -(1/3)\log_2(1/3) - (2/3)\log_2(2/3)$. Hence, by (4.2), using (4.3) for N equals $M, 3n/4 - M$, and $3n/4$,

$$\begin{aligned} P(j^*) &= \frac{\Theta\left(\frac{1}{\sqrt{M}} 2^{M \cdot H(1/3)}\right) \Theta\left(\frac{1}{\sqrt{3n/4-M}} 2^{(3n/4-M) \cdot H(1/3)}\right)}{\Theta\left(\frac{1}{\sqrt{3n/4}} 2^{(3n/4) \cdot H(1/3)}\right)} \\ &= \Theta\left(\frac{\sqrt{3n/4}}{\sqrt{M}\sqrt{3n/4-M}}\right) = \Theta\left(n^{-\beta/2}\right), \end{aligned}$$

where the last equality follows from (4.1). Hence, every $j \in \mathbb{N}$ admits $P(j) \leq P(j^*) = O(n^{-\beta/2})$. ∎

4.4. Proof of Proposition 4.1.

Proof. If $k > n^\beta/4$, then the proposition holds (as $kn^{-\beta/2} > 1$). Thus, assume that $k \leq n^\beta/4$. Let $A \sim \mathcal{D}(X_1)$ be a random partition of X to Y and Z such that $A(X_1) \subset Y$. By the definition of $\mathcal{D}(X_1)$, we think of A as obtained by the following randomized process: let X_2 be a random subset of $\overline{X_1} = X \setminus X_1$ of size $n/4$; then let A be a random partition such that $A(X_1 \cup X_2) = Y$. Recall that Y_v is the set of Y variables that occur in the subcircuit of Ψ_A rooted at v. Thus, $|Y_v| = |X_v \cap X_1| + |X_v \cap X_2|$. Hence,

$$|X_v \cap X_1| \leq |Y_v| \leq |X_v \cap X_1| + n/4.$$

There are three cases to consider. In the first two cases X_v either has small intersection with $\overline{X_1}$ or has large intersection with $\overline{X_1}$, and then v is always unbalanced. In the third case we use Proposition 4.2 to show that v is unbalanced with high probability.

Case 1. Assume $\left|X_v \cap \overline{X_1}\right| \leq n^\beta/4$. Then, $|X_v| = \left|X_v \cap \overline{X_1}\right| + |X_v \cap X_1| \leq n^\beta/4 + |X_v \cap X_1|$. Thus, since $|X_v| \geq n^\beta$, it follows that $|Y_v| \geq |X_v \cap X_1| \geq |X_v| - n^\beta/4 \geq |X_v|/2 + n^\beta/4$. Hence, $a(v) \leq |Z_v| = |X_v| - |Y_v| \leq |X_v|/2 - n^\beta/4$. Thus, $d(v) = |X_v|/2 - a(v) \geq n^\beta/4$. Therefore, since $k \leq n^\beta/4$, it follows that v is not k-unbalanced in Ψ_A with probability 0.

Case 2. Assume $\left|X_v \cap \overline{X_1}\right| \geq 3n/4 - n^\beta/4$. Then, $|X_v| = \left|X_v \cap \overline{X_1}\right| + |X_v \cap X_1| \geq 3n/4 - n^\beta/4 + |X_v \cap X_1|$. Hence, $a(v) \leq |Y_v| \leq |X_v \cap X_1| + n/4 \leq |X_v| - n/2 + n^\beta/4$. Thus, since $|X_v| \leq n - n^\beta$, it follows that $d(v) = |X_v|/2 - a(v) \geq -|X_v|/2 + n/2 - n^\beta/4 \geq n^\beta/4$. Thus, since $k \leq n^\beta/4$, it follows that v is not k-unbalanced in Ψ_A with probability 0.

Case 3. Assume

$$n^\beta/4 < \left|X_v \cap \overline{X_1}\right| < 3n/4 - n^\beta/4. \tag{4.4}$$

Denote $Y_1 = X_v \cap X_1$ and $Y_2 = X_v \cap X_2$. Thus, $|Y_v| = |Y_1| + |Y_2|$. Note that Y_1 is a fixed subset of X, and $|Y_2|$ has the hypergeometric distribution $\mathcal{H}(|\overline{X_1}|, n/4, |X_v \cap \overline{X_1}|)$. Thus, by (4.4) and Proposition 4.2, $|Y_2|$ takes any specific value with probability $O(n^{-\beta/2})$. Denote $\mu = |X_v|/2$. Hence,

$$\begin{aligned}
\Pr_{A \sim \mathcal{D}(X_1)}[v \text{ is not } k\text{-unbalanced in } \Psi_A] &\leq \Pr_{A \sim \mathcal{D}(X_1)}[\mu - k \leq |Y_v| \leq \mu + k] \\
&\leq \sum_{j=\lfloor \mu - k \rfloor}^{\lceil \mu + k \rceil} \Pr_{A \sim \mathcal{D}(X_1)}[|Y_v| = j] \\
&= O\left(k n^{-\beta/2}\right). \qquad \square
\end{aligned}$$

5. Small syntactically multilinear arithmetic circuits compute polynomials of low rank. In this section we prove that a small syntactically multilinear arithmetic circuit computes a polynomial whose partial derivative matrix is not of full rank (for some partition of the variables). Formally, we have the following theorem.

THEOREM 5.1. *Let Ψ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ computing f. Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$ be two sets of variables (where $n = 2m$ and m is even). If for all partitions A of X to Y and Z*

$$\operatorname{Rank}\left(L_{f^A}\right) = 2^m,$$

then

$$|\Psi| = \Omega\left(\frac{n^{4/3}}{\log^2 n}\right).$$

The rest of this section is devoted for the proof of Theorem 5.1. In section 5.1 we introduce the notion of leveled gates and state a lemma. In section 5.2 we use the lemma to prove Theorem 5.1.

5.1. Few leveled gates means low rank. Let Φ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the variables $X = \{x_1, \ldots, x_n\}$. Fix $\tau = 3 \log n$. Define $\mathcal{L}(\Phi, \tau)$, the set of *lower leveled* gates in Φ, as $\mathcal{L}(\Phi, \tau)$, the set of all gates u in Φ, such that $2\tau < |X_u| < n - 2\tau$, and u has a father u' such that $|X_{u'}| \geq n - 2\tau$.

The following lemma shows that, if the set of lower leveled gates in a circuit is small, then the partial derivative matrix of a polynomial computed by the circuit is not of full rank (for some partition of the variables).

LEMMA 5.2. *Let Φ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ computing f. Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$ be two sets of variables (where $n = 2m$ and m is even). Let $\tau = 3 \log n$, and let $\mathcal{L} = \mathcal{L}(\Phi, \tau)$ be the set of lower leveled gates in Φ (as defined above). Let $c > 0$ be a small enough constant ($c = 1/1000$ suffices). Assume $|\mathcal{L}| < \frac{c}{\tau} n^{1/3}$. Then, there exists a partition A of X to Y and Z such that*

$$\operatorname{rank}(L_{f^A}) < 2^{m-1}.$$

We defer the proof of Lemma 5.2 to section 5.4. We use Lemma 5.2 to prove Theorem 5.1.

5.2. Proof of Theorem 5.1.

Proof. Let Ψ' be the arithmetic circuit computing all n partial derivatives of f given by Theorem 3.1. Let $\tau = 3 \log n$, and let $\mathcal{L} = \mathcal{L}(\Psi', \tau)$ be the set of lower leveled gates in Ψ' (as defined in section 5.1). Define $\mathcal{U} = \mathcal{U}(\Psi', \tau)$, the set of *upper leveled* gates in Ψ', by

$$\mathcal{U} = \{u' \text{ is a gate in } \Psi' \ : \ n - 2\tau \leq |X_{u'}| \text{ and } u' \text{ has a son } u \text{ such that } u \in \mathcal{L}\}.$$

To prove the theorem, we will bound from below the size of $\mathcal{U}$.

Let $i \in [n]$. Set $g_i = \frac{\partial f}{\partial x_i}$. Let v_i be the gate computing g_i in Ψ'. Denote by Ψ'_i the arithmetic circuit Ψ'_{v_i}. Define $\mathcal{L}_i = \mathcal{L}(\Psi'_i, \tau)$ to be the set of lower leveled gates in Ψ'_i. The following claim gives two properties of $\mathcal{L}_i$.

CLAIM 5.3. *For every $i \in [n]$,*

1. $\mathcal{L}_i \subseteq \mathcal{L}$.
2. $|\mathcal{L}_i| \geq \frac{c}{\tau} n^{1/3}$, *where c is the constant from Lemma* 5.2.

Proof. Proof of 1: Note that for every gate u in Ψ'_i, the set X_u in Ψ' and in Ψ'_i is the same set. Let $u \in \mathcal{L}_i$. Thus, $2\tau < |X_u| < n - 2\tau$ and u has a father u' in Ψ'_i such that $|X_{u'}| \geq n - 2\tau$. Since u' is a father of u in Ψ', we have $u \in \mathcal{L}$. Hence, $\mathcal{L}_i \subseteq \mathcal{L}$.

Proof of 2: For every partition A of X to Y and Z, we have $g_i^A = (\frac{\partial f}{\partial x_i})^A = \frac{\partial f^A}{\partial A(x_i)}$, which implies using Proposition 2.5 (since L_{f^A} is of full rank) that $\operatorname{Rank}(L_{g_i^A}) = 2^{m-1}$. Hence, by Lemma 5.2, since Ψ'_i computes g_i, $|\mathcal{L}_i| \geq \frac{c}{\tau} n^{1/3}$, where c is the constant from Lemma 5.2. ∎

For a gate v in Ψ', define

$$C_v = |\{i \in [n] \ : \ v \text{ is a gate in } \Psi'_i\}|.$$

For $i \in [n]$, define

$$\mathcal{U}_i = \{u' \in \mathcal{U} \ : \ u' \text{ is a gate in } \Psi'_i\}.$$

Thus, for all $i \in [n]$, we have $\mathcal{U}_i \subseteq \mathcal{U}$. Hence,

$$\sum_{i \in [n]} |\mathcal{U}_i| = |\{(u', i) \ : \ u' \in \mathcal{U} \text{ and } i \in [n] \text{ are such that } u' \text{ is a gate in } \Psi'_i\}|$$

$$= \sum_{u' \in \mathcal{U}} C_{u'}. \tag{5.1}$$

Let $i \in [n]$. By property 1 of Claim 5.3, $\mathcal{L}_i \subseteq \mathcal{L}$. Hence, for every gate $u \in \mathcal{L}_i$, there is a corresponding gate $u' \in \mathcal{U}_i$, which is a father of u. Thus, since the in-degree of the gates in $\mathcal{U}_i$ is 2, we have

$$|\mathcal{L}_i| \leq 2|\mathcal{U}_i|. \tag{5.2}$$

Recall that, for all $u' \in \mathcal{U}$, it holds that $|X_{u'}| \geq n - 2\tau$. Thus, by Corollary 3.2, every $u' \in \mathcal{U}$ admits $C_{u'} \leq n - |X_{u'}| \leq n - (n - 2\tau) = 2\tau$. Thus, by (5.2), (5.1), and property 2 of Claim 5.3,

$$\frac{c}{\tau} n^{4/3} \leq \sum_{i \in [n]} |\mathcal{L}_i| \leq 2 \sum_{i \in [n]} |\mathcal{U}_i| = 2 \sum_{u' \in \mathcal{U}} C_{u'} \leq 2|\mathcal{U}| \cdot 2\tau.$$

Hence, by property 2 of Theorem 3.1, since $\tau = 3 \log n$,

$$|\Psi| = \Omega\left(|\Psi'|\right) = \Omega\left(|\mathcal{U}|\right) = \Omega\left(\frac{n^{4/3}}{\log^2 n}\right). \qquad \square$$

5.3. Unbalancing the lower leveled gates of a small arithmetic circuit. In the rest of this section we prove Lemma 5.2. First we prove that the set of lower leveled gates in a small arithmetic circuit can be made simultaneously unbalanced.

PROPOSITION 5.4. *Under the same assumptions as in Lemma* 5.2*, there exists a partition A of X to Y and Z such that every $u \in \mathcal{L}$ is τ-unbalanced in Φ_A.*

Proof. For a gate v in Φ define $\tilde{X}_v$ by

$$\tilde{X}_v = \begin{cases} X_v, & |X_v| \leq n/2, \\ X \setminus X_v, & |X_v| > n/2. \end{cases}$$

Every partition A of X to Y and Z defines a partition of $\tilde{X}_v$ to $\tilde{Y}_v$ and $\tilde{Z}_v$:

$$\tilde{Y}_v = \left\{ y \in Y \ : \ A^{-1}(y) \in \tilde{X}_v \right\} \text{ and } \tilde{Z}_v = \left\{ z \in Z \ : \ A^{-1}(z) \in \tilde{X}_v \right\}.$$

For every partition A of X to Y and Z and for every gate v in Φ_A,

$$d(v) = \frac{|Y_v| + |Z_v|}{2} - \min(|Y_v|, |Z_v|) = \frac{|\tilde{Y}_v| + |\tilde{Z}_v|}{2} - \min(|\tilde{Y}_v|, |\tilde{Z}_v|),$$

which implies that

$$v \text{ is } \tau\text{-unbalanced in } \Phi_A \Leftrightarrow \frac{|\tilde{Y}_v| + |\tilde{Z}_v|}{2} - \min(|\tilde{Y}_v|, |\tilde{Z}_v|) \geq \tau. \tag{5.3}$$

Partition $\mathcal{L}$ into two sets:

$$\mathcal{L}_{small} = \left\{ v \in \mathcal{L} \ : \ |\tilde{X}_v| \leq n^{2/3} \right\}, \quad \mathcal{L}_{big} = \mathcal{L} \setminus \mathcal{L}_{small} = \left\{ v \in \mathcal{L} \ : \ |\tilde{X}_v| > n^{2/3} \right\}.$$

For all $u \in \mathcal{L}_{small}$, it holds that $|\tilde{X}_u| \leq n^{2/3}$. Define $X'_1 = \bigcup_{u \in \mathcal{L}_{small}} \tilde{X}_u$. Since $|\mathcal{L}| < \frac{c}{\tau} n^{1/3}$, it follows that $|X'_1| \leq \frac{c}{\tau} n^{1/3} n^{2/3} < n/4$. Hence, there exists a set $X_1 \subseteq X$ of size $n/4$ such that, for all $u \in \mathcal{L}_{small}$, we have $\tilde{X}_u \subseteq X_1$ (X_1 is some superset of X'_1 of size $n/4$). Let $A \sim \mathcal{D}(X_1)$ be a random partition of X to Y and Z such that $A(X_1) \subset Y$ (see section 4 for the definition of $\mathcal{D}(X_1)$).

For all $u \in \mathcal{L}_{small}$, it holds that $|\tilde{Y}_u| + |\tilde{Z}_u| = |\tilde{X}_u| \geq 2\tau$ and $\tilde{Z}_u = \emptyset$ (as $A(\tilde{X}_u) \subseteq A(X_1) \subset Y$). Thus, by (5.3), every $u \in \mathcal{L}_{small}$ is τ-unbalanced in Φ_A (with probability 1). Note that every $u \in \mathcal{L}_{big}$ admits $n^{2/3} < |X_u| < n - n^{2/3}$. Thus, by Proposition 4.1 for $\beta = 2/3$, and since $|\mathcal{L}| < \frac{c}{\tau} n^{1/3}$,

$$\mathop{\mathbb{E}}_{A \sim \mathcal{D}(X_1)} \left[|\{ u \in \mathcal{L} \ : \ u \text{ is not } \tau\text{-unbalanced in } \Phi_A \}| \right] \leq O\left(|\mathcal{L}_{big}| \tau n^{-1/3} \right) < 1$$

(for c small enough). Hence, there exists a partition A such that every $u \in \mathcal{L}$ is τ-unbalanced in Φ_A. $\square$

5.4. Proof of Lemma 5.2.

Proof. Let r be a gate computing f in Φ. Assume $|X_r| < n-2$. Let A be a partition of X to Y and Z. Thus, in Φ_A we have $a(r) \leq b(r) < m-1$. Hence, by Proposition 2.2, $\mathrm{Rank}(L_{f^A}) < 2^{m-1}$, which proves the lemma. Thus, assume $|X_r| \geq n-2$.

Since $|\mathcal{L}| < \frac{c}{\tau} n^{1/3}$, by Proposition 5.4 there exists a partition A of X to Y and Z such that every $u \in \mathcal{L}$ is τ-unbalanced in Φ_A. Denote by Ψ the arithmetic circuit Φ_A. In the rest of the proof we focus on Ψ. Recall that, for a gate u in Ψ, Y_u is the set of Y variables that occur in Ψ_u, and Z_u is the set of Z variables that occur in Ψ_u.

Define an order on $\mathcal{L}$ that respects the order of Ψ; that is, $\mathcal{L} = \{u_1, \ldots, u_\ell\}$, where $\ell = |\mathcal{L}|$, and for every $i, j \in [\ell]$ such that $i < j$, there is no directed path from u_i to u_j in Ψ. For $i \in [\ell]$, denote $h_i = \widehat{\Psi}_{u_i}$, the polynomial computed by u_i in Ψ, denote $Y_i = Y_{u_i}$, and denote $Z_i = Z_{u_i}$. For a gate v in Ψ, we say that v is *substituted* by α (which is a field element or a variable) in Ψ if the edges going into v are deleted, and v is relabeled by α.

The following proposition shows how to write the polynomial $f^A \in \mathbb{G}[Y, Z]$ (i.e., the polynomial computed by r in Ψ) as a function of the polynomials $h_1, \ldots, h_\ell \in \mathbb{G}[Y, Z]$ (i.e., the polynomials computed by $u_1, \ldots, u_\ell$ in Ψ).

PROPOSITION 5.5.

$$f^A = \sum_{i \in [\ell]} g_i h_i + g,$$

where $g, g_1, \ldots, g_\ell \in \mathbb{G}[Y, Z]$ are multilinear polynomials such that

1. *for all $i \in [\ell]$, the set of variables that occur in g_i and the set $Y_i \cup Z_i$ are disjoint; and*
2. *g is the polynomial computed by r in Ψ after substituting (in Ψ) each $u \in \mathcal{L}$ by 0.*

Proof. To prove the proposition we follow an inductive process that is based on the following claim.

CLAIM 5.6. *Let Υ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the sets of variables Y and Z. Let $\tilde{r}$ be a gate in Υ computing a polynomial $\tilde{f}$. Let v be a gate in Υ such that $Y_v \cup Z_v \neq \emptyset$. Denote by $\tilde{h} = \widehat{\Upsilon}_v$ the polynomial computed by v in Υ. Then, there exist two multilinear polynomials $\tilde{g}_1, \tilde{g}_2 \in \mathbb{G}[Y, Z]$ such that $\tilde{f} = \tilde{g}_1 \tilde{h} + \tilde{g}_2$, where*

1. *the set of variables that occur in $\tilde{g}_1$ and the set $Y_v \cup Z_v$ are disjoint; and*
2. *$\tilde{g}_2$ is the polynomial computed by $\tilde{r}$ in Υ after substituting (in Υ) v by 0.*

Proof. Let t be a new variable. Let Υ_1 be the arithmetic circuit Υ after substituting (in Υ) v by t. Let $G \in \mathbb{G}[Y, Z, t]$ be the polynomial computed by $\tilde{r}$ in Υ_1. Since Υ is syntactically multilinear and since $Y_v \cup Z_v \neq \emptyset$, it follows that G is linear in t; that is, G is of the form $G = \tilde{g}_1 t + \tilde{g}_2$, where $\tilde{g}_1$ and $\tilde{g}_2$ are two multilinear polynomials in $\mathbb{G}[Y, Z]$. Since $\tilde{f}$ is G after substituting (in G) t by $\tilde{h}$, we have $\tilde{f} = \tilde{g}_1 \tilde{h} + \tilde{g}_2$. Recall that $\mathcal{M}_{\Upsilon_1}(t)$ is the set of variables that multiply t in Υ_1 (see section 2.4). The set of variables that occur in $\tilde{g}_1$ is a subset of $\mathcal{M}_{\Upsilon_1}(t)$. Since Υ is syntactically multilinear, the sets $\mathcal{M}_{\Upsilon_1}(t)$ and $Y_v \cup Z_v$ are disjoint. Hence, the set of variables that occur in $\tilde{g}_1$ and the set $Y_v \cup Z_v$ are disjoint. Since $\tilde{g}_2 = G\big|_{t=0}$, we have that $\tilde{g}_2$ is the polynomial computed by $\tilde{r}$ in Υ after substituting (in Υ) v by 0. ∎

For $i \in [\ell]$, define Ψ^i to be the arithmetic circuit Ψ after substituting (in Ψ) $u_1, \ldots, u_i$ by 0. We now describe the inductive process.

First step. Recall that r computes f^A in Ψ, and u_1 computes h_1 in Ψ. Hence, by Claim 5.6 (for $\Upsilon = \Psi$, $\tilde{r} = r$, and $v = u_1$), since $Y_1 \cup Z_1 \neq \emptyset$, there exist two multilinear polynomials $g_1, g_1' \in \mathbb{G}[Y, Z]$ such that

$$f^A = g_1 h_1 + g_1',$$

where the set of variables that occur in g_1 and the set $Y_1 \cup Z_1$ are disjoint, and g_1' is the polynomial computed by r in Ψ^1. We continue in a similar manner.

Inductive step. Assume $g_1, \ldots, g_i$ are already defined (where $i \in [\ell - 1]$), and r computes g_i' in Ψ^i. Since there are no directed paths from the gates $u_1, \ldots, u_i$ to u_{i+1} in Ψ, the polynomial computed by u_{i+1} in Ψ^i is h_{i+1}, and the set of variables that occur in $\Psi^i_{u_{i+1}}$ is the same as the set of variables that occur in $\Psi_{u_{i+1}}$. Hence, by Claim 5.6 (for $\Upsilon = \Psi^i$, $\tilde{r} = r$, and $v = u_{i+1}$), since $Y_{i+1} \cup Z_{i+1} \neq \emptyset$, there exist two multilinear polynomials $g_{i+1}, g_{i+1}' \in \mathbb{G}[Y, Z]$ such that

$$g_i' = g_{i+1} h_{i+1} + g_{i+1}',$$

where the set of variables that occur in g_{i+1} and the set $Y_{i+1} \cup Z_{i+1}$ are disjoint, and g_{i+1}' is the polynomial computed by r in Ψ^{i+1}.

Thus,

$$f^A = g_1 h_1 + g_1' = g_1 h_1 + g_2 h_2 + g_2' = \cdots = \sum_{i \in [\ell]} g_i h_i + g,$$

where, for all $i \in [\ell]$, the set of variables that occur in g_i and the set $Y_i \cup Z_i$ are disjoint, and $g = g_\ell'$ is the polynomial computed by r in Ψ after substituting (in Ψ) $u_1, \ldots, u_\ell$ by 0. □

The following claim shows that the partial derivative matrices of $g_1 h_1, \ldots, g_\ell h_\ell$ are of low rank.

CLAIM 5.7. *For every $i \in [\ell]$,* $\mathrm{Rank}(L_{g_i h_i}) \leq 2^{m-\tau}$.

Proof. Fix $i \in [\ell]$. Denote by Y' the set of Y variables that occur in g_i and by Z' the set of Z variables that occur in g_i, and denote $a' = \min(|Y'|, |Z'|)$. By property 1 of Proposition 5.5, $(Y' \cup Z') \cap (Y_i \cup Z_i) = \emptyset$. Thus, $|Y'| + |Z'| \leq n - 2b(u_i)$, which implies $a' \leq m - b(u_i)$. Hence, by Proposition 2.2,

$$\mathrm{Rank}(L_{g_i}) \leq 2^{a'} \leq 2^{m - b(u_i)}.$$

Since $u_i \in \mathcal{L}$, u_i is τ-unbalanced. Thus, $d(u_i) = b(u_i) - a(u_i) \geq \tau$. Hence, by Proposition 2.2,

$$\mathrm{Rank}(L_{h_i}) \leq 2^{a(u_i)} \leq 2^{b(u_i) - \tau}.$$

Thus, since $(Y' \cup Z') \cap (Y_i \cup Z_i) = \emptyset$, by Proposition 2.4,

$$\mathrm{Rank}(L_{g_i h_i}) \leq 2^{m - b(u_i) + b(u_i) - \tau} = 2^{m - \tau}.$$ □

The following proposition shows that the total degree of g is small.

PROPOSITION 5.8. *The total degree of g is at most 4τ.*

Proof. Denote by Ψ^ℓ the arithmetic circuit Ψ after substituting (in Ψ) each gate $u \in \mathcal{L}$ by 0. For a gate v in Ψ^ℓ, denote by $td(v)$ the total degree of the polynomial computed by v in Ψ^ℓ. Every gate in Ψ^ℓ is also a gate in Φ. For a gate v in Ψ^ℓ, define

X_v to be the set of X variables that occur in Φ_v. Note that every gate v in Ψ^ℓ admits $td(v) \leq |X_v|$.

The following claim shows that, if X_v is large, then $td(v)$ is small (where v is a gate in Ψ^ℓ).

CLAIM 5.9. *Let v be a gate in Ψ^ℓ, and let $k = n - |X_v|$. Assume that $k \leq 2\tau$. Then, $td(v) \leq 4\tau - k$.*

Proof. The proof is by induction on the structure of Ψ^ℓ (that is, we consider a gate v only after considering v's two sons). Since $|X_v| = n - k \geq n - 2\tau$, it follows that v is not an input gate in Ψ^ℓ. Let v_1 and v_2 be the two sons of v in Ψ^ℓ. Let $k_1 = n - |X_{v_1}|$ and $k_2 = n - |X_{v_2}|$. Since $X_v = X_{v_1} \cup X_{v_2}$, it follows that $k \leq k_1$ and $k \leq k_2$. Consider the following two cases.

Case 1: v is an addition gate. First, we claim that $td(v_1) \leq 4\tau - k$ and $td(v_2) \leq 4\tau - k$. Consider v_1 without loss of generality. There are three cases.

(a) Assume $n - 2\tau \leq |X_{v_1}|$. Thus, $k_1 \leq 2\tau$. Hence, by induction, $td(v_1) \leq 4\tau - k_1 \leq 4\tau - k$.

(b) Assume $2\tau < |X_{v_1}| < n - 2\tau$. Since $|X_v| \geq n - 2\tau$, it follows that $v_1 \in \mathcal{L}$. Hence, v_1 is an input gate labeled by 0 in Ψ^ℓ, which implies $td(v_1) = 0 \leq 4\tau - k$.

(c) Assume $|X_{v_1}| \leq 2\tau$. Since $k \leq 2\tau$, we have $td(v_1) \leq |X_{v_1}| \leq 2\tau \leq 4\tau - k$.

Hence, since v is an addition gate, $td(v) \leq \max(td(v_1), td(v_2)) \leq 4\tau - k$.

Case 2: v is a product gate. Assume without loss of generality that $|X_{v_1}| \geq |X_{v_2}|$. Since $X_v = X_{v_1} \cup X_{v_2}$, it follows that $|X_{v_1}| \geq |X_v|/2 > 2\tau$ (for large enough n). Hence, there are two cases.

(a) Assume $n - 2\tau \leq |X_{v_1}|$. Thus, $k_1 \leq 2\tau$. Hence, by induction, $td(v_1) \leq 4\tau - k_1$. Since Φ is syntactically multilinear, $|X_v| = |X_{v_1}| + |X_{v_2}|$, which implies $|X_{v_2}| = k_1 - k$. Thus, $td(v_2) \leq |X_{v_2}| \leq k_1 - k$. Hence, $td(v) = td(v_1) + td(v_2) \leq 4\tau - k_1 + k_1 - k = 4\tau - k$.

(b) Assume $2\tau < |X_{v_1}| < n - 2\tau$. Since $|X_v| \geq n - 2\tau$, it follows that $v_1 \in \mathcal{L}$. Hence, v_1 is an input gate labeled by 0 in Ψ^ℓ, which implies $td(v) = 0 \leq 4\tau - k$. ∎

Since $|X_r| \geq n - 2$, by Claim 5.9, it follows that $td(r) \leq 4\tau$. Since r computes g in Ψ^ℓ, the proposition follows. ∎

By Propositions 5.8 and 2.6, since $\tau = 3 \log n$, we have

$$\mathrm{Rank}(L_g) \leq 2^{(4\tau+1)\log m} \leq 2^{\tau^3}.$$

By Proposition 5.5, $f^A = \sum_{i \in [\ell]} g_i h_i + g$. Thus, by Claim 5.7 and Proposition 2.3,

$$\mathrm{Rank}(L_{f^A}) \leq \sum_{i \in [\ell]} 2^{m-\tau} + 2^{\tau^3} < 2^{m-1},$$

where the last inequality holds for large enough n, as $\ell = |\mathcal{L}| < \frac{c}{\tau} n^{1/3}$ and $\tau = 3 \log n$. ∎

6. The construction. For a field $\mathbb{F}$ and a set of variables T, we denote by $\mathbb{F}[T]$ the *ring of polynomials* over the field $\mathbb{F}$ and the set of variables T, and we denote by $\mathbb{F}(T)$ the *field of rational functions* over $\mathbb{F}$ in the set of variables T. Let $X = \{x_1, \ldots, x_n\}$, $\Omega = \{\omega_1, \ldots, \omega_n\}$, $Y = \{y_1, \ldots, y_m\}$, and $Z = \{z_1, \ldots, z_m\}$ be four sets of variables (where $n = 2m$). Let $\mathbb{F}$ be a field, and let $\mathbb{G} = \mathbb{F}(\Omega)$ be the field of rational functions over $\mathbb{F}$ in the set of variables Ω. Note that a polynomial in $\mathbb{F}[X, \Omega]$ can also be thought of as a polynomial in $\mathbb{G}[X]$.

In this section, we construct a polynomial $f \in \mathbb{F}[X, \Omega]$ such that the following hold:

- Thinking of f as a polynomial in $\mathbb{G}[X]$, for every partition A of X to Y and Z, the partial derivative matrix of f^A has full rank over $\mathbb{G}$ (recall that $f^A \in \mathbb{G}[Y,Z]$ is the polynomial f after substituting every $x \in X$ by $A(x) \in Y \cup Z$).
- f is explicit in the sense that f is in the class VNP, which is Valiant's algebraic analogue of NP. Moreover, the coefficient of every monomial in f, as a polynomial in $\mathbb{F}[X,\Omega]$, is either 0 or 1.

6.1. Definition of f. For a set $B \subset [n]$ of size m, denote by $i_1,\ldots,i_m$ the elements of B in an increasing order (that is, $B=\{i_1,\ldots,i_m\}$ and $i_1<\cdots<i_m$), and denote by $j_1,\ldots,j_m$ the elements of $[n]\setminus B$ in an increasing order (that is, $[n]\setminus B=\{j_1,\ldots,j_m\}$ and $j_1<\cdots<j_m$). Define r_B, a multilinear monomial in $\mathbb{F}[\Omega]$, by $r_B=\prod_{\ell\in B}\omega_\ell$, and define g_B, a multilinear polynomial in $\mathbb{F}[X]$, by $g_B=\prod_{\ell\in[m]}(x_{i_\ell}+x_{j_\ell})$. Define

$$f=\sum_B r_B g_B, \tag{6.1}$$

where the sum is over all sets $B\subset[n]$ of size m. Thus, $f\in\mathbb{F}[X,\Omega]$ is a multilinear polynomial over the field $\mathbb{F}$ and the sets of variables X and Ω. We think of f also as a polynomial in $\mathbb{G}[X]$.

6.2. The partial derivative matrix of f^A has full rank. The following theorem states that, thinking of f as a polynomial in $\mathbb{G}[X]$, for any partition A of X to Y and Z, the partial derivative matrix of f^A has full rank. Formally, we have the following theorem.

THEOREM 6.1. *Let $f\in\mathbb{G}[X]$ be the polynomial defined in* (6.1)*. Then, for any partition A of X to Y and Z, the partial derivative matrix of f^A has full rank (over $\mathbb{G}$).*

We defer the proof of Theorem 6.1 to section 6.4. We remark that, the larger the set Ω is, the simpler it is to construct a polynomial f that satisfies Theorem 6.1. For the purpose of our lower bound, we need Ω to be as small as possible (and Ω such that $|\Omega|=|X|$ suffices).

6.3. f is explicit. In [9], Valiant defined an algebraic theory, analogous to the theory of NP-completeness. The analogue of NP, according to Valiant's theory, is called VNP. In this section, we show that f is in the class VNP.

For simplicity, instead of using the formal definition of VNP, we use a criterion (given by Valiant) for a polynomial to be in VNP. Valiant's criterion states that a polynomial f is in VNP if the coefficient of a monomial in f can be computed efficiently; that is, there exists a polynomial-time Turing machine M such that, given as input the degrees of the variables in a monomial p, M outputs the coefficient of p in f. (In fact, Valiant's criterion is stronger. For more details see Proposition 2.20 in [2].)

We use Valiant's criterion to show that f is in VNP. Let $r\in\mathbb{F}[\Omega]$ and $g\in\mathbb{F}[X]$ be two monic multilinear monomials. To prove that f is in VNP, we describe an efficient algorithm that outputs the coefficient of rg in f. The algorithm is as follows.

If the total degree of r is not m, then the coefficient of rg in f is 0. Therefore, assume that the total degree of r is m. Thus, r is of the form $r=\prod_{\ell\in B}\omega_\ell$ for a set $B\subset[n]$ of size m. Let $i_1,\ldots,i_m$ be the elements of B in an increasing order, and let $j_1,\ldots,j_m$ be the elements of $[n]\setminus B$ in an increasing order. Since

$$g_B=\prod_{\ell\in[m]}(x_{i_\ell}+x_{j_\ell})=\sum_{D\subseteq[m]}\prod_{\ell\in D}x_{i_\ell}\prod_{\ell\in[m]\setminus D}x_{j_\ell},$$

it follows that the coefficient of rg in f is 1 iff g is of the form $g = \prod_{\ell \in D} x_{i_\ell} \prod_{\ell \in [m] \setminus D} x_{j_\ell}$ for some $D \subseteq [m]$ (otherwise, the coefficient is 0). Checking whether g is of the form $g = \prod_{\ell \in D} x_{i_\ell} \prod_{\ell \in [m] \setminus D} x_{j_\ell}$ is straightforward. Hence, f is in VNP.

6.4. Proof of Theorem 6.1.

Proof. We first prove the following lemma, which is a generalization of Theorem 6.1.

LEMMA 6.2. *Let T be a set of variables, and let $\mathbb{F}(T)$ be the field of rational functions over $\mathbb{F}$ in the set of variables T. Let $k \in \mathbb{N}$. Let $s_1, \ldots, s_k \in \mathbb{F}[T]$ be different monic multilinear monomials. Let $h_1, \ldots, h_k \in \mathbb{F}[Y, Z]$ be multilinear polynomials. Denote $h = \sum_{i \in [k]} s_i h_i$ a polynomial in $\mathbb{F}[Y, Z, T]$. Thus, h can be viewed also as a polynomial in $\mathbb{F}(T)[Y, Z]$. Assume that there exists $\ell \in [k]$ such that the partial derivative matrix of h_ℓ has full rank over $\mathbb{F}$. Then, the partial derivative matrix of h has full rank over $\mathbb{F}(T)$.*

Proof. The proof is by induction on the size of T.

Induction base. Assume $|T| = 0$. Since $s_1, \ldots, s_k$ are *different monic* monomials in $\mathbb{F}[T] = \mathbb{F}$, it follows that $k = 1$ and $s_1 = 1$. Hence, $h = h_1$, and the partial derivative matrix of h_1 has full rank over $\mathbb{F} = \mathbb{F}(T)$, which proves the lemma.

The induction step is based on the following claim (the claim is well known, and we omit its proof).

CLAIM 6.3. *Let P and Q be two $M \times M$ matrices with entries in a field $\mathbb{H}$. Let t be a variable. Then,*

$$\exists\ a_1, \ldots, a_{M-1} \in \mathbb{H}\ :\ \det(tQ + P) = t^M \det(Q) + \det(P) + \sum_{d=1}^{M-1} a_d t^d,$$

where $\det(\cdot)$ is the determinant.

Induction step. Assume $|T| > 0$. Let $t \in T$ be a variable. Denote $T' = T \setminus \{t\}$, and denote by $\mathbb{F}(T')$ the field of rational functions over $\mathbb{F}$ in the set of variables T'. Since $s_1, \ldots, s_k$ are multilinear monomials, assume without loss of generality that $h = t \sum_{i=1}^{k'} s'_i h_i + \sum_{i=k'+1}^{k} s_i h_i$, where $k' \in [k]$, $s'_1, \ldots, s'_{k'} \in \mathbb{F}[T']$ are different monic multilinear monomials, and $s_{k'+1}, \ldots, s_k \in \mathbb{F}[T']$ are different monic multilinear monomials. Recall that the partial derivative matrix of h_ℓ has full rank over $\mathbb{F}$. Assume without loss of generality that $\ell \leq k'$ (similar arguments hold for $\ell > k'$). Denote $h' = \sum_{i=1}^{k'} s'_i h_i$ and $h'' = \sum_{i=k'+1}^{k} s_i h_i$. By induction, it follows that $L_{h'}$ has full rank over $\mathbb{F}(T')$, which implies that $\det(L_{h'}) \neq 0$. Hence, since $L_h = t L_{h'} + L_{h''}$, using Claim 6.3 (for $M = 2^m$, $Q = L_{h'}$, $P = L_{h''}$, and $\mathbb{H} = \mathbb{F}(T')$), we have that $\det(L_h) \neq 0$, which implies that L_h has full rank over $\mathbb{F}(T)$. ∎

Fix a partition A of X to Y and Z, and let $B_0 = \{i \in [n]\ :\ A(x_i) \in Y\}$. Recall that g_{B_0} is the polynomial defined in section 6.1. The following claim shows that the partial derivative matrix of $g^A_{B_0}$ is a permutation matrix (a permutation matrix is a matrix obtained by permuting the rows of the identity matrix).

CLAIM 6.4. *The partial derivative matrix of $g^A_{B_0}$ is a permutation matrix.*

Proof. Denote by $i_1, \ldots, i_m$ the elements of B_0 in an increasing order, and denote by $j_1, \ldots, j_m$ the elements of $[n] \setminus B_0$ in an increasing order. By definition of g_{B_0},

$$g^A_{B_0} = \Big(\prod_{\ell \in [m]} \big(x_{i_\ell} + x_{j_\ell}\big) \Big)^A = \prod_{\ell \in [m]} \big(A(x_{i_\ell}) + A(x_{j_\ell})\big).$$

Note that, for every $\ell \in [m]$, we have $A(x_{i_\ell}) \in Y$ and $A(x_{j_\ell}) \in Z$. Thus, there exists a permutation $\pi : [m] \to [m]$ such that $g^A_{B_0} = \prod_{\ell \in [m]} \left(y_\ell + z_{\pi(\ell)}\right)$. Hence, the partial derivative matrix of $g^A_{B_0}$ is a permutation matrix. □

By Claim 6.4, the set B_0 is such that the partial derivative matrix of $g^A_{B_0}$ has full rank over $\mathbb{F}$. Hence, using Lemma 6.2 (for $T = \Omega$), since $\{r_B\}_{B \subset [n] : |B| = m}$ is a set of different monic multilinear monomials in $\mathbb{F}[\Omega]$, it follows that the partial derivative matrix of f^A has full rank over the field $\mathbb{G}$. Since A is an arbitrary partition of X to Y and Z, the theorem follows. □

7. The lower bound: Proof of Theorem 1.1.

Proof. Denote $\mathbb{G} = \mathbb{F}(\Omega)$ the field of rational functions over $\mathbb{F}$ in the set of variables Ω. We can think of Φ as a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the set of variables X: every input gate in Φ labeled by $\omega \in \Omega$ is thought of as labeled by a field element in $\mathbb{G}$, and every other input gate in Φ is labeled by either a field element in $\mathbb{F} \subseteq \mathbb{G}$ or a variable in X. The number of variables in Φ is n (instead of $2n$). The polynomial computed by Φ is f, but we think of f as a polynomial in $\mathbb{G}[X]$. By Theorem 6.1, for all partitions A of X to Y and Z,

$$\mathrm{Rank}(L_{f^A}) = 2^m,$$

where the rank is over $\mathbb{G}$. Hence, by Theorem 5.1,

$$|\Phi| = \Omega\left(\frac{n^{4/3}}{\log^2 n}\right). \qquad \square$$

REFERENCES

[1] W. Baur and V. Strassen, *The complexity of partial derivatives*, Theoret. Comput. Sci., 22 (1983), pp. 317–330.

[2] P. Burgisser, *Completeness and Reduction in Algebraic Complexity Theory*, Algorithms Comput. Math. 7, Springer-Verlag, Berlin, Heidelberg, 2000.

[3] J. Morgenstern, *How to compute fast a function and all its derivatives, a variation on the theorem of Baur–Strassen*, ACM SIGACT News, 16 (1985), pp. 60–62.

[4] N. Nisan and A. Wigderson, *Lower bound on arithmetic circuits via partial derivatives*, Comput. Complexity, 6 (1996), pp. 217–234.

[5] R. Raz, *Multi-linear formulas for permanent and determinant are of super-polynomial size*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 633–641.

[6] R. Raz, *Separation of multilinear circuit and formula size*, Theory Comput., 2 (2006), pp. 121–135.

[7] R. Raz and A. Shpilka, *Deterministic polynomial identity testing in non commutative models*, Comput. Complexity, 14 (2005), pp. 1–19.

[8] V. Strassen, *Die berechnungskomplexitat von elementarsymmetrischen funktionen und von interpolationskoeffizienten*, Numer. Math., 20 (1973), pp. 238–251.

[9] L. G. Valiant, *Completeness classes in algebra*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing, ACM, New York, 1979, pp. 249–261.

SIAM J. COMPUT.
Vol. 38, No. 4, pp. 1648–1659

COST-DISTANCE: TWO METRIC NETWORK DESIGN*

ADAM MEYERSON†, KAMESH MUNAGALA‡, AND SERGE PLOTKIN§

Abstract. We present the COST-DISTANCE problem: finding a Steiner tree which optimizes the sum of edge *costs* along one metric and the sum of source-sink *distances* along an unrelated second metric. We give the first known $O(\log k)$ randomized approximation scheme for COST-DISTANCE, where k is the number of sources. We reduce several common network design problems to COST-DISTANCE, obtaining (in some cases) the first known logarithmic approximation for them. These problems include single-sink buy-at-bulk with variable pipe types between different sets of nodes, facility location with buy-at-bulk–type costs on edges (integrated logistics), constructing single-source multicast trees with good cost and delay properties, priority Steiner trees, and multilevel facility location. Our algorithm is also easier to implement and significantly faster than previously known algorithms for buy-at-bulk design problems.

Key words. approximation algorithms, network design, Steiner trees, facility location

AMS subject classifications. 68W25, 68W20

DOI. 10.1137/050629665

1. Introduction.

Consider designing a network from the ground up. We are given a set of customers and need to place various servers and network links in order to cheaply provide sufficient service. If we only need to place the servers, this becomes the facility location problem [36], and constant-factor approximations are known (please refer to [28] for a complete list of citations). If a single server handles all customers, and we impose the additional constraint that the set of available network link types is the same for every pair of nodes (subject to constant scaling factors on cost), then this is the single-sink buy-at-bulk problem [35, 4]. We give the first known approximation for the general version of this problem to optimize both placement of servers and network topology.

We reduce the network design problem to the following theoretical framework, which we call the COST-DISTANCE problem: We are given a graph with a single distinguished sink node (server). Every edge in this graph can be measured along two metrics; the first will be called *cost*, and the second will be *length*. Note that the two metrics are entirely unrelated and that there may be any number of parallel edges in the graph. We are given a set of sources (customers). The objective is to construct a Steiner tree connecting the sources to the sink while minimizing the combined sum of the *cost* of the edges in the tree and sum over sources of the weighted *distance* (i.e., total *length*) from source to sink. Note that this definition is a direct generalization of both the shortest path tree and the minimum cost Steiner tree. If *costs* and *lengths*

*Received by the editors April 21, 2005; accepted for publication (in revised form) August 12, 2008; published electronically December 10, 2008. A preliminary version of this work appeared in the 41st IEEE Symposium on Foundations of Computer Science, 2000. This work was done while the authors were at Stanford University, and was supported by ARO grant DAAG55-98-1-0170 and ONR grant N00014-98-1-0589.

http://www.siam.org/journals/sicomp/38-4/62966.html

†Department of Computer Science, University of California, Los Angeles, CA 90095-1596 (awm@cs.ucla.edu).

‡Department of Computer Science, Duke University, Durham, NC 27708 (kamesh@cs.duke.edu). This author's research was supported by the NSF via a CAREER award and grant CNS-0540347.

§Department of Computer Science, Stanford University, Stanford, CA 94305 (plotkin@cs.stanford.edu).

are proportional, then constant-factor approximations [24, 5] are known.

We obtain the first general approximation algorithm for this problem with unrelated metrics. We prove an expected competitive ratio of $O(\log |S|)$ (where S is the set of sources) for our randomized algorithm. The algorithm is fairly simple to implement and runs in a relatively fast $O(|S|^2(m + n \log n))$ time bound.

Theoretical significance. In section 5, we show that many standard problems in network design can be reduced to COST-DISTANCE, showing that this problem forms a general theoretical framework for network design. In particular, we describe simple reductions from single-sink buy-at-bulk and the metric facility location problem. We demonstrate that a natural combination of facility location and buy-at-bulk can be solved by reduction to COST-DISTANCE. In fact, we can generalize single-sink buy-at-bulk to account for a scenario where not all network link types are available between every pair of nodes, or where costs do not scale linearly. This better models real-life situations where certain types of hardware may not be available (or may not be practical to install) in certain locations. This problem was subsequently called "integrated logistics" by Ravi and Sinha [34]. Our algorithm provides the first known approximation for this more general problem. We also obtain among the first known combinatorial approximations for the metric multilevel uncapacitated facility location problem [23], the priority Steiner tree problem [11], and for constructing single-source multicast trees with good cost and average per receiver delay [33].

From a more theoretical standpoint, consider routing single-source traffic through a graph where each edge has some function relating the total traffic along the edge to the cost of routing that traffic. If all functions are convex increasing (nondecreasing derivative), then exact solutions are known using min-cost flow techniques. We present the first approximations for the case where all functions are *concave increasing* (nonincreasing derivative). Previous work on buy-at-bulk [4, 3, 35] required that the concave functions between each pair of nodes be identical up to a constant scaling factor; we eliminate this requirement. We present all these connections in more detail in section 5.

Technical contributions. The chief technical contribution is to show that the COST-DISTANCE problem can be solved by layered aggregation of demands. We perform aggregation in pairs by constructing matchings iteratively on a suitably defined complete graph. Our algorithm and analysis exploit the connection of COST-DISTANCE to the budgeted version of the problem [29, 26], where the goal is to find a tree with low cost in the c metric such that the diameter is no more than L in the l metric. This problem has an $O(\log |S|, \log |S|)$ bicriteria approximation on the cost and diameter via layered aggregation by Marathe et al. [29]. We need to make one nontrivial modification when the objective becomes aggregate distance instead of diameter—this is discussed in section 3.1 and makes our algorithm randomized.

The technical framework of layered aggregation pervades all subsequent combinatorial algorithms for single-sink aggregation problems, such as access network design [19], single-sink buy-at-bulk [20], and the rent-or-buy problem [21].

In addition to generalizing previous results, our algorithm is easy to implement and has a small running time. This makes it the algorithm of choice for many of the problems we have previously described. For example, previous algorithms for single-sink buy-at-bulk depended on methods of randomly selecting trees which approximate stretch [6, 7, 9, 10, 16]. The algorithm for access network design [3] depended on a linear programming relaxation.

Previous results. If the cost and distance metrics are proportional, the offline version of COST-DISTANCE has a constant factor approximation [5, 24], and there is

an online algorithm performing a small number of reroutings of existing paths [17]. If the cost and distance metrics are unrelated, this problem has no previously known approximation algorithm. Previous results for many related network design problems are discussed in detail in section 5.

2. The Cost-Distance problem. We are given a graph $G = (E, V)$ along with a set of source vertices $S \subset V$ which need to be connected to a single-sink vertex $t \in V$. We have two metrics along this graph. We will call the first metric cost $c : E \mapsto \Re^+$ and the second metric length $l : E \mapsto \Re^+$. We are also given a weighting function $w : S \mapsto \Re^+$ on the sources. We denote the two metrics on an edge e as $(c(e), l(e))$.

We are asked to find a connected subgraph $G' = (E', V') \subset G$ which contains all sources ($S \subset V'$) and the sink ($t \in V'$) such that the following sum is minimized:

$$\sum_{e \in E'} c(e) + \sum_{s \in S} w(s) L'(s, t).$$

Here $L'(s, t)$ is the total length of the min-length path from s to t along the edges of G'.

Our algorithm will give an $O(\log |S|)$ approximation to this sum. It is important to notice that our approximation ratio does not depend on the number of edges, since there may potentially be a large number of edges connecting the same pair of nodes ($m \gg n^2$).

3. The algorithm. The algorithm works by pairing up sources (or pairing sources with sink) until only the sink remains. At each stage we find a matching on the nodes and then choose one node from each matched pair to be "center." We transport the weight from the noncenter node to the center, paying the appropriate edge costs and weight times distance costs. We then repeat this process on the centers until the sink is the only remaining node. Details of the algorithm are as follows:

1. Define $S_0 = S \cup \{t\}$ and $w_0 = w$. Create empty set E'.
2. Set $i = 0$.
3. For every pair of nonsink nodes $(u, v) \in S_i$:
   - Find the shortest $u - v$ path in G according to the distance function on the edges: $M_{uv}(e) = c(e) + \frac{2w_i(u)w_i(v)}{w_i(u)+w_i(v)} l(e)$.
   - Let $K_i(u, v)$ be the length of this path under the distance function $M_{uv}(e)$.
4. For every nonsink node $u \in S_i$:
   - Find the shortest $u - t$ path in G according to the metric $M_{ut}(e) = c(e) + w_i(u)l(e)$.
   - Let $K_i(u, t)$ be the length of this path under metric $M_{ut}(e)$.
5. Find a matching between nodes in S_i such that the number of unmatched nodes plus half the number of matched nodes is at most S_i/α and the value of $\sum_{(u,v)matched} K_i(u, v)$ is at most β times the value of the minimum K_i-cost perfect matching. We assume α and β are known constants.
6. For each matched pair (u, v) add the edges on the path defining $K_i(u, v)$ to the set E'.
7. Create an empty set S_{i+1}.
8. For each pair of nonsink matched nodes (u, v):
   - Choose u to be the center with probability $w_i(u)/(w_i(u) + w_i(v))$. Otherwise v will be the center.

- Add the chosen center to S_{i+1} and assign the center a weight $w_{i+1}(\mathit{center}) = w_i(u) + w_i(v)$.

9. Add all unmatched nodes $u \in S_i$ to S_{i+1} and define $w_{i+1}(u) = w_i(u)$.
10. Add the sink to S_{i+1}.
11. If S_{i+1} contains only the sink, we are done. Otherwise increment i and return to step 3.
12. We return $G' = (E', V')$, where E' is the set of edges we constructed and V' is the set of adjacent nodes.

Each time through the steps, the size of our set S_i is reduced by α. Thus the process terminates after $\log_\alpha |S|$ iterations.

3.1. Discussion. The overall algorithm is very similar to that in [29], where the bicriteria (cost, diameter) version is solved. There is however one very important and nontrivial difference. Every time there is an aggregation, a choice has to be made about the "center" for the aggregate demand. In the case of [29], any center works since they are approximating the diameter of the tree. In our case, such a choice would be disastrous, and we instead resort to choosing the center at random with probability proportional to the demand aggregated there (step 8). This method has been subsequently derandomized by Chekuri, Khanna, and Naor [13] using the dual of a natural linear programming formulation.

The metric used for constructing the matching is $M_{uv}(e) = c(e) + \frac{2w_i(u)w_i(v)}{w_i(u)+w_i(v)} l(e)$. This is the expected cost of transporting one demand of the pair (u, v) to the other location, when the choice of which location to transport to is made at random in proportion to its demand value.

A little more detail is needed in step 5. We could find the min-cost perfect matching on the set in polynomial time, obtaining $\alpha = 2$ and $\beta = 1$. Polynomial-time algorithms are known for min-cost perfect matching on nonbipartite graphs [32]. However, these algorithms tend to be impractical.[1] The following simpler procedure will work for us, causing only a small constant loss in our approximation ratio. We will find the cheapest pair of nodes to connect (minimum $K_i(u, v)$) and match them. We then remove these two nodes from consideration and repeat. We continue this process until half the nodes have been matched. The jth pair which we choose to match must have had matching cost at most equal to the $(2j-1)$st cheapest edge in the perfect matching, according to the K_i metric. It follows that our total K_i-cost is at most half the K_i-cost of the perfect matching, guaranteeing $\alpha = 4/3$ and $\beta = 1/2$.

Each iteration of this algorithm finds shortest paths between all pairs in S_i. Since the metric is different for each pair, we cannot use all-pairs shortest-path computations. Instead we perform $|S_i|^2$ single-pair shortest paths. We first take $O(m)$ time to compute the metric on every edge. Using Dijkstra's algorithm, we can compute the shortest path between a single pair of nodes in $O(m + n \log n)$ time. The matching step (step 5) can be performed in $O(|S_i|^2 \log |S_i|)$ time. It follows that iteration i takes at most $O(|S_i|^2(m + n \log n))$ time. Since the size of S_i reduces by constant α at each iteration, when we sum over iterations the total running time looks like $O(|S|^2(m + n \log n))$.

4. Analysis. The optimal solution will be a tree, which we will call T^*. To see this, notice that we can take any graph and produce the shortest-path (according to the length metric) tree connecting the sink to all sources. This shortest-path tree

[1] We could use the $O(n^2 \log n)$ approximation algorithm in [18] to get $\beta = 2$ and $\alpha = 2$. Here, n is the total number of nodes in the graph.

will have total cost at most the total cost of the graph and a distance-to-sink from every source node equal to the distance-to-sink from the graph. It follows that the optimal solution must be a tree, since a nontree solution immediately gives rise to a tree solution with equal or superior total value.

We define the following quantities:

$C^* = \sum_{e \in T^*} c(e)$.

$L^*(v) =$ total length of edges along the path from v to the sink in T^*.

$D^* = \sum_{v \in S} w(v) L^*(v)$.

The total "value" of the optimal solution which we will need to approximate is $C^* + D^*$. At each stage in our algorithm, we have some set of nodes S_i which we are trying to connect. We define the following potential function:

$$D_i^* = \sum_{v \in S_i} w_i(v) L^*(v).$$

Notice that $D_0^* = D^*$.

Since our algorithm is randomized, we need to analyze the expected performance. Each stage of the algorithm transports some weight from matched nodes to chosen centers. We can define the value of stage i to be the total cost of the edges used in stage i matching plus the cost to transport the weight across the appropriate edges to the center. The total value of our solution will then be the sum of the values of the stages.

We first prove a lemma bounding the expected potential function at each stage.

LEMMA 4.1. *For every stage i, $\mathbf{E}[D_i^*] \leq D^*$.*

Proof. We will prove this by induction. For $i = 0$ we know $D_0^* = D^*$. Consider stage $i > 0$. Suppose we matched u and v in our previous matching. The contribution of u and v to D_{i-1}^* was $w_{i-1}(u)L^*(u) + w_{i-1}(v)L^*(v)$. We choose a random center. The expected distance from center to sink is now

$$\frac{w_{i-1}(u)L^*(u) + w_{i-1}(v)L^*(v)}{w_{i-1}(u) + w_{i-1}(v)}.$$

The weight of the new center is $w_{i-1}(u) + w_{i-1}(v)$. It follows that the new center's expected contribution to D_i^* is also $w_{i-1}(u)L^*(u) + w_{i-1}(v)L^*(v)$. Of course, unmatched nodes contribute equally to both potentials, and nodes matched with the source contribute less to D_i^* since their weight will disappear. Thus the expected value of D_i^* is at most D_{i-1}^*. It follows that $\mathbf{E}[D_i^*] \leq \mathbf{E}[D_{i-1}^*]$, and the inductive hypothesis implies that $\mathbf{E}[D_i^*] \leq D^*$. ∎

We will now relate the value of a stage to the metric K_i on which we approximated a min-cost perfect matching. This will allow us to bound the expected value of each stage. This lemma previously appeared in [29].

LEMMA 4.2. *Given a tree $T = (E, V)$ and a set of nodes $S \subseteq V$, there exists a perfect matching of the nodes in S which uses each edge of the tree at most once.*

Proof. We prove this by induction on the number of edges in the tree. If the tree includes zero weight edges, then $|V| = 1$ and the result is trivially true. Consider a larger tree. Suppose $v \in V$ is a leaf of this tree. If v is not included in S, then we can remove v and the edge connecting it to its parent from the tree to produce a smaller tree, T'. We inductively produce a perfect matching of the nodes in S on T' and use the same matching for T. If v is included in S, then we consider v's parent node. If the parent node is also in S, then we match v with its parent. We then remove v and its edge from the tree to produce T' and inductively match the rest of S on T'. If the

parent node is not in S, we produce S' by removing v from S and adding v's parent. We again produce T' and match the nodes. Some node u is matched to v's parent. We will use the identical matching to the one on T' except that we will match v with u by adding the edge from v to its parent to the relevant path. This produces the desired matching. ∎

LEMMA 4.3. *The expected value of stage i is at most $\beta(2D^* + C^*)$.*

Proof. Consider the tree T^*. Using Lemma 4.2, there exists a matching of the nodes in S_i using only edges in T^*, such that no edge is used more than once. This matching has edges with total cost at most C^*. The fraction $2w_i(u)w_i(v)/(w_i(u) + w_i(v))$ is at most twice the minimum of the two weights. Each edge in our matching would have to be along the path-to-source in the optimal tree for one of the two matched nodes. It follows that

$$\sum_{(u,v)matched} K_i(u,v) \leq C^* + 2D_i^*.$$

The min-cost perfect matching along metric K_i must do at least this well. Since the matching we actually use has cost at most β times the min-cost perfect matching, we guarantee a matching of K_i-cost at most $\beta(C^* + 2D_i^*)$. We need to relate this cost to the value of the stage.

The value of the stage is the total cost to transfer weight from matched nodes to their centers. Suppose we match u and v. If we choose v as the center, then we need to transport u's weight over to v. This induces a value of $w_i(u)l(u,v)$ in addition to the value induced by the cost of edges used. On the other hand, if we choose u as center, then we pay $w_i(v)l(u,v)$ plus edge costs. The expected value is thus

$$\frac{w_i(v)w_i(u)l(u,v) + w_i(u)w_i(v)l(u,v)}{w_i(u) + w_i(v)} + c(u,v).$$

Notice that this expected value is exactly $K_i(u,v)$. It follows that the expected value of the stage is equal to the total K_i-cost of the matching found—at most $\beta(C^* + 2D_i^*)$. This of course depends on D_i^*, a random variable with expected value at most D^* (as per Lemma 4.1). It follows that the expected value of stage i is at most $\beta(2D^* + C^*)$, as desired. ∎

THEOREM 4.1. *We obtain approximation ratio $2\beta \log_\alpha |S| = O(\log |S|)$ to the optimal.*

Proof. The expected value of our solution is equal to the sum of the expected value of the stages. This gives us a total value $\mathbf{E}[V] \leq \sum_i \mathbf{E}[V_i]$. Using Lemma 4.3 and our bound on the total number of stages, we can bound this by $\mathbf{E}[V] \leq \beta(\log_\alpha |S|)(2D^* + C^*)$. Since the optimal solution has value $D^* + C^*$, this proves the desired approximation ratio. ∎

Using the described greedy algorithm to find a matching, we will attain expected approximation ratio $\log_{4/3} |S|$; exact perfect matchings would improve this to $2 \log_2 |S|$. There will be a small additional loss in the last stages, where an uneven number of nodes could cause a few additional steps; however, our total approximation will remain bounded by an expected $O(\log |S|)$.

5. Relation to network design problems. We will demonstrate approximation-preserving reductions from many commonly encountered network design probems to special cases of COST-DISTANCE. We emphasize that for all these problems, our algorithm produces a logarithmic approximation ratio, while being (in general) simpler to implement and faster to run than previously known algorithms.

5.1. Multicast tree design. In this problem, we are given a network $G(V, E)$ with costs and delays on the edges. We have a source node $s \in V$ and receivers $R \subseteq V$ for multicast data. The goal is to construct a tree T connecting the source to R so that the sum of the cost of the tree and the delays seen by the receivers is minimized. This problem is equivalent to Cost-Distance if the distance metric is the delays. This problem has been extensively studied in the networks community [8, 15, 22, 25, 27, 33, 39], and many heuristics have been proposed. We present the first approximation for this problem. Our approximation ratio is $O(\log |R|)$.

5.2. Facility location. We are given a weighted undirected graph $G(V, E)$ with a cost per unit demand $c : E \to \Re$ on the edges, which forms a metric. We have a set of demand points $D \subseteq V$ with demands d_i and a set of facility locations $F \subseteq V$ with facility costs f_i. The goal is to open a subset of the facilities and assign demands to the open facilities so that the sum of the cost of opening the facilities and the cost of routing the demand to the facilities is minimized.

For edge e in the graph, the bicriteria cost function is $(0, c(e))$. We add a dummy sink and connect it to all the facilities. For facility i, the cost of the edge is $(f_i, 0)$. The demand points will be our source vertices ($S = D$) and their weights will be equal to the demands ($w(v) = d_v$). The cost of a Cost-Distance solution on this modified graph is identical to the cost of its corresponding facility-location solution, so it follows that the reduction is approximation-preserving.

We can also consider the capacitated version of this problem, where facility i has capacity u_i. We can open multiple copies of a facility, but each copy opened at location i costs f_i. Again, we modify the graph exactly as before but assign cost $(f_i, \frac{f_i}{u_i})$ on the edge connecting the sink to facility i. This causes the loss of an additional factor of two (at most) in the approximation ratio, which is seen as follows. Suppose $d > 0$ amount of demand is routed to a facility. Then, the cost paid by the Cost-Distance version is $f_i(1 + \frac{d}{u_i})$, whereas the cost paid by the actual facility location version is $f_i\lceil \frac{d}{u_i} \rceil$. Clearly $(1 + \frac{d}{u_i}) \leq 2\lceil \frac{d}{u_i} \rceil$.

We have therefore obtained an $O(\log(|D|))$ approximation to these problems. Note that several constant-factor approximations are known for this problem. The best known approximation ratio is 1.52 and is due to Mahdian, Ye, and Zhang [28], which builds on a long line of previous work. Please refer to this paper for previous work on this problem.

5.3. Extended single-sink buy-at-bulk. In this problem [35], we are given a weighted graph $G(V, E)$ with length function $l : E \to \Re$. A subset $S \subseteq V$ of nodes has demands d_i. We have a special sink node t to which all this demand must be routed. The demand must be routed by choosing a tree and buying pipes along this tree. There are K types of pipes. The type i pipe has cost c_i per unit length and capacity u_i. We assume that K is $O(\text{poly}(|V|))$. The goal is to minimize the total cost of pipe bought.

We modify the graph as follows. Replace every edge e in the graph with K parallel edges $e_1, e_2, \ldots, e_K$. The edge e_i has bicriteria cost $(l(e)c_i, l(e)\frac{c_i}{u_i})$. The weight of a node is its demand. This new graph is the instance of Cost-Distance that we solve. Intuitively, $l(e)c_i$ is the fixed cost of using pipe i, and $l(e)\frac{c_i}{u_i}$ is the incremental cost of routing demand.

It is implicit in the work of [4, 35] that the optimum tree with the modified cost function is no more than a factor 2 away from the optimum tree for the original problem. This is seen as follows. Suppose that in the original problem, $d > 0$

amount of demand was routed on edge e using cables of type i so that the cost paid is $l(e)c_i\lceil\frac{d}{u_i}\rceil$. In the COST-DISTANCE version, the cost paid for routing the same demand is $l(e)c_i + l(e)\frac{c_i}{u_i}d = l(e)c_i\left(1+\frac{d}{u_i}\right)$. This is clearly at most a factor of 2 away from $l(e)c_i\lceil\frac{d}{u_i}\rceil$ provided $d > 0$.

The best previously known approximation for this problem was $O(\log|S|\log\log|S|)$, which follows by applying the techniques in [10] to the algorithm in [4]. These algorithms are based on the work in [6, 7, 9, 10], which show how to approximate any finite metric by a tree metric so that the distance between any two nodes in the graph is approximated well. For the special case of $K = 1$, Salman et al. [35] showed a constant-factor approximation by using previous results [5, 24] on balancing Steiner trees with shortest-path trees.

Subsequent to our work, several constant-factor approximations have been obtained for this problem; the first one is due to Guha, Meyerson, and Munagala [20], and the most recent (and best) is due to Gupta, Kumar, and Roughgarden [21].

All previous approximations assumed that all the K pipes are available between all pairs of nodes; it is straightforward to see that we can do away with this restriction. This problem arises naturally in network design. There may be a fixed cost of laying cables which depends on the location but is independent of the type of cable being laid (perhaps the cost of installing the cable outweighs the cost of the cable itself). Alternatively, certain types of services might not be available in certain locations. Our algorithm is the first to handle these generalizations.

5.4. Combining facility location with buy-at-bulk. We can define a combination of the previous problems as follows. We are given the same graph as in the (capacitated) facility location problem and also a set of K pipe types just as in the buy-at-bulk problem. We wish to open facilities and construct a forest routing the demands to the facilities. The demands must be routed by buying pipes along the edges of the forest. We wish to optimize the sum of the cost of laying out the pipes and the cost of opening the facilities.

This problem arises, for example, in placing caches over the web and connecting the demand points to the caches by laying out links of some fixed types (like T1, OC10, etc.). We wish to optimize the total cost of placing the caches and buying the links to route the demands.

It should be clear that the combination of the modifications we made in the previous problems gives an instance of COST-DISTANCE. The approximation ratio is therefore $O(\log|D|)$. This holds even if the set of available pipes differs for different pairs of nodes. As far as we are aware, this is the first approximation algorithm for this problem. Ravi and Sinha [34] subsequently gave improved approximations for this problem and called it "integrated logistics."

5.5. Priority Steiner trees. This problem [11] generalizes the Steiner tree problem in the following way. Each edge e has a priority p_e in addition to cost $c(e)$. Each terminal t also has a priority p_t. The goal is to connect these terminals to a sink using a Steiner tree such that all edges on the path from a terminal t to the sink have priority at most p_t. The goal is to minimize the total cost of the edges used. Charikar, Naor, and Schieber [11] present an $O(\log|R|)$ approximation, where R is the set of terminals.

This problem is a special case of COST-DISTANCE, as shown by a reduction due to Chuzhoy et al. [14] that we outline below. The algorithm presented in section 3 then yields an $O(\log|R|)$ approximation, matching the best known guarantee.

Specifically, a priority Steiner tree instance can be converted in an approximation-preserving fashion to an instance of the extended single-sink buy-at-bulk problem defined in section 5.3. Let $C = \max_e c(e)$, and assume $\min_e c_e \geq 1$ by scaling. Suppose there are k priority levels. For each node v with priority i, we set its demand $d_v = (nC)^{5(k-i)}$, and for each edge e with priority i, the capacity of the corresponding cable type is $u_e = (nC)^{5(k-i)+2}$, and its cost per unit cable is $c(e)$. The length of the edge $l(e) = 1$. Given a solution to the priority Steiner tree, suppose edge e is used. Then, purchase one cable along this edge. Suppose this edge has priority i; then the total demand with priority at least i is at most $n \sum_{l \geq i} (nC)^{5(k-l)} \leq (nC)^{5(k-i)+2} = u_e$. This shows that the purchases yield a feasible solution. This yields a solution to the buy-at-bulk problem with the same cost as the priority Steiner tree. Conversely, given a solution to the buy-at-bulk problem, a demand with priority i will be routed only along edges of priority at most i; otherwise, at least $(nC)^3$ cables of higher priority need to be purchased along the violated edge, which costs at least $(nC)^3$. However, the total cost over all demands of purchasing cables along paths that satisfy the priorities is at most n^2C, which is only cheaper. Therefore, in the optimal buy-at-bulk solution, all demands with priority i are routed on paths such that each edge in the path has priority at most i; furthermore, only one copy of any edge is purchased. These sets of edges are feasible for the priority Steiner tree. A solution to the buy-at-bulk instance can be transformed to obtain a solution for the priority Steiner tree instance, thus completing the reduction.

5.6. Multilevel facility location. In the k-level facility location problem, we are given a graph $G(V, E)$ with a cost function c on the edges and a set of demand nodes D. We have to route each demand through k levels of facilities. The cost of placing a facility of level i at location v is f_{iv}. The demand first goes to a facility of level 1, from there to a facility of level 2, and so on until it reaches a facility of level k. As in the classical facility location problem, the goal is to optimize the total cost of facility placement and the cost of routing the demands through the levels of facilities.

This problem has been extensively studied in operations research literature [23, 37, 38, 2, 36]. We note that Shmoys, Tardos, and Aardal [36] present a 4 approximation for the case when the number of levels k is 2, and Aardal, Chudak, and Shmoys [1] subsequently present a 3 approximation for general k. Both these algorithms are based on solving linear program relaxations and rounding the resulting solutions.

We present the first known combinatorial approximation for general values of k. Our approximation ratio is $O(\log |D|)$. We reduce the multilevel facility location problem to Cost-Distance. We first make k copies of G which we denote $G_1, G_2, \ldots, G_k$. We construct a new graph G' from these copies as follows. For an edge $e \in E$, its bicriteria cost in each of the copies is $(0, c(e))$. For $v \in G$, we add an edge between its corresponding vertices in G_i and G_{i+1} with cost $(f_{iv}, 0)$. We call these facility edges. We add a dummy sink node s and connect it to all vertices in G_k with edges of cost $(f_{kv}, 0)$ for vertex v. The demand nodes D map to the corresponding nodes in G_1.

Note that the only way to move from G_i to G_{i+1} is to take one of the level i facility edges. This means that the multilevel facility location problem is equivalent to Cost-Distance on the graph G' with sink s and the demands in G_1. We therefore have the following theorem.

Theorem 5.1. *The* Cost-Distance *algorithm yields an* $O(\log |D|)$ *approximation for the* k*-level uncapacitated metric facility location problem running in* $O(k|D|^2(|E| + |V| \log k|V|))$ *time.*

5.7. Concave functions. Suppose we are given a graph and a set of sources and demands, and we wish to route all the demand to a single-sink node. For every pair of nodes in the graph, we are given a *concave function* which determines the cost of routing between those nodes given the amount of demand to be transported. Suppose further that this concave function for an edge e is of the form $f_e(d) = \min_{i=1}^{r_e}(a_{ie} + b_{ie}d)$. We can convert this problem to an instance of COST-DISTANCE as follows: For edge e, replace it by r_e parallel edges $e_1, e_2, \ldots, e_{r_e}$, where $c(e_i) = a_{ie}$ and $l(e_i) = b_{ie}$. Given d amount of demand, it is clear that in the new instance, the edge with index $\operatorname{argmin}_i(a_{ie} + b_{ie}d)$ will be used, so that the cost of transportation is precisely $\min_i(a_{ie} + b_{ie}d)$. This is precisely the cost of transporting the demand along edge e in the original instance. Therefore, if the cost of each edge is a piecewise linear monotonically nondecreasing concave function with polynomially many segments, this can be converted in polynomial time to an instance of COST-DISTANCE, hence leading to an $O(\log |S|)$ approximation.

6. Conclusion. We presented a general framework, COST-DISTANCE, for studying several network design problems which are unified by the themes of *economies of scale* and *single-sink*. We presented a simple to implement and intuitive randomized algorithm that achieves an $O(\log |S|)$ approximation, where S is the set of terminals. Since the preliminary version [31] appeared, this algorithm has been derandomized [13] using a linear programming formulation. Meyerson [30] presented a randomized online algorithm with a polylogarithmic competitive ratio. It was shown recently by Chuzhoy et al. [14] that this problem is $\Omega(\log \log |S|)$ hard to approximate, ruling out the possibility of a constant-factor approximation. For the multiterminal version of the problem, Chekuri et al. [12] present a polylogarithmic approximation based on rounding a natural linear programming relaxation. We note that despite all this subsequent work, our algorithm still achieves the best known approximation ratio for the single-terminal COST-DISTANCE problem.

REFERENCES

[1] K. AARDAL, F. CHUDAK, AND D. B. SHMOYS, *A 3-approximation algorithm for the k-level uncapacitated facility location problem*, Inform. Process. Lett., 72 (1999), pp. 161–167.

[2] K. AARDAL, M. LABBÉ, J. LEUNG, AND M. QUEYRANNE, *On the two-level uncapacitated facility location problem*, INFORMS J. Comput., 8 (1996), pp. 289–301.

[3] M. ANDREWS AND L. ZHANG, *The access network design problem*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1998, pp. 40–49.

[4] B. AWERBUCH AND Y. AZAR, *Buy-at-bulk network design*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1997, pp. 542–547.

[5] B. AWERBUCH, A. BARATZ, AND D. PELEG, *Cost-sensitive analysis of communication protocols*, in Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1990, pp. 177–187.

[6] Y. BARTAL, *Probabilistic approximation of metric spaces and its algorithmic applications*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1996, pp. 184–193.

[7] Y. BARTAL, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 161–168.

[8] K. BHARATH-KUMAR AND J. M. JAFFE, *Routing to multiple destinations in computer networks*, IEEE Trans. Comm., 31 (1983), pp. 343–351.

[9] M. CHARIKAR, C. CHEKURI, A. GOEL, AND S. GUHA, *Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median*, in Proceedings of the 30th ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 114–123.

[10] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin, *Approximating a finite metric by a small number of tree metrics*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1998, pp. 379–388.

[11] M. Charikar, J. Naor, and B. Schieber, *Resource optimization in QoS multicast routing of real-time multimedia*, in Proceedings of IEEE INFOCOM, IEEE Communications Society, Washington, DC, 2000, pp. 1518–1527.

[12] C. Chekuri, M.-T. Hajiaghayi, G. Kortsarz, and M. Salavatipour, *Approximation algorithms for non-uniform buy-at-bulk network design problems*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2006, pp. 677–686.

[13] C. Chekuri, S. Khanna, and J. Naor, *A deterministic algorithm for the cost-distance problem*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 232–233.

[14] J. Chuzhoy, A. Gupta, J. Naor, and A. Sinha, *On the approximability of some network design problems*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 943–951.

[15] M. Doar and I. Leslie, *How bad is naive multicast routing?*, in Proceedings of IEEE INFOCOM, IEEE Communications Society, Washington, DC, 1992, pp. 82–89.

[16] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 448–455.

[17] A. Goel and K. Munagala, *Extending greedy multicast routing to delay sensitive applications*, Algorithmica, 33 (2002), pp. 335–352.

[18] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM J. Comput., 24 (1995), pp. 296–317.

[19] S. Guha, A. Meyerson, and K. Munagala, *Hierarchical placement and network design problems*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2000, pp. 603–612.

[20] S. Guha, A. Meyerson, and K. Munagala, *A constant factor approximation for the single sink edge installation problems*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 383–388.

[21] A. Gupta, A. Kumar, and T. Roughgarden, *Simpler and better approximation algorithms for network design*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, ACM, New York, 2003, pp. 365–372.

[22] S.-P. Hong, H. Lee, and B. H. Park, *An efficient multicast routing algorithm for delay-sensitive applications with dynamic membership*, in Proceedings of IEEE INFOCOM, IEEE Communications Society, Washington, DC, 1998, pp. 1433–1440.

[23] L. Kaufman, M. vanden Eede, and P. Hansen, *A plant and warehouse location problem*, Operations Research Quarterly, 28 (1977), pp. 547–557.

[24] S. Khuller, B. Raghavachari, and N. Young, *Balancing minimum spanning and shortest path trees*, Algorithmica, 14 (1994), pp. 305–321.

[25] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, *Multicast routing for multimedia communication*, IEEE/ACM Trans. Networking, 1 (1993), pp. 286–292.

[26] G. Kortsarz and D. Peleg, *Approximating shallow-light trees*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1997, pp. 103–110.

[27] L. Kou, G. Markowsky, and L. Berman, *A fast algorithm for Steiner trees*, Acta Inform., 15 (1981), pp. 141–145.

[28] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, Springer-Verlag, London, 2002, pp. 229–242.

[29] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, III, *Bicriteria network design problems*, Computing Research Repository: Computational Complexity, 1998, http://arxiv.org/abs/cs.CC/9809103.

[30] A. Meyerson, *Online algorithms for network design*, in Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, 2004, pp. 275–280.

[31] A. Meyerson, K. Munagala, and S. Plotkin, *Cost-distance: Two metric network design*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2000, pp. 624–630.

[32] C. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Dover, New York, 1998.

[33] M. PARSA, Q. ZHU, AND J. J. GARCIA-LUNA-ACEVES, *An iterative algorithm for delay-constrained minimum-cost multicasting*, IEEE/ACM Trans. Networking, 6 (1998), pp. 361–374.

[34] R. RAVI AND A. SINHA, *Integrated logistics: Approximation algorithms combining facility location and network design*, in Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization, Cambridge, MA, 2002, pp. 212–229.

[35] F. S. SALMAN, J. CHERIYAN, R. RAVI, AND S. SUBRAMANIAN, *Buy-at-bulk network design: Approximating the single-sink edge installation problem*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1997, pp. 619–628.

[36] D. B. SHMOYS, É. TARDOS, AND K. AARDAL, *Approximation algorithms for facility location problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 265–274.

[37] D. TCHA AND B. LEE, *A branch-and-bound algorithm for the multi-level uncapacitated location problem*, European J. Oper. Res., 18 (1984), pp. 35–43.

[38] T. VAN ROY AND D. ERLENKOTTER, *A dual based procedure for dynamic facility location*, Management Sci., 28 (1982), pp. 1091–1105.

[39] L. WEI AND D. ESTRIN, *The trade-offs of multicast trees and algorithms*, in Proceedings of the International Conference on Computer Communications and Networks, IEEE Computer Society, Washington, DC, 1994.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1661–1694

UNIVERSAL ARGUMENTS AND THEIR APPLICATIONS*

BOAZ BARAK† AND ODED GOLDREICH‡

Abstract. We put forward a new type of computationally sound proof system called universal arguments. Universal arguments are related but different from both CS proofs (as defined by Micali [*SIAM J. Comput.*, 37 (2000), pp. 1253–1298]) and arguments (as defined by Brassard, Chaum, and Crépeau [*J. Comput. System Sci.*, 37 (1988), pp. 156–189]. In particular, we adopt the *instance-based* prover-efficiency paradigm of CS proofs but follow the computational-soundness condition of argument systems (i.e., we consider only cheating strategies that are implementable by *polynomial-size circuits*). We show that universal arguments can be constructed based on standard intractability assumptions that refer to polynomial-size circuits (rather than based on assumptions that refer to subexponential-size circuits as used in the construction of CS proofs). Furthermore, these protocols have a constant number of rounds and are of the public-coin type. As an application of these universal arguments, we weaken the intractability assumptions used in the non–black-box zero-knowledge arguments of Barak [in *Proceedings of the* 42*nd IEEE Symposiun on Foundations of Computer Science*, 2001]. Specifically, we only utilize intractability assumptions that refer to polynomial-size circuits (rather than assumptions that refer to circuits of some "nice" superpolynomial size).

Key words. probabilistic proof systems, computationally sound proof systems, zero-knowledge proof systems, proofs of knowledge, probabilistic checkable proofs, collision-resistant hashing, witness indistinguishable proof systems, error-correcting codes, tree hashing

AMS subject classifications. 94A60, 68Q01

DOI. 10.1137/070709244

1. Introduction.

Various types of *probabilistic* proof systems have played a central role in the development of computer science in the last two decades. The best known ones are interactive proofs [20], zero-knowledge proofs [20], and probabilistic checkable proofs (PCP) [15, 5, 14, 2], but other notions such as various types of computationally sound proofs (e.g., arguments [12] and CS proofs [23]) and multiprover interactive proofs [11] have made a prominent appearance as well. *Do we really need yet another type of probabilistic proof system?*

We believe that the answer is positive: the number of different (related) notions that "we need" is exactly the number of different notions that are natural, interesting, and/or useful. Confining ourselves to usefulness, we note that the new type of computationally sound proof system introduced in this paper has emerged in the context of trying to improve the constructions of non–black-box zero-knowledge arguments of Barak [6]. Furthermore, this proof system seems inherent to certain diagonalization techniques used in [6] and (in a different context) in [13].

1.1. Motivation: Applying diagonalization in cryptography.

A naive idea, which was discarded for decades in cryptography, is constructing a cryptographic scheme by "diagonalization," for example, enumerating all probabilistic polynomial-

*Received by the editors November 26, 2007; accepted for publication (in revised form) June 2, 2008; published electronically December 19, 2008. An extended abstract appeared in *Proceedings of the* 17*th IEEE Conference on Computational Complexity*, 2002, pp. 194–203.
http://www.siam.org/journals/sicomp/38-5/70924.html
†Department of Computer Science, Princeton University, Princeton, NJ 08544 (boaz@cs.princeton.edu). This research was supported by NSF grants CNS-0627526 and CCF-0426582, US-Israel BSF grant 2004288, and Packard and Sloan fellowships.
‡Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel (oded.goldreich@weizmann.ac.il). This research was partially supported by the Israel Science Foundation grant 460/05.

time adversaries and making sure that each of them fails. The main reason that this idea was discarded is that the resulting scheme will (necessarily) be more complex than the class of adversaries it defeats, while in cryptography a scheme should withstand adversaries that are (at least slightly) more complex than the scheme.

Still, as observed by Canetti, Goldreich, and Halevi [13] and Barak [6], a small twist on diagonalization may be useful in cryptography. The twist is using diagonalization in order to build a "trapdoor" so that the trapdoor can be used in some "imaginary setting" (e.g., by the simulator [6]) but not in the "real" setting (e.g., an actual execution of the proof system [6]).[1] Thus, the complexity of the imaginary setting is affected by the diagonalization, whereas the complexity of the real setting is independent of the diagonalization. Specifically, in the context of Barak's zero-knowledge arguments [6], the trapdoor is knowledge of the (adversarial) verifier's strategy, where this strategy (which is the locus of diagonalization) may be any polynomial-size circuit (where the polynomial is determined only after the proof system is specified). In the actual execution, the (zero-knowledge) prover does not use this trapdoor (but rather uses an NP witness to the real input), and so its complexity is independent of the complexity of the trapdoor (i.e., the cheating verifier's strategy). However, the simulator uses the trapdoor, and so its complexity depends on the latter (and so every polynomial-size adversary yields a related polynomial-time simulation).

For the foregoing idea to make sense, the verifier should not be able to distinguish the case in which the (real) prover uses an NP witness to the real input from the case in which the (simulated) prover uses the trapdoor (i.e., the cheating verifier's strategy). Indeed, Barak's protocol utilizes a witness indistinguishable (WI) proof for which both the NP witness (to the real input) and the trapdoor (i.e., the verifier's strategy) are valid witnesses. Thus, the honest verifier strategy in the WI proof must be independent of the length of the witness used by the prover. This is because, in one of the cases, the length of the witness is determined only after the proof system is specified (i.e., in the simulation, the length of the trapdoor is a polynomial, but this polynomial is determined and fixed only after the proof system is specified).

We conclude that in order to use diagonalization as described above, we should have a (WI) proof system that is capable of handling any "NP statement" (and not merely statements in any a priori fixed NP set). Put in other words, we need a *single* proof system that can be used to provide proofs for *any* set S in $\mathcal{NP}$ such that the running time and communication needed for verifying that $x \in S$ is bounded by a fixed (i.e., *single*) polynomial in $|x|$, which does *not* depend on the set S. In particular, it may be the case that $S \in \mathrm{Ntime}(p)$, where $p(\cdot)$ is a polynomial that is *larger* than the fixed polynomial bounding the verifier's complexity. We stress that, in contrast, typically when the phrase "proof systems for $\mathcal{NP}$" is used, the intended meaning is that every set $S \in \mathcal{NP}$ has a different proof system (and the complexity of verifying that $x \in S$ is bounded by an *S-dependent polynomial* in $|x|$).

1.2. The notion of universal arguments. For the sake of simplicity, we define and present proof systems for only the *universal set* $S_{\mathcal{U}}$ defined such that the tuple (M, x, t) is in $S_{\mathcal{U}}$ if M is a nondeterministic machine that accepts x within

[1] In [13], the "imaginary setting" is an implementation of the random oracle by a function ensemble (shown not to exist), whereas the "real setting" is the ideal (random oracle model) setting in which the scheme uses a random oracle. (Indeed, our perspective here is opposite to the one in [13], where the random oracle is considered "imaginary" and its implementations by function ensembles are considered "real.")

t steps.[2] This suffices for handling any NP set via a single protocol, because every $\mathcal{NP}$ set S is linear-time reducible to $S_{\mathcal{U}}$ (e.g., via the mapping $x \mapsto (M_S, x, 2^{|x|})$, where M_S is any fixed nondeterministic polynomial-time machine that decides S). Thus, a proof system for $S_{\mathcal{U}}$ allows us to handle all "NP statements" (in a uniform manner): for any $S \in \mathcal{NP}$, when wishing to verify the assertion "x in S," the verifier should just use the proof system of $S_{\mathcal{U}}$ on input $(M_S, x, 2^{|x|})$, where M_S is as described above. (In fact, $S_{\mathcal{U}}$ is $\mathcal{NE}$-complete by an analogous linear-time reduction.)[3]

We also consider the natural witness relation for $S_{\mathcal{U}}$, denoted $R_{\mathcal{U}}$: the pair $((M, x, t), w)$ is in $R_{\mathcal{U}}$ if M (viewed here as a two-input deterministic machine) accepts (x, w) within t steps. Loosely speaking, a *universal argument* system (or a *universal argument system for* $S_{\mathcal{U}}$) is a two-party protocol (P, V), for common inputs of the form (M, x, t), that satisfies the following:

Efficient verification. The *total* time spent by the (probabilistic) verifier V is polynomial in length of the common input (i.e., polynomial in $|(M, x, t)| = O(|M| + |x| + \log t)$). In particular, all messages exchanged in the protocol have a length that is so bounded.

Completeness by a relatively efficient prover. For every $((M, x, t), w)$ in $R_{\mathcal{U}}$, on common input (M, x, t), when P is given auxiliary input w, it always convinces V. Furthermore, the total time spent by P in this case is bounded by a fixed polynomial in ($|M|$ and) $T_M(x, w)$, where $T_M(x, w) \leq t$ is the number of steps taken by M on the input (x, w).

Computational soundness. For every polynomial-size circuit family $\{C_n\}_{n \in \mathbf{N}}$ and every $(M, x, t) \in \{0,1\}^n \setminus S_{\mathcal{U}}$, the probability that, on common input (M, x, t), the ("cheating") circuit C_n succeeds in fooling V (into accepting (M, x, t)) is negligible (as a function of n).

(The actual definition appears in section 2.)

Relation to prior notions. Universal arguments are related but different from both CS proofs (as defined by Micali [23]) and arguments (as defined by Brassard, Chaum, and Crepeau [12]). Specifically

1. the efficient-verification condition is identical in all definitions (except that arguments are typically defined only for individual sets in $\mathcal{NP}$);
2. the "completeness by a relatively efficient prover" condition follows the instance-based paradigm of CS proofs (but provides the prover with an auxiliary input);
3. the computational-soundness condition is exactly as it is in argument systems (and is typically weaker than the one in CS proofs).

Thus, in a sense, universal arguments are a hybrid of arguments and CS proofs. Indeed, universal arguments are weaker than CS proofs, but the point is that we will be able to construct a universal argument based on a weaker assumption than the ones that seem necessary for constructing CS proofs (cf. [23, 6]).

We comment that computational soundness seems unavoidable in any proof system for $S_{\mathcal{U}}$ that satisfies the efficient-verification condition (even just "uniformly for

[2] One nice feature of $S_{\mathcal{U}}$ is that it comes with a natural measure of the complexity of instances: the complexity of (M, x, t) is the actual time it takes M to accept x (when using the best sequence of nondeterministic choices). (Similarly, the complexity of (M, x, t) coupled with the witness w is the actual time it takes M to accept x when using w as the sequence of nondeterministic choices.) Such a complexity measure is pivotal to the refined formulation of the prover complexity condition.

[3] That is, for $S \in \mathrm{Ntime}(e)$, where $e(n) = 2^{cn}$ for some constant c, we use the reduction $x \mapsto (M_S, x, 2^{c|x|})$. Furthermore, every set in $\mathcal{NEXP}$ is polynomial-time (but not linear-time) reducible to $S_{\mathcal{U}}$.

all $\mathcal{NP}$"). In contrast, statistical soundness (coupled with the standard notion of efficient verification) would have implied that $S_{\mathcal{U}}$ (or "just" all $\mathcal{NP}$) is in DSPACE(p), for some fixed polynomial p. (The reason is that the total communication in such a protocol must be upper bounded by a fixed polynomial p and that an optimal prover strategy can be implemented in space p.)

On natural applications of universal arguments. A strange-looking aspect of universal arguments is that, on some YES-instances, the designated prover may run in more time than allowed to cheating provers (i.e., a fixed polynomial in $T_M(.,.)$ may be larger than an arbitrary polynomial in the length of the common input).[4] However, in typical applications (such as ours), the designated prover will never be invoked on such inputs (i.e., on inputs requiring the prover to run for a time that is superpolynomial in their length). Actually, we shall use only the fact that a universal-argument system guarantees the following behavior with respect to *any* polynomial p (which may be selected after the system is specified):

1. For every $(y,w) \in R_{\mathcal{U}}$ such that $y = (M,x,t)$ and $t \leq p(|x|)$, it holds that (given y and w) the designated prover convinces the verifier to accept y, and the total time used by the prover is $\text{poly}(|M|+t) = \text{poly}(|y|)$. Specifically, there exists a fixed polynomial q_0 that is associated with the universal-argument system such that the running time of the designated prover on input (M,x,t) (and w) is $q_0(|M|+t) \leq q_0(|M|+p(|x|)) < (q_0 \circ p)(|y|)$.
2. For every polynomial q and every sufficiently long $y = (M,x,t) \notin S_{\mathcal{U}}$, it holds that no $q(|y|)$-size circuit can convince the verifier to accept y.

Thus, in this restricted case, the complexity bound considered in the soundness condition (of item 2) may exceed the complexity of the designated prover when handling YES-inputs of the type mentioned in item 1 (i.e., $(M,x,t) \in S_{\mathcal{U}}$ such that $t \leq p(|x|)$). Needless to say, the foregoing items cover all NP sets (where a set $S \in \mathcal{NP}$ is handled by selecting the polynomial p such that $S \in \text{Ntime}(p)$ holds).

1.3. The construction of universal arguments. By adapting the construction of Kilian [21], one can show that the existence of *strong* collision-resistant hashing functions implies the existence of universal arguments (and even CS proofs for $S_{\mathcal{U}}$; cf. Micali [23]). By *strong* collision-resistant hashing we mean families of functions for which collisions are hard to find even by using *subexponential-size* circuits. The goal, achieved in this paper, is to construct universal arguments based only on *standard* collision-resistant hashing, that is, families of functions for which collisions are hard to find by *polynomial-size* circuits. That is, we obtain the following theorem.

THEOREM 1.1 (our main result). *The existence of* (standard) *collision-resistant hashing functions implies the existence of universal arguments. Furthermore, these proof systems are of the public-coin*[5] *type and use a constant number of rounds.*

Our construction of universal arguments adapts Kilian's construction [21] in a quite straightforward manner. Our contribution is in the analysis of this construction. Unlike in previous analyses (as in [21, 23]), when establishing computational soundness

[4] This phenomena does not occur in arguments [12] and in CS proofs [23]: arguments were defined only for individual sets in $\mathcal{NP}$, and so the issue never arises. In the case of CS proofs, the definition of computational soundness relates to cheating provers of size exponential in the security parameter, which is typically set to be linear in the length of the common input [23]. Thus, the cheating provers are always allowed more running time than the designated prover (since its running time is always at most exponential in the length of the common input). However, it seems that allowing the adversaries time that is exponential in the security parameter requires using intractability assumptions that refer to exponential (or subexponential) circuits.

[5] Also known as (a.k.a.) Arthur–Merlin systems (cf. [3]).

via contradiction, we cannot afford to derive a collision-forming circuit of size that is polynomial in the worst-case time complexity of the designated prover (because the designated prover may have (worst-case) complexity that is superpolynomial (in the input length)).[6] We need to derive a collision-forming circuit of size that is polynomial in the input length. Indeed, doing so allows us to use standard collision-resistant hashing (rather than strong ones).

The analysis is further complicated by our desire to establish a "proof-of-knowledge" property, which is needed for our main application. This is discussed next.

1.4. Application to zero-knowledge arguments. Barak's construction [6] of non–black-box zero-knowledge arguments (for any set in $\mathcal{NP}$) uses a witness indistinguishable (WI) argument of knowledge for $R_{\mathcal{U}}$.[7] In his protocol, the prover uses this WI argument (of knowledge) to prove that it knows either an NP witness for the original common input or a program that fits the verifier functionality (as reflected in the challenge-respond exchange that follows). Thus, as a first step, we need to transform our universal argument (of knowledge) into a corresponding WI universal argument (of knowledge). The transformation essentially follows Barak's transformation [6], but then we encounter a second place where Barak uses a superpolynomial hardness assumption: Barak uses a collision-resistant hashing function to hash "$S_{\mathcal{U}}$-witnesses" (into fixed-length strings), where the length of these witnesses is bounded by some superpolynomial function (but not by any polynomial). Consequently, a collision on such long strings yields only the violation of a superpolynomial collision-resistant assumption. To avoid superpolynomial hardness assumptions, we hash these witnesses by combining "tree hashing" (as in Kilian's construction [21]) with an error-correcting code. Specifically, first the witness string is encoded using an error-correcting code, and then the tree hashing is applied to the result. Thus, if two different strings are hashed to the same value, then we can form a collision with respect to the basic hashing function (used in the tree hashing) by considering a uniformly selected leaf (which is quite likely to be assigned different values under an error-correction coding of different strings). Combining the foregoing, we obtain the following theorem.

THEOREM 1.2 (our main application). *The existence of* (standard) *collision-resistant hashing functions implies the existence of* (non–black-box) *zero-knowledge arguments for any set in $\mathcal{NP}$ such that these protocols have the following additional properties:*

1. *The protocol has a constant number of rounds and uses only public coins.*
2. *The simulator runs in strict* (rather than expected) *probabilistic polynomial time.*
3. *The protocol remains zero-knowledge when, say, n^2 copies are executed concurrently.*[8]

[6] Specifically, Kilian's construction [21] uses a probabilistic checkable proof (PCP) system, and the hypothetical violation of its computational soundness is shown to yield a (relatively small) collision-forming circuit, but this circuit is bigger than the length of the corresponding PCP oracle (which, in the case of $S_{\mathcal{U}}$, is exponential in the input length). Instead, we show how to obtain a collision-forming circuit that is smaller than the length of the corresponding PCP oracle.

[7] In fact, Barak uses a CS proof (of knowledge) for $R^f_{\mathcal{U}} \subset R_{\mathcal{U}}$, where f is any "nice" superpolynomial function (e.g., $f(n) = n^{\log_2 n}$) and $((M,x,t),w)$ is in $R^f_{\mathcal{U}}$ only if $t \leq f(|x|)$. He constructs such a CS proof assuming the existence of hashing functions that are resilient with respect to f-size circuits (rather than subexponential hardness which would have been required for a CS proof for $R_{\mathcal{U}}$).

[8] In fact, we can tolerate $p(n)$ concurrent copies, where p is an arbitrary polynomial that is fixed a priori; that is, the argument system depends on the polynomial p.

Recall that, assuming $\mathcal{NP} \not\subseteq \mathcal{BPP}$, each of the three extra properties requires a non–black-box simulator (and that such protocols were first presented in [6]).[9] Thus, Theorem 1.2 establishes the main result of [6] *under a weaker assumption*: we only assume the existence of hashing functions that are resilient with respect to polynomial-size circuits (rather than with respect to circuits of some superpolynomial size).

1.5. Organization. In section 2 we define universal arguments, and in section 3 we show how to construct them (using any family of collision-resistant hash functions). The application to the construction of non–black-box zero-knowledge arguments is presented in section 4. Appendix B contains a revised treatment of the notion of a nonoblivious commitment scheme, which may be of independent interest (because it augments the initial treatment provided in [16, section 4.9.2.1] and corrected in [17, section C.3.3]).

Comment. The current version of this paper differs from prior versions mainly in section 4.1. The presentation in prior versions (see [8]) relied on the existence of constant-round, public-coin *strong*-WI proofs of knowledge for any NP set. Unfortunately, contrary to prior misconceptions (cf. [16, section 4.6]), such protocols are not known to exist [17, Appendix C.3]. Indeed, this gap can be bypassed—as done in [7] and in the current version by using a rather ugly patch—but our hope was to bridge the gap (i.e., prove the existence of the said strong-WI protocols) and maintain the original presentation. Having failed to do so for several years, we decided to archive the current version.

2. The definition of universal arguments. Let us start with some general notions. For an integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$. We denote by $\mu : \mathsf{N} \to [0,1]$ an unspecified *negligible function*; that is, for every positive polynomial p and all sufficiently large n, it holds that $\mu(n) < 1/p(n)$. We say that an event occurs with *overwhelmingly high probability* if it occurs with probability at least $1 - \mu(n)$, where n is the relevant security parameter. For a pair of (interactive) strategies denoted by (P, V), we denote by $(P(w), V)(y)$ the output of V when interacting with $P(w)$ on common input y, where $P(w)$ denotes the functionality of P when given auxiliary input w.

In continuation of the discussion in section 1.2, we now define universal argument systems (for $S_{\mathcal{U}}$). Recall that $S_{\mathcal{U}} = \{(M, x, t) : \exists w \text{ s.t. } ((M, x, t), w) \in R_{\mathcal{U}}\}$, where $((M, x, t), w) \in R_{\mathcal{U}}$ if M accepts (x, w) within t steps. Let $T_M(x, w)$ denote the number of steps made by M on input (x, w); indeed, if $((M, x, t), w) \in R_{\mathcal{U}}$, then $T_M(x, w) \leq t$. Recall that $|(M, x, t)| = O(|M| + |x| + \log t)$; that is, t is given in binary. In the following definition, we incorporate a (weak) proof-of-knowledge property (which was mentioned in sections 1.3 and 1.4 but not in section 1.2).[10]

DEFINITION 2.1 (universal argument). *A universal-argument system is a pair of strategies denoted by (P, V) that satisfies the following properties:*

Efficient Verification. There exists a polynomial p such that for any $y = (M, x, t)$, the total time spent by the (probabilistic) *verifier strategy V, on the common input y, is at most $p(|y|)$. In particular, all messages exchanged in the protocol have a length smaller than $p(|y|)$.*

[9] Indeed, see [6, 7] for a discussion of various results regarding the impossibility of achieving the above via a black-box simulator. We stress that the current discussion refers to protocols of negligible soundness error.

[10] Indeed, the (weak) proof-of-knowledge property may be considered an auxiliary feature, which need not be mandated by the basic definition of a "universal argument" (i.e., Definition 2.1). Our choice to include this property in Definition 2.1 is motivated by the specific context and results of the present work.

Completeness via a relatively efficient prover. For every $((M,x,t),w)$ *in* $R_{\mathcal{U}}$,

$$\Pr[(P(w),V)(M,x,t)=1]=1.$$

Furthermore, there exists a polynomial p *such that for every* $((M,x,t),w)\in R_{\mathcal{U}}$ *the total time spent by* $P(w)$*, on the common input* (M,x,t)*, is at most* $p(|M|+T_M(x,w))\le p(|M|+t)$.

Computational Soundness. For every polynomial-size circuit family $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$ *and every* $(M,x,t)\in\{0,1\}^n\setminus S_{\mathcal{U}}$,

$$\Pr\left[\left(\widetilde{P}_n,V\right)(M,x,t)=1\right] \;<\; \mu(n),$$

where $\mu:\mathsf{N}\to[0,1]$ *is a negligible function.*[11]

A weak proof-of-knowledge property. For every positive polynomial p *there exists a positive polynomial* p' *and a probabilistic polynomial-time oracle machine* E *such that the following holds:*[12] *for every polynomial-size circuit family* $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$ *and every sufficiently long* $y=(M,x,t)\in\{0,1\}^*$*, if* $\Pr[(\widetilde{P}_n,V)(y)=1]>1/p(|y|)$*, then*

$$\Pr_r\left[\begin{array}{l}\exists w=w_1\cdots w_t\in R_{\mathcal{U}}(y)\\ \quad(\forall i\in[t])\quad E_r^{\widetilde{P}_n}(y,i)=w_i\end{array}\right] \;>\; \frac{1}{p'(|y|)}, \tag{2.1}$$

where $R_{\mathcal{U}}(y)\stackrel{\text{def}}{=}\{w:(y,w)\in R_{\mathcal{U}}\}$ *and* $E_r^{\widetilde{P}_n}(.,.)$ *denotes the function defined by fixing the random tape of* E *to equal* r *and providing the resulting* E_r *with oracle access to* $\widetilde{P}_n$*. The oracle machine* E *is called a* (knowledge) extractor.

A few comments regarding the weak proof-of-knowledge property are in place. First, note that the condition "$\forall i\in[t]$ it holds that $E_r^{\widetilde{P}_n}(y,i)=w_i$" means that $E_r^{\widetilde{P}_n}(y,.)$ is an *implicit representation* of the string $w=w_1\cdots w_t$ (i.e., any specific bit of w is obtained by instantiating the second input to $E_r^{\widetilde{P}_n}(y,.)$ accordingly). If $\Pr[(\widetilde{P}_n,V)(y)=1]>1/p(|y|)$, then at least an $1/p'(|y|)$ fraction of the possible r's yields such implicit representations of some string in $R_{\mathcal{U}}(y)$, but these strings are not necessarily equal (i.e., different r's may yield different strings in $R_{\mathcal{U}}(y)$). Implicit (rather than explicit) representation is required here because we want the extractor to run in polynomial time, whereas the length of the strings in $R_{\mathcal{U}}(y)$ may not be bounded by any polynomial (in $|y|$). Finally, we note that the weak proof-of-knowledge property is indeed weaker than the standard definition of a proof of knowledge (cf. [9], [16, section 4.7], and Footnote 12), but it suffices for the applications that we have in mind.

3. The construction of universal arguments. As mentioned in section 1.3, by adapting the construction of Kilian [21], one can easily show that the existence of *strong* collision-resistant hashing functions implies the existence of universal arguments (and even CS proofs for $S_{\mathcal{U}}$; cf. Micali [23]). Here we show how a similar adaptation, when using only *standard* collision-resistant hashing functions, yields universal arguments. Our focus is on demonstrating the computational soundness of this construction, which should now be established under a weaker assumption than the one used in [21, 23].

[11] That is, the function μ is fixed depending on $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$ and on an infinite sequence of inputs that contains at most one input per each input length. Equivalently, one may fix the function μ depending only on the polynomial bounding the size of the circuit family.

[12] Indeed, the polynomial p' as well as the (polynomial) running time of E may depend on the polynomial p (which determines the noticeable threshold probability).

3.1. Motivation. In order to explain the difficulty and its resolution, let us recall the basic construction of Kilian [21] (used also by Micali [23]), as adapted to our setting.

Our starting point is a $\mathcal{PCP}[\text{poly}, \text{poly}]$ system for $S_{\mathcal{U}} \in \mathcal{NEXP}$, which is used in the universal-argument system as follows. The verifier starts by sending the prover a hashing function. The prover constructs a PCP proof/oracle (corresponding to the common input and its own auxiliary input), places the bits of this oracle at the leaves of a polynomial-depth full binary tree, and places in each internal node the hash value obtained by applying the hashing function to the labels (i.e., contents) of its children. The prover sends the *label of the root* to the verifier, which responsed by sending a random tape of the type used by the PCP verifier. Both parties determine the queries corresponding to this tape, and the prover responds with the values of the corresponding leaves, along with the labels of the vertices along the paths from these leaves to the root (as well as the labels of the siblings of these vertices). The verifier checks that this sequence of labels matches the corresponding applications of the hashing function and also emulates the PCP verifier. Ignoring (for a moment) the issue of the prover's complexity, the problem we consider next is that of establishing computational soundness.

The naive approach is to consider how the prover responds to each of the possible random tapes sent to it. If the prover answers consistently (i.e., with leaf labels that depend only on the leaf location), then we obtain a PCP oracle, and soundness follows by the soundness of the PCP scheme. On the other hand, inconsistent labels for the same leaf yield a (hashing) collision somewhere along the path to the root. However, in order to find such a collision, we must spend time proportional to the size of the tree, which yields a contradiction only in the case that the hashing function is supposed to withstand adversaries that use that much time. Note that the size of the tree is picked by the (adversarial) prover, and since we wish to handle $S_{\mathcal{U}}$ (or merely "only" all of $\mathcal{NP}$) we do not have an a priori polynomial bound on the size of the tree that the prover is allowed to use (because no such bound exists for the designated prover). But in such a case, if the tree is exponential (or even merely super polynomial) in the security parameter, then we derive a contradiction only when using hashing functions of subexponential (respectively, superpolynomial) security.

In contrast to the (above) naive approach, the approach taken here is to consider each leaf separately rather than all leaves together. That is, the naive analysis distinguishes the case that the prover answers inconsistently on *some* leaf from the case that it answers consistently on *all* leaves. Instead, we consider each leaf separately and distinguish the case that the prover answers inconsistently on *this leaf* from the case that it answers consistently on *this leaf*. Loosely speaking, we call a leaf *good* if the prover answers consistently on it and observe that if a big fraction of the leaves is good, then soundness follows by the soundness of the PCP scheme (regardless of the contents of other leaves). On the other hand, if sufficiently many leaves are not good, then we obtain a collision by picking a random leaf (hoping that it is not good) and obtaining inconsistent labels for it. This requires being able to uniformly select a random tape that causes the PCP verifier to make the corresponding query, a property which is fortunately enjoyed by the relevant PCP systems.

We warn that the above is merely a rough description of the main idea in our analysis. Furthermore, in order to establish the proof-of-knowledge property of our construction, we need to rely on an analogous property of the PCP system (which again happens to be satisfied by the relevant PCP systems).

3.2. The PCP system in use. We first recall the basic definition of a PCP system. Loosely speaking, a probabilistically checkable proof (PCP) system consists of a probabilistic polynomial-time verifier having access to an oracle which represents a proof in redundant form. Typically, the verifier accesses only a few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. It is required that if the assertion holds, then the verifier always accepts (i.e., when given access to an adequate oracle); whereas if the assertion is false, then the verifier must reject with high probability (as specified in an adequate bound), no matter which oracle is used. The basic definition of the PCP setting is given in item 1 of Definition 3.1 below. Typically, the complexity measures introduced in item 2 of Definition 3.1 are of key importance, but this is not the case in the current work.

DEFINITION 3.1 (PCP—basic definition).

1. *A* probabilistically checkable proof (PCP) system with error bound $\epsilon : \mathsf{N} \to [0,1]$ for a set S *is a probabilistic polynomial-time oracle machine (called* verifier*), denoted by* V *satisfying the following:*
   - Completeness. *For every* $x \in S$ *there exists an oracle* π_x *such that* V*, on input* x *and access to oracle* π_x*, always accepts* x*.*
   - Soundness. *For every* $x \notin S$ *and every oracle* π*, machine* V*, on input* x *and access to oracle* π*, rejects* x *with probability at least* $1 - \epsilon(|x|)$*.*
2. *Let* r *and* q *be integer functions. The complexity class* $\mathcal{PCP}_\epsilon[r(\cdot), q(\cdot)]$ *consists of sets having a PCP system with error bound* ϵ *in which the verifier, on any input of length* n*, makes at most* $r(n)$ *coin tosses and at most* $q(n)$ *oracle queries.*

Note that if S has a PCP system with error bound ϵ, then $S \in \mathcal{PCP}_\epsilon[p(\cdot), p(\cdot)]$ for some polynomial p. Here we will care only that $S_{\mathcal{U}} \in \mathcal{NE}$ has a PCP system with an exponentially decreasing error bound (i.e., $\epsilon(n) = 2^{-n}$). Instead of caring about the refined complexity measures (of item 2), we will care about the following *additional properties* which are satisfied by some PCP systems, where only some of these properties were explicitly considered before (see discussion below).

DEFINITION 3.2 (PCP—auxiliary properties). *Let* V *be a PCP verifier with error* $\epsilon : \mathsf{N} \to [0,1]$ *for a set* $S \in \mathcal{NEXP}$*, and let* R *be a corresponding witness relation. That is, if* $S \in \mathrm{Ntime}(t(\cdot))$*, then we refer to a* polynomial-time decidable *relation* R *satisfying* $x \in S$ *if and only if there exists* w *of length at most* $t(|x|)$ *such that* $(x, w) \in R$*. We consider the following auxiliary properties:*

Relatively efficient oracle construction. This property holds if there exists a polynomial-time algorithm P *such that, given any* $(x, w) \in R$*, algorithm* P *outputs an oracle* π_x *that makes* V *always accept (i.e., as in the completeness condition).*

Nonadaptive verifier. This property holds if the verifier's queries are determined based only on the input and its internal coin tosses, independently of the answers given to previous queries. That is, V *can be decomposed into a pair of algorithms* Q *and* D *such that on input* x *and random tape* r*, the verifier makes the query sequence* $Q(x, r, 1), Q(x, r, 2), \ldots, Q(x, r, p(|x|))$*, obtains the answers* $b_1, \ldots, b_{p(|x|)}$*, and decides according to* $D(x, r, b_1 \cdots b_{p(|x|)})$*, where* p *is some fixed polynomial.*

Efficient reverse sampling. This property holds if there exists a probabilistic polynomial-time algorithm S *such that, given any string* x *and integers* i *and* j*, algorithm* S *outputs a uniformly distributed* r *that satisfies* $Q(x, r, i) = j$*, where* Q *is as defined in the previous item.*

A proof-of-knowledge property. This property holds if there exists a probabilistic polynomial-time oracle machine E such that the following holds:[13] *for every x and π, if $\Pr[V^\pi(x) = 1] > \epsilon(|x|)$, then there exists $w = w_1 \cdots w_t$ such that $(x, w) \in R$ and $\Pr[E^\pi(x, i) = w_i] > 2/3$ holds for every $i \in [t]$.*

Nonadaptive PCP verifiers were explicitly considered in several works, and in fact in some sources PCP is defined in terms of nonadaptive verifiers. (Needless to say, almost all PCP systems use nonadaptive verifiers.) The oracle-construction and proof-of-knowledge properties are implicit in some works, and are known to hold for many PCP systems (although we are not aware of a text that contains a proof of this fact). To the best of our knowledge, the reverse-sampling property was not considered before. Nevertheless, it can be verified that any $S \in \mathcal{NEXP}$ has a PCP system that satisfies all of the foregoing properties.

THEOREM 3.3. *For every $S \in \mathcal{NEXP}$ and for every $\epsilon : \mathsf{N} \to [0, 1]$ such that for some positive polynomial p it holds that $\epsilon(n) > 2^{-p(n)}$ for all n, there exists a PCP system with error ϵ for S such that this PCP satisfies the four properties listed in Definition* 3.2.

Proof sketch. For $S \in \mathrm{Ntime}(t(\cdot))$, we consider the $\mathcal{PCP}_{1/2}[O(\log t(\cdot)), \mathrm{poly}(\cdot)]$ system presented by Babai et al. [5] (i.e., the starting point of Arora et al. [2, 1]). (We stress that this PCP system, unlike the one of Feige et al. [14], uses oracles of length polynomial in t.) This PCP system is evidently nonadaptive, and it is well known to satisfy the oracle-construction property. It is also known (alas less known) that this PCP system satisfies the proof-of-knowledge property. Finally, it is easy to see that this PCP system (as well as any reasonable PCP system we know of) also satisfies the reverse-sampling property.[14] *Further details regarding the proof of all of the foregoing facts can be found in Appendix* A. Thus, we obtain a PCP system with error $1/2$ (for S) that satisfies all of the auxiliary properties listed in Definition 3.2. To obtain the desired error of ϵ, we apply straightforward error reduction, while noting that this process does not affect the oracle, and so the resulting (error-reduced) PCP preserves all of the auxiliary properties. □

3.3. The actual construction. The construction is an adaptation of Kilian's construction [21] (used also by Micali [23]). Using Theorem 3.3, we start with a PCP system with error $\epsilon(n) = 2^{-n}$ for $S_{\mathcal{U}}$ that satisfies the auxiliary properties in Definition 3.2. Actually, the corresponding witness relation will not be $R_{\mathcal{U}}$ as defined in section 1.2 but rather a minor modification of it, denoted $R'_{\mathcal{U}}$: the pair $((M, x, t), (w, 1^{t'}))$ is in $R'_{\mathcal{U}}$ if M accepts (x, w) in $t' \leq t$ steps. (The purpose of the modification is to obtain a relation that is decidable in polynomial time, as required in Definition 3.2.) Let V_{pcp} denote the aforementioned PCP system (or rather its verifier) and $P_{\mathrm{pcp}}, Q_{\mathrm{pcp}}, D_{\mathrm{pcp}}, S_{\mathrm{pcp}}$, and E_{pcp} denote the auxiliary algorithms (or machines) guaranteed by Definition 3.2 (i.e., P_{pcp} is the oracle-constructing procedure, Q_{pcp} determines the verifier's queries, D_{pcp} describes the verifier's final decision, S_{pcp} provides reverse sampling, and E_{pcp} is the "witness extractor" guaranteed in the proof-of-knowledge property).

[13] For negligible ϵ (as used below), this proof-of-knowledge property is stronger than the *standard* proof-of-knowledge property (as in [9] and [16, section 4.7.1]). Indeed, the proof-of-knowledge property in Definition 3.2 is analogous to the definition of a *strong* proof-of-knowledge (as in [16, section 4.7.6]).

[14] This property follows from the structure of the standard PCP systems. In our case, the system consists of a *sum check* (Lund et al. [22]) and a *low-degree test*. In both tests, the queries are selected in a very simple manner, and what is complex (at least in the case of low-degree tests) is the analysis of the test.

A second ingredient used in the construction is a family of *collision-resistant hashing* functions. That is, a collection of (uniformly polynomial-time computable) functions $\{h_\alpha : \{0,1\}^* \to \{0,1\}^{|\alpha|}\}$ such that for every (nonuniform) family of polynomial-size circuits $\{C_n\}_{n\in\mathsf{N}}$

$$\Pr_{\alpha\in\{0,1\}^n}[C_n(\alpha) = (x,y) \text{ s.t. } x \neq y \text{ and } h_\alpha(x) = h_\alpha(y)] = \mu(n),$$

where μ is a negligible function. Such families can be constructed based on, for example, the conjectured intractability of factoring (integers); see, e.g., [17, section 6.2.3].

CONSTRUCTION 3.4 (a universal argument for $S_{\mathcal{U}}$).

Common input: $y = (M, x, t)$, *supposedly in* $S_{\mathcal{U}}$. *Let* $n \stackrel{\text{def}}{=} |y|$.

Auxiliary input to the prover: w *such that supposedly* $(y, w) \in R_{\mathcal{U}}$ *holds.*

First verifier step (V1): *Uniformly select* $\alpha \in \{0,1\}^n$ *and send it to the prover.*

First prover step (P1): *When describing the prover's actions, we assume that* $(y, w) \in R_{\mathcal{U}}$.

1. Preliminary action by the prover. *The prover invokes* M *on input* (x, w) *and obtains* $t' = T_M(x, w)$. *Assuming that* $(y, w) \in R_{\mathcal{U}}$ *and letting* $w' = (w, 1^{t'})$, *the prover obtains an* $R'_{\mathcal{U}}$*-witness; that is,* $(y, w') \in R'_{\mathcal{U}}$.
2. Oracle construction. *Invoking* P_{pcp} *on* (y, w'), *the prover obtains* $\pi_y = P_{\text{pcp}}(y, w')$.

   Assume without loss of generality that $|\pi_y|$ *is a power of* 2, *and let* $d \stackrel{\text{def}}{=} \log_2 |\pi_y|$.
3. Construction of a hashing tree. *The prover constructs a binary tree of depth* d *and associates its nodes with binary strings of length at most* d *such that the root is associated with the empty string, and an internal node associated with* γ *has children associated with* $\gamma 0$ *and* $\gamma 1$. *Using the oracle* π_y (*just constructed*) *and the hashing function* h_α (*sent by the verifier*), *the prover* labels *the nodes of this tree as follows:*
   - *The* label of a leaf *associated with* $\gamma \in \{0,1\}^d$ *is the value of* π_y *at position* γ; *that is, this label denoted by* ℓ_γ *equals the answer of the oracle* π_y *to the query* γ.
   - *The* label of an internal node *associated with* $\gamma \in \cup_{i=0}^{d-1}\{0,1\}^i$ *is the value obtained by applying* h_α *to the string* $\ell_{\gamma 0}\ell_{\gamma 1}$. *This label is denoted by* ℓ_γ.

   Thus, the label of the node associated with $\gamma \in \cup_{i=0}^{d}\{0,1\}^i$ *is denoted by* ℓ_γ.
4. The actual message sent *by the prover is the depth of the tree and the label of its root. That is, the prover sends the pair* (d, ℓ_λ) *to the verifier.*

Second verifier step (V2). *The verifier selects uniformly a random tape* r *for the PCP system and sends* r *to the prover.*

Second prover step (P2). *The prover provides the corresponding* (PCP) *answers, augmented by* proofs of the consistency *of these answers with the label of the root as provided in step* (P1).

1. Determining the queries. *Invoking* Q_{pcp}, *the prover determines the sequence of queries that the PCP system makes on a random tape* r. *That is, for* $i = 1, \ldots, m$, *it computes* $q_i = Q_{\text{pcp}}(y, r, i)$, *where* $m \stackrel{\text{def}}{=} \text{poly}(n)$ *is the number of queries made by the system.*
2. The message sent. *For* $i = 1, \ldots, m$ *and* $j = 0, \ldots, d-1$, *the prover sends the pair* $(\ell_{\gamma_{i,j}0}, \ell_{\gamma_{i,j}1})$, *where* $\gamma_{i,j}$ *is the* j*-bit long prefix of* q_i. (Note that this message contains, for every $i \in [m]$, the value of ℓ_{q_i} as

well as information that enables the authentication of this value with respect to ℓ_λ.)

Verifier final decision step (V3): *The verifier checks that the answers provided by the prover would have been accepted by the PCP verifier and that the corresponding proofs of consistency (with the label of the root) are valid. That is, denoting by ℓ'_γ the label provided by the prover for the node associated with γ, the verifier accepts if and only if all of the following checks pass:*

1. *Invoking D_{pcp}, the verifier checks whether, on input y and random tape r, the PCP verifier would have accepted the answer sequence $\ell'_{q_1}, \ldots, \ell'_{q_m}$, where $q_i = Q_{\mathrm{pcp}}(y, r, i)$ (as in step (P2)). That is, it checks whether $D_{\mathrm{pcp}}(y, r, \ell'_{q_1} \cdots \ell'_{q_m}) = 1$, where r is the random tape chosen in step (V2).*
2. *Check whether the labels provided are consistent with the label of the root of the tree as sent in step P1. That is, for $i = 1, \ldots, m$ and $j = 0, \ldots, d-1$, check whether $\ell'_{\gamma_{i,j}} = h_\alpha(\ell'_{\gamma_{i,j}0}\ell'_{\gamma_{i,j}1})$, where $\gamma_{i,j}$ is the j-bit long prefix of q_i and $\ell'_\lambda \stackrel{\mathrm{def}}{=} \ell_\lambda$.*

We denote the foregoing verifier and prover strategies by V and P, respectively.

We highlight the fact that Construction 3.4 uses a constant number of rounds and is of the public-coin type. Furthermore, Construction 3.4 satisfies the first two requirements of Definition 2.1; that is, the verifier's strategy is implementable in probabilistic polynomial time, and completeness holds with respect to a prover strategy that (when given $y = (M, x, t)$ and w as above) runs in time polynomial in $T_M(x, w)$. We thus focus on establishing the two last requirements of Definition 2.1. In fact, computational soundness follows from the weak proof-of-knowledge property, because if some adversary can convince the verifier to accept y with nonnegligible probability, then the extractor (given oracle access to that adversary) outputs a valid witness for membership of y in $S_{\mathcal{U}}$ (which implies that y is indeed in $S_{\mathcal{U}}$). Thus, it suffices to establish the latter.

3.4. Establishing the weak proof-of-knowledge property. This subsection contains the main technical contribution of the current section. The novel aspect in the analysis is the use of a "local definition of a conflict" (i.e., considering conflicting values for individual oracle bits rather than conflicting values for the entire oracle) and the use of reverse sampling for deriving (in polynomial time) hashing collisions when given a conflict on any bit position in the oracle.

LEMMA 3.5. *Construction 3.4 satisfies the weak proof-of-knowledge property of Definition 2.1, provided that the family $\{h_\alpha\}$ is indeed collision-resistant.*

Combining Lemma 3.5 with the foregoing discussion, we establish Theorem 1.1.

Proof. Fixing any polynomial p, we present a probabilistic polynomial-time knowledge extractor that extracts witnesses from any feasible prover strategy that makes V accept with probability above the threshold specified by p. Specifically, for any family of (deterministic) polynomial-size circuits representing a possible prover strategy and for all sufficiently long y's, if the prover convinces V to accept y with probability at least $1/p(|y|)$, then, with noticeable probability (i.e., $1/p'(|y|)$), the knowledge extractor (given oracle access to the strategy) outputs the bits of a corresponding witness. ∎

We fix an arbitrary family $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$ of (deterministic) polynomial-size circuits representing a possible prover strategy and a generic n and $y \in \{0,1\}^n$ such that $\Pr[(\widetilde{P}_n, V)(y) = 1] > \varepsilon \stackrel{\mathrm{def}}{=} 1/p(n)$. We consider a few key notions regarding the interaction of $\widetilde{P}_n$ and the designated verifier V on common input y. First, we consider

notions that refer to a specific interaction (corresponding to a fixed sequence of verifier coins, which consists of a pair of choices (α, r) that the verifier takes in steps (V1) and (V2), respectively):

- The ith *query* in such an interaction is $q_i = Q_{\mathrm{pcp}}(y, r, i)$, where r is the step (V2) message.
- The ith *answer* supplied by (the prover) $\widetilde{P}_n$ is the label (i.e., ℓ'_{q_i}) that the prover has provided (in step (P2)) for the leaf (associated with) the ith query (i.e., $q_i = Q_{\mathrm{pcp}}(y, r, i)$). The *corresponding authentication* is the corresponding sequence of pairs $\langle(\ell'_{\gamma_{i,j}0}, \ell'_{\gamma_{i,j}1}) : j = 0, \ldots, d-1\rangle$, where $\gamma_{i,j}$ is the j-bit long prefix of q_i.
  Note that the various parts of the prover's message in step (P2) are a function of α and r, and thus the notation ℓ'_γ is actually a shorthand for $\ell'_\gamma(\alpha, r)$.
- The ith answer supplied by $\widetilde{P}_n$ is said to be *proper* if the corresponding authentication passes the verifier's test (in step (V3)); that is, $\ell'_{\gamma_{i,j}} = h_\alpha(\ell'_{\gamma_{i,j}0}\ell'_{\gamma_{i,j}1})$ holds, for $j = 0, \ldots, d-1$ (where $\gamma_{i,j}$ is as described above and $\ell'_\lambda \stackrel{\mathrm{def}}{=} \ell_\lambda$).

Next, we consider the probability distribution induced by the verifier's coins. Note that these coins consist of the pair of choices (α, r) that the verifier takes in steps (V1) and (V2), respectively. Fixing any $\alpha \in \{0,1\}^n$, we consider the conditional probability denoted by $p_{y,\alpha}$ that the verifier accepts y when choosing α as step (V1). Clearly, for at least an $\varepsilon/2$ fraction of the possible α's, it holds that $p_{y,\alpha} \geq \varepsilon/2$. We fix any such α for the rest of the discussion. We now consider notions that refer to the residual probability space induced by a uniformly distributed $r \in \{0,1\}^{\mathrm{poly}(n)}$ (as selected by the verifier in step (V2)).

- For a query value $q \in \{0,1\}^d$, a query index $i \in [m]$, a possible answer $\sigma \in \{0,1\}$, and a parameter $\delta \in [0,1]$, we say that σ *is δ-strong for* (i, q) if, conditioned on the ith query being q, the probability that $\widetilde{P}_n$ *properly answers* the ith query with σ is at least δ. That is,

$$\Pr_r[\ell'_q = \sigma \text{ is proper} \mid q = Q_{\mathrm{pcp}}(y, r, i)] \;\geq\; \delta\,,$$

  where $\ell'_q = \ell'_{Q_{\mathrm{pcp}}(y,r,i)}(\alpha, r)$ as well as its being proper are determined based on α and r.
  When i and q are understood from the context, we just say that σ is a δ-strong answer.
- We say that a query $q \in \{0,1\}^d$ has *δ-conflicting answers* if there exist i and j (possibly $i = j$) such that 0 is δ-strong for (i, q) and 1 is δ-strong for (j, q).

We stress that throughout the rest of the analysis we consider a fixed $\alpha \in \{0,1\}^n$ and a uniformly distributed $r \in \{0,1\}^{\mathrm{poly}(n)}$.

Our goal is to show that if $p_{y,\alpha} \geq \varepsilon/2$, then we can extract a witness for y by using $\widetilde{P}_n$. We first show that using $\widetilde{P}_n$, we can reconstruct an adequate PCP oracle that convinces V_{pcp} (with probability at least $\mathrm{poly}(\varepsilon)$), although the strategy of $\widetilde{P}_n$ need not be consistent with any such oracle. That is, although a priori the strategy of $\widetilde{P}_n$ may answer queries in an inconsistent fashion, we shall show that in order to be convincing the answers of $\widetilde{P}_n$ must be "essentially" consistent.

As a preparation to the oracle reconstruction procedure, we first show (in Claim 3.5.1) that answers that are not adequately strong are quite rare (because

they are useless for convincing V). Indeed, the oracle reconstruction will be based on adequately strong answers (i.e., answers that appear frequently in interactions). Next, we show (via Claim 3.5.2) that reconstructing the oracle based on strong answers is essentially well-defined, because queries that have conflicting answers (which are both adequately strong) occur rarely in the interaction.

CLAIM 3.5.1. *The probability that the verifier accepts while receiving only δ-strong answers is at least $p_{y,\alpha} - m\delta$.*

(Recall that m is the number of queries asked by the PCP verifier V_{pcp}.) Thus, picking $\delta = p_{y,\alpha}/2m$, we may focus on the case that all of the prover's answers are δ-strong (since this case occurs with nonnegligible probability).

Proof. The key observation is that whenever the verifier accepts, it is the case that all answers are proper. Intuitively, answers that are not δ-strong (i.e., are rarely proper) are unlikely to appear in such interactions. Specifically, we just upper bound the probability that, for a uniformly distributed r, there exists $i \in [m]$ such that the answer $\ell'_{Q_{\mathrm{pcp}}(y,r,i)}$ is proper but not δ-strong for $(i, Q_{\mathrm{pcp}}(y,r,i))$. Fixing any i and any possible value of q_i, by definition (of being proper but not δ-strong), the probability that the answer ℓ'_{q_i} is proper but not δ-strong for (i,q_i) is smaller than δ. Averaging over the possible values of q_i (as emerging from $Q_{\mathrm{pcp}}(y,\cdot,i)$) and taking a union bound over all $i \in [m]$, the claim follows. □

CLAIM 3.5.2. *There exists a probabilistic polynomial-time oracle machine that, for any δ, given α and oracle access to $\widetilde{P}_n$, finds collisions with respect to h_α with a success probability that is polynomially related to δ/m and to the probability that the verifier makes a query that has δ-conflicting answers. That is, let η_α denote the probability that* (after choosing α in step (V1)) *the verifier makes a query that has δ-conflicting answers. Then, the probability of finding a collision* (i.e., $z' \neq z''$ such that $h_\alpha(z') = h_\alpha(z'')$) *is at least $\eta_\alpha \delta^2/m^3$.*

Thus, on a typical α (or rather on all but a negligible fraction of the α's) and for $\delta > 1/\mathrm{poly}(n)$, the quantity η_α must be negligible, because otherwise we derive a contradiction to the collision-resistant hypothesis of the family $\{h_\alpha\}$. Consequently, for $\delta = p_{y,\alpha}/2m > 1/\mathrm{poly}(n)$, we may focus on the case that the prover's answers are not $(\delta/2)$-conflicting.

Proof. We uniformly select $r \in \{0,1\}^{\mathrm{poly}(n)}$ and $i \in [m]$, hoping that $q_i = Q_{\mathrm{pcp}}(y,r,i)$ is δ-conflicting (which is the case with probability at least η_α/m). Uniformly selecting $i', i'' \in [m]$, and invoking the reverse-sampling algorithm S_{pcp} on inputs (y,i',q_i) and (y,i'',q_i), respectively, we obtain uniformly distributed r' and r'' that satisfy $q_i = Q_{\mathrm{pcp}}(y,r',i')$ and $q_i = Q_{\mathrm{pcp}}(y,r'',i'')$. We now invoke $\widetilde{P}_n$ twice, feeding it with α and r' (resp., α and r'') in the first (resp., second) invocation. Assuming that q_i is δ-conflicting, with probability at least $(\delta/m)^2$, both answers to q_i will be proper but with opposite values. Thus, with probability at least $(\eta_\alpha/m)\cdot(\delta/m)^2$, we have obtained two different *proper* answers to the same query q_i. In such a case, the authentication information corresponding to these two (proper) answers yields a collision under h_α because of the following considerations:

- Both answers to query q_i are authenticated with respect to the same pair (d, ℓ_λ) that was sent by $\widetilde{P}_n$ in step (P1). Note that these different values are provided as the label of the (same) leaf associated with the string $q_i \in \{0,1\}^d$.
- Each of the different values for the same leaf (associated with q_i) is authenticated with respect to the same value of the root (i.e., ℓ_λ). This means that a collision under h_α must occur somewhere along the path from the leaf to the root. The details follow.

Recall that when invoked with input α and r' (resp., α and r''), the circuit $\widetilde{P}_n$ provides authenticating information corresponding to the leaf q_i. This information takes the form of a sequence of pairs $\langle(\ell'_{\gamma_{i,j}0}, \ell'_{\gamma_{i,j}1}) : j = 0, \ldots, d-1\rangle$ (resp., $\langle(\ell''_{\gamma_{i,j}0}, \ell''_{\gamma_{i,j}1}) : j = 0, \ldots, d-1\rangle$), where $\gamma_{i,j}$ denotes the j-bit long prefix of q_i. Note that $\ell'_{q_i} \neq \ell''_{q_i}$ (i.e., the answers to q_i are different), while $\ell'_\lambda = \ell_\lambda = \ell''_\lambda$ (since both sequences refer to the same root value ℓ_λ). Thus, there exists a $j \in \{0, \ldots, d-1\}$ such that $\ell'_{\gamma_{i,j}} = \ell''_{\gamma_{i,j}}$ and $\ell'_{\gamma_{i,j+1}} \neq \ell''_{\gamma_{i,j+1}}$. It follows that $h_\alpha(\ell'_{\gamma_{i,j}0}\ell'_{\gamma_{i,j}1}) = \ell'_{\gamma_{i,j}} = \ell''_{\gamma_{i,j}} = h_\alpha(\ell''_{\gamma_{i,j}0}\ell''_{\gamma_{i,j}1})$ but $\ell'_{\gamma_{i,j}0}\ell'_{\gamma_{i,j}1} \neq \ell''_{\gamma_{i,j}0}\ell''_{\gamma_{i,j}1}$.

The claim follows. □

Suppose for a moment that (for $\delta = p_{y,\alpha}/2m$) *all* of the prover's answers are δ-strong, but none is $(\delta/2)$-conflicting. Then, we can use the prover's answers in order to construct (and not merely claim the existence of) an oracle for the PCP system that makes V_{pcp} accept with probability at least $p_{y,\alpha}/2$. Specifically, let the qth bit of the oracle be σ if and only if there exists an i such that σ is δ-strong for (i, q). This setting of the oracle bits can be determined in probabilistic polynomial time by using the reverse-sampling algorithm S_{pcp} to generate multiple samples of interactions in which these specific oracle bits are queried. Specifically, we want to determine the qth bit we generate, for every $i \in [m]$, multiple samples of interactions in which the ith oracle query equals q and determine the answer by using the gap provided by the hypothesis that for some i there is an answer that is δ-strong for (i, q), whereas (by the nonconflicting hypothesis) for every j the opposite answer is not $(\delta/2)$-strong for (j, q).

Recall that the foregoing outline relies on the simplifying assumption by which *all* of the prover's answers are δ-strong, but none is $(\delta/2)$-conflicting. In general, some queries may either have no strong answers or be conflicting (although these cases will occur rarely). In such a case, the procedure may fail to recover the corresponding entries in the PCP oracle, but this will not matter much (because with sufficiently high probability the PCP verifier will not query these badly recovered locations).

The oracle-recovery procedure. We present a probabilistic polynomial-time oracle machine that, on input (y, α) and $q \in \{0,1\}^d$ and oracle access to the prover $\widetilde{P}_n$, outputs a candidate for the qth bit of a PCP oracle. The procedure operates as follows, where $T \stackrel{\text{def}}{=} \text{poly}(n/\delta)$ and $\delta = \varepsilon/4m$:

1. For $i = 1, \ldots, m$ and $j = 1, \ldots, T$, invoke S_{pcp} on input (y, i, q) and obtain $r_{i,j}$.
2. For $i = 1, \ldots, m$ and $j = 1, \ldots, T$, invoke $\widetilde{P}_n$, feeding it with α and $r_{i,j}$, and if the ith answer is proper, then *record* (i, j) *as supporting this answer value.*
3. If for some $i \in [m]$, there are $(2\delta/3) \cdot T$ records for the form $(i, \cdot)$ supporting the value $\sigma \in \{0,1\}$, then define σ as a *candidate.* That is, σ is a *candidate* if there exists an i and at least $(2\delta/3) \cdot T$ different j's such that the ith answer of $\widetilde{P}_n(\alpha, r_{i,j})$ is proper and has value σ.
4. If a single value of $\sigma \in \{0,1\}$ is defined as a candidate, then set the qth bit accordingly. (Otherwise, do whatever you please.)

We call the query q *good* if it does not have $(\delta/2)$-conflicting answers and there exists an $i \in [m]$ and a bit value that is δ-strong for (i, q). As shown below, for a good query, with overwhelmingly high probability, the foregoing procedure will define the latter value as a unique candidate (less than $(\delta/2) \cdot T$, for every i'). Let us denote the PCP oracle induced by the above procedure by π.

CLAIM 3.5.3. *Let $\delta = \varepsilon/4m$, and recall that $p_{y,\alpha} \geq \varepsilon/2$. Suppose that the probability that V makes a query that has $(\delta/2)$-conflicting answers is at most $p_{y,\alpha}/4$. Then, with probability at least $1-2^{-n}$ taken over the reconstruction of π, the probability that ${V_{\mathrm{pcp}}}^{\pi}(y)$ accepts is lower bounded by $p_{y,\alpha}/4$.*

Proof. By definition, for each good query q, there exist $i \in [m]$ and $\sigma \in \{0,1\}$ such that σ is δ-strong for (i,q), whereas (for every i') the opposite bit $\bar{\sigma}$ is not $\delta/2$-strong for (i',q). It follows that, with probability at least $1-2^{-n^2}$, the bit σ is declared (in step 3) a candidate for the qth oracle bit, because each invocation of $\widetilde{P}_n(\alpha, S_{\mathrm{pcp}}(y,i,q))$ supports σ with probability at least δ. Similarly, with probability at least $1-m\cdot 2^{-n^2}$, the bit $\bar{\sigma}$ is not declared a candidate (for the qth oracle bit), because for every i' each invocation of $\widetilde{P}_n(\alpha, S_{\mathrm{pcp}}(y,i',q))$ supports $\bar{\sigma}$ with probability at most $\delta/2$. Thus, with probability at least $1-|\pi|\cdot(m+1)\cdot 2^{-n^2} > 1-2^{-n}$, the oracle π reconstructed by the procedure is such that for every good q the qth bit of π equals the (single) answer that is δ-strong. Combining the hypothesis (regarding $(\delta/2)$-conflicting answers) with Claim 3.5.1, we conclude that with probability at least $(p_{y,\alpha}-m\delta)-(p_{y,\alpha}/4) \geq p_{y,\alpha}/4$, taken solely over the choice of the random tape r, the verifier (of the interactive argument) accepts while making only good queries and receiving only δ-strong answers. Fixing any such random tape r and recalling that the (interactive argument) verifier V accepts only if V_{pcp} accepts the answers that V obtained, we conclude that in the latter case (i.e., when r is such that V accepts while making only good queries and receiving only δ-strong answers) it holds that ${V_{\mathrm{pcp}}}^{\pi}$ accepts too (when using the same r). The claim follows. □

Note that so far we have analyzed the behavior of the oracle-recovery procedure with respect to individual α's. Specifically, Claim 3.5.3 asserts that in some cases (i.e., those satisfying the claim's hypothesis) the procedure succeeds (i.e., yields an adequate oracle), whereas Claim 3.5.2 asserts that in the remaining relevant cases one can form collisions under the corresponding hashing function. Using the hypothesis that the family of hashing functions is collision-resistant, we infer that typically (i.e., for a random α) the oracle-recovery procedure works well. The details follow.

We call α *useful* if it satisfies the hypothesis of Claim 3.5.3 (i.e., if $p_{y,\alpha} \geq \varepsilon/2$ and the probability that V makes a query that has $(\varepsilon/8m)$-conflicting answers is at most $p_{y,\alpha}/4$). Indeed, Claim 3.5.3 asserts that, *for any useful α, with probability at least $1-2^{-n}$, the oracle-recovery procedure yields an oracle π such that ${V_{\mathrm{pcp}}}^{\pi}(y)$ accepts with probability at least $p_{y,\alpha}/4 \geq \varepsilon/8$.* By Claim 3.5.2, if the main hypothesis of Claim 3.5.3 does not hold (for α), then a collision (w.r.t. h_α) is formed with probability at least $(p_{y,\alpha}/4)\cdot(\varepsilon/8m)^2/m^3$. Thus, with overwhelmingly high probability over the choice of α, either $p_{y,\alpha}$ is negligible or the main hypothesis of Claim 3.5.3 holds for α. Recalling that $p_{y,\alpha} \geq \varepsilon/2$ with probability at least $\varepsilon/2$ (over the choices of α), it follows that in such a case the hypothesis of Claim 3.5.3 may fail only with negligible probability (for such a random α). We conclude that *a random α is useful with probability at least $(\varepsilon/2)-\mu(n) > \varepsilon/4$.*

So far we showed that if the interactive argument accepts with nonnegligible probability, then with similar probability we can reconstruct an oracle that convinces the PCP verifier with nonnegligible probability. The weak proof-of-knowledge property (of the interactive argument) now follows from the corresponding property of the PCP system. Specifically, combining the PCP extractor with the foregoing oracle-recovery procedure, we obtain the desired extractor (detailed next).

Extractor for the argument system. On input (y,i), where $y=(M,x,t)$ and $i \in [t]$, and access to a prover strategy $\widetilde{P}_n$, the extractor operates as follows (using $\delta = \varepsilon/4m$):

1. Uniformly select $\alpha \in \{0,1\}^n$, hoping that α is useful (which holds with probability at least $\varepsilon/4$). Fix α for the rest of the discussion.
2. Uniformly select coins ω for the oracle-recovery procedure, and fix ω for the rest of the discussion. Note that the oracle-recovery procedure (implicitly) provides oracle access to a PCP oracle π, which is determined by ($\widetilde{P}_n$, y, α, and ω).

   We say that ω is α-*useful* if the foregoing oracle-recovery procedure defines an oracle π such that ${V_{\rm pcp}}^{\pi}(y)$ accepts with probability at least $\varepsilon/8$. Recall that, by Claim 3.5.3, if α is useful, then, with probability at least $1-2^{-n}$, a random ω is α-useful.
3. Invoke $E_{\rm pcp}(y,i)$, providing it with oracle access to π. This means that each time $E_{\rm pcp}(y,i)$ makes a query q, we invoke the oracle-recovery procedure on input (y,q) (and with α and ω as fixed above) and obtain the qth bit of π, which we return as an answer to $E_{\rm pcp}(y,i)$. When $E_{\rm pcp}(y,i)$ provides an answer (supposedly the ith bit of a suitable witness w for y), we just output this answer.

Recall that if α is useful, and ω is α-useful, then with probability at least $2/3$ (over the coins of $E_{\rm pcp}$), the output of $E_{\rm pcp}$ (and thus of our extractor) will be correct (i.e., will yield the desired bit of a fixed witness for y). This follows by the proof-of-knowledge property of the PCP system, which refers to convincing $V_{\rm pcp}$ with probability at least 2^{-n}, while here $V_{\rm pcp}$ is convinced with probability at least $\varepsilon/8 > 2^{-n}$. Using suitable amplification, we can obtain (for each bit in the witness) the correct answer with probability at least $1-2^{-2n}$ (over the coins of $E_{\rm pcp}$), depending again on α and ω being useful. Denoting the coins of the amplified $E_{\rm pcp}$ by ρ, we infer that, for at least a fraction $1-t\cdot 2^{-2n} \geq 1-2^{-n}$ of the possible ρ's, the amplified $E_{\rm pcp}$ provides correct answers *for each of the possible t-bit locations*. We call such ρ's (α,ω)-*useful*.

Let us denote the foregoing extractor by E. The running time of E is dominated by the running time of the oracle-recovery procedure, whereas the latter is dominated by the $\mathrm{poly}(n/\varepsilon)$ invocations of $\widetilde{P}_n$ (during the oracle-recovery procedure). Using $\varepsilon = 1/p(n)$, it follows that E runs in polynomial time (specifically, the running time is polynomial in n and $p(n)$). The random choices of E correspond to the above three steps; that is, they consist of α, ω, and ρ. Whenever they are all useful (i.e., α is useful, ω is α-useful, and ρ is (α,ω)-useful), the extractor E recovers correctly each of the bits of a suitable witness (for y). The event in the condition (i.e., α, ω, and ρ being adequately useful) occurs with probability at least $(\varepsilon/4)\cdot(1-2^{-n})\cdot(1-2^{-n}) > \varepsilon/5 = 1/5p(n)$. Letting $p'(n) = 5p(n)$, the lemma follows. □

4. Application to zero-knowledge arguments. Using Theorem 1.1, we prove Theorem 1.2 in two steps:

1. Using any constant-round public-coin universal argument, we derive one that is witness indistinguishable. Here we follow the paradigm of "encrypted interactions" (introduced in [10] and also used in Barak's paper [6]). The construction and its analysis, which appear in section 4.1, differ from prior versions that appeared (or are outlined) in [8, 7, 17].
2. Using the result of the first step, we modify Barak's main construction [6] of a zero-knowledge argument (for any $S \in \mathcal{NP}$) such that it can be analyzed based on standard collision-resistant hashing. Specifically, rather than using any collision-resistant hashing (for the very first message in his protocol), we use tree hashing (as in step (P1) of Construction 3.4) composed with an error-correcting code. The construction and its analysis appear in section 4.2.

The reasons for the various modifications will be discussed in the corresponding subsections. But before turning to the constructions, we stress that the notion of witness indistinguishability (which is typically applied to proofs/argument systems for specific sets in $\mathcal{NP}$) needs to be redefined when applied to universal arguments. Specifically, this property should apply to any "fragment" of $R_{\mathcal{U}}$ that can be recognized in time that is polynomial in the length of the common input (where the polynomial is fixed after the universal argument system is specified). That is, we refer to the following definition.

DEFINITION 4.1 (witness indistinguishable universal argument). *A universal argument system (P,V) is called witness indistinguishable (WI) if, for every polynomial p, every polynomial-size circuit family $\{V_n^*\}_{n\in\mathsf{N}}$, and every three sequences $\langle y_n=(M_n,x_n,t_n):n\in\mathsf{N}\rangle$, $\langle w_n^1:n\in\mathsf{N}\rangle$, and $\langle w_n^2:n\in\mathsf{N}\rangle$ such that $|y_n|=n$, $t_n\leq p(|x_n|)$, and $(y_n,w_n^1),(y_n,w_n^2)\in R_{\mathcal{U}}$, the probability ensembles $\{\langle P(w_n^1),V^*\rangle(y_n)\}_{n\in\mathsf{N}}$ and $\{\langle P(w_n^2),V^*\rangle(y_n)\}_{n\in\mathsf{N}}$ are computationally indistinguishable, where $\langle P(w),V^*\rangle(y)$ denotes the output of V^* when interacting with $P(w)$ on common input y.*

Note the analogy to the discussion at the end of section 1.2 (regarding "natural applications of universal arguments").

4.1. Constructing witness indistinguishable universal arguments. Our starting point is any constant-round, public-coin universal argument (for $S_{\mathcal{U}}$) denoted by $(P_{\mathrm{ua}},V_{\mathrm{ua}})$. For the sake of simplicity, we assume (without loss of generality) that, on any n-bit long common input, each message sent by either parties has length $m=\mathrm{poly}(n)$. Using the public-coin clause this means that the protocol proceeds in rounds, where in each round the verifier selects uniformly an m-bit string, and the prover responds with an m-bit string determined based on its inputs and the messages it has received so far. We denote by c_{ua} the (constant) number of such rounds; in the case of Construction 3.4, $c_{\mathrm{ua}}=2$.

A second ingredient used in the construction is a (constant-round, public-coin) *nonoblivious statistically binding commitment scheme*. Loosely speaking, such a scheme allows a sender to "commit" to a value such that the value remains hidden from the receiver and still the sender is "committed" to this value. Furthermore, "statistically binding" means that, with high probability, if the commitment phase is concluded successfully, then there exists at most one value v that can be later revealed as a proper decommitment, whereas "nonoblivious" means that the sender actually knows this value v. For further discussion see Appendix B, where we also show that such a scheme can be constructed based on the existence of one-way functions. We denote this commitment scheme by $\mathtt{C}$ and the corresponding decommitment verification by $\mathtt{D}$. Furthermore, we denote by $(d,c)\leftarrow\mathtt{C}(v)$ an execution of $\mathtt{C}$ in which the sender enters the value v, the receiver obtains the commitment value c, and the sender obtains corresponding decommitment information d satisfying $\mathtt{D}((v,d),c)=1$. Recall that statistically binding means that, with overwhelmingly high probability, the protocol yields a commitment c such that if for any v,d,v', and d' it holds that $\mathtt{D}((v,d),c)=1$ and $\mathtt{D}((v',d'),c)=1$, then it must be that $v=v'$. The nonoblivious condition means that the sender knows the (at most) one value v for which there exists d such that $\mathtt{D}((v,d),c)=1$ holds; that is, there exists a knowledge extractor (as in the definition of the proofs of knowledge (see, e.g., [16, section 4.7.1])) that can extract from the sender this value v.

A third (and last) ingredient used in the construction is a (constant-round, public-coin) zero-knowledge proof *of constant soundness error* for some NP-complete set. Such a proof system can be constructed based on the existence of one-way func-

tions (see, e.g., [16, Chapter 4, Exercise 20]).[15] Let us denote such a system by $(P_{\rm zk}, V_{\rm zk})$.

The construction presented next is based on an "encrypted" emulation of the execution of the universal argument system $(P_{\rm ua}, V_{\rm ua})$. That is, the prover in our protocol responds to the verifier's messages by sending commitments to the messages that $P_{\rm ua}$ would have sent. This encrypted form of the prover's messages does not impair the new verifier that merely emulates $V_{\rm ua}$, because $V_{\rm ua}$ is of the public-coin type. At the end of the emulation phase, the prover provides a zero-knowledge proof that the committed values correspond to a transcript that $V_{\rm ua}$ would have accepted. The zero-knowledge property of the foregoing protocol is quite intuitive, and so we focus on its weak proof-of-knowledge property. Indeed, the fact that the commitment scheme is nonoblivious is used for extracting the corresponding transcript, but the acceptability of this transcript is guaranteed only when the new prover convinces the new verifier with constant probability (which originates in the constant soundness error of the zero-knowledge proof). To handle any nonnegligible acceptance probability, we repeat the foregoing protocol for a superlogarithmic number of times, where these repetitions are performed in parallel (so to maintain a constant number of rounds). These repetitions do not necessarily preserve the zero-knowledge property of the basic protocol, but they preserve its witness indistinguishability property. The resulting protocol is described next.

CONSTRUCTION 4.2 (a witness indistinguishable universal argument).

Common input: $y = (M, x, t)$, *supposedly in* $S_{\mathcal{U}}$. *Let* $n \stackrel{\rm def}{=} |y|$.

Auxiliary input to the prover: w *such that supposedly* $(y, w) \in R_{\mathcal{U}}$ *holds.*

Part 1. Encrypted emulations of $(P_{\rm ua}, V_{\rm ua})$. *The parties perform* n *parallel emulations of the* $(P_{\rm ua}, V_{\rm ua})$ *protocol, where each emulation is performed in a partially encrypted manner. Specifically, the verifier generates random messages exactly as* $V_{\rm ua}$*, but the prover answers with commitments to the corresponding responses of* $P_{\rm ua}$*. That is, for* $i = 1, \ldots, c_{\rm ua}$*, the parties emulate in parallel* n *copies of the* i*th round of* $(P_{\rm ua}, V_{\rm ua})$ *as follows:*

1. *The verifier selects uniformly at random* $r_i^1, \ldots, r_i^n \in \{0,1\}^m$ *and sends these strings to the prover.*
2. *The prover determines the answers of* $P_{\rm ua}$ *and responds with commitments to them. That is, for* $j = 1, \ldots, n$*, the prover first determines* $a_i^j \leftarrow P_{\rm ua}(y, w; r_1^j, \ldots, r_i^j)$*. Next, the parties invoke* n *parallel executions of* $\mathtt{C}$*, where the prover plays the sender and the verifier plays the receiver, such that the* j*th copy yields the output pair* $(s_i^j, e_i^j) \leftarrow \mathtt{C}(a_i^j)$*. If the verifier detects improper termination in any of these executions, then it halts and rejects.*

Part 2. Proving that $V_{\rm ua}$ accepts in the encrypted emulations. *The parties invoke the proof system* $(P_{\rm zk}, V_{\rm zk})$*, where the prover's goal is proving that the transcripts* $(r_1^j, e_1^j, \ldots, r_{c_{\rm ua}}^j, e_{c_{\rm ua}}^j)$ *generated in Part* 1 *correspond to encryptions* (or rather commitments) *of accepting* $(P_{\rm ua}, V_{\rm ua})$ *transcripts. That is, the parties run* n *parallel copies of* $(P_{\rm zk}, V_{\rm zk})$ *such that, in the* j*th copy, the NP statement being proved refers to the input* $(y, r_1, e_1, \ldots, r_{c_{\rm ua}}, e_{c_{\rm ua}})$ *and asserts that there exists* $((a_1^j, s_1^j), \ldots, (a_{c_{\rm ua}}^j, s_{c_{\rm ua}}^j))$ *such that*

[15] This construct replaces the *strong WI* proof system *of negligible soundness error* assumed in [8]. Unfortunately, contrary to prior misconceptions (cf. [16, section 4.6]), such protocols are not known to exist [17, Appendix C.3]. In fact, in our construction, we may use a *strong WI* proof system *of constant soundness error* (for some NP-complete set), but we do not know of such a protocol that is not zero-knowledge as well.

1. *for* $i = 1, \ldots, c_{\mathrm{ua}}$, *it holds that* $\mathtt{D}((a_i^j, s_i^j), e_i^j) = 1$;
2. $V_{\mathrm{ua}}(y; r_1^j, a_1^j, \ldots, r_{c_{\mathrm{ua}}}^j, a_{c_{\mathrm{ua}}}^j) = 1$.

Needless to say, the prover executes this copy of P_{zk} *using the NP witness* $((a_1^j, s_1^j), \ldots, (a_{c_{\mathrm{ua}}}^j, s_{c_{\mathrm{ua}}}^j))$, *where this sequence of pairs is as determined by it in Part* 1.

Note that the length of the NP statement being proven (*as well as the length of the corresponding NP witness*) *is bounded by a fixed polynomial in* $n + m$ (*and thus by a fixed polynomial in* n).

We denote the above verifier and prover strategies by V *and* P, *respectively.*

Clearly, Construction 4.2 is constant-round and public-coin and satisfies the first two requirements of Definition 2.1; that is, the verifier's strategy is implementable in probabilistic polynomial-time, and completeness holds with respect to a prover strategy that (given $y = (M, x, t)$ and w as above) runs in time polynomial in $T_M(x, w)$. To establish that Construction 4.2 is a WI universal argument, it remains to prove the following two properties of Construction 4.2:

1. the weak proof-of-knowledge property, which in turn implies also the computational soundness property;
2. the witness indistinguishability property (as per Definition 4.1).

We mention that (unlike in [8, Lemma 4.2]) the following proofs do not take advantage of the fact that the basic ingredients are constant-round protocols. We start with the witness indistinguishability property because its proof is significantly simpler.

LEMMA 4.3. *Construction* 4.2 *is WI* (as per Definition 4.1), *provided that* $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$ *is zero-knowledge and that* $\mathtt{C}$ *is computationally hiding.*

Proof. We view Construction 4.2 as the result of n parallel executions of a basic protocol denoted by (P', V') such that (P', V') consists of performing a single encrypted execution of $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$ followed by a (single) execution of $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$, which refers to this encrypted transcript (and is aimed at establishing that this encrypted transcript encodes an accepting transcript of $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$). Intuitively, this basic protocol is zero-knowledge, because it can be simulated by generating dummy commitments and invoking the simulator of $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$ on these commitments. The computational indistinguishability of this simulation from the real execution is argued as follows:

- As a mental experiment, we consider a hybrid distribution in which the simulator of $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$ is invoked on an encrypted transcript of $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$.
- The computationally hiding property of the commitment scheme implies that the output of our simulator (which invokes the simulator of $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$ on dummy commitments) is computationally indistinguishable from the foregoing hybrid distribution (which invokes the same simulator on commitments that correspond to a transcript of $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$).
- By our hypothesis regarding the simulator of $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$, the hybrid distribution is computationally indistinguishable from the real execution of the basic protocol (in which $(P_{\mathrm{zk}}, V_{\mathrm{zk}})$ is invoked on the encrypted transcript of $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$).

Having established the zero-knowledge property of the basic protocol, we conclude that the basic protocol (i.e., (P', V')) is WI (in the sense of Definition 4.1). Furthermore, when confining our attention to common and auxiliary inputs that are admissible as per Definition 4.1, it is the case that P' runs in polynomial time (when given adequate auxiliary inputs). Thus, the witness indistinguishability of P' (on such inputs) is preserved under parallel composition (cf., e.g., [16, section 4.6.2]). The lemma follows. □

LEMMA 4.4. *Construction* 4.2 *satisfies the weak proof-of-knowledge property of Definition* 2.1, *provided that so does* $(P_{\rm ua}, V_{\rm ua})$ *and that* $\mathtt{C}$ *is statistically binding and nonoblivious.*

Proof. Here we decompose Construction 4.2 in a different way, considering Part 1 as a whole and Part 2 as a whole. We start with an overview of the proof. Loosely speaking, the nonoblivious property of the commitment scheme guarantees that we can extract the cleartext version of all encrypted transcripts (of $(P_{\rm ua}, V_{\rm ua})$), but at this point it is unclear whether any of these transcripts is accepting. Using the statistically binding property of the commitment scheme, we distinguish the case that some of these transcripts are accepting from the case that none of them is accepting. Using the (constant error) soundness of $(P_{\rm zk}, V_{\rm zk})$, we infer that the second case (which refers to multiple failures) happens with negligible probability, and so we may ignore it. Thus, either V detects an improper execution of the commitment scheme or we are able to extract an accepting transcript of $(P_{\rm ua}, V_{\rm ua})$. It follows that if V is convinced with some noticeable probability p, then for some $j \in [n]$, with probability at least p/n (or so), the jth transcript is accepting and extractable. At this point the lemma follows by invoking the weak proof-of-knowledge property of $(P_{\rm ua}, V_{\rm ua})$. We now turn to the actual proof.

Fixing an arbitrary prover strategy for Construction 4.2, we shall derive a related strategy for the underlying universal-argument system such that the circuit complexity (resp., the success probability) of the resulting strategy will be related to the circuit complexity (resp., the success probability) of the original strategy. Specifically, let us consider an arbitrary family $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$ of (deterministic) polynomial-size circuits representing a possible prover strategy in the system (P, V). Fixing a generic n and $y \in \{0,1\}^n$, we let $p_y \stackrel{\rm def}{=} \Pr[(\widetilde{P}_n, V)(y) = 1]$. Using $\widetilde{P}_n$, we shall construct a corresponding prover strategy $\widetilde{P_{\rm ua}}$ that makes $V_{\rm ua}$ accept y with probability $\Omega((p_y - \mu(n))/n)$, where μ is some negligible function. Thus, once we are done constructing $\widetilde{P_{\rm ua}}$, the weak proof-of-knowledge property of Construction 4.2 will follow from the weak proof-of-knowledge property of the underlying universal argument system.

We thus turn to constructing $\widetilde{P_{\rm ua}}$, which starts by uniformly selecting $j \in [n]$ and tries to extract the cleartext messages that are encrypted (or rather committed) in the transcript of the jth interaction performed by $\widetilde{P}_n$ in Part 1. Specifically, after selecting j, the strategy $\widetilde{P_{\rm ua}}$ operates in $c_{\rm ua}$ iterations, where in each iteration it obtains from $\widetilde{P}_n$ a sequence of encrypted messages and extracts from it a (cleartext) message that it sends to $V_{\rm ua}$. That is, for $i = 1, \ldots, c_{\rm ua}$, the following occurs:

1. The strategy $\widetilde{P_{\rm ua}}$ obtains from the (real) verifier $V_{\rm ua}$ a (uniformly distributed) string denoted by $r_i \in \{0,1\}^m$.
2. The strategy $\widetilde{P_{\rm ua}}$ selects uniformly $r_i^1, \ldots, r_i^{j-1}, r_i^{j+1}, \ldots, r_i^n \in \{0,1\}^m$ and feeds $r_i^1, \ldots, r_i^{j-1}, r_i, r_i^{j+1}, \ldots, r_i^n$ to $\widetilde{P}_n$, obtaining a residual sender (i.e., a committer) for the current round of executions of the nonoblivious commitment scheme. Using this residual sender and the knowledge extractor guaranteed for the nonoblivious commitment scheme, the strategy $\widetilde{P_{\rm ua}}$ extracts the (unique) cleartext message that may correspond to the jth copy of the current executions of the nonoblivious commitment scheme. (Note that the nonoblivious/proof-of-knowledge property is preserved when the protocol is executed multiple times in parallel.)
3. The strategy $\widetilde{P_{\rm ua}}$ sends the aforementioned answer to the (real) verifier $V_{\rm ua}$.

Note that we do not claim, at this point, that the extracted values correspond to proper decommitments and furthermore that they yield an accepting transcript. At this point we wish only to establish that $\widetilde{P_{\mathrm{ua}}}$ extracts the only values that may be properly decommitted.

We now turn to the analysis of the size of $\widetilde{P_{\mathrm{ua}}}$ and its success probability. These quantities are determined by the size of $\widetilde{P}_n$ and its success probability denoted by p_y. We may indeed focus on the case that p_y is a noticeable function of $n = |y|$ (i.e., $p_y > 1/\mathrm{poly}(n)$). Recall that $\widetilde{P}_n$ executes $c_{\mathrm{ua}} + 1$ subprotocols, where the first c_{ua} protocols are n parallel executions of the nonoblivious commitment scheme and the last protocol consists of n parallel executions of the (constant-error) zero-knowledge proof system. (Indeed, we neglect the fact that each of the first c_{ua} protocols actually consists of a random $n \cdot m$-bit long message sent from the verifier to the prover, followed by n parallel executions of the nonoblivious commitment scheme.)[16] We first prove that, with probability at least $p_y/2$, the entire execution produces a transcript such that for every $i \in \{0, 1, \ldots, c_{\mathrm{ua}}\}$, given the partial transcript of the i previous executions, the next execution is successful with probability at least $p_y/2c_{\mathrm{ua}}$. This follows from the following general claim.

CLAIM 4.4.1. *For a finite set Ω and an integer m, consider an arbitrary $G \subseteq \Omega^m$, and let $\rho = |G|/|\Omega|^m$. For $i \in [m]$, a sequence $(e_1, \ldots, e_m) \in \Omega^m$ is called i-good if $\Pr_{e \in \Omega}[(e_1, \ldots, e_{i-1}, e) \in G_i] \geq \frac{\rho}{2(m-1)}$, where $(e'_1, \ldots, e'_i) \in G_i$ if and only if there exist $e'_{i+1}, \ldots, e'_m \in \Omega$ such that $(e'_1, \ldots, e'_m) \in G$. Then, at least half of the sequences in G are i-good for every $i \in [m]$.*

Needless to say, the foregoing sequences correspond to the executions of the different $m = c_{\mathrm{ua}} + 1$ protocols, and i-good sequences correspond to transcripts in which, given the partial transcript of the $i-1$ previous (successful) executions, the next execution is successful with probability at least $\frac{p_y}{2(m-1)}$. Actually, we give up on execution transcripts that are not fully successful, and yet for every $i \in [c+1]$, given the partial transcript of the $i-1$ previous executions, the next execution is successful with probability at least $\frac{p_y}{2(m-1)}$.

Proof. Let $B_i \stackrel{\mathrm{def}}{=} \{(e_1, \ldots, e_{i-1}) \in \Omega^{i-1} : \Pr_{e \in \Omega}[(e_1, \ldots, e_{i-1}, e) \in G_i] < \rho/2(m-1)\}$ be the set of all $(i-1)$-long prefixes of sequences that are not i-good. Note that $B_1 = \emptyset$, because $\Pr_{e \in \Omega}[e \in G_1] \geq \Pr_{\overline{e} \in \Omega^m}[\overline{e} \in G] = \rho$. Now, on one hand, every sequence in G that has no prefix in $\cup_{i=1}^m B_i$ is i-good for every i. On the other hand, the number of sequences in G that have a prefix in B_i is less than $\frac{\rho}{2(m-1)} \cdot |\Omega|^m$, because each such sequence has an i-long prefix $(e_1, \ldots, e_{i-1}, e_i) \in G_i$, whereas $\Pr_{e \in \Omega}[(e_1, \ldots, e_{i-1}, e) \in G_i] < \rho/2(m-1)$. The claim follows. ∎

Turning back to Construction 4.2, we note that (by Claim 4.4.1) a $p_y/2$ fraction of all executions is fully successful and furthermore each partial transcript of these executions leads to success in the next execution with probability at least $p_y/2c_{\mathrm{ua}}$. Thus, (1) in these executions each of the c_{ua} rounds of nonoblivious commitments is successful with probability at least $p_y/2c_{\mathrm{ua}}$, and (2) the same holds for the interactive proofs that take place in Part 2.

The first foregoing fact implies that (in these executions), for every choice of j and for $i = 1, \ldots, c_{\mathrm{ua}}$, the strategy $\widetilde{P_{\mathrm{ua}}}$ can extract the value committed in the jth copy of the ith iteration of Part 1 (or rather the only value that can be decommitted as such) by invoking $\widetilde{P}_n$ for $\frac{\mathrm{poly}(n)}{p_y/2c_{\mathrm{ua}}}$ times (and failing with negligible probability). Thus, we

[16] This ignored (random) message can be viewed as belonging to the previous subprotocol.

can bound the size of $\widetilde{P_{\rm ua}}$ by $\mathrm{poly}(n)/p_y$ times the size of $\widetilde{P}_n$ and conclude that (in these executions) $\widetilde{P_{\rm ua}}$ answers with messages that equal the values committed in the corresponding executions of $\widetilde{P}_n$ (or rather the only values that can be decommitted as such).

Combining the second foregoing fact, which refers to Part 2, with the constant soundness error of each execution of the proof system, it follows that at least one of the n parallel executions that took place in Part 1 corresponds (i.e., can be properly decommitted) to a $(P_{\rm ua}, V_{\rm ua})$-transcript that is accepting, because otherwise the probability that all of these executions of the interactive proof are successful is at most $s^n \ll p_y/2c_{\rm ua}$, where $s < 1$ denotes the constant soundness error of the proof system. We stress that the proper decommitment requirement guarantees that this transcript corresponds to the messages sent by $\widetilde{P_{\rm ua}}$.

Hence, with probability at least $(p_y/2) - \mu(n) > p_y/3$, for every choice of j, the strategy $\widetilde{P_{\rm ua}}$ sends messages that correspond to the only possible proper decommitments of the corresponding commitment of $\widetilde{P}_n$, and at least one of these n possible transcripts (corresponding to the choice of $j \in [n]$) is accepting. Conditioned on the foregoing, with probability at least $1/n$, the strategy $\widetilde{P_{\rm ua}}$ selects j such that the recovered $(P_{\rm ua}, V_{\rm ua})$-transcript is accepting. We conclude that $\widetilde{P_{\rm ua}}$ convinces $V_{\rm ua}$ with probability at least $p_y/3n$, and the lemma follows. □

Remarks. We stress again that Construction 4.2 is different from the corresponding construction presented in [8]. It is also different from the patch described in [7, Appendix A.4] and the one outlined in [17, Appendix C.3.3]. An important difference is that the current proof of Lemma 4.4 does not rely on the fact that the protocols employed have a constant number of rounds. We also mention that the main analysis of Construction 4.2, which is provided in the proofs of Lemmas 4.3 and 4.4, proceeds by decomposing the construction in two different ways. A similar strategy is employed in Appendix B (see the proof of Claims B.2.1 and B.2.2).

4.2. Modifying Barak's zero-knowledge argument. Here our starting point is any (constant-round, public-coin) strong WI universal argument (for $S_{\mathcal{U}}$) denoted by $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$.

A second ingredient used in the construction is a tree-hashing scheme denoted by $\mathtt{TH}$, as used in Construction 3.4. Loosely speaking, such a scheme can be applied to arbitrary long strings and allows for the verification of the value of a particular bit in the string within time polynomial in the hash value (and possibly polylogarithmic in the length of the string). We stress that the verification does not require presenting the entire string to which the hashing was applied (but rather only auxiliary authentication information that is specific to that bit position). Recall that tree hashing is constructed based on some "basic" hashing function (which maps $2n$-bit strings to n-bit strings), and that conflicting values assigned to any bit position in the tree hashing yield a collision in the basic hashing. Specifically, when using a basic hashing function indexed by α and applying the tree-hashing procedure to an m-bit long string z that is placed at the leaves of the ($\log_2 m$-deep) tree, we denote the resulting label of the root by $\mathtt{TH}_\alpha(z)$ and denote the corresponding sequence of m authenticators by $\mathtt{auth}_\alpha(z)$.[17]

[17] That is, $\mathtt{TH}_\alpha(z) = \ell_\lambda$, where ℓ_i is the ith bit of z and $\ell_\gamma = h_\alpha(\ell_{\gamma 0}\ell_{\gamma 1})$. Actually, here it is more natural to let $\mathtt{TH}_\alpha(z) = \ell_{0,0}$, where $\ell_{d,i}$ is the ith bit of $z \in \{0,1\}^{2^d}$ and $\ell_{j,i} = h_\alpha(\ell_{j+1,2i}\ell_{j+1,2i+1})$. Similarly, the ith sequence in $\mathtt{auth}_\alpha(z)$ is $(\ell_{d,2\lfloor i/2\rfloor}, \ell_{d,2\lfloor i/2\rfloor+1}, \ell_{d-1,2\lfloor i/4\rfloor}, \ell_{d-1,2\lfloor i/4\rfloor+1}, \ldots, \ell_{1,2\lfloor i/2^d\rfloor}, \ell_{1,2\lfloor i/2^d\rfloor+1})$.

In addition, we use a standard statistically binding commitment scheme denoted by $\mathtt{C}$ and a binary error-correcting code of constant relative distance and polynomial-time encoding algorithm denoted by $\mathtt{ECC}$. Recall that a commitment scheme as described above can be constructed based on any one-way function (see, e.g., [16, section 4.4.1]), whereas even stronger forms of error-correcting codes are known to exist unconditionally (e.g., consider a concatenation code that combines a Reed–Solomon code with the encoding of the small field elements by a Hadamard code). As in Construction B.2, we let $\mathtt{C}_s(z)$ denote the receiver's view of the commitment phase (of $\mathtt{C}$) when the sender inputs the value z and uses randomness s.

The key idea in our modification of Barak's construction [6] is replacing an arbitrary collision-resistant hashing of strings by the following two-step (hashing) process:

1. Apply the error-correcting code to the input string.
2. Apply the tree hashing to the resulting code word.

The advantage of this two-step (hashing) process over standard hashing is that if two different strings are hashed to the same value, then we can quickly obtain a collision in the basic hashing function (underlying the tree hashing). We stress that this collision is found in time that is polynomial in the hash value (which may be polylogarithmic in the length of the strings being hashed). The reason is that, with (positive) constant probability, a uniformly selected bit position in the code word will have different values in the two code words, and in this case we obtain from the corresponding authentications a collision in the basic hashing. (The foregoing motivational discussion will be clarified by the proof of Lemma 4.6.)

CONSTRUCTION 4.5 (a zero-knowledge argument for $S \in \mathcal{NP}$ (with a corresponding witness relation R_S)).

Common input: *x, supposedly in S. Let $n \stackrel{\text{def}}{=} |x|$.*

Auxiliary input to the prover: *w such that supposedly $(x,w) \in R_S$ holds.*

Part 1. Introducing a trapdoor for the simulation. *The prover commits to a dummy value that allows cheating in the case that this value provides a description of the supposedly random value that is sent by the verifier after getting the said commitment. This is done as follows:*

1. *The verifier uniformly selects $\alpha \in \{0,1\}^n$* (i.e., a basic hash function) *and sends it to the prover.*
2. *The prover sends a dummy commitment* (i.e., a commitment to the value 0^{2n})*; that is, it sends $c \stackrel{\text{def}}{=} \mathtt{C}_s(0^{2n})$ to the verifier, where s denotes the sender's randomness in this execution of the commitment scheme.*
3. *The verifier uniformly selects $r \in \{0,1\}^n$ and sends it to the prover.*

Cheating in Part 2 will become possible if and only if r agrees with c in the sense that one may present a circuit Π such that $\Pi(c) = r$ and $c = C_s(|\mathtt{ECC}(\Pi)|, \mathtt{TH}_\alpha(\mathtt{ECC}(\Pi)))$.

Part 2. Effectively proving that $x \in S$. *Specifically, the prover will prove that he knows either a witness w for $x \in S$* (i.e., $(x,w) \in R_S$) *or a circuit Π such that $\Pi(c) = r$ and $c = C_s(|\mathtt{ECC}(\Pi)|, \mathtt{TH}_\alpha(\mathtt{ECC}(\Pi)))$. This is done as follows: Loosely speaking, the parties invoke the proof system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$ on common input (x, α, c, r), where the prover intends to prove that it knows a tuple (w, m, η, γ, s) such that either $(x,w) \in R_S$ or (η, γ, s) encodes authentication and decommitment information for a circuit Π such that $\Pi(c) = r$, where the encoding clause means that it holds that $\eta = \mathtt{ECC}(\Pi) \in \{0,1\}^m$, $\gamma = \mathtt{auth}_\alpha(\eta)$, and $c = C_s(|\eta|, \mathtt{TH}_\alpha(\eta))$.*[18] *Actually, the parties reduce the above instance*

[18] The reason that we include (in the witness) the authentication information (i.e., $\mathtt{auth}(\mathtt{ECC}(\Pi))$) rather than including Π itself will become clear in the proof of Lemma 4.6. Furthermore, for simplicity and clarity, we explicitly include in the witness both $\mathtt{ECC}(\Pi)$ and its length.

(x, α, c, r) to the triplet $y = (M'_S, (x, \alpha, c, r), 2^n)$, where $y \in \{0,1\}^{\mathrm{poly}(n)}$ is supposedly in $S_{\mathcal{U}}$ and M'_S is such that $M'_S((x, \alpha, c, r), (w, m, \eta, \gamma, s)) \stackrel{\mathrm{def}}{=} 1$ if and only if at least one of the following two conditions holds:

1. *$(x, w) \in R_S$.*
2. *$m = |\eta|$, $c = \mathtt{C}_s(m, \mathtt{TH}_\alpha(\eta))$, $\gamma = \mathtt{auth}_\alpha(\eta)$, and $\Pi(c) = r$, where $\Pi \leftarrow \mathtt{ECC}^{-1}(\eta)$ is a description of a circuit* (i.e., *$\eta = \mathtt{ECC}(\Pi)$*).

When invoking $P_{\text{wi-ua}}$, the prover provides it with the witness $(w, m_0, \eta_0, \gamma_0, s_0)$, where $m_0 = n$ and $\eta_0 = \gamma_0 = s_0 \stackrel{\mathrm{def}}{=} 0^n$ are (short) *dummy values.*

Note that the first condition can be evaluated in (fixed) *polynomial time* (in *$|x|$*)*, whereas the complexity of evaluating the second condition is dominated by the running time of Π on input c. Furthermore, if $(x, w) \in R_S$, then $(y, (w, m_0, \eta_0, \gamma_0, s_0)) \in R_{\mathcal{U}}$ and the running time of $P_{\text{wi-ua}}$ on $(y, (w, m_0, \eta_0, \gamma_0, s_0))$ is a fixed polynomial in $|x|$. We stress that, in any case, the length of the statement being proven is bounded by a fixed polynomial in $|x|$.*

We denote the above verifier and prover strategies by V and P, respectively.

Clearly, Construction 4.5 is constant-round and public-coin and employs a probabilistic polynomial-time verifier strategy. Furthermore, the designated prover satisfies the completeness property while running in polynomial time, given x and w as described above. Demonstrating that Construction 4.5 is zero-knowledge is done by following the ideas of [6]. We start with a rough sketch of this proof and then turn to establish the computational soundness property of Construction 4.5.

Construction 4.5 *is zero-knowledge.* We present a non–black-box simulator that, given the code of any feasible cheating verifier (represented by a polynomial-size circuit family $\{\widetilde{V}_n\}_{n \in \mathbf{N}}$), simulates the interaction of P with that verifier. Specifically, given $\widetilde{V}_n$, the simulator emulates Part 1 of the protocol, except that it sets $c \leftarrow \mathtt{C}_s(|\mathtt{ECC}(\widetilde{V}_n)|, \mathtt{TH}_\alpha(\mathtt{ECC}(\widetilde{V}_n)))$ instead of $c \leftarrow \mathtt{C}_s(0^{2n})$. Next, the simulator emulates Part 2 of the protocol by using the witness $(w_0, |\mathtt{ECC}(\widetilde{V}_n)|, \mathtt{ECC}(\widetilde{V}_n), \mathtt{auth}_\alpha(\mathtt{ECC}(\widetilde{V}_n)), s)$, where $w_0 = 0^n$ is a (short) dummy value, s was selected by the simulator when emulating Part 1, and $\widetilde{V}_n$ was given to it as an (auxiliary) input.

Needless to say, given $\widetilde{V}_n$, the simulator computes $\eta \leftarrow \mathtt{ECC}(\widetilde{V}_n)$ as well as the tree-hash value $\mathtt{TH}_\alpha(\eta)$ and the corresponding sequence of authenticators $\mathtt{auth}_\alpha(\eta)$ in polynomial time. It follows that, for every polynomial bounding the size of the verifier's strategy (i.e., $\widetilde{V}_n$), the simulator runs in poly(n)-time. It is thus left to show that, for every polynomial bounding the size of the verifier's strategy (i.e., $\widetilde{V}_n$), the simulator's output (produced when given $\widetilde{V}_n$ as auxiliary input) is computationally indistinguishable from a real execution (as in Construction 4.5). The proof proceeds as follows:

- As a mental experiment, we consider a hybrid distribution in which Part 1 is performed as in the simulation (i.e., by committing to the tree hashing of the encoding of $\widetilde{V}_n$) but Part 2 is performed as in the real execution (i.e., by using a witness w to the common input x).
- Combining the computationally hiding property of the commitment scheme and the fact that P (when given w) runs in poly(n) time, it follows that the hybrid distribution is computationally indistinguishable from the real execution of the protocol.
- The witness indistinguishability property of $P_{\text{wi-ua}}$ implies that the simulator's output (in which the witness $\widetilde{V}_n$ is used) is computationally indistinguishable from the hybrid distribution (in which the witness w is used). We stress that, when using the witness indistinguishability property of $P_{\text{wi-ua}}$, we refer to witnesses that are verifiable in fixed polynomial (in n) time (be-

cause the polynomial bounding the size of $\widetilde{V}_n$ has been fixed for the current discussion).

Thus, the zero-knowledge feature follows.

Construction 4.5 *is computationally sound.* We show that any feasible cheating strategy for the prover yields a feasible algorithm that forms collisions with respect to the basic hashing family $\{h_\alpha : \{0,1\}^{2|\alpha|} \to \{0,1\}^{|\alpha|}\}_{\alpha \in \{0,1\}^*}$. The main idea is using the (weak) proof-of-knowledge property of $V_{\text{wi-ua}}$ in order to *implicitly* reconstruct error-correcting code words that encode (different) valid witnesses that correspond to different executions of Part 1. We stress that since there is no a priori polynomial bound on the length of such witnesses, we cannot afford to *explicitly* reconstruct them (as done in [6], where only a contradiction to superpolynomial hardness is derived). However, implicit reconstruction of valid code words will suffice, because different witnesses will be encoded by code words that differ on a constant fraction of the bit locations.

LEMMA 4.6. *Construction* 4.5 *is computationally sound* (w.r.t. S)*, provided that the family* $\{h_\alpha\}$ *is indeed collision resistant.*

Proof. Suppose towards the contradiction that there exists a feasible prover strategy that fools V with nonnegligible probability (to accept inputs not in S). Specifically, let $\{\widetilde{P}_n\}_{n \in \mathbb{N}}$ be such a family and p be a polynomial such that for infinitely many $x \notin S$ it holds that $p_x \stackrel{\text{def}}{=} \Pr[(\widetilde{P}_n, V)(x) = 1] > 1/p(|x|)$. Let us fix a generic n and $x \in \{0,1\}^n \setminus S$ such that $p_x > 1/p(n)$. For simplicity, we incorporate this x in $\widetilde{P}_n$. Using $\{\widetilde{P}_n\}_{n \in \mathbb{N}}$, we present a family of circuits $\{C_n\}_{n \in \mathbb{N}}$ that try to form collisions. On input $\alpha \in \{0,1\}^n$, the circuit C_n proceeds as follows:

1. Invoking $\widetilde{P}_n$, on input α, the circuit C_n obtains $c \leftarrow \widetilde{P}_n(\alpha)$. This is supposedly a commitment produced by the cheating prover (in the second step of Part 1 of the protocol).
2. Uniformly selecting $r \in \{0,1\}^n$ and feeding it to $\widetilde{P}_n$ yields a residual prover $\widetilde{P}_n(\alpha, r)$ for Part 2 of the protocol. That is, $\widetilde{P_{\text{wi-ua}}} \stackrel{\text{def}}{=} \widetilde{P}_n(\alpha, r)$ is a prover strategy for the (WI) universal-argument system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$.

   We shall focus on the case that $\widetilde{P_{\text{wi-ua}}}$ convinces $V_{\text{wi-ua}}$ to accept (x, α, c, r) with probability at least $p_x/4 > 1/4p(n)$. In this case, the (weak) proof-of-knowledge property guarantees that we can implicitly reconstruct a witness (w, m, η, γ, s) that satisfies the second condition in Part 2 (because $x \notin S$ makes it impossible to satisfy the first condition (i.e., the condition $(x, w) \in R_S$)). Recall that the second condition implies that η encodes a circuit Π such that $\Pi(c) = r$ and futhermore that $m = |\eta|$, $c = \mathtt{C}_s(m, \mathtt{TH}_\alpha(\eta))$, and $\gamma = \mathtt{auth}_\alpha(\eta)$. In the following two steps, we shall first use the foregoing (implicit) reconstruction procedure to obtain m and next use it to obtain the ith bit of η as well as the corresponding (authentication) segment of γ, for a uniformly selected $i \in [m]$.
3. Invoking the knowledge extractor guaranteed for the system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$, while providing it with oracle access to $\widetilde{P_{\text{wi-ua}}}$, we reconstruct the values at bit locations $n+1, \ldots, 2n$ in the witness (i.e., the integer m that indicates the length of a code word).

   Recall that the values at bit locations $2n+1, \ldots, 2n+m$ are an error-correcting encoding of Π, and the subsequent m strings consist of authentication information for the corresponding bits.
4. The circuit C_n uniformly selects $i \in [m]$. Invoking the knowledge extractor again with oracle access to $\widetilde{P_{\text{wi-ua}}}$, it reconstructs the value of the ith bit of

the code word as well as the values in the bit locations that correspond to the authenticator of the ith bit in the code word.

5. We repeat Steps 2 and 4 with a new uniformly selected $r' \in \{0,1\}^n$ but with the same value of i as selected in Step 4. That is, analogously to Step 2, we first obtain a corresponding prover strategy $\widetilde{P'_{\text{wi-ua}}} \stackrel{\text{def}}{=} \widetilde{P}_n(\alpha, r')$ (for the (WI) universal argument system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$). We need not repeat Step 3 because the statistically binding property of $\mathtt{C}$ guarantees the uniqueness of m (obtained in Step 3). Analogously to Step 4, we invoke the knowledge extractor with oracle access to $\widetilde{P'_{\text{wi-ua}}}$ and obtain the value of the ith bit of a code word (which encodes a circuit Π' such that $\Pi'(c) = r'$) as well as the values in the bit locations that correspond to the authenticator of this bit. Recall that, since $x \notin S$, the witness used by $\widetilde{P'_{\text{wi-ua}}}$ must encode a program Π' such that $\Pi'(c) = r'$. Since (with probability $1 - 2^{-n}$ it holds that) $r \neq r'$ (and $\Pi(c) = r$), it must be the case that $\Pi' \neq \Pi$ (whereas $|\mathtt{ECC}(\Pi)| = m = |\mathtt{ECC}(\Pi')|$). We hope that $\mathtt{ECC}(\Pi)$ and $\mathtt{ECC}(\Pi')$ differ on the ith bit (which happens with constant probability), in which case we obtain authenticators for conflicting values (with respect to the same label of the root of the tree (where the uniqueness of this label is due to the statistically binding property of $\mathtt{C}$)).
6. The circuit C_n examines the authenticators obtained in Steps 4 and 5. If they authenticate conflicting values, then the circuit derives a collision under h_α (as in the proof of Claim 3.5.2).

The foregoing circuit family has a polynomial size (because each C_n is implementable by the same probabilistic polynomial-time oracle machine, which in turn is given access to the polynomial-size $\widetilde{P}_n$). We now turn to analyze the success probability of C_n.

CLAIM 4.6.1. *Given a uniformly distributed $\alpha \in \{0,1\}^n$, with probability at least $1/\text{poly}(n)$, the circuit C_n outputs a collision with respect to h_α.*

Proof. Recall that $\widetilde{P}_n$ makes V accept x with probability $p_x > 1/p(n)$, where the probability is taken over V's random choices in the two parts of Construction 4.5. Moreover, V's choices in Part 1 are (α, r), where α (resp., r) is uniformly selected in $\{0,1\}^n$.

We call α good if the probability that $\widetilde{P}_n$ makes V accept x, conditioned on V selecting α in the first step of Part 1, is at least $p_x/2$. Clearly, at least a $p_x/2$ fraction of the α's is good, and we focus on any such good α.

Similarly, we say that r is *α-good* if the probability that $\widetilde{P}_n$ makes V accept x, conditioned on V selecting α and r in Part 1, is at least $p_x/4$. We denote by $G = G_\alpha$ the set of α-good strings and note that $|G| > (p_x/4) \cdot 2^n$ (since α is good). That is, for every $r \in G$, the residual prover $\widetilde{P}_n(\alpha, r)$ convinces $V_{\text{wi-ua}}$ with probability at least $p_x/4 > 1/4p(n)$, and therefore the knowledge extractor (given oracle access to $\widetilde{P}_n(\alpha, r)$) implicitly extracts the corresponding witness with probability at least $q_n \stackrel{\text{def}}{=} 1/\text{poly}(n)$, where the polynomial depends on the polynomial p.

Observe that, with probability at least $(p_x/4)^2$, both r and r' selected in Steps 2 and 5, respectively, reside in G. Conditioned on this event, we (implicitly) extract both of the corresponding witnesses with probability at least q_n^2. We stress that the two witnesses must be of the same length by virtue of their lengths being committed to in c (sent by the prover in the second step of Part 1). Finally, with constant probability, these witnesses differ in bit position i (selected in Step 4), in which case C_n succeeds in forming a collision under h_α.

We conclude that, for any good α, the circuit C_n succeeds in forming a collision under h_α with probability at least $(p_x/4)^2 \cdot q_n^2 \cdot \Omega(1) = 1/\mathrm{poly}(n)$. Recalling that at least $p_x/2 = 1/\mathrm{poly}(n)$ of the α's are good, the claim follows. □

Using Claim 4.6.1, we derive a contradiction to the hypothesis that $\{h_\alpha\}$ is collision-resistant. The lemma follows. □

Achieving bounded concurrent zero knowledge. We have shown that Construction 4.5 satisfies the first two extra properties asserted in Theorem 1.2. To establish the third extra property, we slightly modify the construction (analogously to the way this is done in [7, Chapter 4], which constitutes the full version of [6]). Specifically, for a suitably chosen polynomial ℓ, in Part 1 we select r uniformly in $\{0,1\}^{\ell(n)+n}$ (rather than in $\{0,1\}^n$), and in Part 2 we relax the second condition (or case) such that now we require that there exists a string $z \in \{0,1\}^{\ell(n)}$ such that $\Pi(c,z) = r$ (rather than requiring that $\Pi(c) = r$). The extra string z allows us to encode information from n^2 concurrent executions of the protocol (see details in [7, section 4.4]) and so enables the simulator strategy to insert a "trapdoor" into the second step of Part 1 of the current execution (and thus proceed in an "execution-by-execution" manner). When demonstrating computational soundness, we merely observe that, for a uniformly distributed $r' \in \{0,1\}^{\ell(n)+n}$ (chosen in Step 5), it is unlikely that Π, which is (implicitly) recovered (in Step 4) obliviously of r', will satisfy $\Pi(c,z') = r'$ for some $z' \in \{0,1\}^{\ell(n)}$.

Appendix A. Auxiliary properties of popular PCP systems. We claim that the PCP system of Babai et al. [5] satisfies all of the auxiliary properties listed in Definition 3.2. As stated in the main text, it is clear that this PCP system is nonadaptive, and thus we turn to establish the following remaining properties:[19]

Relatively efficient oracle construction. The oracle in this PCP system consists of two parts: (1) a low-degree extension of a multivariate function that encodes the original witness, and (2) various partial sums of the values of this polynomial, where the number of these partial sums is smaller than the size of the domain of the low-degree extension. The low-degree polynomial can be constructed in time that is polynomial in the length of the *explicit* description of the aforementioned function, and the same holds with respect to each of the partial sums.

Efficient reverse sampling. The queries in this PCP system are either evaluations of the aforementioned polynomial at various points or queries regarding some partial sums of such values. Specifically, the queries belong either to the low-degree test or to the sum check, respectively. The queries of the low-degree test are points along a random line, while the queries of the sum check are prefixes of a random point. Thus, given i and q, it is easy to select uniformly a random tape for the verifier such that the ith query of the verifier (on this random tape) equals q.

A proof-of-knowledge property. The standard analysis of this PCP system actually establishes that if the verifier rejects with probability smaller than $1/2$, then part of the oracle is (very) close to a low-degree polynomial that extends a multivariate function that encodes a valid witness. Thus, by standard self-correction, we may (probabilistically) recover each bit in this witness in polynomial time. Hence, we obtain a polynomial-time oracle machine E such that for every x and π that satisfy $\Pr[V^\pi(x) = 1] > 1/2$ it follows that there

[19] Needless to say, we assume some familiarity with the work of Babai et al. [5]. In particular, the high-level overview provided in [18, section 9.3.2.2] suffices.

exists $w = w_1 \cdots w_t$ such that $(x, w) \in R$ and $\Pr[E^\pi(x, i) = w_i] > 2/3$ holds for every i.

As stated in the main text, to obtain the desired error of ϵ, we apply straightforward error reduction (on the PCP system) while noting that this process does not affect the oracle, and so the resulting (error-reduced) PCP preserves all of the auxiliary properties (including a proof-of-knowledge property that refers to a lower level of error). This completes the proof of Theorem 3.3.

We comment that the foregoing considerations are not specific to the PCP system of [5] but rather hold also with respect to many other PCP systems.

Appendix B. Nonoblivious commitment schemes. We first recall the definition of nonoblivious commitment schemes, following [16, section 4.9.2.1] and [17, section C.3.3]. This definition augments the definition of a standard commitment scheme as presented in [16, section 4.4.1.1]. The latter definition (i.e., [16, Definition 4.4.1]) refers to a scheme that is computationally hiding and statistically binding. Loosely speaking, such a commitment scheme is an efficient *two-phase* two-party protocol through which one party, called the *sender*, can commit itself to a *value* so that the following two conflicting requirements are satisfied:

Hiding. At the end of the first phase, the other party, called the *receiver*, does not gain any knowledge of the sender's value. This requirement has to be satisfied even if the receiver tries to cheat. The computational version asserts that the receiver's views of interactions regarding any two values used by the sender are computational indistinguishable.

Binding. Given the transcript of the interaction in the first phase, there exists at most one value that the receiver may later (i.e., in the second phase) accept as a legal "opening" of the commitment. This requirement has to be satisfied even if the sender tries to cheat. The statistical version asserts that this property holds with overwhelmingly high probability (even if the sender is compuationally unbounded), where the probability is taken solely over the receiver's randomness.

In addition, one should require that the protocol is *viable* in the sense that if both parties follow it, then, at the end of the second phase, the receiver gets the value committed to by the sender. Throughout this appendix we refer to commitment schemes that are computationally hiding and statistically binding. Actually, for the sake of simplicity, we assume that the binding propery is perfect (i.e., it holds with probability 1).[20]

It is indeed time to define formally the notion of a *proper opening* of a commitment, which is also known as proper decommitment. Typically, one considers a *canonical decommitment*, which consists of all randomness used by the sender in the commit phase. In this case, the value v and the sender's randomness s may be considered a proper decommitment to the receiver's view of the commitment phase if it is consistent with that view (i.e., using these values along with the receiver's randomness yields the transcript of messages seen by the receiver). However, a more

[20] We mention that such (perfectly binding) commitment schemes can be constructed based on any one-way permutation [16, section 4.4.1.2], but only commitment schemes with nonperfect statistical binding are known to be constructible based on any one-way function [16, section 4.4.1.3]. Definition B.1 can be extended (from the case of perfectly binding commitment schemes) to the case of statistically binding commitment schemes by referring to a proof of knowledge for a promise problem, where the promise set consists of all interaction transcripts that can be opened in at most one way (i.e., yielding one value).

general notion of proper decommitment may be beneficial. Specifically, we refer to an arbitrary algorithm $\mathtt{D}$ that satisfies the following two conditions:

1. If the commitment phase is performed properly using the sender's value v and the outcome is a pair (s, c), where s is given to the sender and c is given to the receiver, then $\mathtt{D}((v, s), c) = 1$. This means that (v, s) is accepted as a proper decommitment to c.
2. If the receiver follows the commitment phase properly, then the receiver output c is such that there exists at most one value of v such that the set $\{s : \mathtt{D}((v, s), c) = 1\}$ is nonempty.

Needless to say, the aforementioned canonical decommitment gives rise to such an algorithm $\mathtt{D}$. In what follows, we shall assume (without loss of generality) that the commitment c is contained in the information s has given to the sender.

Recall that the foregoing condition 2 mandates that, even when the sender cheats, there exists at most one value v that can be properly decommitted for the output c (i.e., at most one v satisfies $\{s : \mathtt{D}((v, s), c) = 1\} \neq \emptyset$). This does not mean that the sender necessarily knows v, and furthermore such a v may not exist at all (i.e., $\{s : \mathtt{D}((v', s), c) = 1\} = \emptyset$ may hold for every v'). The following nonobliviousness requirement mandates that the sender "knows" the set of values for which proper decommitment is impossible; that is, if the receiver does not detect cheating, then it is the case that the sender knows v such that for every $v' \neq v$ there is no proper decommitment to c that yields the value v' (i.e., $\{s : \mathtt{D}((v', s), c) = 1\} = \emptyset$ holds).

Definition B.1 (nonoblivious commitment schemes). *A commitment scheme is called* nonoblivious *if its commit phase consists of two subphases such that the first subphase is an ordinary commitment phase and the second subphase is a proof of knowledge of the only possible value that can be properly decommitted with respect to the first subphase. That is, the commitment scheme (S, R) decomposes into $S = (S_1, S_2)$ and $R = (R_1, R_2)$ such that*

1. *(S_1, R_1) is a commitment scheme with a decommitment-verification algorithm $\mathtt{D}$;*
2. *(S_2, R_2) is a proof-of-knowledge system* (with negligible error—see [16, section 4.7.1]) *for the relation*

$$\{(c, v) : \forall v' \neq v \, \forall s' \quad \mathtt{D}((v', s'), c) \neq 1\}, \tag{B.1}$$

and this proof system is invoked on common input c and the prover's auxiliary input (v, s), where (s, c) is a result of applying (S_1, R_1) to the value v.

Note that the relation defined in (B.1) is not necessarily an NP-witness relation (i.e., it is not necessarily polynomial-time recognizable), but the definition of a proof of knowledge applies nevertheless (see [16, section 4.7.1] and further discussions in [9]). A seemingly stronger notion (of nonoblivious commitment schemes) requires a proof of knowledge for an NP-witness relation R' such that (c, v) satisfies (B.1) if and only if there exists $w \in \{0,1\}^{\mathrm{poly}(|c|)}$ such that $(c, (v, w)) \in R'$ holds. We comment that Construction B.2 (below) satisfies this stronger definition.

We mention that, in [17, section C.3.3], Definition B.1 is referred to as a *relaxed version* of the notion of a nonoblivious commitment scheme, whereas in the nonrelaxed notion the second subphase is required to prove knowledge of the input as well as of the proper decommitment information. That is, in the strict (i.e., nonrelaxed) notion of nonoblivious commitment schemes (outlined in [17, section C.3.3]), the second subphase is a proof of knowledge of an input and decommitment information that are consistent with the receiver's view of the first subphase (i.e., a proof-of-knowledge

system for the relation $\{(c,(v,s)) : \mathtt{D}((v,s),c) = 1\}$). (In particular, this implies that this stage is a proof of membership for the corresponding language, hence proving that there does in fact exist a value v that can be properly decommitted.) Originally, we thought that we could provide a constant-round public-coin implementation of the strict notion, but a flaw in our reasoning was discovered by Haitner, Rosen, and Shaltiel. Although the relaxed notion suffices for our applications (as well as for other natural applications (see, e.g., [16, section 4.9.2.2])), we believe that the question of whether the strict notion can be implemented by a constant-round public-coin protocol is of independent interest.

Constructing nonoblivious commitment schemes. Our construction follows the outline provided in [17, section C.3.3]. The basic idea is augmenting a standard commitment scheme with an adequate zero-knowledge proof of knowledge, but the problem is that such a proof of knowledge that is also constant-round and public-coin is not known. As observed in [16, section 4.9.2.1], it suffices to use a *strong*-WI proof of knowledge, but such a protocol is not known either (see [17, section C.3]). We stress that in both cases we have referred to protocols with negligible soundness error and that corresponding protocols with constant soundness error do exist. The solution suggested in [17, section C.3.3] is using multiple commitments (via the standard scheme) and applying a *strong*-WI proof of knowledge of constant soundness error to each of the different commitments. In this setting (of statistically independent inputs), the *strong*-WI property is preserved under parallel executions (cf. [17, Lemma C.3.1]). The following construction merely spells out this suggestion.

CONSTRUCTION B.2 (a nonoblivious commitment scheme). *Let* $\mathtt{C}$ *be a standard statistically binding commitment scheme, and let* $\mathtt{C}_s(x)$ *denote the receiver's view of the commitment phase when the sender inputs the value* x *and uses randomness* s*. The following description refers to committing to the value* v *under the security parameter* n*, where* $|v| \leq \mathrm{poly}(n)$*:*

First commit subphase. *The parties invoke* $\mathtt{C}$*, in parallel, for* n *times. In all invocations the sender enters the value* v*, and in the* i*th invocation the receiver obtains the value* $c_i = \mathtt{C}_{s_i}(v)$*, where* s_i *denotes the sender's randomness.*

These n *invocations yield a standard commitment scheme with a decommitment algorithm* $\mathtt{D}$ *that accepts the value* v *if it is supported by* n *proper decommitments with respect to the basic commitment scheme; that is,* $\mathtt{D}((v,\overline{s}'),\overline{c}) = 1$ *if* $\overline{c} = (c_1,\ldots,c_n)$ *and* $\overline{s}' = (s'_1,\ldots,s'_n)$ *such that for every* $i \in [n]$ *it holds that* $c_i = \mathtt{C}_{s'_i}(v)$*.*

Second commit subphase. *The parties perform, in parallel,* n *copies of the following protocol. In the* i*th copy, the sender proves in zero knowledge that it knows the values* v *and* s_i *that were used in the corresponding copy of the first subphase. That is, the sender proves that it knows* (v,s_i) *such that* $c_i = \mathtt{C}_{s_i}(v)$*. Each of these proofs of knowledge has constant soundness error. If the receiver detects cheating in any of these executions, then it aborts, indicating that the sender is cheating. Otherwise, the receiver accepts the commitment* $(c_1,\ldots,c_n)$*.*

A proper decommitment to the value v *with respect to a transcript* $\overline{c} = (c_1,\ldots,c_n)$ *of the first subphase consists of a sequence of* n *strings* $(s'_1,\ldots,s'_n)$ *such that for every* $i \in [n]$ *it holds that* $c_i = \mathtt{C}_{s'_i}(v)$*.*

Construction B.2 inherits the statistical-binding property of the standard commitment $\mathtt{C}$ (since a proper decommitment in Construction B.2 requires proper decommitments of $\mathtt{C}$ to the same value). We first show that, although the zero-knowledge property may not be preserved in the parallel executions that take place in the second

commit subphase, this subphase preserves the computational-hiding property of the standard commitment $\mathtt{C}$.

CLAIM B.2.1. *Construction* B.2 *is computationally hiding.*

Proof. We view Construction B.2 as consisting of n parallel executions of a basic protocol in which the sender first commits to v and then proves (in zero knowledge) that it knows a proper decommitment. This basic protocol is computationally hiding because it consists of producing a computationally hiding commitment and running a zero-knowledge proof regarding this commitment.[21] Thus, it suffices to prove that any computationally hiding commitment scheme preserves this property under parallel executions. Indeed, the proof proceeds by a hybrid argument and is very similar to the proof of Lemma C.3.1 in [17]. □

It is left to establish the nonoblivious property of Construction B.2. To this end, we assume that the proof of knowledge employed is such that the verifier tosses a single coin (and indeed has knowledge error 1/2).[22] We note that such protocols (of constant soundness error) exist; see, e.g., [16, Chapter 4, Exercise 28.1].

CLAIM B.2.2. *Construction* B.2, *when implemented using a proof of knowledge in which the verifier tosses a single coin, is nonoblivious.*

Proof. We fix an arbitrary execution of the first (commit) subphase and consider a random execution of the second (commit) subphase, which amounts to n parallel executions of the proof-of-knowledge protocol. Fixing an arbitrary (deterministic) sender strategy, the underlying probability space consists of the n random (bit) choices made in the corresponding executions. We denote by p the probability that the receiver is convinced by all of these proofs, where here the probability is taken uniformly over all of the receiver's n (binary) choices. We call the ith copy *good* if, *for each choice of the verifier bit in this copy*, the receiver is convinced (in this copy) with probability at least p/n, where the probability is taken over the receiver's choices in all of the other $n-1$ copies. The current claim follows by combining two facts:

1. If $p > 2^{-n}$, then at least one of the indices is good.
   Otherwise, we reach a contraction by upper bounding the number of points in the probability space that cause the receiver to be convinced. Specifically, if all indices are not good, then for every $i \in [n]$ there exists a bit σ_i such that the number of convincing points in which the ith bit equals $1-\sigma_i$ is less than $(p/n) \cdot 2^{n-1}$. Thus, the total number of convincing points is less than $1 + n \cdot (p/n)2^{n-1}$, because the set of convincing points denoted by C is a subset of
   $$\{\sigma_1 \cdots \sigma_n\} \cup \{\alpha_1 \cdots \alpha_n \in C : \exists i \in [n] \text{ s.t. } \alpha_i = 1 - \sigma_i\}.$$
   It follows that $p \cdot 2^n < 1 + p \cdot 2^{n-1}$, which yields $p < 2^{-(n-1)}$. Recalling that p is defined with respect to a probability space of size 2^n, it follows that $p \leq 2^{-n}$, in contradiction to the hypothesis $p > 2^{-n}$.
2. If i is good, then the corresponding knowledge (i.e., a pair (v_i, s_i) such that $c_i = \mathtt{C}_{s_i}(v_i)$) can be extracted in $\mathrm{poly}(n)/p$ steps.
   By making $O(n/p)$ trials, we can generate a convincing transcript for each of the (two) possible choices of the verifier's random bit in the execution of the ith copy. This implies extraction (by the definition of a proof of knowledge).

[21] Indeed, it is instructive to note that the zero-knowledge property implies the *strong*-WI property (see [16, section 4.6.1.1]), whereas the latter property preserves the computational indistinguishability of $\mathtt{C}$-commitments (to different values).

[22] We believe that the argument can be extended to the general case (or at least to all standard proof-of-knowledge protocols). See the related discussion in [9, Appendix C.2].

Thus, assuming $p > 2^{-n}$, we can extract the relevant decommitment information (i.e., v as well as at least one corresponding s_i such that $c_i = \mathtt{C}_{s_i}(v)$) in time that is inversely proportional to the probability that the receiver is convinced in the second commit subphase. This information guarantess that the corresponding value (i.e., v) is the only value for which a proper decommitment exists. It follows that the second commit subphase (of Construction B.2) constitutes a proof of knowledge, with error 2^{-n}, of the only possible value that can be properly decommitted with respect to the first subphase. The claim follows. □

Recalling that all ingredients used in Construction B.2 can be implemented based on any one-way function (via constant-round public-coin protocols), we get the following theorem.

THEOREM B.3. *The existence of one-way functions implies the existence of nonoblivious commitment schemes. Furthermore, these systems are of the public-coin type and use a constant number of rounds.*

We also mention that the properties of nonoblivious commitment schemes are preserved when many copies of the scheme are executed in parallel (or even concurrently under arbitrary scheduling). The preservation of the standard properties of a commitment scheme follows by a hybrid argument (as employed in the proof of Claim B.2.1). The preservation of the nonoblivious (or rather the proof-of-knowledge) property follows by focusing on each individual copy and considering an auxiliary sender that emulates all of the other copies. (We stress that we do not claim here a reduction in the soundness error of the proof-of-knowledge property but rather a preservation of the proof-of-knowledge property at the same level of soundness error.)

Acknowledgments. We are grateful to the anonymous referees for their careful review and for numerous corrections and suggestions. We are indebted to Iftach Haitner, Alon Rosen, and Ronen Shaltiel for discovering a flaw in a previous presentation (see Appendix B).

REFERENCES

[1] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and intractability of approximation problems*, J. ACM, 45 (1998), p. 501–555.

[2] S. ARORA AND S. SAFRA, *Probabilistic checkable proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[3] L. BABAI, *Trading group theory for randomness*, in Proceedings of the 17th Symposium on Theory of Computing, Providence, RI, 1985, pp. 421–429.

[4] L. BABAI, L. FORTNOW, AND C. LUND, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[5] L. BABAI, L. FORTNOW, L. LEVIN, AND M. SZEGEDY, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd Symposium on Theory of Computing, New Orleans, 1991, pp. 21–31.

[6] B. BARAK, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, 2001, pp. 106–115. A full version appears as [7, Chapter 4].

[7] B. BARAK, *Non-black-box techniques in cryptography*, Ph.D. thesis, Rehovdt, Israel, Weizmann Institute of Science, 2004; Also available at http://www.cs.princeton. edu/∼ boaz/research.php.

[8] B. BARAK AND O. GOLDREICH, *Universal arguments and their applications*, Electronic Colloqium on Computational Complexity, 2001, report TR01-093.

[9] M. BELLARE AND O. GOLDREICH, *On defining proofs of knowledge*, in Advances in Cryptology Cryptol–92, Lecture Notes in Comput. Sci. 740, Springer-Verlag, Berlin, 1993, pp. 390–420.

[10] M. BEN-OR, O. GOLDREICH, S. GOLDWASSER, J. HÅSTAD, J. KILIAN, S. MICALI, AND P. ROGAWAY, *Everything provable is provable in zero-knowledge*, in Cryptology–Crypto'88, Lecture Notes in Comput. Sci. 403, Springer-Verlag, Berlin, 1990, pp. 37–56.

[11] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, *Multi-prover interactive proofs: How to remove intractability*, in Proceedings of the 20th Symposium on Theory of Computing, Chicago, 1988, pp. 113–131.
[12] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp. 156–189.
[13] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, J. ACM, 51 (2004), pp. 557–594.
[14] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, J. ACM, 43 (1996), pp. 268–292.
[15] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, Theoret. Comput. Sci., 134 (1994), pp. 545–557.
[16] O. Goldreich, *Foundation of Cryptography – Basic Tools*, Cambridge University Press, Cambridge, 2001.
[17] O. Goldreich, *Foundation of Cryptography – Basic Applications*, Cambridge University Press, Cambridge, 2004.
[18] O. Goldreich, *Computational Complexity: A Conceptual Perspective*, Cambridge University Press, Cambridge, 2008.
[19] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.
[20] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM Comput., 18 (1989), pp. 186–208.
[21] J. Kilian, *A note on efficient zero-knowledge proofs and arguments*, in Proceedings of the 24th Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 723–732.
[22] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.
[23] S. Micali, *Computationally sound proofs*, SIAM J. Comput., 30 (2000), pp. 1253–1298.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1695–1708

EXPONENTIAL SEPARATION FOR ONE-WAY QUANTUM COMMUNICATION COMPLEXITY, WITH APPLICATIONS TO CRYPTOGRAPHY*

DMITRY GAVINSKY†, JULIA KEMPE‡, IORDANIS KERENIDIS§, RAN RAZ¶, AND RONALD DE WOLF‖

Abstract. We give an exponential separation between one-way quantum and classical communication protocols for a partial Boolean function (a variant of the Boolean hidden matching problem of Bar-Yossef et al.). Previously, such an exponential separation was known only for a relational problem. The communication problem corresponds to a *strong extractor* that fails against a small amount of *quantum* information about its random source. Our proof uses the Fourier coefficients inequality of Kahn, Kalai, and Linial. We also give a number of applications of this separation. In particular, we show that there are privacy amplification schemes that are secure against classical adversaries but not against quantum adversaries; and we give the first example of a key-expansion scheme in the model of bounded-storage cryptography that is secure against classical memory-bounded adversaries but not against quantum ones.

Key words. communication complexity, exponential separation, quantum communication, one-way communication, hidden matching problem, quantum cryptography, bounded-storage model, extractor, streaming model

AMS subject classifications. 81P68, 81P15, 94A05, 94A60, 68P30, 68Q01

DOI. 10.1137/070706550

1. Introduction.

One of the main goals of quantum computing is to exhibit problems where quantum computers are much faster (or otherwise better) than classical computers, preferably exponentially better. The most famous example, Shor's efficient quantum factoring algorithm [32], constitutes a separation only if one is willing to believe that efficient factoring is impossible on a classical computer—proving this would, of course, imply P $\neq$ NP. One of the few areas where one can establish *unconditional* exponential separations is communication complexity.

Communication complexity is a central model of computation, first defined by Yao [36]. It has found applications in many areas [20]. In this model, two parties, Alice

*Received by the editors October 29, 2007; accepted for publication (in revised form) July 18, 2008; published electronically December 19, 2008. The research of the second and third authors was supported in part by ACI Sécurité Informatique SI/03 511 and ANR AlgoQP grants of the French Research Ministry, by an Alon Fellowship of the Israeli Higher Council of Academic Research, by a grant of the Israeli Science Foundation, and by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as contract 015848.

http://www.siam.org/journals/sicomp/38-5/70655.html

†NEC Laboratories America, Inc., Princeton, NJ 08540 (dmitry.gavinsky@gmail.com). This author's research was supported in part by Canada's NSERC. Part of this work was done while this author was at the Institute for Quantum Computing, University of Waterloo.

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (kempe@post.tau.ac.il). Part of this work was done while this author was with LRI & CNRS, Université de Paris-Sud, Orsay, France.

§CNRS & LRI, Université de Paris-Sud, 91405 Orsay Cedex, France (jkeren@gmail.com).

¶Faculty of Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (ran.raz@weizmann.ac.il). Part of this work was done when this author visited Microsoft Research in Redmond, WA, supported by ISF and BSF grants.

‖CWI, 1098 SJ Amsterdam, The Netherlands (rdewolf@cwi.nl). This author's research was supported by a Veni grant from the Netherlands Organization for Scientific Research (NWO) and also partially supported by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as contract 015848.

with input x and Bob with input y, collaborate to solve some computational problem that depends on both x and y. Their goal is to do this with minimal communication. The problem to be solved could be a function $f(x, y)$ or some relational problem where for each x and y several outputs are valid. The protocols could be *interactive (two-way)*, in which case Alice and Bob take turns sending messages to each other; *one-way*, in which case Alice sends a single message to Bob, who then determines the output; or *simultaneous*, where Alice and Bob each pass one message to a third party (the *referee*) who determines the output. The bounded-error *communication complexity* of the problem is the worst-case communication of the best protocol that gives (for every input x and y) a correct output with probability at least $1 - \varepsilon$ for some fixed constant $\varepsilon \in [0, 1/2)$, usually $\varepsilon = 1/3$.

Allowing the players to use *quantum* resources can reduce the communication complexity significantly. Examples of problems where quantum communication gives exponential savings were given by Buhrman, Cleve, and Wigderson for one-way and interactive protocols with zero error probability [8]; by Raz for bounded-error interactive protocols [28]; and by Buhrman et al. for bounded-error simultaneous protocols [9]. The first two problems are partial Boolean functions, while the third one is a total Boolean function. However, the latter separation does not hold in the presence of public coins.[1] Bar-Yossef, Jayram, and Kerenidis [4] showed an exponential separation for one-way protocols and simultaneous protocols with public coins, but they achieved this only for a relational problem, called the *hidden matching problem*. This problem can be solved efficiently by one quantum message of $\log n$ qubits, while classical one-way protocols need to send nearly $\sqrt{n}$ bits to solve it. However, Boolean functions are much more natural objects than relations both in the model of communication complexity and in the cryptographic settings that we consider later in this paper. Bar-Yossef et al. stated a Boolean version of their problem (a partial Boolean function) and conjectured that the same quantum-classical gap holds for this problem as well.

1.1. Exponential separation for a variant of Boolean hidden matching. In this paper we prove an exponential quantum-classical one-way communication gap for a variant of the Boolean hidden matching problem of [4]. Let us first state a non-Boolean communication problem. Suppose Alice has an n-bit string x, and Bob has a sequence M of αn disjoint pairs $(i_1, j_1), (i_2, j_2), \ldots, (i_{\alpha n}, j_{\alpha n}) \in [n] \times [n]$ for some parameter $\alpha \in (0, 1/2]$. This M may be viewed as a partial matching on the graph whose vertices are the n bits $x_1, \ldots, x_n$. We call this an *α-matching*. Together, x and M induce an αn-bit string z defined by the parities of the αn edges:

$$z = z(x, M) = (x_{i_1} \oplus x_{j_1}), (x_{i_2} \oplus x_{j_2}), \ldots, (x_{i_{\alpha n}} \oplus x_{j_{\alpha n}}).$$

Suppose Bob wants to learn some information about z. Let $x \in \{0,1\}^n$ be uniformly distributed, and let M be uniform over the set $\mathcal{M}_{\alpha n}$ of all α-matchings. Note that for any fixed M, a uniform distribution on x induces a uniform distribution on z. Hence Bob (knowing M but not x) knows nothing about z: from his perspective it is uniformly distributed. But now suppose Alice can send Bob a short message. How much can Bob learn about z, given that message and M?

The answer is very different depending on whether the message is quantum or classical. To state this difference, we need to introduce some terminology. For prob-

[1] In fact, whether there exists a superpolynomial separation for a total Boolean function in the presence of public coins is one of the main open questions in the area of quantum communication complexity.

ability distributions p and q whose supports are subsets of a set S, define their *total variation distance* as

$$\| p - q \|_{tvd} = \sum_{i \in S} |p(i) - q(i)|. \tag{1}$$

This distance is 0 if and only if $p = q$; it is 2 if and only if p and q have disjoint supports; and the value lies between 0 and 2 otherwise. Suppose we want to distinguish p from q, given a sample from one of the two. The best probability with which we can succeed is $\frac{1}{2} + \frac{||p-q||_{tvd}}{4}$. This well-known fact gives a clear intuitive meaning to the notion of total variation distance. Modifying the protocol of [4], it is easy to show that a short *quantum* message of about $\log(n)/2\alpha$ qubits allows Bob to learn a bit at a random position in the string z. This already puts a lower bound of 1 on the total variation distance between Bob's distribution on z and the uniform αn-bit distribution.

What about a short *classical* message? Using the birthday paradox, one can show that if Alice sends Bob about $\sqrt{n/\alpha}$ bits of x, then with constant probability there will be one edge (i_ℓ, j_ℓ) for which Bob receives both bits x_{i_ℓ} and x_{j_ℓ}. Since $z_\ell = x_{i_\ell} \oplus x_{j_\ell}$, this gives Bob a bit of information about z. Our key theorem says that this classical upper bound is essentially optimal: if Alice sends much fewer bits, then from Bob's perspective the string z will be close (in total variation distance) to uniformly distributed, so he does not even know one bit of z.

In order to be able to state this precisely, suppose Alice is deterministic and sends c bits of communication. Then her message partitions the set of 2^n x's into 2^c sets, one for each message. A typical message will correspond to a set A of about 2^{n-c} x's. Given this message, Bob knows the random variable X is drawn uniformly from this set A and he knows M, which is his input. Hence his knowledge of the random variable $Z = z(X, M)$ is fully described by the distribution

$$p_M(z) = \Pr[Z = z \mid \text{given } M \text{ and Alice's message}] = \frac{|\{x \in A \mid z(x, M) = z\}|}{|A|}.$$

Our main technical result says that if the communication c is much less than $\sqrt{n/\alpha}$ bits, then for a typical message and averaged over all matchings M, this distribution is very close to uniform in total variation distance. In other words, most of the time Bob knows essentially nothing about z.

THEOREM 1. *Let x be uniformly distributed over a set $A \subseteq \{0,1\}^n$ of size $|A| \geq 2^{n-c}$ for some $c \geq 1$, and let M be uniformly distributed over the set $\mathcal{M}_{\alpha n}$ of all α-matchings, for some $\alpha \in (0, 1/4]$. There exists a universal constant $\gamma > 0$ (independent of n, c, and α), such that for all $\varepsilon \in (0, 2]$, if $c \leq \gamma \varepsilon \sqrt{n/\alpha}$, then*

$$\mathbb{E}_M \left[\| p_M - U \|_{tvd} \right] \leq \varepsilon.$$

Note that the ε in this theorem is not the error probability of a protocol for a Boolean function, but an upper bound on the expected distance between Bob's distribution p_M and the uniform distribution. We prove Theorem 1 using the Fourier coefficients inequality of Kahn, Kalai, and Linial [16], which is a special case of the Bonami–Beckner inequality [7, 5]. We remark that Fourier analysis has been previously used in communication complexity by Raz [27] and Klauck [17].

This result allows us to turn the above communication problem into a partial Boolean function, as follows. Again we give Alice input $x \in \{0,1\}^n$, while Bob now receives two inputs: a partial matching M as before and an αn-bit string w. The

promise on the input is that w is either equal to $z = z(x, M)$ or to its complement $\overline{z}$ (i.e., z with all bits flipped). The goal is to find out which of these two possibilities is the case. We call this communication problem αPM, for *α-partial matching*. As mentioned before, Alice can allow Bob to learn a random bit of z with high probability by sending him an $O(\log(n)/\alpha)$-qubit message. Knowing one bit z_ℓ of z suffices to compute the Boolean function: just compare z_ℓ with w_ℓ. In contrast, if Alice sends Bob much less than $\sqrt{n/\alpha}$ classical bits, then Bob still knows essentially nothing about z. In particular, he cannot decide whether $w = z$ or $w = \overline{z}$! This gives the following separation result for the classical and quantum one-way communication complexities (with error probability fixed to $1/3$, say).

THEOREM 2. *Let $\alpha \in (0, 1/4]$. The classical bounded-error one-way communication complexity of the α-PM problem is $R^1(\alpha\mathrm{PM}) = \Theta(\sqrt{n/\alpha})$, while the quantum bounded-error one-way complexity is $Q^1(\alpha\mathrm{PM}) = O(\log(n)/\alpha)$.*

Fixing α to $1/4$, we obtain the promised exponential quantum-classical separation for one-way communication complexity of $O(\log n)$ qubits vs. $\Omega(\sqrt{n})$ classical bits.

As noted by Aaronson [1, section 5], Theorem 2 implies that his general simulation of bounded-error one-way quantum protocols by deterministic one-way protocols,

$$D^1(f) = O(mQ^1(f) \log Q^1(f)),$$

is tight up to a polylogarithmic factor. Here m is the length of Bob's input. This simulation works for any partial Boolean function f. Taking f to be our αPM for $\alpha = 1/4$, one can show that $D^1(f) = \Theta(n)$, $m = \Theta(n \log n)$, and $Q^1(f) = O(\log n)$. It also implies that his simulation of quantum bounded-error one-way protocols by classical bounded-error one-way protocols,

$$R^1(f) = O(mQ^1(f)),$$

cannot be considerably improved. In particular, the product on the right cannot be replaced by the sum: if we take $f = \alpha$PM with $\alpha = 1/\sqrt{n}$, then by Theorem 2 we have $R^1(f) \approx n^{3/4}$, $m \approx \sqrt{n} \log n$, and $Q^1(f) = O(\sqrt{n} \log n)$.

Remarks. The earlier conference version of this paper [13] had two different communication problems, establishing an exponential one-way separation for both of them in quite different ways. The present paper unifies these two approaches to something substantially simpler.

The original Boolean hidden matching problem stated in [4] is our αPM with $\alpha = 1/2$ (i.e., M is a perfect matching). Theorem 2, on the other hand, assumes $\alpha \leq 1/4$ for technical reasons. By doing the analysis in section 3 a bit more carefully, we can prove Theorem 2 for every α that is bounded away from $1/2$. Note that if $\alpha = 1/2$, then the parity of $z = z(x, M)$ equals the parity of x, so by communicating the parity of x in one bit, Alice can give Bob one bit of information about z. The conference version of this paper showed that one can prove a separation for the case where M is a perfect matching if the promise is that w is "close" to z or its complement (instead of being *equal* to z or its complement). One can think of w in this case as a "noisy" version of $z = z(x, M)$ (or its complement), while the w of our current version can be thought of as starting from a perfect matching M', and then "erasing" some of the $n/2$ bits of the string $z(x, M')$ to get the αn-bit string z (or its complement).

The separation given here can be modified to a separation in the simultaneous message passing model, between the models of classical communication with shared entanglement and classical communication with shared randomness. Earlier, such a separation was known only for a relational problem [4, 12], not for a Boolean function.

1.2. Application: Privacy amplification. Randomness *extractors* extract almost uniform randomness from an *imperfect* (i.e., nonuniform) source of randomness X with the help of an independent uniform seed Y. With a bit of extra work (see section 4), Theorem 1 actually implies that our function $z : \{0,1\}^n \times \mathcal{M}_{\alpha n} \to \{0,1\}^{\alpha n}$ is an extractor:

> If $X \in \{0,1\}^n$ is a random variable with min-entropy at least $n - \gamma\varepsilon\sqrt{n/\alpha}$ (i.e., $\max_x \Pr[X = x] \leq 2^{-(n-\gamma\varepsilon\sqrt{n/\alpha})}$) and Y is a random variable uniformly distributed over $\mathcal{M}_{\alpha n}$, then the random variable $Z := z(X,Y)$ is ε-close to the uniform distribution on $\{0,1\}^{\alpha n}$.

It is in fact a *strong* extractor: the pair (Y,Z) is ε-close to the uniform distribution on $\mathcal{M}_{\alpha n} \times \{0,1\}^{\alpha n}$.[2] Informally, this says that if there is a lot of uncertainty about X, then Z will be close to uniform even if Y is known.[3]

Extractors have found numerous applications in computer science, in particular in complexity theory (see, e.g., [31] and the references therein) and cryptography. One important cryptographic application is that of *privacy amplification*, introduced in [6, 15]. In this setting two parties called *Alice* and *Bob* start with a shared random variable X, about which an *adversary* has partial knowledge. The parties' goal is to generate a secret key Z, about which the adversary would have very little information.

They can achieve this by communicating an independent uniform seed Y over a *public* channel, and using a strong extractor to generate the key $Z(X,Y)$. Our extractor guarantees that if the shared variable X, conditioned upon the adversary's knowledge, has min-entropy at least $n - \gamma\varepsilon\sqrt{n/\alpha}$, then the generated αn-bit key Z, conditioned upon adversary's knowledge, is ε-close to uniform. On the other hand, we show that this scheme is *insecure* against a *quantum* adversary who uses only $O(\log n)$ qubits of storage. This is the first example of a privacy amplification scheme that is safe against classical adversaries with up to $\Theta(\sqrt{n})$ bits of storage (with some small constant in the $\Theta(\cdot)$), but not against quantum adversaries with *exponentially less* quantum storage.

This dependence on whether the adversary has quantum or classical memory is quite surprising, particularly in light of the following two facts. First, privacy amplification based on two-universal hashing provides exactly the same security against classical and quantum adversaries. The length of the key that can be extracted is given by the min-entropy both in the classical [6, 15] and the quantum case (see [19, 30], [29, Chapter 5]). Second, König and Terhal [18] have shown that for protocols that extract just *one* bit, the level of security against a classical and a quantum adversary (with the same information bound) is comparable.

1.3. Application: Key-expansion in the bounded-storage model. In privacy amplification, we can ensure that the adversary has much uncertainty about the random variable X by assuming that he has only bounded storage. The idea of basing cryptography on storage limitations of the adversary was introduced by Maurer [23] with the aim of implementing information-theoretically secure *key-expansion*. In this

[2] Note that $\mathbb{E}_M \left[\| p_M - U \|_{tvd} \right] = \| (Y,Z) - U \|_{tvd}$, where U on the left and right is uniform over different domains.

[3] It should be noted that the parameters of our extractor are quite bad, as far as these things go. First, the uniform input seed Y takes about $\alpha n \log n$ bits to describe, which is more than the αn bits that the extractor outputs; in a good extractor, we want the seed length to be much shorter than the output length. Second, our assumed lower bound on the initial min-entropy is quite stringent. Finally, the distance from uniform can be made polynomially small in n (by putting an $n - n^{1/2-\eta}$ lower bound on the min-entropy of X) but not exponentially small, which is definitely a drawback in cryptographic contexts. Still, this extractor suffices for our purposes here.

setting, a large random variable X is publicly but only temporarily available. Alice and Bob use a shared secret key Y to extract an additional key $Z = Z(X, Y)$ from X, in such a way that the adversary has only limited information about the pair (Y, Z). "Limited information" means that the distribution on (Y, Z) is ε-close to uniform even when conditioned on the information about X that the adversary stored. Thus Alice and Bob have expanded their shared secret key from Y to (Y, Z). Aumann, Ding, and Rabin [3] were the first to prove a bounded-storage scheme secure, and essentially tight constructions have subsequently been found [11, 22, 33].

It is an important open question whether any of these constructions remain secure if the adversary is allowed to store *quantum* information. One may even conjecture that a bounded-storage protocol secure against classical adversaries with a certain amount of memory should be roughly as secure against *quantum* adversaries with roughly the same memory bound. After all, Holevo's theorem [14] tells us that k qubits cannot contain more information than k classical bits. However, a key-expansion scheme based on our extractor refutes this conjecture. The scheme is essentially the same as the above privacy amplification scheme, but we describe it separately because the context is a bit different. Alice and Bob will compute $Z := z(X, Y)$ by applying our extractor to X and Y. If the adversary's memory is bounded by $\gamma\varepsilon\sqrt{n/\alpha}$ bits, then Z will be ε-close to uniform from the adversary's perspective. On the other hand, $O(\log n)$ qubits of storage suffice to learn one or more bits of information about Z, given Y, which shows that (Y, Z) is not good as a key against a quantum adversary. Thus we have an example of a key-expansion scheme that is secure against classical adversaries with nearly $\sqrt{n}$ bits of storage, but insecure against quantum adversaries even with exponentially less quantum storage.

1.4. Application: A separation in the streaming model. In the *streaming model* of computation, the input is given as a stream of bits and the algorithm is supposed to compute or approximate some function of the input, having only space of size S available. See, for instance, [2, 24].

There is a well-established connection between one-way communication complexity and the streaming model: if we view the input as consisting of two consecutive parts x and y, then the content of the memory after x has been processed, together with y, contains enough information to compute $f(x, y)$. Hence, a space-S streaming algorithm for f implies a one-way protocol for f of communication S with the same success probability. The classical lower bound for our Boolean communication complexity problem, together with the observation that our quantum protocol can be implemented in the streaming model, implies a separation between the quantum and classical streaming model. Namely, there is a partial Boolean function f that can be computed in the streaming model with small error probability using quantum space of $O(\log n)$ qubits, but requiring $\Omega(\sqrt{n})$ bits if the space is classical.

Le Gall [21] constructed a problem that can be solved in the streaming model using $O(\log n)$ qubits of space, while any classical algorithm needs $\Omega(n^{1/3})$ classical bits. His $\log n$-vs.-$n^{1/3}$ separation is a bit smaller than our $\log n$-vs.-$\sqrt{n}$, but his separation is for a *total* Boolean function while ours is only partial (i.e., requires some promise on the input). Le Gall's result predates ours, though we learned about it only after finishing the conference version of our paper. We remark also that Le Gall's separation holds only in the streaming model variant where the bits arrive in order, while ours holds in the more general model where we allow the different pieces of the input to arrive in any order.

The algorithm for solving our problem in the streaming model starts out with a

$\log n$-qubit superposition $\frac{1}{\sqrt{n}}\sum_{i=1}^{n}|i\rangle$. Whenever a bit x_i streams by in the input, the algorithm unitarily multiplies basis state $|i\rangle$ with a phase $(-1)^{x_i}$. Whenever an edge (i_ℓ, j_ℓ) streams by, the algorithm measures with operators $E_1 = |i_\ell\rangle\langle i_\ell| + |j_\ell\rangle\langle j_\ell|$ and $E_0 = I - E_1$; in case of outcome E_1, the algorithm records the values i_ℓ and j_ℓ (note that E_1 can be obtained at most once, as the edges are pairwise disjoint). And whenever a bit (i_ℓ, j_ℓ, w_ℓ) streams by, the algorithm unitarily multiplies basis state $|\min(i_\ell, j_\ell)\rangle$ with a phase $(-1)^{w_\ell}$. At the end, with probability 2α the algorithm is left with a classical record of $(i_\ell, j_\ell) \in M$ and the corresponding quantum state $\frac{1}{\sqrt{2}}((-1)^{x_{i_\ell}\oplus w_\ell}|i_\ell\rangle + (-1)^{x_{j_\ell}}|j_\ell\rangle)$. The algorithm can learn the function value $x_{i_\ell} \oplus x_{j_\ell} \oplus w_\ell$ from this by a final measurement.

2. The problem and its quantum and classical upper bounds. We assume basic knowledge of quantum computation [26] and (quantum) communication complexity [20, 34].

Before giving the definition of our variant of the Boolean hidden matching problem, we fix some notation. Part of Bob's input will be a sequence M of αn disjoint edges $(i_1, j_1), \ldots, (i_{\alpha n}, j_{\alpha n})$ over $[n]$, which we call an α-matching. We use $\mathcal{M}_{\alpha n}$ to denote the set of all such matchings. If $\alpha = 1/2$, then the matching is *perfect*; if $\alpha < 1/2$, then the matching is *partial*. We can view M as an $\alpha n \times n$ matrix over $GF(2)$, where the ℓth row has exactly two 1's, at positions i_ℓ and j_ℓ. Let $x \in \{0,1\}^n$. Then the matrix-vector product Mx is an αn-bit string $z = z_1, \ldots, z_\ell, \ldots, z_{\alpha n}$ where $z_\ell = x_{i_\ell} \oplus x_{j_\ell}$. Using this notation, we define the following αPM problem, whose one-way communication complexity we will study.

Alice: $x \in \{0,1\}^n$

Bob: an α-matching M and a string $w \in \{0,1\}^{\alpha n}$

Promise on the input: there is a bit b such that $w = Mx \oplus b^{\alpha n}$ (equivalently, $w = z$ or $w = \overline{z}$)

Function value: b

Actually, most of our analysis will not be concerned with Bob's second input w. Rather, we will show that, given only a short message about x, Bob will know essentially nothing about $z = Mx$. Note that to compute b, it suffices that Bob learns one bit z_ℓ of the string z, since $b = z_\ell \oplus w_\ell$. We will first give quantum and classical upper bounds on the message length needed for this.

Quantum upper bound. Suppose Alice sends a uniform superposition of her bits to Bob:

$$|\psi\rangle = \frac{1}{\sqrt{n}}\sum_{i=1}^{n}(-1)^{x_i}|i\rangle.$$

Bob completes his αn edges to a perfect matching in an arbitrary way, and measures with the corresponding set of $n/2$ 2-dimensional projectors. With probability 2α he will get one of the edges (i_ℓ, j_ℓ) of his input M. The state then collapses to

$$\frac{1}{\sqrt{2}}\left((-1)^{x_{i_\ell}}|i_\ell\rangle + (-1)^{x_{j_\ell}}|j_\ell\rangle\right),$$

from which Bob can obtain the bit $z_\ell = x_{i_\ell} \oplus x_{j_\ell}$ by measuring in the corresponding $|\pm\rangle$-basis. Note that this protocol has so-called zero-sided error: Bob knows when he didn't learn any bit z_ℓ. If Bob is given $O(k/\alpha)$ copies of $|\psi\rangle$, then with high probability (at least while $k \ll \alpha n$) he can learn k distinct bits of z.

Remark. This protocol can be modified to a protocol in the simultaneous message passing model in a standard way, first suggested by Buhrman (see [12]). Alice and Bob share the maximally entangled state $\frac{1}{\sqrt{n}}\sum_i |i,i\rangle$. Alice implements the transformation $|i\rangle \to (-1)^{x_i}|i\rangle$ on her half. Bob performs the measurement with his projectors on his half. If he gets one of the edges of his input, he sends the resulting (i_ℓ, j_ℓ) and w_ℓ to the referee. Now Alice and Bob perform a Hadamard transform on their halves, and measure and send the result to the referee, who has enough information to reconstruct z_ℓ.

Classical upper bound. We sketch an $O(\sqrt{n/\alpha})$ *classical* upper bound. Suppose Alice uniformly picks a subset of $d \approx \sqrt{n/\alpha}$ bits of x to send to Bob. By the birthday paradox, with high probability Bob will have both endpoints of at least one of his αn edges and so he can compute a bit of z (and hence the function value b) with good probability. In this protocol Alice would need to send about $d \log n$ bits to Bob, since she needs to describe the d indices as well as their bit values. However, by Newman's theorem [25], Alice can actually restrict her random choice to picking one out of $O(n)$ possible d-bit subsets, instead of one out of all $\binom{n}{d}$ possible subsets. Hence $d + O(\log n)$ bits of communication suffice. This matches our lower bound up to constant factors.

3. Main proof. In this section we prove our main technical result (Theorem 1), which shows that Bob knows hardly anything about the string $z = Mx$ unless Alice sends him a long message.

3.1. Preliminaries. We begin by providing a few standard definitions from Fourier analysis on the Boolean cube. For functions $f, g : \{0,1\}^n \to \mathbb{R}$ we define their inner product and ℓ_2-norm by

$$\langle f, g\rangle = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)g(x), \quad \| f \|_2^2 = \langle f, f\rangle = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} |f(x)|^2. \tag{2}$$

The Fourier transform of f is a function $\widehat{f} : \{0,1\}^n \to \mathbb{R}$ defined by

$$\widehat{f}(s) = \langle f, \chi_s\rangle = \frac{1}{2^n}\sum_{y\in\{0,1\}^n} f(y)\chi_s(y),$$

where $\chi_s : \{0,1\}^n \to \mathbb{R}$ is the character $\chi_s(y) = (-1)^{y\cdot s}$, with "$\cdot$" being the scalar product; $\widehat{f}(s)$ is the Fourier coefficient of f corresponding to s. We have the following relation between f and $\widehat{f}$:

$$f = \sum_{s\in\{0,1\}^n} \widehat{f}(s)\chi_s.$$

We will use two tools in our analysis, Parseval's identity and the Kahn–Kalai–Linial lemma.

LEMMA 3 (Parseval). *For every function $f : \{0,1\}^n \to \mathbb{R}$ we have $\| f \|_2^2 = \sum_{s\in\{0,1\}^n} \widehat{f}(s)^2$.*

Note in particular that if f is an arbitrary probability distribution on $\{0,1\}^n$ and U is the uniform distribution on $\{0,1\}^n$, then $\widehat{f}(0^n) = \widehat{U}(0^n) = 1/2^n$ and $\widehat{U}(s) = 0$ for nonzero s, hence

$$\| f - U \|_2^2 = \sum_{s\in\{0,1\}^n} (\widehat{f}(s) - \widehat{U}(s))^2 = \sum_{s\in\{0,1\}^n\setminus\{0^n\}} \widehat{f}(s)^2. \tag{3}$$

LEMMA 4 (Kahn–Kalai–Linial [16]). *Let f be a function $f : \{0,1\}^n \to \{-1,0,1\}$. Let $A = \{x \mid f(x) \neq 0\}$, and let $|s|$ denote the Hamming weight of $s \in \{0,1\}^n$. Then for every $\delta \in [0,1]$ we have*

$$\sum_{s \in \{0,1\}^n} \delta^{|s|} \widehat{f}(s)^2 \leq \left(\frac{|A|}{2^n}\right)^{\frac{2}{1+\delta}}.$$

We also need the following combinatorial lemma about uniformly chosen matchings.[4]

LEMMA 5. *Let $v \in \{0,1\}^n$. If $|v| = k$ for even k, then*

$$\Pr_M[\exists\, s \in \{0,1\}^{\alpha n} \text{ s.t. } M^T s = v] = \frac{\binom{\alpha n}{k/2}}{\binom{n}{k}},$$

where the probability is taken uniformly over all α-matchings M.

Proof. We can assume without loss of generality that $v = 1^k 0^{n-k}$. We will compute the fraction of matchings M for which there exists such an s. The total number of matchings M of αn edges is $n!/(2^{\alpha n}(\alpha n)!(n-2\alpha n)!)$. This can be seen as follows: Pick a permutation of n, view the first αn pairs as αn edges, and ignore the ordering within each edge, the ordering of the αn edges, and the ordering of the last $n - 2\alpha n$ vertices. Note that $\exists\ s$ s.t. $M^T s = v$ if and only if M has exactly $k/2$ edges in $[k]$ and $\alpha n - k/2$ edges in $[n]\backslash[k]$. The number of ways to pick $k/2$ edges in $[k]$ (i.e., a perfect matching) is $k!/(2^{k/2}(k/2)!)$. The number of ways to pick $\alpha n - k/2$ edges in $[n] - [k]$ is $(n-k)!/(2^{\alpha n - k/2}(\alpha n - k/2)!(n-2\alpha n)!)$. Hence the probability in the lemma equals

$$\frac{k!/(2^{k/2}(k/2)!) \cdot (n-k)!/(2^{\alpha n - k/2}(\alpha n - k/2)!(n-2\alpha n)!)}{n!/(2^{\alpha n}(\alpha n)!(n-2\alpha n)!)} = \frac{\binom{\alpha n}{k/2}}{\binom{n}{k}}. \qquad \square$$

This probability is exponentially small in k if $\alpha < 1/2$, but it equals 1 if $\alpha = 1/2$ and $v = 1^n$.

3.2. Proof of Theorem 1. In order to prove Theorem 1, consider any set $A \subseteq \{0,1\}^n$ with $|A| \geq 2^{n-c}$ and let $f : \{0,1\}^n \to \{0,1\}$ be its characteristic function (i.e., $f(x) = 1$ if and only if $x \in A$). Let $\varepsilon \in (0,2]$, $\alpha \in (0, 1/4]$, and $1 \leq c \leq \gamma\varepsilon\sqrt{n/\alpha}$ for some γ to be determined later.

With x uniformly distributed over A, we can write down Bob's induced distribution on z as

$$p_M(z) = \frac{|\{x \in A \mid Mx = z\}|}{|A|}.$$

We want to show that p_M is close to uniform for most M. By (3), we can achieve this by bounding the Fourier coefficients of p_M. These are closely related to the Fourier coefficients of f:

$$\begin{aligned}\widehat{p_M}(s) &= \frac{1}{2^{\alpha n}} \sum_{z \in \{0,1\}^{\alpha n}} p_M(z)(-1)^{z \cdot s} \\ &= \frac{1}{|A| 2^{\alpha n}} \left(|\{x \in A \mid (Mx) \cdot s = 0\}| - |\{x \in A \mid (Mx) \cdot s = 1\}|\right)\end{aligned}$$

[4] We use the standard convention $\binom{a}{b} = 0$ whenever $b > a$.

$$= \frac{1}{|A|2^{\alpha n}} \left(|\{x \in A \mid x \cdot (M^T s) = 0\}| - |\{x \in A \mid x \cdot (M^T s) = 1\}| \right)$$

$$= \frac{1}{|A|2^{\alpha n}} \sum_{x \in \{0,1\}^n} f(x)(-1)^{x \cdot (M^T s)}$$

$$= \frac{2^n}{|A|2^{\alpha n}} \cdot \widehat{f}(M^T s). \tag{4}$$

Note that the Hamming weight of $v = M^T s \in \{0,1\}^n$ is twice the Hamming weight of $s \in \{0,1\}^{\alpha n}$.

Using the Kahn–Kalai–Linial lemma, we get the following bound on the level sets of the Fourier transform of f.

LEMMA 6. *For every $k \in \{1, \ldots, 4c\}$ we have $\frac{2^{2n}}{|A|^2} \sum_{v:|v|=k} \widehat{f}(v)^2 \leq \left(\frac{4\sqrt{2}c}{k}\right)^k$.*

Proof. By the Kahn–Kalai–Linial inequality (Lemma 4), for every $\delta \in [0,1]$ we have

$$\frac{2^{2n}}{|A|^2} \sum_{v:|v|=k} \widehat{f}(v)^2 \leq \frac{2^{2n}}{|A|^2} \frac{1}{\delta^k} \left(\frac{|A|}{2^n}\right)^{2/(1+\delta)} = \frac{1}{\delta^k} \left(\frac{2^n}{|A|}\right)^{2\delta/(1+\delta)} \leq \frac{1}{\delta^k} \left(\frac{2^n}{|A|}\right)^{2\delta} \leq \frac{2^{2\delta c}}{\delta^k}.$$

Plugging in $\delta = k/4c$ (which is in $[0,1]$ by our assumption on the value of k) gives the lemma. □

We bound the expected *squared* total variation distance between p_M and U as follows:

$$\mathbb{E}_M[\| p_M - U \|_{tvd}^2] \leq 2^{2\alpha n} \mathbb{E}_M \left[\| p_M - U \|_2^2 \right]$$

$$= 2^{2\alpha n} \mathbb{E}_M \left[\sum_{s \in \{0,1\}^{\alpha n} \setminus \{0^{\alpha n}\}} \widehat{p_M}(s)^2 \right]$$

$$= \frac{2^{2n}}{|A|^2} \mathbb{E}_M \left[\sum_{s \in \{0,1\}^{\alpha n} \setminus \{0^{\alpha n}\}} \widehat{f}(M^T s)^2 \right],$$

where we used, respectively, the Cauchy–Schwarz inequality (recall that our definition of $\| \cdot \|_2^2$ in (2) already includes a factor $1/2^{\alpha n}$), (3), and (4). Note that for each $v \in \{0,1\}^n$, there is at most one $s \in \{0,1\}^{\alpha n}$ for which $M^T s = v$ (and the only s that makes $M^T s = 0^n$ is $s = 0^{\alpha n}$). This allows us to change the expectation over M into a probability and use Lemma 5:

$$= \frac{2^{2n}}{|A|^2} \mathbb{E}_M \left[\sum_{v \in \{0,1\}^n \setminus \{0^n\}} |\{s \in \{0,1\}^{\alpha n} \mid M^T s = v\}| \cdot \widehat{f}(v)^2 \right]$$

$$= \frac{2^{2n}}{|A|^2} \sum_{v \in \{0,1\}^n \setminus \{0^n\}} \Pr_M \left[\exists\, s \in \{0,1\}^{\alpha n} \text{ s.t. } M^T s = v \right] \cdot \widehat{f}(v)^2$$

$$= \frac{2^{2n}}{|A|^2} \sum_{\text{even } k=2}^{2\alpha n} \frac{\binom{\alpha n}{k/2}}{\binom{n}{k}} \sum_{v:|v|=k} \widehat{f}(v)^2.$$

We first upper bound the part of this sum with $k < 4c$. Applying Lemma 6 for each k, and using the standard estimates $(n/k)^k \leq \binom{n}{k} \leq (en/k)^k$ and our upper bound

$c \leq \gamma\varepsilon\sqrt{n/\alpha}$, we get

$$\frac{2^{2n}}{|A|^2}\sum_{\text{even } k=2}^{4c-2}\frac{\binom{\alpha n}{k/2}}{\binom{n}{k}}\sum_{v:|v|=k}\widehat{f}(v)^2 \leq \sum_{\text{even } k=2}^{4c-2}\frac{(2e\alpha n/k)^{k/2}}{(n/k)^k}\left(\frac{4\sqrt{2}c}{k}\right)^k \leq \sum_{\text{even } k=2}^{4c-2}\left(\frac{64e\gamma^2\varepsilon^2}{k}\right)^{k/2}.$$

Picking γ a sufficiently small constant, this is at most $\varepsilon^2/2$ (note that the sum starts at $k=2$).

In order to bound the part of the sum with $k \geq 4c$, note that the function $g(k) := \binom{\alpha n}{k/2}/\binom{n}{k}$ is decreasing for the range of even k up to $2\alpha n$ (which is $\leq n/2$ because $\alpha \leq 1/4$):

$$\begin{aligned}\frac{g(k-2)}{g(k)} &= \frac{\binom{\alpha n}{k/2-1}/\binom{n}{k-2}}{\binom{\alpha n}{k/2}/\binom{n}{k}}\\ &= \frac{(n-k+2)(n-k+1)k/2}{(\alpha n-k/2+1)(k-1)k}\\ &= \frac{(n-k+2)(n-k+1)}{(2\alpha n-k+2)(k-1)}\\ &\geq \frac{n-k+1}{k-1} \geq 1.\end{aligned}$$

We also have $\sum_{v\in\{0,1\}^n}\widehat{f}(v)^2 = \frac{|A|}{2^n}$ by Parseval (Lemma 3), and $\frac{2^n}{|A|} \leq 2^c$ by assumption. Hence

$$\frac{2^{2n}}{|A|^2}\sum_{\text{even } k=4c}^{2\alpha n} g(k)\sum_{v:|v|=k}\widehat{f}(v)^2 \leq 2^c g(4c) \leq \left(\frac{8\sqrt{2}e\alpha c}{n}\right)^{2c} \leq \left(8\sqrt{2}e\gamma\varepsilon\sqrt{\frac{\alpha}{n}}\right)^{2c} \leq \varepsilon^2/2,$$

where in the last step we used $\alpha/n \leq 1$ and $c \geq 1$, and picked γ a sufficiently small constant.

Hence we have shown $\mathbb{E}_M[\| p_M - U \|_{tvd}^2] \leq \varepsilon^2$. By Jensen's inequality we have

$$\mathbb{E}_M[\| p_M - U \|_{tvd}] \leq \sqrt{\mathbb{E}_M[\| p_M - U \|_{tvd}^2]} \leq \varepsilon.$$

This concludes the proof of Theorem 1. Let x be uniformly distributed over a set $A \subseteq \{0,1\}^n$ of size $|A| \geq 2^{n-c}$ for some $c \geq 1$, and let M be uniformly distributed over the set $\mathcal{M}_{\alpha n}$ of all α-matchings for some $\alpha \in (0, 1/4]$. There exists a universal constant $\gamma > 0$ (independent of n, c, and α), such that for all $\varepsilon \in (0,2]$, if $c \leq \gamma\varepsilon\sqrt{n/\alpha}$, then

$$\mathbb{E}_M\left[\| p_M - U \|_{tvd}\right] \leq \varepsilon.$$

The ε^2 upper bound on $\mathbb{E}_M[\| p_M - U \|_{tvd}^2]$ is essentially tight. This can be seen in the communication setting as follows. With probability $\Omega(\varepsilon^2)$ over the choice of M, at least one edge of M will have both endpoints in the first $c = \varepsilon\sqrt{n/\alpha}$ bits. Then if Alice just sends the first c bits of x to Bob, she gives him a bit of z. This makes $\| p_M - U \|_{tvd}$ at least 1, hence $\mathbb{E}_M[\| p_M - U \|_{tvd}^2] = \Omega(\varepsilon^2)$.

3.3. Proof of Theorem 2. Our Theorem 2, stated in the introduction, easily follows from Theorem 1. By the Yao principle [35], it suffices to analyze *deterministic*

protocols under some "hard" input distribution. Our input distribution will be uniform over $x \in \{0,1\}^n$ and $M \in \mathcal{M}_{\alpha n}$. The inputs x and M together determine the αn-bit string $z = Mx$. To complete the input distribution, with probability $1/2$ we set $w = z$ and with probability $1/2$ we set w to z's complement $\overline{z}$.

Fix $\varepsilon > 0$ to a small constant, say $1/1000$. Let $c = \gamma\varepsilon\sqrt{n/\alpha}$, and consider any classical deterministic protocol that communicates at most $C = c - \log(1/\varepsilon)$ bits. This protocol partitions the set of 2^n x's into 2^C sets $A_1, \ldots, A_{2^C}$, one for each possible message. On average, these sets have size 2^{n-C}. Moreover, by a simple counting argument, at most a $2^{-\ell}$-fraction of all $x \in \{0,1\}^n$ can sit in sets of size $\leq 2^{n-C-\ell}$. Hence with probability at least $1-\varepsilon$, the message that Alice sends corresponds to a set $A \subseteq \{0,1\}^n$ of size at least $2^{n-C-\log(1/\varepsilon)} = 2^{n-c}$. In that case, by Theorem 1 and Markov's inequality, for at least a $(1-\sqrt{\varepsilon})$-fraction of all M, the random variable $Z = MX$ (with X uniformly distributed over A) is $\sqrt{\varepsilon}$-close to the uniform distribution U. Given w, Bob needs to decide whether $w = Z$ or $w = \overline{Z}$. In other words, he is given one sample w, and needs to decide whether it came from distribution Z or $\overline{Z}$. As we mentioned after (1), he can do this only if the distributions of Z and $\overline{Z}$ have large total variation distance. But by the triangle inequality

$$\| Z - \overline{Z} \|_{tvd} \leq \| Z - U \|_{tvd} + \| \overline{Z} - U \|_{tvd} = 2\| Z - U \|_{tvd} \leq 2\sqrt{\varepsilon}.$$

Hence Bob's advantage over randomly guessing the function value will be at most ε (for the unlikely event that A is very small) plus $\sqrt{\varepsilon}$ (for the unlikely event that M is such that MX is more than $\sqrt{\varepsilon}$ away from uniform) plus $\sqrt{\varepsilon}/2$ (for the advantage over random guessing when $\| Z - U \| \leq \sqrt{\varepsilon}$). To sum up, if the communication is much less than $\sqrt{n/\alpha}$ bits, then Bob cannot decide the function value with probability significantly better than $1/2$.

4. Viewing our construction as an extractor. So far, we have proved that if the n-bit string X is uniformly distributed over a set A with $|A| \geq 2^{n-c}$ (i.e., a flat distribution on A), and Y is uniformly distributed over all α-matchings, then $(Y, Z(X,Y))$ is close to uniform. In order to conclude the result about extractors mentioned in section 1.2, we need to prove the same result in the more general situation when X has min-entropy at least $n-c$ (instead of just being uniform on a set of size at least 2^{n-c}). However, a result by Chor and Goldreich [10, Lemma 5], based on the fact that any distribution can be thought of as a convex combination of flat distributions, shows that the second statement follows from the first: flat distributions are the "worst distributions" for extractors.

5. Conclusion. In this paper we presented an extractor that is reasonably good when some small amount of *classical* information is known about the random source X (technically: $H_{min}(X) \geq n - O(\sqrt{n/\alpha})$), but that fails miserably if even a very small (logarithmic) amount of *quantum* information is known about X. We gave the following applications of this:

1. An exponential quantum-classical separation for one-way communication complexity for a Boolean function (which, in particular, implies near-optimality of Aaronson's classical simulations of quantum one-way protocols).
2. A classically-secure privacy amplification scheme that is insecure against a quantum adversary.
3. A key-expansion scheme that is secure against memory-bounded classical adversaries, but not against memory-bounded quantum adversaries.
4. An exponential quantum-classical separation in the streaming model of computation.

They all can be viewed as examples where *quantum memory is much more powerful than classical*. This contrasts, for instance, with the results about privacy amplification based on two-universal hashing [19, 30], where quantum memory is not significantly more powerful than classical memory.

Acknowledgments. Many thanks to Oded Regev for his ideas on simplifying the two proofs in the conference version of our paper to the current proof. Also many thanks to Renato Renner for explaining why min-entropy determines the key length when doing privacy amplification based on two-universal hashing (both in the classical and in the quantum case), and him, Yevgeniy Dodis, Christian Schaffner, and Barbara Terhal for discussions about the bounded-storage model. Thanks to Shengyu Zhang for bringing the streaming model to our attention, and to Scott Aaronson for prodding us to make explicit the dependence on α in Theorem 1 in light of the results mentioned at the end of section 1.1. Thanks to Guy Kindler and Jaikumar Radhakrishnan for various references that were needed for the earlier proofs in our conference paper (but that are no longer needed). Finally, thanks to the anonymous SICOMP referees for some comments that improved the presentation of the paper.

REFERENCES

[1] S. Aaronson, *The learnability of quantum states*, Proc. Roy. Soc. London Ser. A, 463 (2007), article 2088.

[2] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments*, J. Comput. System Sci., 58 (1999), pp. 137–147.

[3] Y. Aumann, Y. Z. Ding, and M. O. Rabin, *Everlasting security in the bounded storage model*, IEEE Trans. Inform. Theory, 48 (2002), pp. 1668–1680.

[4] Z. Bar-Yossef, T. S. Jayram, and I. Kerenidis, *Exponential separation of quantum and classical one-way communication complexity*, in Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04), 2004, pp. 128–137.

[5] W. Beckner, *Inequalities in Fourier analysis*, Ann. of Math., 102 (1975), pp. 159–182.

[6] C. H. Bennett, G. Brassard, and J.-M. Robert, *Privacy amplification by public discussion*, SIAM J. Comput., 17 (1988), pp. 210–229.

[7] A. Bonami, *Étude des coefficients de Fourier des fonctions de $L^p(G)$*, Ann. Inst. Fourier (Grenoble), 20 (1970), pp. 335–402.

[8] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum vs. classical communication and computation*, in Proceedings of the 30th ACM Symposium on Theory of Computing (STOC'98), 1998, pp. 63–68.

[9] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, *Quantum fingerprinting*, Phys. Rev. Lett., 87 (2001), article 167902.

[10] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[11] S. Dziembowski and U. Maurer, *Optimal randomizer efficiency in the bounded-storage model*, J. Cryptology, 17 (2004), pp. 5–26.

[12] D. Gavinsky, J. Kempe, O. Regev, and R. de Wolf, *Bounded-error quantum state identification and exponential separations in communication complexity*, in Proceedings of the 38th ACM Symposium on Theory of Computing (STOC'06), 2006, pp. 594–603.

[13] D. Gavinsky, J. Kempe, I. Kerenidis, R. Raz, and R. de Wolf, *Exponential separations for one-way quantum communication complexity, with applications to cryptography*, in Proceedings of the 39th ACM Symposium on Theory of Computing (STOC'07), 2007, pp. 516–525.

[14] A. S. Holevo, *Bounds for the quantity of information transmitted by a quantum communication channel*, Problemy Peredači Informacii, 9 (1973), pp. 3–11 (in Russian); English translation in Problems of Information Transmission, 9 (1973), pp. 177–183.

[15] R. Impagliazzo, L. A. Levin, and M. Luby, *Pseudo-random generation from one-way functions*, in Proceedings of the 21st ACM Symposium on Theory of Computing (STOC'89), 1989, pp. 12–24.

[16] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS'88), 1988, pp. 68–80.

[17] H. Klauck, *Lower bounds for quantum communication complexity*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS'01), 2001, pp. 288–297.

[18] R. König and B. M. Terhal, *The bounded storage model in the presence of a quantum adversary*, IEEE Trans. Inform. Theory, 54 (2008), pp. 749–762.

[19] R. König, U. Maurer, and R. Renner, *On the power of quantum memory*, IEEE Trans. Inform. Theory, 51 (2005), pp. 2391–2401.

[20] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[21] F. Le Gall, *Exponential separation of quantum and classical online space complexity*, in Proceedings of the 18th ACM Symposium on Parallel Algorithms and Architectures (SPAA'06), 2006, pp. 67–73.

[22] C.-J. Lu, *Encryption against storage-bounded adversaries from on-line strong extractors*, J. Cryptology, 17 (2004), pp. 27–42.

[23] U. Maurer, *Conditionally-perfect secrecy and a provably-secure randomized cipher*, J. Cryptology, 5 (1992), pp. 53–66.

[24] M. Muthukrishnan, *Data streams: Algorithms and applications*, Found. Trends Theor. Comput. Sci., 1 (2005), pp. 117–236.

[25] I. Newman, *Private vs. common random bits in communication complexity*, Inform. Process. Lett., 39 (1991), pp. 67–71.

[26] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.

[27] R. Raz, *Fourier analysis for probabilistic communication complexity*, Comput. Complexity, 5 (1995), pp. 205–221.

[28] R. Raz, *Exponential separation of quantum and classical communication complexity*, in Proceedings of the 31st ACM Symposium on Theory of Computing (STOC'99), 1999, pp. 358–367.

[29] R. Renner, *Security of Quantum Key Distribution*, Ph.D. thesis, ETH Zürich, 2005.

[30] R. Renner and R. König, *Universally composable privacy amplification against quantum adversaries*, in Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005), Lecture Notes in Comput. Sci. 3378, Springer, Berlin, 2005, pp. 407–425.

[31] R. Shaltiel, *Recent developments in explicit constructions of extractors*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 77 (2002), pp. 67–95.

[32] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.

[33] S. P. Vadhan, *Constructing locally computable extractors and cryptosystems in the bounded-storage model*, J. Cryptology, 17 (2004), pp. 43–77.

[34] R. de Wolf, *Quantum communication and complexity*, Theoret. Comput. Sci., 287 (2002), pp. 337–353.

[35] A. C.-C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS'77), 1977, pp. 222–227.

[36] A. C.-C. Yao, *Some complexity questions related to distributive computing*, in Proceedings of the 11th ACM Symposium on Theory of Computing (STOC'79), 1979, pp. 209–213.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1709–1727

GRAPH DISTANCES IN THE DATA-STREAM MODEL*

JOAN FEIGENBAUM†, SAMPATH KANNAN‡, ANDREW MCGREGOR§, SIDDHARTH SURI¶, AND JIAN ZHANG‖

Abstract. We explore problems related to computing graph distances in the data-stream model. The goal is to design algorithms that can process the edges of a graph in an arbitrary order given only a limited amount of working memory. We are motivated by both the practical challenge of processing massive graphs such as the web graph and the desire for a better theoretical understanding of the data-stream model. In particular, we are interested in the trade-offs between model parameters such as per-data-item processing time, total space, and the number of passes that may be taken over the stream. These trade-offs are more apparent when considering graph problems than they were in previous streaming work that solved problems of a statistical nature. Our results include the following: (1) *Spanner construction:* There exists a single-pass, $\tilde{O}(tn^{1+1/t})$-space, $\tilde{O}(t^2n^{1/t})$-time-per-edge algorithm that constructs a $(2t+1)$-spanner. For $t = \Omega(\log n/\log\log n)$, the algorithm satisfies the semistreaming space restriction of $O(n \operatorname{polylog} n)$ and has per-edge processing time $O(\operatorname{polylog} n)$. This resolves an open question from [J. Feigenbaum et al., *Theoret. Comput. Sci.*, 348 (2005), pp. 207–216]. (2) *Breadth-first-search (BFS) trees:* For any even constant k, we show that any algorithm that computes the first k layers of a BFS tree from a prescribed node with probability at least 2/3 requires either greater than $k/2$ passes or $\tilde{\Omega}(n^{1+1/k})$ space. Since constructing BFS trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data-stream model. (3) *Graph-distance lower bounds:* Any t-approximation of the distance between two nodes requires $\Omega(n^{1+1/t})$ space. We also prove lower bounds for determining the length of the shortest cycle and other graph properties. (4) *Techniques for decreasing per-edge processing:* We discuss two general techniques for speeding up the per-edge computation time of streaming algorithms while increasing the space by only a small factor.

Key words. stream algorithms, graph distances, spanners

AMS subject classifications. 68Q05, 68Q17, 68Q25, 68W20, 68W25

DOI. 10.1137/070683155

1. Introduction.

In recent years, streaming has become an active area of research and an important paradigm for processing massive data sets [4, 23, 27]. Much of the existing work has focused on computing statistics of a stream of data elements,

*Received by the editors February 20, 2007; accepted for publication (in revised form) August 4, 2008; published electronically December 19, 2008. This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under grant N00014-01-1-0795. It first appeared in the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2005 [21].

http://www.siam.org/journals/sicomp/38-5/68315.html

†Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520-8285 (feigenbaum-joan@cs.yale.edu). This author's research was supported in part by ONR grant N00014-01-1-0795 and NSF grant ITR-0331548.

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (kannan@cis.upenn.edu). This author's research was supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.

§Department of Computer Science, University of Massachusetts, 140 Governors Drive, Amherst, MA 01003-9264. This work was done while the author was at the University of Pennsylvania and was supported in part by NSF grant ITR-0205456.

¶Yahoo! Research, 111 West 40th Street, 17th floor, New York, NY 10018 (suri@yahoo-inc.com). This work was done while the author was at the University of Pennsylvania and was supported in part by NIH grant T32HG000046-05.

‖Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803 (jz@lsu.edu). This work was done while the author was at Yale University and was supported by NSF grant ITR-0331548.

e.g., frequency moments [4, 29], ℓ_p distances [23, 28], histograms [24, 26], and quantiles [25]. More recently, there have been extensions of the streaming research to the study of graph problems [6, 22, 27]. Solving graph problems in this model has raised new challenges, because many existing approaches to the design of graph algorithms are rendered useless by the sequential-access limitation and the space limitation of the streaming model.

Massive graphs arise naturally in many real-world scenarios. Two examples are the *call graph* and the *web graph*. In the call graph, nodes represent telephone numbers and edges correspond to calls placed during some time interval. In the web graph, nodes represent web pages, and the edges correspond to hyperlinks between pages. Also, massive graphs appear in structured data mining, where the relationships among the data items in the data set are represented as graphs. When processing these graphs it is often appropriate to use the streaming model. For example, the graph may be revealed by a web crawler or the graph may be stored on external-memory devices, where being able to process the edges in an arbitrary order improves I/O efficiency. Indeed, the authors of [33] argue that one of the major drawbacks of standard graph algorithms, when applied to massive graphs such as the web, is their need to have random access to the edge set.

In general it seems that most graph algorithms need to access the data in a very adaptive fashion. Since we cannot store the entire graph, emulating a traditional algorithm may necessitate an excessive number of passes over the data. There has been some success in estimating quantities that are of a statistical nature, e.g., counting triangles [6, 11, 31] or estimating frequency and entropy moments of the degrees in a multigraph [12, 15]. However, it seemed for a while that more "complicated" computation was not possible in this model. For example, Buchsbaum, Giancarlo, and Westbrook [10] demonstrated the intrinsic difficulty of computing common neighborhoods in the streaming model with small space. One possible way to ameliorate the situation is to consider algorithms that use $\Theta(n \operatorname{polylog} n)$ space, i.e., space roughly proportional to the number of nodes rather than the number of edges. This space restriction was identified as an apparent "sweet-spot" for graph streaming in a survey article by Muthukrishnan [37] and dubbed the *semistreaming* space restriction. This spurred further research on algorithms for graph problems in the streaming model such as distance estimation [19, 22], matchings [22, 36], and connectivity [21, 41], including the work described here. We will provide further discussion of the results on distance estimation in the next section.

A related model is the *semiexternal model*. This was introduced by Abello, Buchsbaum, and Westbrook [1] for computations on massive graphs. In this model, the vertex set can be stored in memory, but the edge set cannot. However, unlike in our model, random access to the edges, although expensive, was allowed. Finally, graph problems have been considered in a model that extends the stream model by allowing the algorithm to write to the stream during each pass [2, 16]. These *annotations* can then be utilized by the algorithm during successive passes. Aggarwal et al. [2] go further and suggest a model in which *sorting passes* are permitted, and the data-stream is sorted according to a key encoded by the annotations.

Designing algorithms for computing graph distances is a very well studied problem, and graph distances are a natural quantity to study when trying to understand properties of massive graphs such as the diameter of the world wide web [3]. We start with a formal definition of the relevant terms.

DEFINITION 1.1 (graph distance, diameter, and girth). *For an undirected, un-*

weighted graph $G = (V, E)$, we define a distance function $d_G : V \times V \to \{0, \ldots, n-1\}$, where $d_G(u, v)$ is the length of the shortest path in G between u and v. The diameter of G is the length of the longest shortest path, i.e.,

$$\mathrm{Diam}(G) = \max_{u,v \in V} d_G(u, v).$$

The girth of G is the length of the shortest cycle in G, i.e.,

$$\mathrm{Girth}(G) = 1 + \min_{(u,v) \in E} d_{G \setminus (u,v)}(u, v).$$

Classic algorithms such as Dijkstra, Bellman–Ford, and Floyd–Warshall are taught widely [14]. Recent research has focused on computing approximate graph distances [5, 7, 17, 40]. Unfortunately, these algorithms seem to be inherently unsuitable for computing distances in the streaming model; an important subroutine of many of the existing algorithms is the construction of breadth-first-search (BFS) trees, and one of our main results is a lower bound on the computational resources required to compute a BFS tree. For example, Thorup and Zwick provide a construction of distance oracles for approximating distances in graphs [40]. Although *all-pairs*-shortest-path distances can be approximated using this oracle, their oracle construction requires the computation of shortest-path trees for certain vertices. Indeed, many constructions of distance oracles have this requirement, i.e., they need to compute *some* distances between certain pairs of vertices in order to build a data structure from which the all-pairs-shortest-path distances can be approximated.

A common method for approximating graph distances is via the construction of *spanners*.

DEFINITION 1.2 (spanners). *A subgraph $H = (V, E')$ is an (α, β)-*spanner *of $G = (V, E)$ if, for any vertices $x, y \in V$,*

$$d_G(x, y) \le d_H(x, y) \le \alpha d_G(x, y) + \beta.$$

When $\beta = 0$, we call the spanner a multiplicative-spanner *and refer to α as the* stretch factor *of the spanner.*

In [22], a simple semistreaming spanner construction is presented. That algorithm constructs a $(\log n)$-spanner in one pass using $O(n \operatorname{polylog} n)$ space. However, this algorithm needs $O(n)$ time to process each edge in the input stream. Such a per-edge processing time is prohibitive, especially in the streaming model when edges may be arriving in quick succession. The work of [19] studies the construction of $(1 + \epsilon, \beta)$ spanners in the streaming model. However, the algorithm of [19] requires multiple passes over the input stream, while our construction needs only one pass.

Notation and terminology. We refer to an event's occurring "with high probability" if the probability of the event is at least $1 - 1/n^{\Omega(1)}$. We denote the set $\{1, \ldots, t\}$ by $[t]$. Let $\mathcal{P}(S)$ denote the power set of S, i.e., $\{S' : S' \subset S\}$.

1.1. Our results. Our results include the following.

1. *Spanner construction:* There exists a single-pass $O(tn^{1+1/t} \log^2 n)$ space, $O(t^2 n^{1/t} \log n)$ time-per-edge algorithm that constructs a $(2t + 1)$-spanner. For $t = \Omega(\log n / \log \log n)$, the algorithm satisfies the semistreaming space restriction of $O(n \operatorname{polylog} n)$ and has per-edge processing time $O(\operatorname{polylog} n)$. The algorithm is presented in section 3. This result resolves an open question from [22].

2. *BFS trees:* For any even constant k, any algorithm that computes the first k layers of a BFS tree from a prescribed node with probability at least $2/3$ requires either greater than $k/2$ passes or $\tilde{\Omega}(n^{1+1/k})$ space. This result is proved in section 4. Since constructing BFS trees is an important subroutine in many traditional graph algorithms, this demonstrates the need for new algorithmic techniques when processing graphs in the data-stream model.
3. *Lower bounds:* In section 5, we present lower bounds for the following problems:
   (a) *Connectivity and other balanced properties:* We show that testing any of a large class of graph properties, which we refer to as *balanced properties*, in one pass requires $\Omega(n)$ space. This class includes properties such as connectivity and bipartiteness. This result provides a formal motivation for the *semistreaming* space restriction, where algorithms are permitted $O(n \operatorname{polylog} n)$ space.
   (b) *Graph distances and graph diameter:* We show that any single-pass algorithm that returns a t-approximation of the graph distance between two given nodes with probability at least $3/4$ requires $\Omega(n^{1+1/t})$ bits of space. Furthermore, this bound also applies to estimating the diameter of the graph. Therefore, approximating a distance using the above spanner construction is only about a factor of 2 from optimal in terms of the approximation factor achievable for a given space restriction.
   (c) *Girth:* Any p-pass algorithm that ascertains whether the length of the shortest cycle is longer than g requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$ bits of space.
4. *Techniques for decreasing per-edge processing:* In section 6, we present a method for local amortization of per-data-item complexity. We also present a technique for adapting existing partially dynamic graph algorithms to the semistreaming model.

The above results indicate various trade-offs between model parameters and accuracy. These include the smooth trade-off between the space a single-pass algorithm is permitted and the accuracy achievable when estimating graph distances. For multiple-pass algorithms, a smooth trade-off between passes and space is evident when trying to compute the girth of a graph. This trade-off is, in a sense, fundamental as it indicates that the only way to get away with using half the amount of space is essentially to make half as much progress in each pass. The trade-off between space and passes when computing BFS trees indicates that, as we restrict the space, no algorithm can do much better than emulating a trivial traditional graph algorithm and will consequently require an excessive number of passes.

Recent developments. Since the preliminary version of this paper appeared, improved spanner construction algorithms were presented by Baswana [8] and Elkin [18]. Extensions of ideas in section 6 have been developed by Zelke [41, 42].

2. Preliminaries. In this section, we give a formal definition of a graph stream.

DEFINITION 2.1 (graph stream). *For a data-stream $A = \langle a_1, a_2, \ldots, a_m \rangle$, where $a_j \in [n] \times [n]$, we define a graph G on n vertices $V = \{v_1, \ldots, v_n\}$ with edges $E = \{(v_i, v_k) : a_j = (i, k)$ for some $j \in [m]\}$.*

We usually assume that each a_j is distinct, but this assumption is often not necessary. When the data items are not distinct, the model can naturally be extended to consider multigraphs; i.e., an edge (v_i, v_k) has multiplicity equal to $|\{j : a_j = (i, k)\}|$. Similarly, we consider undirected graphs, but the definition can be generalized

to define digraph streams. Sometimes we will consider weighted graphs, and in this case $a_j \in [n] \times [n] \times \mathbb{N}$, where the third component of the data item indicates a weight associated with the edge. Note that some authors have also considered a special case of the model, the adjacency-list model, in which all incident edges are grouped together in the stream [6]. We will be interested in the fully general model.

3. Spanners. In this section, we present a single-pass streaming algorithm that constructs a $(2t+1)$-spanner for an unweighted, undirected graph. The algorithm uses some of the ideas from [7] and adapts them for use in the data-stream model. We then extend this algorithm to construct $(1+\epsilon)(2t+1)$-spanners for weighted, undirected graphs using a geometric grouping technique.

Overview of the algorithm. Intuitively, we want to cover each "dense subgraph" of the graph by a tree of small depth, i.e., one of depth $O(\log n)$, rooted at some node. If there are many edges between two such dense subgraphs, only one representative edge needs to be remembered. Edges between vertices that are not part of such dense subgraphs also need to be remembered, but we will argue that there are not too many of them. The construction requires a delicate balance between trying to include as many nodes as possible in a small number of dense subgraphs and ensuring that the depth of the spanning tree covering each dense subgraph is $O(\log n)$.

Our clusters are similar to those used in [5, 7, 17]. However, the constructions of the clusters in [5, 7, 17] all employ an approach similar to BFS; i.e., the clusters are constructed layer by layer. Such a layer-by-layer process is important to ensure that the clusters constructed in [5,7,17] have small diameters. In the streaming model, this would necessitate multiple passes over the input stream. Our labeling scheme employs a different strategy to control the clusters' diameters, thus bypassing the BFS.

In more detail, in the case in which we are constructing $O(\log n)$-spanners, we consider $\log n/2$ levels. Each node is present at the bottom level and is "promoted" to each successively higher level with probability $1/2$. Whenever a node is present at level i, a cluster at level i is dynamically grown around it as we process the edges. The expected number of clusters at level $i+1$ is half of the expected number of clusters at level i. At the top level, the expected number of clusters is $\sqrt{n}$. As the algorithm goes through the input stream of edges, a node in a lower-level cluster may join a higher-level cluster. The constructed spanner consists of three types of edges: (1) a small-diameter spanning tree for each cluster, (2) one edge between each pair of top-level clusters whose vertex sets have at least one edge between them, and (3) a small number of "other" edges that do not fit into either of these two categories and are necessary to preserve the spanner property.

The above ideas are applicable mutatis mutandis when a t-spanner is sought for other values of t. The description below is for arbitrary t.

The algorithm. A label l used in our construction is a positive integer. Given two parameters n and t, the set of labels L used by our algorithm is generated in the following way. Initially, we have labels $[n]$. We denote by L^0 this set of labels and call them the *level*-0 labels. Independently, and with probability $n^{-1/t}$, each label $l \in L^0$ will be put into a set S^0 and marked as *selected*. From each label l in S^0, we generate a new label $l' = l + n$. We denote by L^1 the set of newly generated labels and call them the *level*-1 labels. We then apply the above selection and new-label-generation procedure on L^1 to get the set of level-2 labels L^2. We continue this until the level-$\lfloor t/2 \rfloor$ labels $L^{\lfloor t/2 \rfloor}$ are generated. If a level-$(i+1)$ label l is generated from a level-i label l', we call l the *successor* of l' and denote it by $\mathrm{Succ}(l') = l$. The set of labels we will use in our algorithm is the union of labels of levels $0, 1, 2, \ldots, \lfloor t/2 \rfloor$, i.e.,

$L = \bigcup_0^{\lfloor t/2 \rfloor} L^i$. Note that L can be generated before the algorithm sees the edges in the stream. But, in order to generate the labels in L, except in the case $t = \Theta(\log n)$, the algorithm needs to know n, the number of vertices in the graph, before seeing the edges in the input stream. For $t = \Theta(\log n)$, a simple modification of the above method can be used to generate L without knowing n, because the probability for a label to be selected is constant.

At first glance, it might appear that our labeling scheme resembles the sequences of sets initially constructed by the algorithm of Thorup and Zwick [40]. In our construction, the generation of the clusters of vertices depends on both the labels and the edge set. Thorup and Zwick, however, generate their sequence of sets independently of the edges. But, this is just the first step in their algorithm. Subsequently, their algorithm computes the exact distance between a fixed vertex and each of the sets of vertices in the sequence. Computing the exact distances between a pair of vertices is difficult in the streaming model. Our algorithm takes a very different approach from [40] to avoid this difficulty.

While going through the stream, our algorithm will label each vertex with labels chosen from L. The algorithm may label a vertex v with multiple labels; however, v will be labeled by at most one label from L^i for $i \in [\lfloor t/2 \rfloor]$. Moreover, if v is labeled by a label l, and l is selected, the algorithm will also label v with the label $\mathrm{Succ}(l)$.

Denote by l^i a label of level i, i.e., $l^i \in L^i$. Let $L(v) = \{l^0, l^{k_1}, l^{k_2}, \ldots, l^{k_j}\}$ be the collection of labels that has been assigned to the vertex v, where $0 < k_1 < \cdots < k_j \leq \lfloor t/2 \rfloor$. Let

$$\mathrm{Height}(v) = \max\{j | l^j \in L(v)\}$$

and $\mathrm{Top}(v) = l^k \in L(v)$ s.t. $k = \mathrm{Height}(v)$. Let $C(l)$ be the collection of vertices that are labeled with the label l.

The sets $L(v)$ and $C(l)$ will grow while the algorithm goes through the stream and labels the vertices. For each $C(l)$, our algorithm stores a tree, $\mathrm{Tree}(l)$, on the vertices of $C(l)$ and the tree is rooted on the first vertex that gets labeled by l. We say an edge (u, v) connects $C(l)$ and $C(l')$ if u is labeled with l and v is labeled with l'. For some pairs of labels $l, l' \in L^{\lfloor t/2 \rfloor}$, our algorithm will store edges that connect $C(l)$ and $C(l')$. We denote by H the set of such edges stored by our algorithm. In addition, for each vertex v, we denote by $M(v)$ the other edges incident to v that are stored by our algorithm. Intuitively, the subgraph induced by $\cup_{v \in V} M(v)$ is the sparse part of the graph G. The spanner constructed by the algorithm is the union of the rooted trees for all the labels, $M(v)$ for all the vertices, and the set H. The detailed algorithm is given in Figure 3.1.

Analysis. We start with two preliminary lemmas showing that, with high probability, the spanner construction requires only a small amount of working space. Note that randomness is introduced in the generation of the labels in Algorithm 1; that is the reason that the bounds stated in both lemmas are true with high probability.

LEMMA 3.1. *With high probability, for all $v \in V$, $|M(v)| = O(tn^{1/t} \log n)$.*

Proof. Let $M^{(i)}(v) \subseteq M(v)$ be the set of edges added to $M(v)$ during the period when $\mathrm{Height}(v) = i$. Let $L(M(v)) = \cup_{(u,v) \in M(v)} L(u)$ be the set of labels that have been assigned to the vertices in $M(v)$. An edge (u, v) is added to $M(v)$ only in step 2(b)ii. Note that, in this case, $\lfloor \frac{t}{2} \rfloor \geq \mathrm{Height}(u) \geq \mathrm{Height}(v)$. Hence, the set $L_v(u)$ is not empty. Also, by the condition in step 2(b)ii, an edge (u, v) is added only when none of the labels in $L_v(u)$ appears in $L(M(v))$. Thus, if we add the edge in this step, we will introduce/add the label(s) in $L_v(u)$ to $L(M(v))$. Because $L_v(u)$

Algorithm 1 (efficient one pass spanner construction).
The input to the algorithm is an unweighted, undirected graph $G = (V, E)$, presented as a stream of edges, and two positive integer parameters n and t.

1. Generate the set L of labels as described. $\forall\ v_i \in V$, label vertex v_i with label $i \in L^0$. If i is selected, label v_i with $\mathrm{Succ}(i)$. Continue this until we encounter a label that is not selected. Set $M(v_i) \leftarrow \emptyset$ and $H \leftarrow \emptyset$.
2. Upon seeing an edge (u, v) in the stream, if $L(v) \cap L(u) \neq \emptyset$, drop the edge. Otherwise, consider the following cases:
   (a) If $\mathrm{Height}(v) = \mathrm{Height}(u) = \lfloor t/2 \rfloor$, and there is no edge in H that connects $C(\mathrm{Top}(v))$ and $C(\mathrm{Top}(u))$, set $H \leftarrow H \cup \{(u, v)\}$.
   (b) Otherwise, assume, without loss of generality, that $\lfloor t/2 \rfloor \geq \mathrm{Height}(u) \geq \mathrm{Height}(v)$. Consider the collection of labels
   $$L_v(u) = \{l^{k_1}, l^{k_2}, \ldots, l^{\mathrm{Height}(u)}\} \subseteq L(u),$$
   where $\mathrm{Height}(v) \leq k_1 \leq k_2 \leq \cdots \leq \mathrm{Height}(u)$. Let $l \in L_v(u)$ be the selected label whose level is the lowest among the selected labels in $L_v(u)$.
      i. If such a label l exists, label the vertex v with the successor $l' = \mathrm{Succ}(l)$ of l, i.e.,
      $$L(v) \leftarrow L(v) \cup \{l'\}.$$
      Incorporate the edge in the rooted tree $\mathrm{Tree}(l')$. If l' is selected, label v with $l'' = \mathrm{Succ}(l')$ and incorporate the edge in the tree $\mathrm{Tree}(l'')$. Continue until we see a label that is not marked as selected. (Note that $\mathrm{Height}(v)$ is increased by this process.)
      ii. If no such label l exists and there is no edge (u', v) in $M(v)$ such that u, u' are labeled with the same label $l \in L_v(u)$, add (u, v) to $M(v)$.
3. After seeing all the edges in the stream, output the union of the rooted trees for all the labels, $M(v)$ for all the vertices, and the set H as the spanner.

FIG. 3.1. *An efficient, one-pass algorithm for computing sparse spanners. See the descriptions before Lemma* 3.1 *for the definitions of* H, $M(v)$, $L(v)$, $C(l)$, $\mathrm{Tree}(l)$, $\mathrm{Succ}(l)$, $\mathrm{Top}(v)$, *and* $\mathrm{Height}(v)$.

is nonempty, it adds at least one new label to $L(M(v))$. This is the case for every edge in $M^{(i)}(v)$. Let B be the set of distinct labels that $M^{(i)}(v)$ adds to $L(M(v))$. Furthermore, because each edge in $M^{(i)}(v)$ adds at least one new label, the size of B is at least $|M^{(i)}(v)|$. Note that the labels in B are not marked as selected. Otherwise, the algorithm would have taken step 2(b)i instead of step 2(b)ii. Hence, the size of B satisfies

$$\Pr(|B| = k) \leq (1 - 1/n^{1/t})^k.$$

Thus, with high probability, $|M^{(i)}(v)| = O(n^{1/t} \log n)$. Because i can take only $O(t)$ values, with high probability,

$$|M(v)| = \sum_i |M^{(i)}(v)| = O(tn^{1/t} \log n). \qquad \square$$

LEMMA 3.2. *With high probability, the algorithm stores* $O(tn^{1+1/t}\log n)$ *edges.*

Proof. The algorithm stores edges in the set H, in the rooted trees for each cluster $C(l)$, and in the sets $M(v)$, for all $v \in V$. By the Chernoff bound and the union bound, with high probability the number of clusters at level $t/2$ is $O(\sqrt{n})$, and the size of the set H is $O(n)$.

For each label l, the algorithm stores a rooted tree for the set of vertices $C(l)$. The rooted tree is formed by the set of edges added when the vertices in $C(l)$ get labeled by the label l. This happens in step 2(b)i. In this step, we add an edge only at the time when a vertex becomes a member of $C(l)$. Therefore, the set of edges forms a tree on $C(l)$. We call this tree the rooted tree of $C(l)$. Note that two vertices u and v in $C(l)$ may share some other label l' of a different level. In this case, they both also belong to $C(l')$. The tree of $C(l')$ may have a path connecting u and v. Hence the subgraph of the spanner induced by the vertices in $C(l)$ is not necessarily a tree. When we say "the rooted tree of $C(l)$," we refer only to the edges added when the vertices in $C(l)$ get labeled by l.

Note that, for each level $i \in [\lfloor t/2 \rfloor]$, a vertex is labeled with at most one label in L^i. Hence, $\sum_{l \in L^i} |C(l)| \leq |V|$. Thus, the number of edges summed over the rooted trees for the labels at level i is $O(n)$, and the total number of edges in all the rooted trees is $O(tn)$. Finally, by Lemma 3.1, with high probability, $|M(v)| = O(tn^{1/t}\log n)$. By the union bound, with high probability, $\sum_{v \in V} |M(v)| = O(tn^{1+1/t}\log n)$. ∎

THEOREM 3.3. *Let G be an unweighted graph. There exists a single-pass $O(tn^{1+1/t}\log^2 n)$ space algorithm that constructs a $(2t+1)$-spanner of G with high probability and processes each edge in $O(t^2 n^{1/t}\log n)$ time.*

Proof. Consider Algorithm 1 in Figure 3.1. At the beginning of the algorithm, for all the labels $l \in L^0$, $C(l)$ is a singleton set, and the depth of the rooted tree for $C(l)$ is zero. We now bound, for label l^i, where $i > 0$, the depth of the rooted tree T^i on the vertices in $C(l^i)$. A tree grows when an edge (u, v) is incorporated into the tree in step 2(b)i. In this case, l^i is a successor of some label l^{i-1} of level $i-1$. Assume that the depth $d_{T^i}(v)$ of the vertex v in the tree is one more than the depth $d_{T^i}(u)$ of the vertex u. Then $u \in C(l^{i-1})$, and the depth $d_{T^{i-1}}(u)$ of u in the rooted tree T^{i-1} of $C(l^{i-1})$ is the same as $d_{T^i}(u)$. Hence, $d_{T^i}(v) = d_{T^{i-1}}(u) + 1$, where T^i is a tree of level i, and T^{i-1} is a tree of level $i-1$. Given that $d_{T^0}(x) = 0$ for all $x \in V$, the depth of a rooted tree for $C(l)$, where l is a label of level i, is at most i.

We proceed to show that, for any edge that the algorithm does not store, there is a path of length at most $2t+1$ that connects the two endpoints of the edge. The algorithm ignores three types of edges. First, if $L(u) \cap L(v) \neq \emptyset$, the edge (u, v) is ignored. In this case, let l be one of the label(s) in $L(u) \cap L(v)$. The nodes u and v are both on the rooted tree for $C(l)$; hence, there is a path of length at most t connecting u and v. Second, (u, v) will be ignored if $\text{Height}(v) = \text{Height}(u) = \lfloor t/2 \rfloor$, and there is already an edge connecting $C(\text{Top}(u))$ and $C(\text{Top}(v))$. In this case, the path connecting u and v has length at most $2t+1$. Finally, in step 2(b)ii, (u, v) will be ignored if there is already another edge in $M(v)$ that connects v to some $u' \in C(l)$, where $l \in L(u)$. Note that u and u' are both on the rooted tree of $C(l)$. Hence, there is a path of length at most $t+1$ connecting u and v.

Hence, the stretch factor of the spanner constructed by Algorithm 1 is $2t+1$. By Lemma 3.2, with high probability, the algorithm stores $O(tn^{1+1/t}\log n)$ edges and requires $O(tn^{1+1/t}\log^2 n)$ bits of space. Also note that the bottleneck in the processing of each edge lies in step 2(b)ii, where, for each label in $L_v(u)$, we need to examine the whole set of $M(v)$. This takes $O(t^2 n^{1/t}\log n)$ time. ∎

Extensions. Once the spanner is constructed, all-pairs-shortest-distances of the graph can be computed from the spanner. This computation does not need to access the input stream and thus can be viewed as postprocessing. We also note that the above algorithm can be used to construct a spanner of a weighted graph $G = (V, E)$ using a geometric grouping technique [13,22]. Namely, we can round each edge weight ω' up to $\min\{\omega(1+\epsilon)^i : i \in \mathbb{Z}, \omega(1+\epsilon)^i \geq \omega'\}$, where ω is the weight of the first edge, and $\epsilon > 0$ is a user-defined accuracy parameter. Let $G^i = (V, E^i)$ be the graph formed from G by removing all edges not of weight $\omega(1+\epsilon)^i$. For each G^i, we construct a spanner in parallel and take the union of these spanners. This leads to the following theorem.

THEOREM 3.4. *Let G be a weighted graph and W be the ratio between the maximum and minimum weights. There exists a single-pass $O(\epsilon^{-1}tn^{1+1/t}\log W \log^2 n)$ space algorithm that constructs a $(1+\epsilon)(2t+1)$-spanner of G with high probability and processes each edge in $O(t^2 n^{1/t}\log n)$ time.*

In the case where $t = \log n/\log\log n$, Algorithm 1 computes a $(2\log n/\log\log n + 1)$-spanner in one pass using $O(n\log^4 n)$ bits of space and processing each edge in $O(\log^4 n)$ time. This answers an open question we posed in [22]. Finally, note that constructing a $(2t+1)$-spanner gives a $(2t+1)$-approximation for the diameter and, indirectly, a $(2t+2)/3$-approximation of the girth. The diameter result is immediate. For the girth approximation, note that, if the constructed spanner is a strict subgraph of G, then the girth of G must have been between 3 and $2t+2$.

4. BFS trees lower bound. In this section, we prove a lower bound on the number of passes required to construct the first l layers of a BFS tree in the streaming model. The result is proved using a reduction from the communication-complexity problem "multivalued pointer chasing." This is a natural generalization of the pointer-chasing problem considered by Nisan and Wigderson [38].

Overview of proof. Nisan and Wigderson [38] considered the problem in which Alice and Bob have functions $f_A : [m] \to [m]$ and $f_B : [m] \to [m]$, respectively. The k-round pointer-chasing problem is to output the result of starting from 1 and alternatively applying f_A and f_B a total of k times, starting with f_A. Nisan and Wigderson proved that, if Bob speaks first, the communication complexity of any k-round communication protocol to solve this problem is $\Omega(m/k^2 - k\log m)$. Jain, Radhakrishnan, and Sen [30] gave a direct sum extension showing that, if there are d pairs of functions and the goal is to perform k-round pointer chasing as above on each pair, the communication-complexity lower bound is approximately d times the bound of [38]. More precisely, they showed a lower bound of $\Omega(dm/k^3 - dk\log m - 2d)$ for the problem.

We show how the lower bound of [30] implies a lower bound on the communication complexity of pointer chasing with "d-valued functions," i.e., functions that map $i \in [m]$ to a subset of $[m]$ of size at most d. If f_A and f_B are such functions, then the result of pointer chasing starting from 1 produces a set of size at most d^k. The key difference between this problem and the problem of [30] is that in the latter, one is concerned only with chasing "like" pointers. That is, if one gets to an element j using the function $f_{A,i}$, one can continue only with $f_{B,i}$. (We will present an example after the formal definition of the appropriate terms.) Nevertheless, we give a reduction that shows that the two problems have fairly similar communication complexity.

Finally, we create a an l-layered graph in which alternate layers have edges corresponding to d-valued functions f_A and f_B. In order to construct the BFS tree, we must solve the l-round, d-valued pointer-chasing problem and then apply the afore-

mentioned lower bound. This will lead to the following theorem.

THEOREM 4.1 (BFS lower bound). *For any constant k, any algorithm that computes the first k layers of a BFS tree with probability at least $2/3$ requires either $k/2$ passes or $\Omega(n^{1+1/k}/(\log n)^{1/k})$ space.*

We now present the above argument formally. Let F_d be the set of functions $f : \mathcal{P}([m]) \to \mathcal{P}([m])$ such that

$$\forall i \in [m],\ |f(i)| \le d \quad \text{and} \quad \forall A \subset [m],\ f(A) = \bigcup_{i \in A} f(i).$$

Throughout the rest of this section, we abuse notation and denote the singleton set $\{i\}$ by i when it appears as the input or the output of a function $f \in F_d$.

DEFINITION 4.2. *Define $g^{k,d} : F_d \times F_d \to \mathcal{P}([m])$ inductively as*

$$g^{0,d}(f_A, f_B) = \{1\} \quad \textit{and} \quad g^{i,d}(f_A, f_B) = \begin{cases} f_A(g^{i-1,d}(f_A, f_B)) & \textit{if } i \textit{ odd,} \\ f_B(g^{i-1,d}(f_A, f_B)) & \textit{if } i \textit{ even.} \end{cases}$$

Define $h^{k,d}$ as the d-fold direct sum of $g^{k,1}$, i.e.,

$$h^{k,d}(\langle f_{A,1}, \ldots, f_{A,d}\rangle, \langle f_{B,1}, \ldots, f_{B,d}\rangle) = \langle g^{k,1}(f_{A,1}, f_{B,1}), \ldots, g^{k,1}(f_{A,d}, f_{B,d})\rangle.$$

EXAMPLE 4.3. *Consider $f_{A,1}, f_{A,2}, f_{B,1}, f_{B,2} \in F_1$, where*

$$\begin{array}{llll} f_{A,1}: 1 \to 1 & f_{A,2}: 1 \to 2 & f_{B,1}: 1 \to 1 & f_{B,2}: 1 \to 3 \\ \quad 2 \to 2 & \quad 2 \to 3 & \quad 2 \to 2 & \quad 2 \to 1 \\ \quad 3 \to 3, & \quad 3 \to 1, & \quad 3 \to 3, & \quad 3 \to 2. \end{array}$$

Let $f_A, f_B \in F_2$ be defined by $f_A(j) := f_{A,1}(j) \cup f_{A,2}(j)$ and $f_B(j) := f_{B,1}(j) \cup f_{B,2}(j)$. Then

$$h^{2,2}(\langle f_{A,1}, f_{A,2}\rangle, \langle f_{B,1}, f_{B,2}\rangle) = \langle 1, 1\rangle, \textit{ whereas } g^{2,2}(f_A, f_B) = \{1, 2, 3\}.$$

Let Alice have function f_A and Bob have function f_B. Let $R^r_\delta(g_{d,k})$ be the r-round randomized communication complexity of $g_{d,k}$ where Bob speaks first, i.e., the number of bits sent in the worst case (over all inputs and random coin tosses) by the best r-round protocol Π in which, with probability at least $1-\delta$, both Alice and Bob learn $g_{d,k}$. The following theorem for $h^{k,d}$ was proved in [30] using an information-theoretic argument in combination with a result by Nisan and Wigderson [38].

THEOREM 4.4 (see Jain, Radhakrishnan, and Sen [30]). $R^k_{1/3}(h^{k,d}) = \Omega(dmk^{-3} - dk \log m - 2d)$.

We now use the above result to prove a bound on the communication complexity of $g^{k,d}$.

THEOREM 4.5. $R^k_{1/3}(g^{k,d}) = \Omega(dm/(k+1)^3 - d(k+1)\log m - 2d - 6d^{k+1} \lg m)$ *if k is even.*

Proof. The proof uses a reduction from $h^{k,d}$. Let $(\langle f_{A,1}, \ldots, f_{A,d}\rangle, \langle f_{B,1}, \ldots, f_{B,d}\rangle)$ be an instance of $h^{k,d}$. Define f^*_A and f^*_B as follows:

$$f^*_A(j) := \{f_{A,i}(j) : i \in [d]\} \text{ and } f^*_B(j) := \{f_{B,i}(j) : i \in [d]\}.$$

Assume that there exists a k-round protocol Π for $g^{k,d}$ that fails with probability at most $1/3$ and communicates $o(dm/(k+1)^3 - d(k+1)\log m - 2d - 12d^{k+1} \lg m - km)$ bits in the worst case. We will show how to transform Π into a $(k+1)$-round protocol

Π' for $h^{k+1,d}$ that fails with probability at most $1/3$ and communicates $o(dm/(k+1)^3 - d(k+1)\log m - 2d)$ bits in the worst case. This contradicts Theorem 4.4 and hence shows that there is no such protocol Π.

Let Π' be a protocol that simulates Π and, in addition, on the jth message $(1 < j \le k+1)$, sends the following set of triples:[1]

$$T_{j-1} = \{\langle a, b, f_{C,a}(b)\rangle : b \in g^{j-2,d}(f_A^*, f_B^*)\}, \quad \text{where} \quad C = \begin{cases} A & \text{if } j \text{ is even,} \\ B & \text{if } j \text{ is odd.} \end{cases}$$

Π is shown to be correct by an inductive argument and requires at most $3d^{j-1}\log m$ additional bits of communication per message, because $b \in g^{j-2,d}(f_A^*, f_B^*)$ and $|g^{j-2,d}(f_A^*, f_B^*)| \le d^{j-2}$, $a \in [d]$, and each $\langle a, b, f_{C,a}(b)\rangle$ can be encoded with at most $3\log m$ bits. However, if Π is successful, then the player who sends the kth message (which is Alice by assumption that k is even and Bob speaks first) of Π also knows $g^{k,d}(f_A^*, f_B^*)$. Hence, she can also send

$$T_{k+1} = \{\langle a, b, f_{A,a}(b)\rangle : b \in g^{k,d}(f_A^*, f_B^*)\}.$$

Hence, after $(k+1)$ rounds, $\bigcup_{i\in[k+1]} T_i$ is known to both parties with probability at least $2/3$ and can be used to deduce $g^{k+1,d}$. The total amount of extra communication required to transmit $\bigcup_{i\in[k+1]} T_i$ is $\sum_{i\in[k+1]} 3d^i \log m \le 6d^{k+1}\log m$. Hence Π' communicates $o(dm/(k+1)^3 - d(k+1)\log m - 2d)$ bits in the worst case. ∎

We are now in a position to prove Theorem 4.1.

Proof. The proof is a reduction from d-valued pointer chasing. Let $m = n/(k+1)$, and let $d = c(m/\log m)^{1/k}$ for some small constant c. Then, since k is constant by Theorem 4.5, $R^k_{1/3}(g^{k,d}) = \Omega(n^{1+1/k}/(\log n)^{1/k})$.

Consider an instance (f_A, f_B) of $g^{k,d}$. The graph described by the stream is on the following set V of $n = (k+1)m$ nodes:

$$V = \bigcup_{0\le i\le k} \{v_1^i, \ldots, v_m^i\}.$$

For $i \in [k]$, we define a set of edges $E(i)$ between $\{v_1^{i-1}, \ldots, v_m^{i-1}\}$ and $\{v_1^i, \ldots, v_m^i\}$ in the following way:

$$E(i) = \begin{cases} \{(v_j^{i-1}, v_\ell^i) : \ell \in f_A(j)\} & \text{if } i \text{ is odd,} \\ \{(v_j^{i-1}, v_\ell^i) : \ell \in f_B(j)\} & \text{if } i \text{ is even.} \end{cases}$$

Suppose there exists an algorithm $\mathcal{A}$ that computes the first k layers of the BFS tree from v_1^1 in p passes using memory M. Let L_r be set of nodes that are at distance exactly r from v_1^1. Note that, for all $r \in [k]$,

$$g^{r,d} = L_r \cap \{v_1^r, \ldots, v_m^r\}.$$

By simulating $\mathcal{A}$ on a stream starting with $\bigcup_{0\le i\le k:\text{even}} E(i)$ and concluding with $\bigcup_{i\in[k]:\text{odd}} E(i)$ in the natural way, we deduce that there exists a $(2p)$-round communication protocol for $g^{k,d}$ that uses only $2pM$ communication. (Note that $2p$ rounds of communication rather than $(2p-1)$ rounds are required, because we required both parties to learn $g^{k,d}$.) Hence, either $2p > k$ or $M = \Omega(n^{1+1/k})$. ∎

[1] Note that on the $(k+1)$th message there is no message of Π to simulate and only T_k is transmitted.

5. Other lower bounds. In this section, we present lower bounds on the space required to estimate graph distances, test whether a graph is connected, and compute the girth of a graph. Our lower bounds are reductions from problems in communication complexity. In the SET-DISJOINTNESS problem, Alice has a length-n binary string $x \in \{0,1\}^n$, and Bob has a length-n binary string $y \in \{0,1\}^n$, where $\sum_{i\in[n]} x_i = \sum_{i\in[n]} y_i = \lfloor n/4 \rfloor$. If Bob is to compute

$$\text{SET-DISJOINTNESS}(x,y) = \begin{cases} 1 & \text{if } x \cdot y = 0, \\ 0 & \text{if } x \cdot y \geq 1 \end{cases}$$

(where $x \cdot y = \sum_{i\in[n]} x_i y_i$) with probability at least 3/4, then it is known that $\Omega(n)$ bits must be communicated between Alice and Bob [32, 39]. In the INDEX problem, Alice has $x \in \{0,1\}^n$, and Bob has $j \in [n]$. If Bob is to compute $\text{INDEX}(x,j) = x_j$ with probability at least 3/4 after a single message from Alice, then it is known that this message must contain $\Omega(n)$ bits (e.g., [34]). To relate our graph-stream problems to these communication problems, we use reductions based upon results from random-graph theory and extremal combinatorics.

5.1. Connectivity and balanced properties. Our first result shows that a large class of problems, including connectivity, cannot be solved by single-pass streaming algorithms in small space. Specifically, we identify a general type of graph property[2] and show that testing any such graph property requires $\Omega(n)$ space.

DEFINITION 5.1 (balanced properties). *We say a graph property $\mathcal{P}$ is* balanced *if there exists a constant $c > 0$ such that, for all sufficiently large n, there exists a graph $G = (V,E)$ with $|V| = n$ and $u \in V$ such that*

$$\min\{|\{v : (V, E \cup \{(u,v)\}) \text{ has } \mathcal{P}\}|, |\{v : (V, E \cup \{(u,v)\}) \text{ has } \neg\mathcal{P}\}| \} \geq cn.$$

In other words, there are $\Omega(n)$ vertices v such that $(V, E \cup \{(u,v)\})$ has $\mathcal{P}$ and $\Omega(n)$ vertices v such that $(V, E \cup \{(u,v)\})$ does not have $\mathcal{P}$.

Many interesting properties are balanced, including connectivity, bipartiteness, and whether there exists a vertex of a certain degree.

THEOREM 5.2. *Testing for any balanced graph property $\mathcal{P}$ with probability 3/4 in a single pass requires $\Omega(n)$ space.*

Proof. Let c be a constant, $G = (V,E)$ be a graph on n vertices, and $u \in V$ be a vertex satisfying the relevant conditions. The proof is by a reduction to the communication complexity of INDEX. Let $(x,j) \in \{0,1\}^{cn} \times [cn]$ be an instance of INDEX. Let $G(x)$ be a relabeling of the vertices of G such that $u = v_n$ and, for $i \in [cn]$, $(V, E \cup \{(v_n, v_i)\})$ has $\mathcal{P}$ if and only if $x_i = 1$. Such a relabeling is possible, because $\mathcal{P}$ does not depend on the labeling of the vertices. Let $e(j) = (v_j, v_n)$. Hence the graph determined by the edges of $G(x)$ and $e(j)$ has $\mathcal{P}$ if and only if $x_j = 1$. Therefore, any single-pass algorithm for testing $\mathcal{P}$ using M bits of work space gives rise to a one-message protocol for solving INDEX, and this implies that $M = \Omega(n)$. ∎

For some balanced graph properties, the above theorem can be generalized. For example, it is possible to show that any p-pass algorithm that determines whether a graph is connected requires $\Omega(np^{-1})$ bits of space [27].

[2] A *graph property* is a boolean function whose inputs are the elements of the adjacency matrix of the graph but whose output is independent of the labeling of the nodes of the graph.

5.2. Graph distances and graph diameter. If we are interested only in estimating the distance between two nodes u and v, it may appear that constructing a graph spanner that gives no special attention to u or v, but rather approximates *all* distances, is an unnecessarily crude approach. In this section, however, we show that the spanner-construction approach yields an approximation at most a factor 2 from optimal. Our result is a generalization of one in [22] that applied to the semistreaming case. Integral to our proof is the notion of a *k-critical edge*.

DEFINITION 5.3. *In a graph $G = (V, E)$, an edge $e = (u, v) \in E$ is* k-critical *if* $d_{G\setminus(u,v)}(u, v) \geq k$.

In Lemma 5.4, we show the existence of a graph G with a large subset of edges E' such that each edge in E' is k-critical, but the removal of all edges in E' leaves a graph with relatively small diameter. The proof uses a probabilistic method.

LEMMA 5.4. *For sufficiently large n and $3 \leq k = o(\log n/\log\log n)$, there exists a set E of edges partitioned into two disjoint sets E_1 and E_2 on a set of n nodes V such that*

1. $|E_2| = \lceil n^{1+1/k}/144 \rceil$,
2. *every edge in E_2 is k-critical for $G = (V, E)$, and*
3. $\mathrm{Diam}(G_1) \leq k + 1$, *where $G_1 = (V, E_1)$.*

Proof. Let $\gamma = 1/k$. Consider choosing a random graph $G' = (V, E')$ on n nodes where each edge is (independently) present with probability $p = 1/(2n^{1-\gamma})$. This is commonly denoted as $G' \sim \mathcal{G}_{n,p}$. We will then construct $G_1 = (V, E_1)$ by deleting each edge in G' with probability $1/2$. We will show that, with nonzero probability, the sets E_1 and $E_2 = \{e \in E' \setminus E_1 : e \text{ is } k\text{-critical for } G'\}$ satisfy the three required properties.

The second property is satisfied by construction. It follows from the fact that, if an edge is k-critical in a graph G, then it is also k-critical in any subgraph of G. We now argue that the third property is satisfied with probability at least $99/100$. First note that the process that generates G_1 is identical to picking $G_1 \sim \mathcal{G}_{n,p/2}$. It can be shown that, with high probability, the diameter of such a graph is less than $1/\gamma + 1$ for sufficiently large n [9, Corollary 10.12].

We now show that the first property is satisfied with probability at least $8/100$. Applying the Chernoff bound and the union bound proves that, with probability at least $99/100$, the degree of every vertex in G' is between $n^\gamma/4$ and $3n^\gamma/4$.

Now consider choosing a random graph and a random edge in that graph simultaneously, i.e., $G' = (V, E') \sim \mathcal{G}_{n,p}$ and an edge $(u, v) \in_R E'$. We prove a lower bound on the probability that (u, v) is k-critical in G'. Let $\Gamma_i(v) = \{w \in V : d_{G'\setminus(u,v)}(v, w) \leq i\}$. For sufficiently large n, conditioned on the event that the maximum degree is at most $3n^\gamma/4$,

$$|\Gamma_k(v)| \leq \sum_{0\leq i\leq k} (3n/4)^{i\gamma} \leq 1.01(3n/4)^{k\gamma} \leq 4(n-1)/5.$$

As G' varies over all possible graphs, by symmetry, each vertex is equally likely to be in $\Gamma_k(v)$. Thus the probability that u is not in this set is at least $1/5$. By Markov's inequality,

$$\Pr\left(|\{(u, v) \in E' : d_{G'\setminus(u,v)}(u, v) \geq k\}| \geq |E'|/9\right) \geq 1/10.$$

Note that, if the degree of every vertex in G' is at least $n^\gamma/4$, then $|E'| \geq n^{1+\gamma}/8$. Hence,

$$\Pr\left(|\{(u, v) \in E' : d_{G'\setminus(u,v)}(u, v) \geq k\}| \geq n^{1+\gamma}/72\right) \geq 9/100.$$

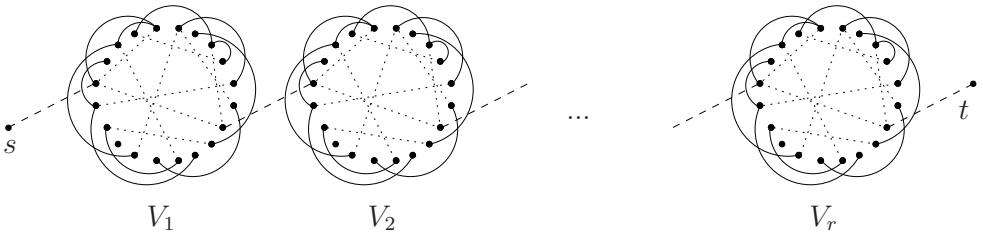


FIG. 5.1. *Diameter lower-bound construction. Edges E_x are dotted, E_j are dashed, and E_m are solid.*

Given that each edge in E' is deleted independently with probability $1/2$ to form E_1, by a further application of the Chernoff bound we deduce that

$$\Pr\left(|\{(u,v) \in E' \setminus E_1 : d_{G' \setminus (u,v)}(u,v) \geq k\}| \geq n^{1+\gamma}/144\right) \geq 8/100.$$

From this set of k-critical edges, we can choose a subset whose size is exactly $\lceil n^{1+\gamma}/144 \rceil$, as required by statement 1. Therefore, all three properties hold with probability at least $1 - 92/100 - 1/100 = 7/100$. □

THEOREM 5.5. *For $3 \leq k = o(\log n / \log\log n)$, any single-pass algorithm that, with probability at least $3/4$, returns $\tilde{D}$ such that*

$$\mathrm{Diam}(G) \leq \tilde{D} \leq (k-1)\,\mathrm{Diam}(G),$$

where G is a weighted graph on n nodes, requires $\tilde{\Omega}(n^{1+1/k})$ space.

Proof. Let $(x, j) \in \{0,1\}^t \times [t]$ be an instance of the INDEX problem. We will show how to transform an algorithm $\mathcal{A}$ for approximating the diameter of a graph into a protocol for INDEX.

Let $G = (V, E = E_1 \cup E_2)$ be a graph on $n' = (144t)^{1/(1+\gamma)}$ nodes with the properties listed in Lemma 5.4. We assume that both Alice and Bob know G and that, moreover, they agree on an ordered list $e_1, \ldots, e_t$ of the edges that are in E_2. This may be assumed, because Alice and Bob can generate identical enumerations of all graphs on n' nodes and all partitions of the edges of each graph into E_1 and E_2, test each graph and partition for the necessary properties, and use the first that passes all of the tests. Finding such a G may take exponential time, but that is all right, because it is only the communication complexity of the resulting INDEX protocol, not the time complexity, that concerns us.

Alice forms the graph $G_x = (V, E_m \cup E_x)$, where $E_x = \{e_i \in E_2 : x_i = 1\}$ and $E_m = E_1$. She then creates the prefix of a stream by taking r (to be determined later) copies of G_x, i.e., a graph on $n'r$ vertices $\{v_1^1, \ldots, v_{n'}^1, v_1^2, \ldots, v_{n'}^2, v_1^3, \ldots, v_{n'}^r\}$ and with edge set $\{(v_j^i, v_k^i) : i \in [r], (v_j, v_k) \in E_x\}$. All these edges have unit weight.

Let j be the index in the instance of INDEX, and let $e_j = (a, b)$. Bob determines the remaining edges E_j as follows: $r-1$ edges of zero weight, $\{(v_b^i, v_a^{i+1}) : i \in [r-1]\}$, and two edges of weight $k+1$, (s, v_a^1) and (v_b^r, t). See Figure 5.1 for a diagram of the construction.

Note that, regardless of the values of x and j, the diameter of the graph described by the stream equals $d_G(s,t)$. Note that $x_j = 1$ implies that $d_G(s,t) = r + 2k + 2$. However, if $x_j = 0$, then $d_G(s,t) = kr + 2k + 2$. Hence, for $r = 2k^2$, the ratio between $kr + 2k + 2$ and $r + 2k + 2$ is at least $k - 1$. Therefore, any single-pass algorithm that approximates the diameter to within a factor of $k - 1$ gives rise to a one-way protocol for solving INDEX. This implies that any such algorithm requires

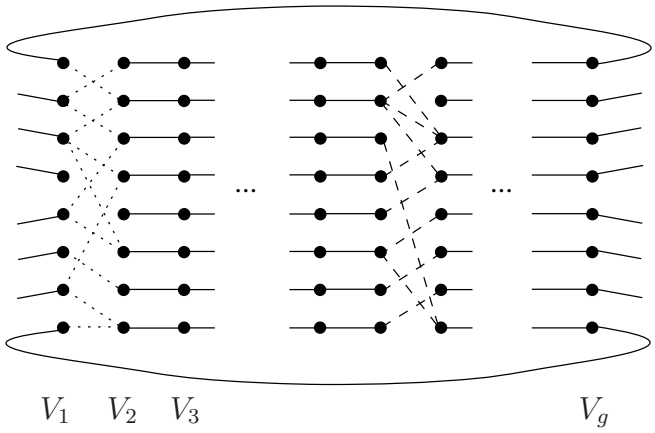


FIG. 5.2. *Girth lower-bound construction. Edges E_x are dotted, E_y are dashed, and E_m are solid.*

$\Omega(n^{1+1/k})$ bits of space, because the total number of nodes in the construction is $n = O((144t)^{1/(1+1/k)}k^2)$. □

5.3. Girth. In this section, we prove a lower bound on the space required by a (multipass) algorithm that tests whether a graph has girth at most g. We shall make use of the following result from [35].

LEMMA 5.6 (see Lazebnik, Ustimenko, and Woldar [35]). *Let $k \geq 1$ be an odd integer, $t = \lfloor \frac{k+2}{4} \rfloor$, and q be a prime power. There exists a bipartite, q-regular graph with at most $2q^{k-t+1}$ nodes and girth at least $k+5$.*

The following lower bound is established with a construction based on Lemma 5.6 that yields a reduction from SET-DISJOINTNESS to girth estimation.

THEOREM 5.7. *For $g \geq 5$, any p-pass algorithm that tests whether the girth of an unweighted graph is at most g requires $\Omega\left(p^{-1}(n/g)^{1+4/(3g-4)}\right)$ space. If g is odd, this can be strengthened to $\Omega\left(p^{-1}(n/g)^{1+4/(3g-7)}\right)$ space.*

Proof. Let q be a prime power; let $k = g-4$ if g is odd, and $k = g-3$ if g is even. Let $t = \lfloor \frac{k+2}{4} \rfloor$. Then,

$$k - t + 1 \leq k - \frac{k+2}{4} + 3/4 + 1 \leq \begin{cases} (3g-7)/4 & \text{if } g \text{ is odd,} \\ (3g-4)/4 & \text{if } g \text{ is even.} \end{cases}$$

Lemma 5.6 implies that there exists a q-regular graph $G' = (L \cup R, E')$ with at most $2n' \leq 2q^{k-t+1}$ nodes and girth at least $g+1$. Let $L = \{l_1, \ldots, l_{n'}\}$ and $R = \{r_1, \ldots, r_{n'}\}$ and, for each $i \in [n']$, $D_i = \Gamma(l_i)$.

We let $(x, y) \in \{0,1\}^r \times \{0,1\}^r$ be an instance of SET-DISJOINTNESS where $r = n'q$. It will be convenient to write $x = x^1 \ldots x^{n'}$ and $y = y^1 \ldots y^{n'}$, where $x^i, y^j \in \{0,1\}^q$. We will show how to transform a p-pass algorithm $\mathcal{A}$ for testing whether the girth of a graph is at most g into a protocol for SET-DISJOINTNESS. If $\mathcal{A}$ uses M bits of working memory, then the protocol will transmit $O(pM)$ bits. Hence $M = \Omega(p^{-1}n'q)$.

Alice and Bob construct a graph G based upon G', x, and y as follows. For $i \in [g]$, let $V_i = \{v_1^i, \ldots, v_{n'}^i\}$. For each $i \in [n']$, let $D_i(x) \subset D_i$ be the subset of D_i whose characteristic vector is x^i. $D_i(y)$ is defined similarly. There are three sets of edges on

these nodes:

$$E_m = \bigcup_{j=\lfloor g/2 \rfloor + 1}^{g} \{(v_i^j, v_i^{j+1}) : i \in [n']\},$$
$$E_x = \{(v_i^1, v_j^2) : j \in D_i(x), i \in [n']\}, \text{ and}$$
$$E_y = \{(v_j^{\lfloor g/2 \rfloor}, v_i^{\lfloor g/2 \rfloor + 1}) : j \in D_i(y), i \in [n']\}.$$

See Figure 5.2 for a diagram of the construction.

Note that $\text{Girth}(G) = g$ if there exists i such that $D_i(x) \cap D_i(y) \neq \emptyset$, i.e., x and y are not disjoint. However, if x and y are disjoint, then the shortest cycle is of length at least $4 + 2\lfloor \frac{g-2}{2} \rfloor \geq g + 1$. Hence, determining whether the girth is at most g determines whether x and y are disjoint. □

6. Toward fast per-item processing. In section 3, we gave a spanner construction that processes each edge much faster than previous spanner-construction algorithms. In this section, we explore two general methods for decreasing the per-edge computation time of a streaming algorithm. As a consequence, we will show how some results from [20] give rise to efficient graph-stream algorithms.

Our first observation is that we can locally amortize per-edge processing by using some of our storage space as a *buffer* for incoming edges. While the algorithm processes a time-consuming edge, subsequent edges can be buffered subject to the availability of space. This yields a potential decrease in the minimum allowable time between the arrival of consecutive pairs of incoming edges.

THEOREM 6.1. *Consider a streaming algorithm that runs in space $S(n)$ and uses computation time $\tau(m, n)$ to process the entire stream. This streaming algorithm can be simulated by a one-pass streaming algorithm that uses $O(S(n) \log n)$ storage space and has worst-case time per-edge $\tau(m, n)/S(n)$.*

Next we turn to capitalizing on work done to speed up *dynamic graph algorithms.* Dynamic graph algorithms allow edges to be inserted and deleted in any order and the current graph to be queried for a property $\mathcal{P}$ at any point. *Partially dynamic algorithms*, on the other hand, are those that allow only edge insertions and querying. In [20], the authors describe a technique called *sparsification* and use it to speed up existing dynamic graph algorithms that decide whether a graph has property $\mathcal{P}$. Sparsification is based on maintaining strong certificates throughout the updates to the graph.

DEFINITION 6.2. *For any graph property $\mathcal{P}$ and graph G, a* strong certificate *for G is a graph G' on the same vertex set such that, for any H, $G \cup H$ has property $\mathcal{P}$ if and only if $G' \cup H$ has it as well.*

It is easy to see that strong certificates obey a transitivity property: If G' is a strong certificate of property $\mathcal{P}$ for graph G, and G'' is a strong certificate for G', then G'' is a strong certificate for G. Strong certificates also obey a compositional property. If G' and H' are strong certificates of $\mathcal{P}$ for G and H, then $G' \cup H'$ is a strong certificate for $G \cup H$.

In order to achieve their speedup, the authors of [20] ensure that the certificates they maintain are not only strong but also sparse. A property is said to have *sparse certificates* if there is some constant c such that for every graph G on an n-vertex set, we can find a strong certificate for G with at most cn edges. Maintaining $\mathcal{P}$ over a sparse certificate allows an algorithm to run on a dense graph, using the (smaller) computational time required for a sparse graph.

TABLE 6.1
One-pass, $O(n \operatorname{polylog} n)$ space streaming algorithms given by Theorem 6.3.

Problem	Time/edge
Bipartiteness	$\alpha(n)$
Connected comps.	$\alpha(n)$
2-vertex connected comps.	$\alpha(n)$
3-vertex connected comps.	$\alpha(n)$
4-vertex connected comps.	$\log n$
MST	$\log n$
2-edge connected comps.	$\alpha(n)$
3-edge connected comps.	$\alpha(n)$
4-edge connected comps.	$n\alpha(n)$
Constant edge connected comps.	$n \log n$

In the streaming model, we are concerned only with edge insertion and need only query the property at the end of the stream. Moreover, observe that a sparse certificate fits in space $O(n \operatorname{polylog} n)$. The following theorem states that any algorithm that could be sped up via the three major techniques described in [20] yields a one-pass, $O(n \operatorname{polylog} n)$ space, streaming algorithm with the improved running time per input edge.

THEOREM 6.3. *Let $\mathcal{P}$ be a property for which we can find a sparse certificate in time $f(n, m)$. Then there exists a one-pass, semistreaming algorithm that maintains a sparse certificate for $\mathcal{P}$ using $f(n, O(n))/n$ time per edge.*

Proof. Let the edges in the stream be denoted $e_1, e_2, \ldots, e_m$. Let G_i denote the subgraph given by $e_1, e_2, \ldots, e_i$. Inductively, assume we have a sparse certificate C_{jn} for G_{jn}, where $1 \leq j \leq \lfloor m/n \rfloor$, constructed in time $f(n, O(n))/n$ per edge. Also, inductively assume that we have buffered the next n edges, $e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}$. Let $T = C_{jn} \cup \{e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}\}$. By the composability of strong certificates, T is a strong certificate for $G_{(j+1)n}$. Let $C_{(j+1)n}$ be the sparse certificate of T. By the transitivity of strong certificates, $C_{(j+1)n}$ is a sparse certificate of $G_{(j+1)n}$. Since C_{jn} is sparse, $|T| = (c+1)n$. Thus, computing $C_{(j+1)n}$ takes time $f(n, O(n))$. By Theorem 6.1, this results in $f(n, O(n))/n$ time per edge, charged over $e_{jn+1}, e_{jn+2}, \ldots, e_{(j+1)n}$. This computation can be done while the next n edges are being buffered. If $k = n\lfloor m/n \rfloor$, then the final sparse certificate will be $C_k \cup \{e_{k+1}, e_{k+2}, \ldots, e_m\}$. □

We note that, for $f(n, m)$ that is linear or sublinear in m, a better speedup may be achieved by buffering more than n edges, which is possible when we have more space. In [20], the authors provide many algorithms for computing various graph properties which they speed up using sparsification. Applying Theorem 6.3 to these algorithms yields the list of streaming algorithms outlined in Table 6.1. For $l \geq 2$, the l-vertex and l-edge connectivity problems either have not been explicitly considered in the streaming model or have algorithms with significantly slower time per edge [22].

Acknowledgments. We thank Martin Sauerhoff for helpful comments on the results in section 4 and the anonymous referees for their comments on the presentation.

REFERENCES

[1] J. ABELLO, A. L. BUCHSBAUM, AND J. WESTBROOK, *A functional approach to external graph algorithms*, Algorithmica, 32 (2002), pp. 437–458.

[2] G. AGGARWAL, M. DATAR, S. RAJAGOPALAN, AND M. RUHL, *On the streaming model augmented with a sorting primitive*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2004, pp. 540–549.

[3] R. Albert, H. Jeong, and A.-L. Barabasi, *The diameter of the world wide web*, Nature, 401 (1999), p. 130.
[4] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments*, J. Comput. System Sci., 58 (1999), pp. 137–147.
[5] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear time construction of sparse neighborhood covers*, SIAM J. Comput., 28 (1998), pp. 263–277.
[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, *Reductions in streaming algorithms, with an application to counting triangles in graphs*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 623–632.
[7] S. Baswana and S. Sen, *A simple linear time algorithm for computing a* $(2k-1)$*-spanner of* $O(n^{1+1/k})$ *size in weighted graphs*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Eindhoven, The Netherlands, 2003, pp. 384–396.
[8] S. Baswana, *Streaming algorithm for graph spanners—single pass and constant processing time per edge*, Inform. Process. Lett., 106 (2007), pp. 110–114.
[9] B. Bollobás, *Random Graphs*, Academic Press, London, 1985.
[10] A. L. Buchsbaum, R. Giancarlo, and J. Westbrook, *On finding common neighborhoods in massive graphs*, Theoret. Comput. Sci., 299 (2003), pp. 707–718.
[11] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, *Counting triangles in data streams*, in Proceedings of the ACM Symposium on Principles of Database Systems, ACM, New York, 2006, pp. 253–262.
[12] A. Chakrabarti, G. Cormode, and A. McGregor, *A near-optimal algorithm for computing the entropy of a stream*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2007, pp. 328–335.
[13] E. Cohen, *Fast algorithms for constructing t-spanners and paths with stretch t*, SIAM J. Comput., 28 (1998), pp. 210–236.
[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, McGraw-Hill, New York, 2001.
[15] G. Cormode and S. Muthukrishnan, *Space efficient mining of multigraph streams*, in Proceedings of the ACM Symposium on Principles of Database Systems, ACM, New York, 2005, pp. 271–282.
[16] C. Demetrescu, I. Finocchi, and A. Ribichini, *Trading off space for passes in graph streaming problems*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2006, pp. 714–723.
[17] M. Elkin, *Computing almost shortest paths*, ACM Trans. Algorithms, 1 (2005), pp. 283–323.
[18] M. Elkin, *Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Wroclaw, Poland, 2007, pp. 716–727.
[19] M. Elkin and J. Zhang, *Efficient algorithms for constructing* $(1+\epsilon,\beta)$*-spanners in the distributed and streaming models*, Distributed Computing, 18 (2006), pp. 375–385.
[20] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification—a technique for speeding up dynamic graph algorithms*, J. ACM, 44 (1997), pp. 669–696.
[21] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *Graph distances in the streaming model: The value of space*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 745–754.
[22] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *On graph problems in a semi-streaming model*, Theoret. Comput. Sci., 348 (2005), pp. 207–216.
[23] J. Feigenbaum, S. Kannan, M. J. Strauss, and M. Viswanathan, *An approximate* L^1*-difference algorithm for massive data streams*, SIAM J. Comput., 32 (2002), pp. 131–151.
[24] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *Fast, small-space algorithms for approximate histogram maintenance*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 389–398.
[25] M. Greenwald and S. Khanna, *Efficient online computation of quantile summaries*, in Proceedings of the ACM International Conference on Management of Data, ACM, New York, 2001, pp. 58–66.
[26] S. Guha, N. Koudas, and K. Shim, *Approximation and streaming algorithms for histogram construction problems*, ACM Trans. Database Syst., 31 (2006), pp. 396–438.
[27] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, in External Memory Algorithms, AMS, Providence, RI, 1999, pp. 107–118.
[28] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 2000, pp. 189–197.

[29] P. Indyk and D. P. Woodruff, *Optimal approximations of the frequency moments of data streams*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2005, pp. 202–208.
[30] R. Jain, J. Radhakrishnan, and P. Sen, *A direct sum theorem in communication complexity via message compression*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Eindhoven, The Netherlands, 2003, pp. 300–315.
[31] H. Jowhari and M. Ghodsi, *New streaming algorithms for counting triangles in graphs*, in Proceedings of the International Conference on Computing and Combinatorics, Kunming, Yunnan, China, 2005, pp. 710–716.
[32] B. Kalyanasundaram and G. Schnitger, *The probabilistic communication complexity of set intersection*, SIAM J. Discrete Math., 5 (1992), pp. 545–557.
[33] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, *Extracting large-scale knowledge bases from the web*, in Proceedings of the International Conference on Very Large Data Bases, Edinburgh, Scotland, 1999, pp. 639–650.
[34] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[35] F. Lazebnik, V. A. Ustimenko, and A. J. Woldar, *A new series of dense graphs of high girth*, Bull. AMS, 32 (1995), pp. 73–79.
[36] A. McGregor, *Finding graph matchings in data streams*, in Proceedings of APPROX-RANDOM, Berkeley, CA, 2005, pp. 170–181.
[37] S. Muthukrishnan, *Data Streams: Algorithms and Applications*, Now Publishers, Boston, 2005.
[38] N. Nisan and A. Wigderson, *Rounds in communication complexity revisited*, SIAM J. Comput., 22 (1993), pp. 211–219.
[39] A. A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 106 (1992), pp. 385–390.
[40] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 183–192.
[41] M. Zelke, *k-connectivity in the semi-streaming model*, http://arxiv.org/abs/cs/0608066, 2006.
[42] M. Zelke, *Optimal per-edge processing times in the semi-streaming model*, Inform. Process. Lett., 104 (2007), pp. 106–112.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1728–1760

PRIVATE APPROXIMATION OF SEARCH PROBLEMS*

AMOS BEIMEL†, PAZ CARMI†, KOBBI NISSIM†, AND ENAV WEINREB‡

Abstract. Many approximation algorithms have been presented in the last decades for $\mathcal{NP}$-hard search problems. The focus of this paper is on cryptographic applications, where it is desirable to design algorithms which do not leak unnecessary information. Specifically, we are interested in private approximation algorithms—efficient approximation algorithms whose output does not leak information not implied by the optimal solutions to the search problems. Privacy requirements add constraints on the approximation algorithms; in particular, known approximation algorithms usually leak a lot of information. For functions, Feigenbaum et al. [*ACM Trans. Algorithms*, 2 (2006), pp. 435–472] presented a natural requirement that a private algorithm should not leak information not implied by the original function. Generalizing this requirement to relations is not straightforward as an input may have many different outputs. We present a new definition that captures a minimal privacy requirement from such algorithms; applied to an input instance, it should not leak any information that is not implied by its *collection of exact solutions*. We argue that our privacy requirement is natural and quite minimal. We show that, even under this minimal definition of privacy, for well-studied problems such as vertex cover and max exact 3SAT, private approximation algorithms are unlikely to exist even for poor approximation ratios. Similarly to Halevi et al. [in *Proceedings of the 33rd ACM Symposium on Theory of Computing*, ACM, New York, 2001, pp. 550–559], we define a relaxed notion of approximation algorithms that leak (a little) information, and demonstrate the applicability of this notion by showing near optimal approximation algorithms for max exact 3SAT that leak a little information.

Key words. secure computation, private approximation, solution-list algorithms, vertex cover

AMS subject classifications. 68Q17, 94A60

DOI. 10.1137/060671899

1. Introduction.

Approximation algorithms are currently one of the main research fields in theoretical computer science. The design of algorithms for approximating computationally hard problems has attracted substantial attention in the last few decades, as has the research on proving hardness of approximation. Frequently, approximation algorithms are applied to sensitive data, as in the distributed cryptographic setup of secure computation. In this paper we study privacy issues related to approximation algorithms of search problems.

We consider an abstract client-server setting. This scenario, besides being interesting on its own, is important since the multiparty distributed setting can be reduced to this model using secure function evaluation protocols [28, 12] (see the discussion below). In the client-server setting, the server $\mathcal{S}$ is willing to let the client $\mathcal{C}$ learn a specific functionality f of its input. The standard requirement in this setting is that no other information would be leaked to $\mathcal{C}$. For concreteness, assume that $\mathcal{S}$ holds a graph G and consider the following examples:

*Received by the editors October 10, 2006; accepted for publication (in revised form) August 9, 2008; published electronically December 19, 2008. An extended abstract of this work appeared in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), 2006. This work was partially supported by the Frankel Center for Computer Science.

http://www.siam.org/journals/sicomp/38-5/67189.html

†Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel (beimel@cs.bgu.ac.il, carmip@gmail.com, kobbi@cs.bgu.ac.il). Part of this work was done while the first author was on sabbatical at the University of California, Davis, partially supported by the David and Lucile Packard Foundation. The third author's research was partially supported by the Israel Science Foundation grant 860/06.

‡CWI Amsterdam, NL-1098 SJ Amsterdam, The Netherlands (e.n.weinreb@cwi.nl).

1. If $f(G)$ is the diameter of G, then $\mathcal{S}$ can simply compute $f(G)$ and send it to $\mathcal{C}$. It is clear that $\mathcal{C}$ learns $f(G)$ but no other information about G.
2. If $f(G)$ is the number of perfect matchings in G, which is a well-known $\#\mathcal{P}$-complete problem, then $\mathcal{C}$ and $\mathcal{S}$ have to settle for an approximation $\hat{f}$. This raises the question of which approximation to use, as $\mathcal{S}$ is willing to reveal only $f(G)$, but $\hat{f}$ may leak some other information. This was the setting in the work by Feigenbaum et al. [10]. They defined the notion of *private approximation* that combines two requirements: (i) *approximation:* $\hat{f}$ is an approximation to f; and (ii) *functional privacy:* $\hat{f}(G)$ can be simulated given $f(G)$. In particular, functional privacy implies that if $f(x) = f(y)$, then $\hat{f}(x)$ and $\hat{f}(y)$ are indistinguishable. It turns out that an efficient private approximation algorithm exists for the number of perfect matchings in a graph [10].
3. A more general and more typical case is when the server $\mathcal{C}$ needs to learn a "solution" for an optimization problem, rather than the value of the objective function.[1] For example, one is usually interested in finding a vertex cover of minimum size in G, rather than just learning the size of the minimum vertex cover. Even if $\mathcal{C}$ and $\mathcal{S}$ are willing to settle for an approximation (i.e., finding a cover which is not much larger than the minimum), it is not clear which cover $\mathcal{C}$ should learn; the framework of private approximations of functions does not address this problem as there may be many solutions to the search problem.

This more general case, where one seeks a computation not of a function but of a solution to an optimization problem, is the focus of this work.

1.1. Our contribution. Our main conceptual contribution is supplying a minimal definition of private approximation of search problems. The generalization of the definition of (functional) private approximation to search problems is not straightforward. As one input may have many different outputs, it is not clear in which cases we need $\hat{f}(x)$ and $\hat{f}(y)$ to be indistinguishable. For our example of vertex cover, it seems that a minimal requirement is that a private approximation algorithm should not distinguish between graphs G_1, G_2 that are equivalent in the sense that they have exactly the same set of solutions; i.e., every minimum cover for G_1 is also a minimum cover for G_2 and vice versa. Note that this is a rather weak requirement, as it does not restrict the approximation algorithm with respect to nonequivalent graphs.[2] Furthermore, it might reveal more information than a single solution to the problem.

In general, given a search problem, we say that two inputs are equivalent if they have the same set of solutions; an algorithm is private if it does not distinguish between equivalent inputs. This definition is a generalization of the privacy requirement in secure function evaluation where each input has a single solution and the protocol should not distinguish between inputs with the same solution. Furthermore, this definition is a generalization of the requirement for private approximation of functions [10]. Thus, we believe that our definition is natural. Furthermore, it is minimal

[1] An optimization problem is a problem where for each input there is a set of feasible solutions and there is an objective function; the goal is given an input to find a feasible solution with a minimum (or maximum) objective vale.

[2] For the vertex cover problem, the number of equivalence classes of graphs is exponential in $|V|$; thus, there may be many equivalence classes that contain only a few graphs. This is in sharp contrast with privacy of the vertex cover problem size that divides the graphs to only $|V|$ equivalence classes, as there are $|V|$ possible answers to the function.

in the sense that weakening it will result in algorithms that cannot be considered private. It is plausible that there are meaningful definitions of privacy for search problems that are incomparable to our definition.

Impossibility results for private approximation of the optimization objective value for several $\mathcal{NP}$-hard problems (e.g., approximating the size of a minimum vertex cover) were proven by Halevi et al. [16] (a detailed summary of their results appears in section 1.2). In view of the result of [16], it seems that we already know all there is to know about private approximation of optimization problems. However, impossibility results for private approximation of the optimization objective value (e.g., approximating the size of a minimum vertex cover) do not imply impossibility results for the task of finding a feasible solution with near optimal optimization value (e.g., finding a small vertex cover). For nonprivate computation, if there is an efficient algorithm whose output is a near optimal solution, then one can compute its value. However, this reduction does not preserve privacy as it leaks more information than implied by the optimization value. Furthermore, for private approximation the above two tasks are incomparable as the privacy requirement of "not learning information" is applied to a different output. For example, consider two graphs G_1 and G_2 such that the sizes of minimum vertex covers of G_1 and G_2 are the same, but the sets of minimum covers are different. A private approximation algorithm for the search problem of vertex cover can return covers of different size, while a private approximation algorithm for the size of the vertex cover must return the same answer for G_1 and G_2.

Impossibility of private approximation. To understand the nature of private approximation of search problems, we concentrate in this work on two optimization problems with different characteristics—vertex cover and max exact 3SAT. The first problem is a minimization problem, while the second is a maximization problem. More importantly, for vertex cover we want a solution satisfying *all* constraints (covering all edges), and we compromise by allowing a solution whose size is not optimal. In contrast, for max exact 3SAT we seek a solution satisfying *most* constraints (i.e., clauses). The same methods we present in this paper for these two problems are applicable for other optimization problems (e.g., max-cut). Our first technical contribution is an impossibility result for private approximation of vertex cover.

Informal Theorem. *If $\mathcal{RP} \neq \mathcal{NP}$, then vertex cover cannot be approximated privately even within an approximation ratio as poor as $n^{1-\epsilon}$.*

A similar result is shown for max exact 3SAT. This means that although our notion of privacy is minimal and it seems that any reasonable notion of privacy for search problems should imply it, there are natural problems for which it is too strong.

Our proof techniques for the impossibility results are different from those used for obtaining the inapproximability results for the functional version of the problem [16]. We show how to use a private approximation algorithm for a problem in order to solve the problem *exactly* in polynomial time. All our lower bounds have the same structure, where a solution is constructed in an iterative manner. For example, for vertex cover, in each iteration a node is examined. If that node appears in some optimal solution, we add it to the solution and remove it and its neighbors from the graph. If there exists some optimal solution that does not include this node, we remove it from the graph. When both conditions are met, we choose one of them arbitrarily. The crux of the algorithm is that the private approximation algorithm is used for deciding which of the conditions hold.

Algorithms that leak a little information. In view of the impossibility results, it is natural to look for a relaxation of the definition. In the setting of functional privacy,

Halevi et al. [16] defined the notion of *almost private algorithms* that are allowed to leak (a little) information beyond what is disclosed by the exact functionality. This notion falls elegantly within our definitional framework, where it is generalized to search problems. We say that an algorithm leaks at most k bits if it refines each equivalence class by dividing it to at most 2^k subclasses. That is, the algorithm leaks only the subclass, which can be described by k bits. For max exact 3SAT, this relaxation results in a tremendous improvement.

INFORMAL THEOREM. *There exists an efficient deterministic approximation algorithm for max exact 3SAT with a near optimal approximation ratio of* $7/8 - \epsilon$ *that leaks only* $O(\log \log n)$ *bits of information (where the exact constant in the "O" notation depends on* ϵ*).*

Interestingly, the algorithm for max exact 3SAT and the algorithm we describe for vertex cover fall into a class of approximation algorithms that we call *solution-list algorithms*. This class of algorithms provides much stronger privacy guarantees. Intuitively, for every input size n, the solution-list algorithm determines in advance (before seeing the input) a list of possible outcomes. Upon seeing the actual input, the algorithm is restricted to output a solution from the list. Thus, the number of bits the algorithm leaks is at most the logarithm of the size of the list. Solution-list algorithms provide stronger privacy guarantees as they protect every pair of inputs (not only pairs of equivalent inputs). For max exact 3SAT, our algorithm efficiently computes a list of $\text{poly}((\log n)/\epsilon)$ assignments such that for every exact 3CNF (conjunctive normal form) formula ϕ there exists an assignment in the list that is a good approximation for ϕ; hence the algorithm leaks $O(\log \log n + \log 1/\epsilon)$ bits. For vertex cover, we obtain a solution-list algorithm that is also interesting, but not as dramatic as that for max exact 3SAT, and we obtain a tradeoff between the amount of leakage and approximation quality.

We also show an impossibility result for approximating vertex cover while leaking information.

INFORMAL THEOREM. *If* $\mathcal{RP} \neq \mathcal{NP}$*, then vertex cover cannot be approximated within an approximation ratio as poor as* $n^{1-\epsilon}$ *while leaking at most* $O(\epsilon \log n)$ *bits.*

The above result was improved in [5], where it was proven that any algorithm that $n^{1-\epsilon}$-approximates vertex cover must leak $\Omega(n^\epsilon)$ bits. This implies that the solution-list algorithm for the problem is optimal up to a constant factor.

Private computation of search problems in $\mathcal{P}$. The notion of private search is applicable also to search problems in $\mathcal{P}$. For example, suppose a server holds a graph G and a client wants to learn a maximum matching of this graph. What kind of information can the client derive from the server's output? Can the client rule out many input graphs given the answer? It is somewhat surprising that the problem of private computation of search problems in $\mathcal{P}$ was not considered in previous papers. For many problems in $\mathcal{P}$, there is an efficient private algorithm solving the search problem while satisfying our minimal definition of privacy. One option is choosing the lexicographically first exact solution (e.g., for maximum matching and shortest path). Another option is constructing a randomized algorithm that chooses a random exact solution according to some distribution. Both approaches are somewhat problematic. Following our work, in [6] two stronger definitions of private computation of search problems were suggested and discussed, and algorithmic techniques were developed for constructing algorithms satisfying these stronger definitions.

For some problems finding a solution privately imposes additional constraints on the algorithm. We show that these constraints can make the problem much harder.

INFORMAL THEOREM. *There exists a search problem which can be efficiently solved without privacy constraints but cannot be efficiently and privately solved unless* $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$.

Multiparty approximations of search problems. As mentioned before, the abstract client-server setting we consider in this paper is important since the multiparty distributed setting can be reduced to this model. Consider the following scenario in the multiparty model: There is some secret input shared by the parties (that is, all parties together can compute the input, while smaller sets of parties have no information on the input). The parties want to solve some search problem on the input and make the output public without disclosing extra information. Clearly, our impossibility results hold for this model as well. Furthermore, using general secure function evaluation protocols [28, 12], efficient private algorithms in the server/client model can be transformed into protocols in the multiparty model. That is, the efficient algorithm and the sharing of the secret input imply that there is a small circuit, whose inputs are the inputs of the parties, computing the private functionality. Using the constructions of [28, 12], there is a private protocol computing this private functionality.

1.2. Related work. Feigenbaum et al. [10] initiated the discussion of private approximation of functions. They observed that combining approximation algorithms and secure function evaluation protocols might result in protocols that are not private as the output of the approximation algorithm might leak information. The definition of functional privacy put forward by [10] is a simulation-based definition, where the simulator's input is the exact value $f(x)$ and its output distribution is computationally indistinguishable from $\hat{f}(x)$. Under this definition, they provided a protocol for approximating the Hamming distance of two n-bit strings with communication complexity $\tilde{O}(\sqrt{n})$, and polynomial-time solutions for approximating the permanent and other natural $\#\mathcal{P}$ problems. Other private approximation protocols published since include [11, 22, 19]. In particular, Indyk and Woodruff [19] provide a polylogarithmic communication approximation for the Hamming distance and a secure approximation of the nearest neighbor search problem.

Inapproximability results for computing the *size* of a minimum vertex cover within approximation $n^{1-\epsilon}$ were proved by Halevi et al. [16]. Their proof uses a special *sliding-window* reduction that given a SAT instance ϕ and an integer z constructs an instance G of vertex cover such that if ϕ is satisfiable, then G has a vertex cover of size z, and otherwise any vertex cover for G is of size at least $z+1$. These techniques do not apply in our setting, as the large number of equivalence classes does not allow a simple averaging argument such as that used in [16].

The notion of almost private approximation was introduced in [16]. Their definition modifies that of [10] by allowing the simulator to consult a deterministic predicate of the input. They showed that by this slight compromise in privacy, one can get fairly good approximations for any problem that admits a good deterministic approximation. For the functional version of vertex cover this yields an approximation ratio 4 while leaking one bit (more generally, there is a tradeoff between the leakage and the approximation ratio). A similar relaxation of privacy is the notion of additional information in secure two-party protocols [3]. Related ideas can be found in the study of knowledge complexity [15, 14, 7, 13, 27].

We next compare the possibility and impossibility results for vertex cover and max exact 3SAT to nonprivate computation and privately approximating the size of the solutions.

Vertex cover. Vertex cover can be (nonprivately) $(2-o(1))$-approximated in deterministic polynomial time [24, 4, 17], and if $\mathcal{P} \neq \mathcal{NP}$, then there is no polynomial-time 1.3606-approximation algorithm for vertex cover [9].[3] Furthermore, if $\mathcal{NP} \not\subseteq \mathcal{BPP}$, then for every constant $\epsilon > 0$ there is no private $n^{1-\epsilon}$-approximation algorithm for the size of the minimum vertex cover [16]. However, there is a 4-approximation algorithm for the size of the minimum vertex cover which leaks one bit, and more generally, there is a tradeoff between the leakage and the approximation ratio [16]. In contrast, we show that if $\mathcal{RP} \neq \mathcal{NP}$, then this problem cannot be privately $n^{1-\epsilon}$-approximated even if a leakage of $(\epsilon \log n)/6$ bits is allowed. In [5] this result was improved showing that every algorithm $n^{1-\epsilon}$-approximating the vertex cover problem must leak $\Omega(n^{\epsilon})$ bits.

Max exact 3SAT. The max exact 3SAT problem can be 7/8-approximated [20], and if $\mathcal{P} \neq \mathcal{NP}$ there is no polynomial-time $(7/8 + \epsilon)$-approximation algorithm for it [18]. The trivial algorithm that returns 7/8 is a private 7/8-approximation algorithm for the functional version of max exact 3SAT. We show that if $\mathcal{RP} \neq \mathcal{NP}$, then the search problem of max exact 3SAT cannot be privately $n^{1-\epsilon}$-approximated; however, with $O(\log \log n + \log 1/\epsilon)$ leakage there is a $(7/8 - \epsilon)$-approximation algorithm for it.

Organization. In section 2 we define private algorithms for search problems. In section 3 we provide impossibility results for the minimum vertex cover and the max exact 3SAT search problems. In section 4 we discuss algorithms that leak (a little) information and describe such algorithms for both search problems. Later, in section 5, we prove our strongest impossibility result, showing vertex cover cannot be privately approximated even if a leakage of $O(\log n)$ bits is allowed. In section 6, we present an impossibility result for a search problem in $\mathcal{P}$. In Appendix A, we discuss an equivalent definition for private algorithms for search problems. In Appendices B and C, we complete the proofs of the impossibility results.

2. Definition of private algorithms with respect to a privacy structure. There are two different aspects of private algorithms—the utility of the algorithm (what should be computed) and the privacy requirement (what should be protected, that is, what information should not be revealed by the computation). In computing functions this is quite straightforward; we want to compute (or approximate) a function, and we want to protect inputs with the same output. For search algorithms, we know what should be computed. However, since these algorithms may output different outputs on the same input, it is less clear what should be protected. We thus separate the specification of what we want to protect from what we compute. In general, we require the output of the algorithm on certain pairs of inputs to be indistinguishable. In this section we define the pairs of inputs that should be protected by a private algorithm.

DEFINITION 2.1 (privacy structure). *A privacy structure $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is an equivalence relation on instances. For $\langle x, y\rangle \in \mathcal{R}$, we use the notation $x \equiv_{\mathcal{R}} y$.*

We will discuss only privacy structures of the form $\mathcal{R} = \cup_{n \in \mathcal{N}} \mathcal{R}_n$, where $\mathcal{R}_n$ is an equivalence relation between instances of size n, such as graphs on n vertices or Boolean formulae over n variables. We say that an algorithm $\mathcal{A}$ is *private* with respect to a privacy structure $\mathcal{R}$ if the results of executing $\mathcal{A}$ on two $\mathcal{R}$-equivalent inputs are computationally indistinguishable.

[3] A hardness result with a larger nonapproximability factor was proved in [21]: Assuming the unique games conjecture, there is no polynomial-time $(2 - \epsilon)$-approximation algorithm for vertex cover for every constant $\epsilon > 0$.

DEFINITION 2.2 (private algorithm). *Let $\mathcal{R}$ be a privacy structure. A randomized polynomial-time algorithm $\mathcal{A}$ is* private *with respect to $\mathcal{R}$ if for every randomized polynomial-time algorithm $\mathcal{D}$ and for every positive polynomial $p(\cdot)$ there exists some $n_0 \in \mathbb{N}$ such that for every $x, y \in \{0,1\}^*$ such that $x \equiv_{\mathcal{R}} y$ and $|x| = |y| \geq n_0$*

$$\left| \Pr[\mathcal{D}(\mathcal{A}(x), x, y) = 1] - \Pr[\mathcal{D}(\mathcal{A}(y), x, y) = 1] \right| \leq \frac{1}{p(|x|)},$$

where the probabilities are taken over the random choices of $\mathcal{A}$ and $\mathcal{D}$. That is, when $x \equiv_{\mathcal{R}} y$, an algorithm $\mathcal{D}$ given an output of $\mathcal{A}$ cannot distinguish whether the input of $\mathcal{A}$ is x or y.

Example 2.3. Let $f : \{0,1\}^* \to \mathbb{N}$ be a function. Define

$$\mathcal{R}_f = \{\langle x, y\rangle : |x| = |y|, f(x) = f(y)\}.$$

The relation $\mathcal{R}_f$ is the relation implicitly considered when discussing private computation of functions. Furthermore, this is the relation implicitly considered in [10, 16] when considering private approximation of the objective function, where the privacy requirement is with respect to inputs with the same objective value.

In Appendix A, we present equivalent definitions of privacy which require "semantic security" and prove that these definitions are equivalent to Definition 2.2.

We next recall the definition of a search problem and define the privacy structure associated with it.

DEFINITION 2.4. *A bivariate relation Q is polynomially bounded if there exists a constant c such that $|w| \leq |x|^c$ for every $\langle x, w\rangle \in Q$. The* decision problem *for Q is, given an input x, to decide if there exists a w such that $\langle x, w\rangle \in Q$ or not. The* search problem *for Q is, given an input x, to* find *a w such that $\langle x, w\rangle \in Q$ if such a w exists.*

Search problems are the more common algorithmic task of finding a solution to a problem (rather than deciding whether the problem has a solution or not). To define a private solution or private approximation of a search problem, one must determine the privacy structure the algorithm should respect. It is a *minimal* requirement to demand that if two inputs have the same set of answers to the search problem, the output of the approximation algorithm should not enable distinguishing between them.

DEFINITION 2.5 (privacy structure of a search problem). *The privacy structure $\mathcal{R}_Q$ related to a relation Q is defined as follows: $x \equiv_{\mathcal{R}_Q} y$ iff*

- *$|x| = |y|$, and*
- *$\langle x, w\rangle \in Q$ iff $\langle y, w\rangle \in Q$ for every w.*

That is, $x \equiv_{\mathcal{R}_Q} y$ if they have the same set of solutions.

We first give an example of a search problem in $\mathcal{P}$.

Example 2.6. Let maxMatch be the maximum matching relation; that is, $\langle G, M\rangle \in$ maxMatch if M is a maximum matching in G. In this case, $\mathcal{R}_{\text{maxMatch}}$ contains all pairs of graphs $G_1 = \langle V, E_1\rangle, G_2 = \langle V, E_2\rangle$ for which M is a maximum matching for G_1 iff it is a maximum matching for G_2.

We give two examples of privacy structures, for specific relations, that would be the focus of this paper.

Example 2.7. A vertex cover of an undirected graph $G = (V, E)$ is a set $C \subseteq V$ that covers all edges in G; that is, $C \cap \{u, v\} \neq \emptyset$ for every $(u, v) \in E$. Let minVC be the minimum vertex cover relation; that is, $\langle G, C\rangle \in$ minVC if C is a minimum vertex cover in G. In this case, $\mathcal{R}_{\text{minVC}}$ contains all pairs of graphs $G_1 = \langle V, E_1\rangle, G_2 =$

$\langle V, E_2\rangle$ for which $C \subseteq V$ is a minimum vertex cover for G_1 iff it is a minimum vertex cover for G_2.

To understand the difference between privately approximating the size of the vertex and privately finding an approximated minimum vertex cover, consider two isomorphic graphs. The size of the minimum vertex cover in these two graphs is the same; thus, a private algorithm approximating the minimum vertex cover size has to "protect" them. However, in general, these two graphs have different sets of minimum vertex covers; thus, an algorithm finding an approximated minimum vertex cover can behave differently on these graphs.

Example 2.8. An exact 3CNF formula is a CNF formula that contains exactly three different literals in each clause. Let maxE3SAT be the max exact 3SAT relation; that is, $\langle \phi, a\rangle \in$ maxE3SAT if ϕ is an exact 3CNF formula over n variables, and a is an assignment to the n variables that satisfies the maximum possible number of clauses in ϕ. In this case, the privacy structure $\mathcal{R}_{\text{maxE3SAT}}$ contains all pairs of exact 3CNF formulae ϕ_1, ϕ_2 over n variables for which an assignment a satisfies the maximum possible number of clauses in ϕ_1 iff it satisfies the maximum possible number of clauses in ϕ_2. We stress that these two assignments need not satisfy the same number of clauses in ϕ_1 and ϕ_2.

We note that a related definition of private approximation was recently presented in the context of the nearest neighbor problem [19]. Their privacy requirement is that instances with identical sets of *approximate* solutions should not be told apart by the private approximation algorithm. This definition may be cast in our framework by constructing a privacy structure where instances that have the same collection of approximate solutions are considered equivalent.

Approximation algorithms. We next define approximation algorithms for a minimization problem (the definition for maximization problems is analogous). Given a bivariate relation Q and an objective function $f : \{0,1\}^n \to \mathbb{R}$ we define

$$\text{OPT}(x) = \min\{f(w) : \langle x, w\rangle \in Q\}.$$

We consider the minimization problem that given an input x finds a solution w such that $\langle x, w\rangle \in Q$ and $f(w) = \text{OPT}(x)$. The $c(n)$-approximation version of the problem is to find a solution w such that $f(w) \le c(n)\cdot\text{OPT}(x)$. Formally, we have the following definition.

DEFINITION 2.9. *For a function $c : \mathbb{N} \to \mathbb{R}$, we say that a randomized algorithm $\mathcal{A}$ is a $c(n)$-approximation algorithm for a minimization problem $\langle Q, f\rangle$ if for every input x with probability one it finds a solution w such that $\langle x, w\rangle \in Q$ and $f(w) \le c(n)\cdot\text{OPT}(x)$ (if x has a solution).*

A weaker definition of approximation requires that $\text{E}(f(\mathcal{A}(x))) \le c(n)\cdot\text{OPT}(x)$; namely, the expected value of the function f applied to the solution returned by the approximation algorithm is bounded by $c(n)\cdot\text{OPT}(x)$. Our impossibility result for vertex cover holds also for this definition. However, for maxE3SAT, the behavior with respect to the two definitions is different. The algorithm that returns a random assignment is a private algorithm for maxE3SAT in which the expected fraction of clauses satisfied is 7/8. In contrast, we prove that every private randomized approximation algorithm according to Definition 2.9 has an approximation ratio at least $n^{1-\epsilon}$ for every constant $\epsilon > 0$.

3. Impossibility results for private approximation.

3.1. Impossibility results for private approximation of vertex cover. In this section we show that private approximation of the vertex cover search problem with respect to $\mathcal{R}_{\text{minVC}}$ (defined in Example 2.7) is a hard task. We start with defining private approximation of vertex cover and then prove impossibility results for both the deterministic and the randomized settings.

DEFINITION 3.1 (private approximation of vertex cover). *An algorithm $\mathcal{A}$ is a* private $c(n)$-approximation *algorithm for* minVC *if* (i) *$\mathcal{A}$ runs in polynomial time;* (ii) *$\mathcal{A}$ is a $c(n)$-approximation algorithm for* minVC*, that is, for every graph G with n vertices, it returns with probability* 1 *a vertex cover whose size is at most $c(n)$ times the size of the smallest vertex cover of G;* (iii) *$\mathcal{A}$ is private with respect to $\mathcal{R}_{\text{minVC}}$.*

To illustrate our definitions, we present a private $(n/\log n)$-approximation algorithm for the vertex cover problem. This algorithm is based on the polynomial-time algorithm of [26] that returns a minimum vertex cover if the size of the vertex cover is at most $\log n$. Actually, in this case there are at most n^2 such covers,[4] and the algorithm can efficiently compute all of them. Thus, we can define any rule to choose one of them (e.g., the lexicographically first, a random vertex cover, or a cover chosen such that it satisfies the stronger definitions of privacy given in [6]). To approximate minVC we do the following:

> If there is a cover of size at most $\log n$,
> then return the lexicographically first minimum vertex cover.
> Otherwise, return the entire set of vertices.

We show impossibility results for privately approximating vertex cover in the deterministic and in the randomized setting.

THEOREM 3.2. *Let $\epsilon > 0$ be a constant.*

1. *If $\mathcal{P} \neq \mathcal{NP}$, then there is no* deterministic *private $n^{1-\epsilon}$-approximation algorithm for the search problem of* minVC.
2. *If $\mathcal{RP} \neq \mathcal{NP}$, then there is no* randomized *private $n^{1-\epsilon}$-approximation algorithm for the search problem of* minVC.

Part 1 of Theorem 3.2 is proven in the rest of this section. Part 2 of Theorem 3.2 is a special case of Theorem 5.1 proven in Appendix B.

3.1.1. Relevant and critical vertices. The framework for proving Theorem 3.2 is the following: We assume the existence of the appropriate private approximation algorithm and derive a greedy algorithm that solves vertex cover *exactly*. The following definitions are central for both the deterministic and the randomized cases.

DEFINITION 3.3 (critical vertices and relevant vertices). *Let $G = \langle V, E\rangle$ be a graph and $v \in V$ be a vertex of G. We say that v is* critical *for G if* every *minimum vertex cover of G contains v. We say that v is* relevant *for G if there* exists *a minimum vertex cover of G that contains v.*

OBSERVATION 3.4. *Every vertex is relevant or noncritical (or both).*

Example 3.5. To illustrate the relation $\mathcal{R}_{\text{minVC}}$ we show a pair of graphs that are equivalent under the relation. One way to create such a pair is to pick a graph and identify a vertex that is critical for this graph. For example, vertex v_3 in Figure 3.1(a) is a critical vertex. To get the second graph we connect the critical vertex to some

[4]The algorithm of [26] first finds a maximal matching of size at most $\log n$ (the size of any matching is a lower bound on the size of a vertex cover). Let P be the endpoints of the edges in this matching (thus, $|P| \leq 2\log n$). In [26] it is proven that any minimum cover is composed of a subset $P' \subseteq P$ and all the neighbors of $P \setminus P'$. As there are at most n^2 subsets of P, there are at most n^2 minimum vertex covers for the graph.

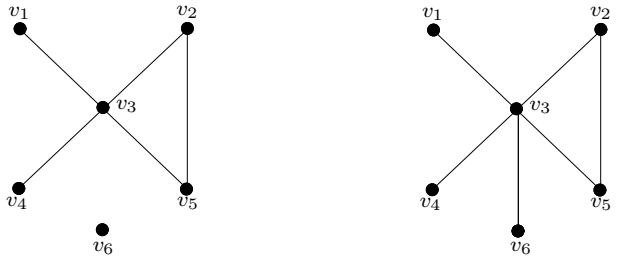


FIG. 3.1. *A pair of graphs equivalent under* $\mathcal{R}_{\text{minVC}}$.

other vertex. In Figure 3.1(b), vertex v_3 is connected to v_6. It is easy to verify that the set of minimum covers in both graphs is $\{\{v_3, v_2\}, \{v_3, v_5\}\}$. The equivalence can also be derived from Claim 3.8 below.

We design a greedy algorithm for exactly solving the vertex cover problem that in each step solves the following problem.

DEFINITION 3.6 (the relevant-noncritical problem).
INPUT: *A graph* $G = \langle V, E\rangle$ *and a vertex* $v \in V$.
OUTPUT: *One of the following:* (i) v *is relevant for* G. (ii) v *is noncritical for* G.

Algorithm `Greedy Vertex Cover`

INPUT: A graph $G = \langle V, E\rangle$.
OUTPUT: A minimum vertex cover of G.

1. If $V = \emptyset$, return $\emptyset$.
2. Pick a vertex $v \in V$ and execute Algorithm `Relevant-Noncritical` on G and v.
3. If the answer is "RELEVANT":
   (a) Run `Greedy Vertex Cover` on the graph $G \setminus \{v\}$. Denote the answer by C_v.
   (b) Return $C_v \cup \{v\}$.
4. If the answer is "NONCRITICAL":
   (a) Let $N(v)$ be the neighbors of v in G, that is, $N(v) = \{u : (u, v) \in E\}$.
   (b) Run `Greedy Vertex Cover` on the graph $G \setminus (\{v\} \cup N(v))$. Denote the answer by $C_{N(v)}$.
   (c) Return $C_{N(v)} \cup N(v)$.

FIG. 3.2. *A greedy algorithm using Algorithm* `Relevant-Noncritical` *to find a minimum vertex cover.*

In Figure 3.2, we describe a greedy algorithm for vertex cover that in each step uses an algorithm that solves the Relevant-Noncritical problem. In subsequent sections, we solve the latter using oracle access to private approximation algorithms for vertex cover. The following claim asserts the correctness of the greedy algorithm.

CLAIM 3.7. *If Algorithm* `Relevant-Noncritical` *is correct and runs in polynomial time, then Algorithm* `Greedy Vertex Cover` *is correct and runs in polynomial time.*

Proof. The proof is by induction on $|V|$. The algorithm is trivially correct for

$V = \emptyset$. Now suppose $V \neq \emptyset$, and we start from the case where v is relevant. In this case, there is a minimum cover C for G that contains v. Denote $d = |C|$, and note that the set $C \setminus \{v\}$ is a cover of size $d-1$ for the graph $G \setminus \{v\}$. By the induction hypothesis, the set C_v is a minimum cover for $G \setminus \{v\}$. We claim that $C_v \cup \{v\}$ is a minimum cover for G. First note that it is indeed a cover of G as any edge adjacent to v is covered by v, and all the rest are covered by C_v. Since C_v is a minimum cover of $G \setminus \{v\}$, it is of size at most $d-1$. Therefore, the size of $C_v \cup \{v\}$ is d, and it is a minimum cover of G.

In the other case, v is not critical for G. Thus, there is a minimum cover C' for G that does not contain v and therefore contains all vertices in $N(v)$. Denote $d' = |C'|$ and $h = |N(v)|$. In this case the set $C' \setminus N(v)$ is a cover of size $d' - h$ of the graph $G \setminus (N(v) \cup \{v\})$. By the induction hypothesis, the set $C_{N(v)}$ is a minimum cover for $G \setminus (N(v) \cup \{v\})$. We claim that $C_{N(v)} \cup N(v)$ is a minimum cover for G. First note that it is indeed a cover of G as any edge adjacent to $N(v) \cup \{v\}$ is covered by $N(v)$, and all other edges are covered by $C_{N(v)}$. Since $C_{N(v)}$ is a minimum cover of $G \setminus (N(v) \cup \{v\})$, it is of size at most $d' - h$. Therefore, the size of $C_{N(v)} \cup N(v)$ is d', and it is a minimum cover of G.

It is straightforward to verify that if `Relevant-Noncritical` runs in polynomial time, then the greedy algorithm runs in polynomial time. □

3.1.2. Combinatorial claims. The following combinatorial claims are helpful in designing algorithms for the Relevant-Noncritical problem in both the deterministic and the randomized settings. Intuitively, a private approximation algorithm must be "sensitive" to small changes in the set of minimum vertex covers of its input graph. We study the connection between the $\mathcal{R}_{\text{minVC}}$ relation and the set of critical and relevant vertices in a graph.

CLAIM 3.8. *Let $G = \langle V, E \rangle$ be a graph, $u, v \in V$ such that $(u, v) \notin E$, and $G^* = \langle V, E^* \rangle$, where $E^* = E \cup (u, v)$. If u is critical for G, then $G \equiv_{\mathcal{R}_{\text{minVC}}} G^*$.*

Proof. We first show that every minimum vertex cover of G is a minimum vertex cover of G^*. Let C be a minimum cover of G. As u is critical for G, we deduce that $u \in C$. Therefore, C covers the edge (u, v), and thus it is a cover of G^*. Note that every cover of G^* is also a cover of G, and thus C is a minimum cover of G^*.

For the other direction, let C^* be a minimum cover of G^*. Let c be the size of a minimum cover of G. As u appears in at least one minimum cover of G, which is also a cover of G^*, the size of C^* is at most c. On the other hand, as $E \subseteq E^*$, the set C^* is also a cover of G, and thus the size of C^* is exactly c. Therefore, C^* is a minimum cover of G. □

We will later see that if a vertex u is not in the result of the private approximation algorithm $\mathcal{A}$, then it is noncritical for the input graph G. However, if the vertex cover size of the input graph is large, the approximation algorithm may return the entire set V as its result. To avoid this, we add a large set of isolated vertices to G. (The size of this set is a function of the approximation ratio.) The mere fact that an isolated vertex is noncritical for G is of course not helpful. Nevertheless, we gain information by connecting this isolated vertex to the vertex v and running $\mathcal{A}$ on the new graph. It will also be helpful to consider duplicating the graph G and connecting the isolated vertex to both copies of the original vertex v.

DEFINITION 3.9 (the graphs G_2 and $G\binom{i}{\wedge}$). *Let $G = \langle V, E \rangle$ be a graph, $v \in V$ be a vertex, I be a set of vertices, and $i \in I$. The graph G_2 is defined as $G_2 = \langle V_2, E_2 \rangle$, where $V_2 \stackrel{def}{=} (V \times \{1,2\}) \cup I$ and $E_2 \stackrel{def}{=} \{(\langle u, j \rangle, \langle w, j \rangle) : (u, w) \in E, j \in \{1,2\}\}$. The graph $G\binom{i}{\wedge}$ is defined as $G\binom{i}{\wedge} = \langle V_2, E(\overset{i}{\wedge}) \rangle$, where $E(\overset{i}{\wedge}) \stackrel{def}{=} E_2 \cup \{(\langle v, j \rangle, i) : j \in \{1,2\}\}$.*

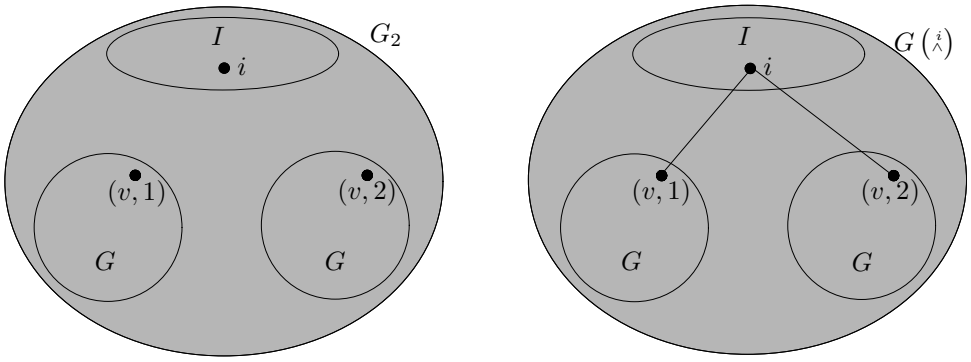


FIG. 3.3. *The graphs G_2 and $G(\binom{i}{\wedge})$.*

The graphs G_2 and $G\binom{i}{\wedge}$ are illustrated in Figure 3.3. The following claims summarize the properties of G_2 and $G\binom{i}{\wedge}$.

CLAIM 3.10. *If v is critical for G, then $G_2 \equiv_{\mathcal{R}_{\text{minVC}}} G\binom{i}{\wedge}$.*

Proof. Since G_2 contains two separate copies of G and v is critical for G, the vertices $\langle v,1\rangle$ and $\langle v,2\rangle$ are critical for G_2. Hence, by Claim 3.8, adding the edges $(\langle v,1\rangle, i)$ and $(\langle v,2\rangle, i)$ does not change the set of minimum vertex covers of the graph. □

CLAIM 3.11. *If v is not relevant for G, then i is critical for $G\binom{i}{\wedge}$.*

Proof. Assume toward contradiction that the vertex i is noncritical for $G\binom{i}{\wedge}$. Hence, there must be a minimum cover $C(\binom{i}{\wedge})$ of $G\binom{i}{\wedge}$ that contains $\langle v,1\rangle$ and $\langle v,2\rangle$. The intersection of $C(\binom{i}{\wedge})$ with each copy of G contains a cover of G that contains the appropriate copy of v. As v is not relevant for G, these covers are not optimal. Let c be the size of a minimum vertex cover of G. Then $\left|C(\binom{i}{\wedge})\right| \geq 2(c+1) = 2c+2$. On the other hand, let C be a minimum cover of G of size c. Then, the set $(C \times \{1,2\}) \cup \{i\}$ is a cover of $G\binom{i}{\wedge}$ of size $2c+1$, in a contradiction to the minimality of $C(\binom{i}{\wedge})$. Hence, i is critical for $G\binom{i}{\wedge}$. □

3.1.3. Impossibility result for deterministic private approximation. In this section we show an algorithm that solves the Relevant-Noncritical problem, given an oracle access to a deterministic private approximation algorithm for minVC.

CLAIM 3.12. *Let $\mathcal{A}$ be a deterministic private approximation algorithm for* minVC*, let $G = \langle V, E\rangle$ be a graph, and denote $W = \mathcal{A}(G)$. Then for any two different vertices $v_1, v_2 \in V \backslash W$, the vertex v_1 is not critical for G (and, similarly, v_2 is not critical).*

Proof. As v_1 and v_2 are not in W, and W is a cover of G, we infer that $(v_1, v_2) \notin E$. Let $E^* = E \cup \{(v_1, v_2)\}$, and define $G^* = \langle V, E^*\rangle$. We now consider a hypothetical execution of the algorithm $\mathcal{A}$ on G^* and denote $W^* = \mathcal{A}(G^*)$. The set W^* must cover the edge (v_1, v_2) and thus $W^* \neq W$.

If v_1 is critical for G, then, by Claim 3.8, the sets of minimum-size vertex cover of G and G^* are equal. However, since $\mathcal{A}$ is deterministic and private, and $W \neq W^*$, the set of minimum vertex covers of G and G^* must be different. Therefore, v_1 is not critical for G. □

In Figure 3.4, we present Algorithm `Relevant-Noncritical`. It takes a graph $G = \langle V, E\rangle$ and a vertex $v \in V$ as inputs and uses a private $n^{1-\epsilon}$-approximation

algorithm $\mathcal{A}$ to solve the Relevant-Noncritical problem.

Algorithm `Relevant-Noncritical`

INPUT AND OUTPUT: See Definition 3.6.

1. Let I be a set of vertices of size $(4n)^{1/\epsilon} - 2n$.
2. Construct the graph G_2 from G and I as in Definition 3.9.
3. Execute $\mathcal{A}$ on G_2 and denote $W_2 = \mathcal{A}(G_2)$.
4. Choose any vertex $v' \in I \backslash W_2$ (there must be at least two such vertices as the approximation algorithm cannot return all the sets I).
5. Construct $G(\overset{v'}{\wedge})$ from G, I, and v' as in Definition 3.9.
6. Execute $\mathcal{A}$ on $G(\overset{v'}{\wedge})$ and denote $W(\overset{v'}{\wedge}) = \mathcal{A}(G(\overset{v'}{\wedge}))$.
7. If $W_2 \neq W(\overset{v'}{\wedge})$ return "NOT CRITICAL." Else return "RELEVANT."

FIG. 3.4. *Algorithm* `Relevant-Noncritical` *for a private deterministic* $\mathcal{A}$.

The correctness of Algorithm `Relevant-Noncritical` stems from the following claims.

CLAIM 3.13. *If* $W_2 \neq W(\overset{v'}{\wedge})$, *then* v *is not critical for* G.

Proof. Assume toward contradiction that v is critical for G. By Claim 3.10, the graphs G_2 and $G\left(\overset{v'}{\wedge}\right)$ have the same set of minimum vertex covers. Hence, from the privacy of $\mathcal{A}$, we get that $W_2 = \mathcal{A}(G_2) = \mathcal{A}\left(G\left(\overset{v'}{\wedge}\right)\right) = W(\overset{v'}{\wedge})$, contradicting $W_2 \neq W(\overset{v'}{\wedge})$. ∎

We next show that there are at least two vertices we can choose in step 4 of the algorithm.

CLAIM 3.14. *Let* $\epsilon > 0$ *be a constant. There exist at least two vertices in* $I \setminus W_2$.

Proof. Let $N = |I| + 2n = (4n)^{1/\epsilon}$ be the number of vertices in G_2. The size of the minimum vertex cover of G_2 is twice the size of the minimum vertex cover of G; thus, it is at most $2(n-1)$. Since $\mathcal{A}$ is an $N^{1-\epsilon}$-approximation algorithm for vertex cover, the size of $W_2 = \mathcal{A}(G_2)$ is at most

$$\begin{aligned} 2(n-1) \cdot N^{1-\epsilon} &\leq 2n \cdot N^{1-\epsilon} - 2 \;=\; 2n \cdot ((4n)^{1/\epsilon})^{1-\epsilon} - 2 \;=\; \frac{(4n)^{1/\epsilon}}{2} - 2 \\ &\leq (4n)^{1/\epsilon} - 2n - 2 \;=\; |I| - 2. \end{aligned}$$

Consequently, there are at least two vertices in $I \setminus W_2$. ∎

CLAIM 3.15. *If* $W_2 = W(\overset{v'}{\wedge})$, *then* v *is relevant for* G.

Proof. As $W_2 = W(\overset{v'}{\wedge})$ and $v' \notin W_2$, we get $v' \notin W(\overset{v'}{\wedge})$. Since there are at least two vertices in $I \setminus W_2$, we can apply Claim 3.12 and deduce that the vertex v' is not critical for $G\left(\overset{v'}{\wedge}\right)$. Applying Claim 3.11, we infer that v is relevant for G. ∎

We extend the techniques described in this section to get an impossibility result with respect to a weaker notion of private approximation defined in section 4 (see Theorem 5.1).

3.2. Impossibility results for private approximation of exact 3SAT. Similar results are obtained for private approximation of maxE3SAT as stated in the following theorem.

THEOREM 3.16. *Let $\epsilon > 0$ be a constant.*

1. *If $\mathcal{P} \neq \mathcal{NP}$, then there is no* deterministic *private $1/n^{1-\epsilon}$-approximation algorithm for the search problem of* maxE3SAT.
2. *If $\mathcal{RP} \neq \mathcal{NP}$, then there is no* randomized *private $1/n^{1-\epsilon}$-approximation algorithm for the search problem of* maxE3SAT.

See Appendix C for a proof of the deterministic case. The proof of the randomized case follows, with modifications similar to those made in Appendix B for minVC.

4. Algorithms that leak a little information. As demonstrated in the previous section, in some cases it is impossible to design an efficient algorithm which is private with respect to a privacy structure $\mathcal{R}$. However, letting $\mathcal{R}'$ be a refinement of $\mathcal{R}$, we view a private algorithm with respect to $\mathcal{R}'$ as a private algorithm with respect to $\mathcal{R}$ that *leaks* information. The amount of information leaked is quantified according to the relation between $\mathcal{R}$ and $\mathcal{R}'$.

DEFINITION 4.1 (k-refinement). *Let $\mathcal{R}$ and $\mathcal{R}'$ be two privacy structures over $\{0,1\}^*$ and $k : \mathbb{N} \to \mathbb{N}$. We say that $\mathcal{R}'$ is a $k(n)$-refinement of $\mathcal{R}$ if $\mathcal{R}' \subseteq \mathcal{R}$ and for every $n \in \mathbb{N}$ every equivalence class of $\mathcal{R}$ of strings of size n is a union of at most $2^{k(n)}$ equivalence classes of $\mathcal{R}'$.*

DEFINITION 4.2 (algorithms leaking $k(n)$-bits). *Let $\mathcal{R}$ be a privacy structure. A randomized polynomial-time algorithm $\mathcal{A}$* leaks at most $k(n)$ bits *with respect to $\mathcal{R}$ if there exists a privacy structure $\mathcal{R}'$ such that* (i) *$\mathcal{R}'$ is a $k(n)$-refinement of $\mathcal{R}$, and* (ii) *$\mathcal{A}$ is private with respect to $\mathcal{R}'$.*

Informally, the algorithm leaks only the subclass of the equivalence class of the input; this subclass can be described by k bits; thus, the algorithm leaks at most k bits.

4.1. Solution-list algorithms. Solution-list algorithms are algorithms whose outcome is always in a small predetermined set, that is, in a set that is a function of $|x|$ but not of x itself. With respect to privacy, solution-list algorithms are valuable as they leak only a few bits with respect to any privacy structure—at most logarithmic in the number of their possible outcomes.

DEFINITION 4.3 (solution-list algorithm). *We say that a deterministic algorithm $\mathcal{A}$ is a $K(n)$-*solution-list *algorithm if for every $n \in \mathbb{N}$*

$$|\{\, y : \exists x \in \{0,1\}^n \text{ such that } \mathcal{A}(x) = y \,\}| \leq K(n).$$

That is, a solution-list algorithm is an algorithm that, for every input size n, "chooses" its outputs from a set of at most $K(n)$ possible outcomes.

We define the universal relation, denoted $\mathcal{U}^* = \cup_{n \in \mathcal{N}} \mathcal{U}^*_n$, as the privacy structure where every two instances of the same size are equivalent. Note that any privacy structure is a refinement of $\mathcal{U}^*$; hence if an algorithm is private with respect to $\mathcal{U}^*$, it is also private with respect to any privacy structure,[5] and similarly, if an algorithm leaks at most k bits with respect to $\mathcal{U}^*$, then such is the case with respect to any privacy structure.

OBSERVATION 4.4. *Any $K(n)$-solution-list algorithm leaks at most* $\log K(n)$ *bits with respect to $\mathcal{U}^*$, and, in particular, the algorithm leaks at most* $\log K(n)$ *bits with respect to every privacy structure.*

[5] An algorithm $\mathcal{A}$ is private with respect to $\mathcal{U}^*$ if only the instance size may be learned from its outcome.

4.2. Solution-list algorithms for exact 3SAT. In this section we present a $(7/8-\epsilon)$-approximation algorithm for maximum satisfiability on exact 3CNF formulae that leaks a little information, i.e., $O(\log\log n+\log 1/\epsilon)$ bits. The algorithm is a solution-list algorithm as defined in Definition 4.3. The approximation in our algorithm is nearly optimal, as by the result of Håstad [18], if $\mathcal{P}\neq\mathcal{NP}$, then there is no polynomial-time $(7/8+\epsilon)$-approximation algorithm for this problem.[6]

We start with a simple motivating example. Consider the following simple algorithm for the max-SAT problem.

If 0^n satisfies at least half of the clauses in ϕ, return 0^n. Otherwise, return 1^n.

For every clause, either 0^n or 1^n satisfy the clause. Thus, 0^n or 1^n satisfy at least half of the clauses in ϕ, and this is a 1/2-approximation of max-SAT. Since there are only two possible answers, this algorithm leaks at most one bit.

CLAIM 4.5. *There is a* 7/8*-approximation algorithm for* maxE3SAT *that leaks at most* $O(\log n)$ *bits with respect to* $\mathcal{R}_{\text{maxE3SAT}}$. *Furthermore, for every* $\epsilon>0$, *there is a* $(7/8-\epsilon)$*-approximation algorithm for* maxE3SAT *that leaks at most* $O(\log\log n+\log 1/\epsilon)$ *bits with respect to* $\mathcal{R}_{\text{maxE3SAT}}$.

Proof. We first describe the 7/8-approximation algorithm. Toward this goal, we construct for every n a list of poly(n) assignments such that for every exact 3CNF formula with n variables there is an assignment in the list that satisfies at least 7/8 of the clauses of the formula. Furthermore, there is an efficient algorithm that generates this list. Thus, the 7/8-approximation algorithm, with input ϕ, a formula with n variables, constructs this list and chooses the first assignment in the list that satisfies at least 7/8 of the clauses in ϕ.

We next explain how to construct the list, using ideas of the randomized 7/8-approximation algorithm of Johnson [20]. Fix a clause with three different literals. If we pick an assignment at random, then with probability at least 7/8 it satisfies the clause. Now, fix any exact 3CNF formula. If we pick an assignment at random, then the expected fraction of satisfied clauses is at least 7/8. Thus, there exists at least one assignment that satisfies a fraction of at least 7/8 of the clauses in the formula. This is true even if we pick the assignments from a 3-wise independent space. As there is a 3-wise independent space of size $O(n^3)$, this implies the existence of the list. To generate the assignments we can use any of the constructions of 3-wise independent spaces, e.g., the construction based on polynomials (see, e.g., [23, 1, 8]).

We next describe the $(7/8-\epsilon)$-approximation algorithm. As in the previous case, it suffices to show how to efficiently construct, for every n and $\epsilon>0$, a list of poly$(\frac{\log n}{\epsilon})$ assignments such that for every exact 3CNF formula with n variables there is an assignment in the list that satisfies at least $7/8-\epsilon$ of the clauses of the formula. To construct the list, notice that if we pick an assignment from an almost 3-wise independent space, that is, from an $(\epsilon,3)$-wise independent space, then the probability that a given clause is satisfied is at least $7/8-\epsilon$. Thus, the expected fraction of satisfied clauses is at least $7/8-\epsilon$, and there exists at least one assignment that satisfies a fraction of at least $7/8-\epsilon$ of the clauses in the formula. There are $(\epsilon,3)$-wise independent spaces of size poly$(\frac{\log n}{\epsilon})$. To generate the assignments we can use any of the constructions of [25, 2]. □

The next claim shows that the size of the solution list used in the proof of Claim 4.5 is nearly optimal.

[6] Any $(7/8+\epsilon)$-approximation solution-list algorithm that uses poly(n) solutions would imply that $\mathcal{NP}\subseteq\mathcal{P}/\text{poly}$ even if the list cannot be efficiently constructed.

CLAIM 4.6. *Every solution-list algorithm for* maxE3SAT *that achieves approximation ratio better than* $1/2$ *uses at least* $\log n - 1$ *solutions.*

Proof. Assume a solution-list algorithm for maxE3SAT, and let $a_1, \ldots, a_t$, where $t < \log n - 1$, be its list of possible output assignments on formulae over n variables $x_1, \ldots, x_n$. To each variable x_i we assign a *label* that is the concatenation of the truth values assigned to x_i by the t assignments $\langle a_1(x_i), \ldots, a_t(x_i)\rangle$. As there are at most 2^t different labels and $n/2^t > 2$, there exist three distinct variables $x_{i_1}, x_{i_2}, x_{i_3}$ that share the same label; i.e., for all $1 \le j \le t$ it holds that $a_j(x_{i_1}) = a_j(x_{i_2}) = a_j(x_{i_3})$.

Consider the formula $\phi = (x_{i_1} \vee x_{i_2} \vee x_{i_3}) \wedge (\neg x_{i_1} \vee \neg x_{i_2} \vee \neg x_{i_3})$. It is easy to see that ϕ is satisfied by exactly those assignments which do not assign the same truth value to all three variables $x_{i_1}, x_{i_2}, x_{i_3}$. However, as this is not the case for any of the t assignments, each of them satisfies exactly one clause in ϕ, achieving an approximation factor of at most $1/2$. □

4.3. Solution-list algorithms for vertex cover. In this section we present almost private search algorithms for minimum vertex cover. These algorithms are interesting but not as dramatic as the algorithms for maxE3SAT. For any $0 < \epsilon < 1$, there is an $n^{1-\epsilon}$-approximation algorithm that leaks $2n^\epsilon$ bits. The algorithms are solution-list algorithms, and we will prove that no solution-list algorithm can do better for this problem.

CLAIM 4.7. *For every* $0 < \epsilon < 1$*, there is an* $n^{1-\epsilon}$*-approximation algorithm for the minimum vertex cover problem which leaks at most* $2n^\epsilon$ *bits.*

Proof. The algorithm proceeds as follows:

INPUT: A graph G with n vertices.

1. Execute any deterministic 2-approximation algorithm for VC on G, and get a cover C.
2. Let $\ell \leftarrow 2n^\epsilon$.
3. Partition the n vertices into ℓ fixed sets, $V_1, \ldots, V_\ell$, each of size $n/\ell = n^{1-\epsilon}/2$.
4. Let $C' \leftarrow \bigcup_{\{i:V_i \cap C \neq \emptyset\}} V_i$.
5. Return C'.

The algorithm first finds a small cover. Then, if V_i contains at least one vertex in this cover, the algorithm returns the entire set V_i. This implies that the size of C' is at most $|C|n^{1-\epsilon}/2$, and since $|C|$ is at most twice the size of the minimum vertex cover, this algorithm is an $n^{1-\epsilon}$-approximation algorithm.

Notice that the algorithm has 2^ℓ possible outputs (it chooses only which of the sets V_i is in its output). That is, this is a solution-list algorithm with a list of size 2^ℓ; thus, it leaks at most $\ell = 2n^\epsilon$ bits. □

The private algorithms for maxE3SAT and for minVC that we presented are solution-list algorithms. For maxE3SAT, the algorithm generates the entire list and chooses the best candidate in the list. For minVC this is not possible as the size of the list is big. The deterministic algorithm generates a "good" candidate from the list without generating the entire list.

We next claim that any solution-list algorithm for minVC cannot use a shorter list than the algorithm we presented (up to a constant factor).

CLAIM 4.8. *Any solution-list algorithm that* $n^{1-\epsilon}$*-approximates* minVC *uses at least* $2^{n^\epsilon/6}$ *solutions.*

Proof. Assume that there is a list of covers that $n^{1-\epsilon}$-approximates minVC; that is, for every graph with n vertices and minimum vertex cover of size d, there exists

a cover of size at most $n^{1-\epsilon}d$ in the list. We construct a "big" family of graphs and show that every cover in the list covers "a few" graphs in the family; thus the size of the list must be "big."

Consider the following family of graphs. Each graph is defined by a subset I of size $d \stackrel{\text{def}}{=} n^\epsilon/6$. The graph G_I contains all edges between I and $V \setminus I$. Notice that the number of graphs in this family is

$$\binom{n}{d} \geq (n/d)^d. \tag{4.1}$$

The set I of size d is a vertex cover of the graph G_I. Since we assume that the list $n^{1-\epsilon}$-approximates minVC, there is a cover in the list that covers G_I and contains at most $n^{1-\epsilon}d = n^{1-\epsilon}n^\epsilon/6 = n/6$ vertices. However, every vertex cover of G_I contains either all vertices in I (and possibly vertices from $V \setminus I$) or all vertices of $V \setminus I$ (and possibly vertices from I). Since $|V \setminus I| = n - d > n/6$, this cover must contain I. The number of graphs in the family that a given cover of size at most $n/6$ covers is at most

$$\binom{n/6}{d} \leq \left(\frac{en/6}{d}\right)^d. \tag{4.2}$$

Thus, by (4.1) and (4.2), the number of covers in the list is at least

$$\frac{(n/d)^d}{(en/6d)^d} \geq 2^d = 2^{n^\epsilon/6}. \qquad \square$$

We emphasize that Claim 4.8 applies only to the size of lists used by solution-list algorithms and does not imply that there are no polynomial-time $n^{1-\epsilon}$-approximation algorithms that leak $o(n^\epsilon)$ bits.[7] However, it was proven in [5] that such approximation algorithms do not exist if $\mathcal{RP} \neq \mathcal{NP}$.

5. Impossibility result for vertex cover approximation that leaks $\log n$ bits. In this section we show that it is unlikely that there is an efficient approximation algorithm for minVC that leaks $\log n$ bits of information. Specifically, if there is such an $n^{1-\epsilon}$-approximation algorithm $\mathcal{A}$ that leaks at most $\frac{\epsilon}{6}\log n$ bits, then $\mathcal{RP} = \mathcal{NP}$.

THEOREM 5.1. *Let $\epsilon > 0$ be a constant. If $\mathcal{RP} \neq \mathcal{NP}$, then there is no randomized $n^{1-\epsilon}$-approximation algorithm for the search problem of* minVC *that leaks at most $\frac{\epsilon}{6}\log n$ bits.*

As in the proof of Theorem 3.2, we assume the existence of such an approximation algorithm and deduce an algorithm that solves vertex cover. Again, we do this by designing an algorithm that solves the Relevant-Noncritical problem (see Definition 3.6) and thus, by Claim 3.7, solves vertex cover. This algorithm, given the input G and v and an oracle access to an approximation algorithm $\mathcal{A}$ that leaks k bits, applies $\mathcal{A}$ on a set of inputs and decides whether v is relevant or noncritical for G according to the results. Note that Algorithm `Relevant-Noncritical` for the (perfectly) private case is not applicable; here, even assuming $\mathcal{A}$ is deterministic, the fact that $\mathcal{A}(G_1) \neq \mathcal{A}(G_2)$ does not directly imply that G_1 and G_2 are not equal under $\mathcal{R}_{\text{minVC}}$. However, as $\mathcal{A}$ leaks at most k bits, if there are $2^k + 1$ graphs with different outputs, then at least two of them are not equivalent.

[7] It can be proved that the algorithm we presented leaks $\Omega(n^\epsilon)$ bits.

We first deal in section 5.1 with a deterministic $\mathcal{A}$ that leaks at most one bit; then we deal in section 5.2 with a deterministic $\mathcal{A}$ that leaks at most $O(\log n)$ bits. The proof of the impossibility result for a randomized $\mathcal{A}$ that leaks at most $O(\log n)$ bits is deferred to Appendix B. We believe that presenting the proof through these stages is helpful for the reader.

Our inputs for Algorithm `Relevant-Noncritical` are generally constructed from a number of copies of the original input graph G and a big set of isolated vertices I. In each such graph we connect the different copies of the input vertex v with vertices from I and sometimes connect two different vertices from I. We will use the following notation to address these graphs.

DEFINITION 5.2. *Let $G = \langle V, E\rangle$ be a graph, $v \in V$ be some fixed vertex, I be a set of vertices (where $I \cap V = \emptyset$), t, m be indices such that $0 \le t \le m$, and $i_1, \ldots, i_m, j_1, \ldots, j_{2t} \in I$ be distinct vertices. The graph*

$$G\left(\begin{smallmatrix} j_1 & j_2 & & & j_{2t-1} & j_{2t} & i_{t+1} & & i_m \\ \vee & & & & \vee & & \wedge & & \wedge \\ i_1 & & \cdots & & i_t & & & \cdots & \end{smallmatrix}\right)$$

is defined as follows: its vertex set is $(V \times \{1, \ldots, 2m\}) \cup I$, and its edges are $E = E_m \cup E_\vee \cup E_\wedge$, where

$$\begin{aligned} E_m &\stackrel{def}{=} \{(\langle u, \ell\rangle, \langle w, \ell\rangle) : (u, w) \in E, \ell \in \{1, \ldots, m\}\}, \\ E_\vee &\stackrel{def}{=} \{(i_\ell, j_{2\ell-1}), (i_\ell, j_{2\ell}) : \ell \in \{1, \ldots, t\}\}, \text{ and} \\ E_\wedge &\stackrel{def}{=} \{(\langle v, 2\ell-1\rangle, i_\ell), (\langle v, 2\ell\rangle, i_\ell) : \ell \in \{t+1, \ldots, m\}\}. \end{aligned}$$

Informally, the graph has $2m$ copies of G (see Figure 5.1). It has t "vees," where the ℓth vee is associated with copies $2\ell-1$ and 2ℓ of G. It has $m-t$ "wedges," where the ℓth wedge connects i_ℓ to the copies of v in copies $2\ell-1$ and 2ℓ of G, for $t < \ell \le m$.

We next present two combinatorial lemmas, whose role is similar to Claims 3.10 and 3.11 in the case where $\mathcal{A}$ was perfectly private.

CLAIM 5.3. *Let $1 \le t \le m$, and $i_1, \ldots, i_m, j_1, \ldots, j_{2t}, i'_{t+1}, \ldots, i'_m \in I$ be distinct vertices. Furthermore, let*

$$H \stackrel{def}{=} G\left(\begin{smallmatrix} j_1 & j_2 & & & j_{2t-1} & j_{2t} & i_{t+1} & & i_m \\ \vee & & & & \vee & & \wedge & & \wedge \\ i_1 & & \cdots & & i_t & & & \cdots & \end{smallmatrix}\right) \quad \text{and} \quad H' \stackrel{def}{=} G\left(\begin{smallmatrix} j_1 & j_2 & & & j_{2t-1} & j_{2t} & i'_{t+1} & & i'_m \\ \vee & & & & \vee & & \wedge & & \wedge \\ i_1 & & \cdots & & i_t & & & \cdots & \end{smallmatrix}\right).$$

If v is critical for G, then $H \equiv_{\mathcal{R}_{\text{minVC}}} H'$.

Proof. By Claim 3.10, the graphs H and H' are unions of graphs that are equivalent; thus, H and H' are equivalent. ∎

CLAIM 5.4. *Let $0 \le t' < t \le m$ and $i_1, \ldots, i_m, j_1, \ldots, j_{2t} \in I$ be distinct vertices. Furthermore, let*

$$\begin{aligned} H &\stackrel{def}{=} G\left(\begin{smallmatrix} j_1 & j_2 & & & j_{2t'-1} & j_{2t'} & & & j_{2t-1} & j_{2t} & i_{t+1} & & i_m \\ \vee & & & & \vee & & & & \vee & & \wedge & & \wedge \\ i_1 & & \cdots & & i_{t'} & & \cdots & & i_t & & & \cdots & \end{smallmatrix}\right) \quad \text{and} \\ H' &\stackrel{def}{=} G\left(\begin{smallmatrix} j_1 & j_2 & & & j_{2t'-1} & j_{2t'} & i_{t'+1} & & i_m \\ \vee & & & & \vee & & \wedge & & \wedge \\ i_1 & & \cdots & & i_{t'} & & & \cdots & \end{smallmatrix}\right). \end{aligned}$$

If v is nonrelevant for G, then $H \equiv_{\mathcal{R}_{\text{minVC}}} H'$.

Proof. Note that the vertices $i_1, \ldots, i_m$ are critical for both H and H': It is straightforward that i_ℓ is critical for $\begin{smallmatrix} j_{2\ell-1} & j_{2\ell} \\ \vee & \\ i_\ell & \end{smallmatrix}$. By Claim 3.11, vertex i_ℓ is critical for $\begin{smallmatrix} i_\ell \\ \wedge \end{smallmatrix}$. Therefore, the two graphs have the same set of minimum vertex covers; hence they are equivalent under $\mathcal{R}_{\text{minVC}}$. ∎

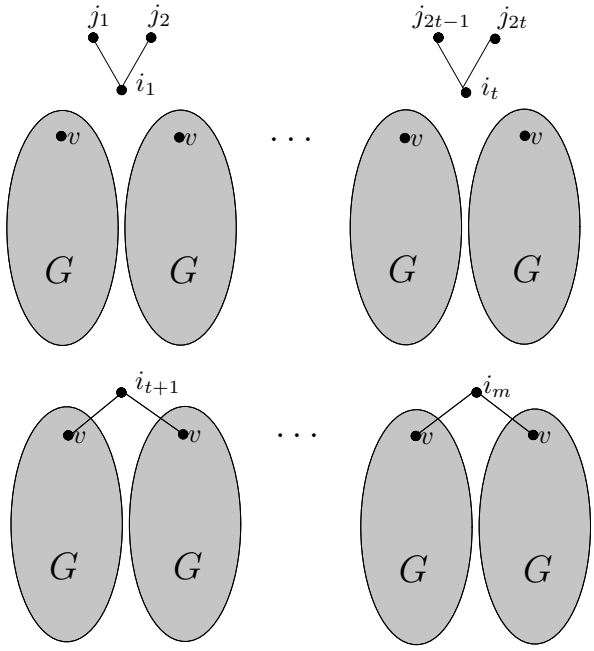


FIG. 5.1. *The graph* $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \ \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$.

In both claims we have the same graph H. In Claim 5.3, the graph H' has the same "vees" as H; however, the "wedges" have different vertices from I (namely, $i'_1, \ldots, i'_{t-1}$). In Claim 5.4, the sequence of i's is the same in H and H'; however, there are "vees" in H' in some places where there are "wedges" in H.

5.1. Handling one bit leakage. To simplify our presentation, we first present Algorithm `RelNonCrit—one bit` in Figure 5.2. This algorithm assumes that the deterministic algorithm $\mathcal{A}$ leaks at most one bit. This case is simpler and describes some of the ideas used for the $\log n$ bits leakage case.

CLAIM 5.5. *Algorithm* `RelNonCrit—one bit` *is correct.*

Proof. By Claim 5.3, if v is critical for G, then all graphs considered in step 1 of the algorithm are equivalent. As $\mathcal{A}$ leaks at most 1 bit, there can be at most two different answers on equivalent graphs. Hence, if there are more than two different answers in step 1, vertex v is noncritical for G. Similarly, if there are more than two different answers in step 2, vertex v is noncritical for G. Thus, if the algorithm outputs "Noncritical," vertex v is noncritical for G, and the algorithm is correct.

Else, let i_1, j_1, j_2 be the vertices chosen in step 2 of the algorithm, and fix any $i_2, j_3, j_4 \in I$ that did not appear in the result of any execution of $\mathcal{A}$ in Algorithm `RelNonCrit—one bit` (both in step 1 and in step 2), and note the following.

1. $\mathcal{A}(G\left(\overset{i_1}{\wedge} \ \overset{i_2}{\wedge}\right))$ *does not contain* any of $\{i_1, j_1, j_2\}$ (by the choice of the algorithm in step 2) and *does not contain* any of $\{i_2, j_3, j_4\}$.
2. $\mathcal{A}(G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \ \overset{i_2}{\wedge}))$ *contains* at least one of $\{i_1, j_1\}$ (since the graph contains the edge (i_1, j_1)) and *does not contain* any of $\{i_2, j_3, j_4\}$.
3. $\mathcal{A}(G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \ \overset{j_3 j_4}{\underset{i_2}{\vee}}))$ *contains* at least one of $\{i_2, j_3\}$ (since the graph contains the edge (i_2, j_3)).

Algorithm `RelNonCrit`—`one bit`

INPUT: A graph $G = \langle V, E \rangle$ and a vertex $v \in V$.
OUTPUT: One of the following: (i) The vertex v is relevant for G. (ii) The vertex v is not critical for G.

1. Execute $\mathcal{A}$ on all the graphs of the form $G(\overset{i_1}{\wedge}\ \overset{i_2}{\wedge})$, where $i_1, i_2 \in I$.
   (a) If these executions result in more than two different answers, return "Noncritical,"
   (b) Else, remove from I all the vertices that appeared in any of the above results.
2. Pick arbitrary $i_1, j_1, j_2 \in I$ and execute $\mathcal{A}$ on all the graphs of the form $G(\overset{j_1 j_2}{\underset{i_1}{\vee}}\ \overset{i_2}{\wedge})$, where $i_2 \in I \setminus \{i_1, j_1, j_2\}$.
   (a) If these executions result in more than two different answers, return "Noncritical."
   (b) Else, return "Relevant."

FIG. 5.2. *Algorithm* `RelNonCrit`—`one bit`.

Thus, these three results of $\mathcal{A}$ are all different. However, by Claim 5.4, if the vertex v is not relevant for G, then the three graphs are equivalent. Since $\mathcal{A}$ leaks at most one bit, there cannot be three different answers on three equivalent graphs. Thus, if the algorithm outputs "Relevant," vertex v is relevant for G, and the algorithm is correct. □

Similarly to the case where $\mathcal{A}$ is perfectly private, it is enough to set $|I| = O((4n)^{1/\epsilon} - 2n)$ to ensure there are enough vertices in I that are not returned by $\mathcal{A}$ (see Claim 3.14). As $|I|$ is polynomial in n, the number of calls to $\mathcal{A}$ in the algorithm is polynomial in n, and each execution of $\mathcal{A}$ runs in polynomial time. Thus, we have proved that if $\mathcal{P} \neq \mathcal{NP}$, then there is no deterministic $n^{1-\epsilon}$-approximation algorithm for vertex cover that leaks at most one bit.

5.2. Handling $\log n$-bits leakage. We next want to accommodate a deterministic algorithm $\mathcal{A}$ that leaks $\log n$ bits. Using an algorithm similar to that used in the one-bit leakage case with graphs that contain more copies of G, we can handle Algorithm $\mathcal{A}$ which leaks more bits. However, if $\mathcal{A}$ leaks $\omega(1)$ bits, the number of executions of $\mathcal{A}$ will be too big. To reduce the number of calls to $\mathcal{A}$, we choose the vertices from I at random.

In Figure 5.3, we present Algorithm `RelNonCrit`—log n `bits`, which assumes that Algorithm $\mathcal{A}$ is a deterministic $n^{1-\epsilon}$-approximation algorithm that leaks at most $\frac{\epsilon}{6} \log n$ bits. Algorithm `RelNonCrit`—log n `bits` is a randomized algorithm which returns a correct answer with probability at least $1 - \delta$ for some $0 < \delta < 1$. Given a graph G with n vertices, we construct the graphs from Definition 5.2. These graphs have $N \stackrel{\text{def}}{=} (12n/\delta)^{2/\epsilon}$ vertices, $2m$ copies of G, and a disjoint set of vertices I. We choose the number of copies to be $2m$, where

$$m \stackrel{\text{def}}{=} N^{\epsilon/6} = (12n/\delta)^{1/3}. \tag{5.1}$$

The size of the set I is $|I| = N - 2mn$. In the proof of Claim 5.7 it will become clear why we made these choices.

In the following we assume that, on graphs with n vertices, $\mathcal{A}$ leaks at most $k(n) = \frac{\epsilon}{6} \log n$ bits. Recall that we execute $\mathcal{A}$ on graphs with N vertices; thus, the

Algorithm `RelNonCrit`—log n `bits`

INPUT: A graph $G = \langle V, E \rangle$, a vertex $v \in V$, and a number $0 < \delta < 1$.
OUTPUT: One of the following: (i) The vertex v is relevant for G. (ii) The vertex v is not critical for G. The algorithms errs with probability at most δ.

1. Choose distinct $i_1, \ldots, i_m, j_1, \ldots, j_{2m}$ at random from I.
2. For $t = 0$ to $m - 1$ do:
   (a) Let $G_t = G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$.
   (b) If $\mathcal{A}(G_t) \cap \{i_{t+1}, j_{2t+1}\} \neq \emptyset$, then return "Noncritical."
3. ($*$ all m sets do not contain the two vertices $*$)
   RETURN "Relevant."

FIG. 5.3. *Algorithm* `RelNonCrit`—log n `bits`.

approximation ratio is $N^{1-\epsilon}$, and the leakage is bounded by $k(N) = \frac{\epsilon}{6} \log N = \log m$. The next sequence of claims asserts the correctness of Algorithm `RelNonCrit`—log n `bits`.

CLAIM 5.6. *If v is nonrelevant, then Algorithm* `RelNonCrit`—log n `bits` *does not return "Relevant."*

Proof. Let $G_m = G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2m-1} j_{2m}}{\underset{i_m}{\vee}})$. If the algorithms returns "Relevant," then the for loop finishes. Thus, the sets $\mathcal{A}(G_0), \ldots, \mathcal{A}(G_m)$ are $m+1 = 2^{k(n)} + 1$ different sets; for $t' < t$ the cover $\mathcal{A}(G_t)$ must contain at least one of the vertices $i_{t'+1}, j_{2t'+1}$ (as it must cover the edge $(i_{t'+1}, j_{2t'+1})$), while $\mathcal{A}(G_{t'})$ contains none of them. Thus, there are $2^{k(n)} + 1$ graphs with pairwise different answers of $\mathcal{A}$, and, since $\mathcal{A}$ leaks at most $k(n)$ bits, at least two of the graphs are not equivalent according to $\mathcal{R}_{\text{minVC}}$. Thus, by Claim 5.4, the vertex v is relevant for G. ∎

CLAIM 5.7. *Let $0 \leq t \leq m$. For every $i_1, \ldots, i_t, j_1, \ldots, j_{2t} \in I$, consider the following experiment: choose $i_{t+1}, \ldots, i_m$ and j_{2t+1} at random with uniform distribution from I and return* 1 *if*

$$\mathcal{A}\left(G\left(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge}\right)\right) \cap \{i_{t+1}, j_{2t+1}\} \neq \emptyset.$$

If v is critical, then the probability that this experiment returns 1 *is at most δ/m.*

Proof. We first prove that we chose N, the number of vertices in the graphs we construct, such that the size of each cover $\mathcal{A}$ returns is at most $|I|/\alpha$, where $\alpha \stackrel{\text{def}}{=} 2m^2/\delta$. We will need the following requirement for our computations:

$$|I| = N - 2mn \geq N/2. \tag{5.2}$$

That is, we need to show that $N/2 \geq 2mn$. As $m = N^{\epsilon/6} \leq N^{1/6}$, it suffices to require that $N \geq (4n)^{6/5}$. By the choice of N

$$N = (12n/\delta)^{2/\epsilon} \geq (12n)^2 \geq (4n)^{6/5}$$

(since $0 < \delta, \epsilon \leq 1$); thus (5.2) holds.

We next upper-bound the size of the covers that $\mathcal{A}$ outputs. The size of a minimum vertex cover of these graphs is at most $(2n+1)m \leq 3nm$. Since $\mathcal{A}$ is an $N^{1-\epsilon}$-approximation algorithm for vertex cover, the size of its output is at most $3 \cdot nm \cdot N^{1-\epsilon}$.

Recall that $N = (\frac{12n}{\delta})^{2/\epsilon}$ and $m = N^{\epsilon/6}$, and thus, by (5.1),

$$12nm^3 = \delta(12n/\delta) \cdot N^{\epsilon/2} = \delta((12n/\delta)^{2/\epsilon})^{\epsilon/2} \cdot N^{\epsilon/2} = \delta N^{\epsilon/2} N^{\epsilon/2} = \delta N^{\epsilon}. \tag{5.3}$$

Therefore, the size of the cover returned by $\mathcal{A}$ is at most

$$3 \cdot nm \cdot N^{1-\epsilon} = \frac{3nmN}{N^{\epsilon}} \leq \frac{6nm|I|}{N^{\epsilon}} = \frac{12nm^3}{\delta N^{\epsilon}} \cdot \frac{\delta |I|}{2m^2} = \frac{\delta |I|}{2m^2} = \frac{|I|}{\alpha},$$

where the inequality above follows from (5.2) and the last two equalities follow (5.3) and the definition of α. Thus, with probability at most $1/\alpha$, a random vertex from I is in a given answer.

We now analyze the probability that the experiment returns 1. If v is critical, then, by Claim 5.3, every two choices of $i_{t+1}, \ldots, i_m$ result in equivalent graphs according to $\mathcal{R}_{\text{minVC}}$. Since $\mathcal{A}$ leaks at most $k(N) = \frac{\epsilon}{6} \log N = \log m$ bits, there are at most $2^{k(N)} = m$ different answers for all the different choices of $i_{t+1}, \ldots, i_m$. Thus, the size of the union of all the answers is at most $\frac{m|I|}{\alpha}$. The probability that at least one of the vertices i_{t+1}, j_{2t+1} is in the union of the m answers is, by the union bound, at most $\frac{2m}{\alpha}$. Thus, the probability of the experiment returning 1 is at most $\frac{2m}{\alpha} = \delta/m$, as required. ∎

Claim 5.8. *If vertex v is critical, then the probability that Algorithm* `RelNonCrit`—log n `bits` *returns "Noncritical" is at most δ.*

Proof. Algorithm `RelNonCrit`—log n `bits` repeats the experiment of Claim 5.7 m times and returns "Noncritical" if one of the experiments returns 1. Thus, by Claim 5.7 and the union bound, if vertex v is critical, then the probability that Algorithm `RelNonCrit`—log n `bits` returns "Noncritical" is at most δ. ∎

Algorithm `RelNonCrit`—log n `bits` executes the polynomial-time algorithm $\mathcal{A}$ m times on graphs with N vertices, where $N = (12n/\delta)^{2/\epsilon}$. Thus, we have the following claim.

Claim 5.9. *If $\mathcal{A}$ runs in polynomial time and $0 \leq \epsilon < 1$ is a constant, then the running time of Algorithm* `RelNonCrit`—log n `bits` *is* poly(n/δ).

We summarize the results of this section in the next lemma.

Lemma 5.10. *Let $\epsilon > 0$ be a constant. If $\mathcal{RP} \neq \mathcal{NP}$, then there is no deterministic $n^{1-\epsilon}$-approximation algorithm for the search problem of* vertex cover *that leaks at most $\frac{\epsilon}{6} \log n$ bits.*

Proof. Algorithm `Greedy Vertex Cover`, which solves vertex cover, executes at most n times Algorithm `RelNonCrit`—log n `bits` with graphs of size at most n. We execute these calls with $\delta = \frac{1}{4n}$ (where n is the original number of vertices in G); thus, all together, the error is at most $1/4$. By Claim 5.9, the running time of Algorithm `Greedy Vertex Cover` is polynomial. Thus, if there is an $n^{1-\epsilon}$-approximation algorithm for the search problem of vertex cover that leaks at most $k(n) = \frac{\epsilon}{6} \log n$ bits, then there is a polynomial-time randomized algorithm for minimum vertex cover that errs with probability $1/4$. This implies that $\mathcal{NP} \subseteq \mathcal{BPP}$.

To contradict $\mathcal{RP} \neq \mathcal{NP}$, this algorithm is transformed to a one-sided error algorithm for the decision problem of vertex cover: Given $\langle G, s \rangle$, decide if G has a vertex cover of size at most s. The transformation is simple; execute the algorithm for the search problem of vertex cover. If this algorithm returns a set that covers G and its size is at most s, return "yes"; otherwise return "no." ∎

6. Impossibility result for private computation of a search problem in $\mathcal{P}$. An algorithm solving a search problem can return any solution to the problem.

An algorithm solving a search problem *privately* has additional requirements on the solutions that it returns. In this section, we show that these additional requirements can make the problem much harder. That is, we show that there is a relation Q whose search problem is in $\mathcal{P}$; however, unless $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$, there is no polynomial-time private algorithm for Q with respect to $\mathcal{R}_Q$.

DEFINITION 6.1. *Let G be a graph and $C_1, C_2 \subseteq V$. We define the relation Q as follows: $\langle G, C_1\rangle$ and C_2 are in Q (that is, $\langle\langle G, C_1\rangle, C_2\rangle \in Q$) if $|C_1| = |C_2|$ and C_1 and C_2 are cliques in G.*

Clearly, the search problem of Q is easy; given $\langle G, C_1\rangle$, return C_1. Assume that there is a private algorithm for $\mathcal{R}_Q$. That is, if C_1 and C_2 are two disjoint cliques of the same size in a graph G, then a private algorithm has to return the same output distribution on $\langle G, C_1\rangle$ and $\langle G, C_2\rangle$. Intuitively, this implies that given a clique in the graph, there is an efficient algorithm that finds another clique. We will prove that this is impossible unless $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$.

THEOREM 6.2. *If $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$, then there is no polynomial-time private algorithm for the search problem of Q with respect to the privacy structure $\mathcal{R}_Q$.*

Proof. We assume toward contradiction that there exists a polynomial-time randomized private algorithm $\mathcal{A}$ for the search problem of Q with respect to the privacy structure $\mathcal{R}_Q$. We will use $\mathcal{A}$ to prove that the $\mathcal{NP}$-complete problem CLIQUE is in $\mathcal{P}/\text{poly}$. That is, we construct a sequence of polynomial-length advice strings $\langle a_n\rangle_{n\in\mathbb{N}}$ and a polynomial-time algorithm $\mathcal{B}$ such that, given a graph G with n vertices, an integer k, and the advice a_n, Algorithm $\mathcal{B}$ decides if G contains a clique of size k.

Given two graphs $G_0 = \langle V_0, E_0\rangle$ and $G_1 = \langle V_1, E_1\rangle$, where $V_1 \cap V_2 = \emptyset$, we define their disjoint union $G_0 \cup G_1$ as the graph $G = \langle V, E\rangle$, where $V = V_0 \cup V_1$ and $E = E_0 \cup E_1$. Every clique in $G_0 \cup G_1$ is either a clique in G_0 or a clique in G_1. Assume that C_0 and C_1 are cliques of size k in G_0 and G_1, respectively. Then, $\langle G_0 \cup G_1, C_0\rangle$ and $\langle G_0 \cup G_1, C_1\rangle$ have the same set of cliques of size k; that is, they are in $\mathcal{R}_Q$.

Algorithm $\mathcal{A}'$

INPUT: Two graphs G_0, G_1 with disjoint sets of vertices and a set C.
PROMISE: C is a clique in $G_0 \cup G_1$.

1. Execute $\mathcal{A}$ on $\langle G_0 \cup G_1, C\rangle$
   (a) Let j be the index such that $\mathcal{A}$ returns a clique in G_j.
   (b) Return j.

FIG. 6.1. *Algorithm $\mathcal{A}'$, which gets two graphs and a clique, executes $\mathcal{A}$ on their union, and checks in which graph $\mathcal{A}$ returns a clique.*

As a first step, we construct in Figure 6.1 an algorithm $\mathcal{A}'$ that will be used to construct Algorithm $\mathcal{B}$ and the advice strings. This algorithm gets two graphs G_0, G_1 and a clique C in one of them, executes $\mathcal{A}$ on their union, and checks in which graph $\mathcal{A}$ returns a clique. If only one graph G_i has a clique of size $|C|$, then, by the correctness requirement of $\mathcal{A}$, Algorithm $\mathcal{A}'$ must return i. However, if both graphs have a clique of size $|C|$, then, by the privacy requirement of $\mathcal{A}$, the output distribution of $\mathcal{A}'$ is approximately the same when C is a clique in G_0 and when C is a clique in G_1. That is, there is an integer n_0 such that for every pair of graphs with at least n_0 vertices, the difference between the probabilities that $\mathcal{A}'$ returns 0 in the two cases is small—for concreteness, at most 1/3 (the probabilities are taken over the random inputs of

algorithm $\mathcal{A}$).

We construct the advice string a_n as the concatenation of advice strings $a_{n,1}, \ldots, a_{n,n}$, where $a_{n,k}$ is used to decide if a graph G with n vertices has a clique of size k. Fix an integer n and an integer k, where $n \geq n_0$ and $1 \leq k \leq n$. For every graph G with n vertices that has a clique of size k, we fix some clique C_G of size k in G. Let G_0 and G_1 be two graphs that have a clique of size k.

DEFINITION 6.3. *We say that G_0 loses to G_1 if Algorithm $\mathcal{A}'$ returns 1 on input $\langle G_0, G_1, C_{G_0}\rangle$ with probability at least 1/3.*

That is, G_0 loses to G_1 if, when given a clique in G_0, Algorithm $\mathcal{A}$ "magically" manages to return with probability 1/3 a clique in the graph G_1; thus, G_0, C_{G_0} can aid in finding a clique in G_1.

If G_0 does not lose to G_1, then $\mathcal{A}'$ returns 0 with probability at least 2/3, and by the privacy requirement, with probability at least 1/3 Algorithm $\mathcal{A}'$ returns 0 on input $\langle G_0, G_1, C_{G_1}\rangle$.

OBSERVATION 6.4. *Let G_0 and G_1 be two graphs with n vertices that have a clique of size k. If G_0 does not lose to G_1, then G_1 loses to G_0. (It is possible that G_0 loses to G_1 and G_1 loses to G_0.)*

Thus, for every set of graphs with n vertices and a clique of size k, there exists a graph G_0 in the set that loses to at least half the graphs in the set. This is the idea of constructing the advice string $a_{n,k}$.

Construction of $a_{n,k}$

1. $a_{n,k} \leftarrow \emptyset$.
2. Initialize L as the set of all graphs with n vertices that have a clique of size k.
3. While $L \neq \emptyset$ do:
   (a) Choose a graph G in L that loses to at least half of the graphs in L.
   (b) $a_{n,k} \leftarrow a_{n,k} \cup \{\langle G, C_G\rangle\}$.
   (c) $L \leftarrow L \setminus \{G_1 : G \text{ loses to } G_1\}$.

Since there are $2^{O(n^2)}$ graphs with n vertices, the advice $a_{n,k}$ contains $O(n^2)$ graphs. We are ready to describe the nonuniform algorithm $\mathcal{B}$ that, given a graph G with n vertices, an integer k, and the advice a_n, decides if G contains a clique of size k.

Algorithm $\mathcal{B}$

INPUT: G, k, and $a_{n,k}$.

1. For every G_0 in $a_{n,k}$ execute $\mathcal{A}'$ on $\langle G_0, G, C_{G_0}\rangle$.
2. If there exists an execution of $\mathcal{A}'$ which returns 1, return "G has a clique of size k,"
3. Otherwise, return "FAIL."

If G does not have a clique of size k, then, by the correctness of $\mathcal{A}$, Algorithm $\mathcal{A}'$ always returns 0, and $\mathcal{B}$ always returns "FAIL." If G has a clique of size k, then there exists a graph G_0 in $a_{n,k}$ that loses to G; thus, with probability at least 1/3, Algorithm $\mathcal{A}'$ returns 1 on input $\langle G_0, G, C_{G_0}\rangle$, and, thus, with probability at least 1/3, Algorithm $\mathcal{B}$ returns "G has a clique of size k." That is, $\mathcal{B}$ is a randomized algorithm with advice strings that decides if $\langle G, k\rangle \in$ CLIQUE with a one-sided error of 2/3. By a standard amplification and union-bound arguments, we can get a deterministic polynomial-time nonuniform algorithm that decides if $\langle G, k\rangle \in$ CLIQUE without error. □

Appendix A. Semantic-security flavored definitions of private algorithms. In Definition 2.2, private algorithms are defined as those whose outputs on $\mathcal{R}$-equivalent instances are computationally indistinguishable. To shed a little more light on this definition we now compare it with two other definitions. Definition A.1 is of semantically secure private algorithms, where the outcome of the private algorithm $\mathcal{A}$ on x is simulated by an efficient simulator that is given access to an arbitrary representative y of the equivalence class of x. Definition A.2 is a (seeming) weakening of this definition, where the simulator is potentially much more powerful. Here, the simulator is given access to an arbitrary oracle O of its choice that answers queries about the equivalence class of x. For example, one may choose an oracle O that enumerates in lexicographic order the values y such that $x \equiv_{\mathcal{R}} y$, hence empowering the simulator to perform computations that may be otherwise intractable. It turns out that Definitions 2.2, A.1, and A.2 are equivalent. This gives a further indication that Definition 2.2 is a rather minimal notion of privacy.

DEFINITION A.1 (private algorithm—semantic security). *Let $\mathcal{R}$ be a privacy structure. A randomized polynomial-time algorithm $\mathcal{A}$ is* semantically private *with respect to $\mathcal{R}$ if there exists a randomized polynomial-time simulator $\mathcal{S}$, such that for every randomized polynomial-time algorithm $\mathcal{D}$ and for every positive polynomial $p(\cdot)$, there exists some $n_0 \in \mathbb{N}$ such that for every $x, y \in \{0,1\}^*$, where $x \equiv_{\mathcal{R}} y$ and $|x| = |y| \geq n_0$,*

$$\left| \Pr[\mathcal{D}(\mathcal{A}(x), x, y) = 1] - \Pr[\mathcal{D}(\mathcal{S}(y), x, y) = 1] \right| \leq \frac{1}{p(|x|)}.$$

That is, the simulator, on input y, outputs a distribution that is indistinguishable from the output of $\mathcal{A}$ on x.

For a string $y \in \{0,1\}^*$ define $\mathrm{Eq}(y) = \{z : y \equiv_{\mathcal{R}} z\}$ to be its equivalence class. Let $O : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an oracle. We use the notation $O(\mathrm{Eq}(y), \cdot)$ to denote the oracle obtained from O by "hardwiring" its first input to $\mathrm{Eq}(y)$. Informally, O is a collection of oracles—one per each equivalence class. The oracle $O(\mathrm{Eq}(y), \cdot)$ answers queries about the equivalence class of y. For an oracle machine S, we use $S^{O(\mathrm{Eq}(y),\cdot)}$ to denote the Turing machine S with oracle access to $O(\mathrm{Eq}(y), \cdot)$. This way, S may use O to learn facts about the equivalence class of y.

DEFINITION A.2 (private algorithm—weak semantic security). *Let $\mathcal{R}$ be a privacy structure. A randomized polynomial-time algorithm $\mathcal{A}$ is* weakly semantically private *with respect to $\mathcal{R}$ if there exist an oracle $O : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ and a randomized polynomial-time oracle machine[8] $\mathcal{S}$ (the simulator), such that for every randomized polynomial-time algorithm $\mathcal{D}$ and for every positive polynomial $p(\cdot)$, there exists some $n_0 \in \mathbb{N}$ such that, for every $x, y \in \{0,1\}^*$ such that $|x| = |y| \geq n_0$ and $x \equiv_{\mathcal{R}} y$,*

$$\left| \Pr[\mathcal{D}(\mathcal{A}(x), x, y) = 1] - \Pr[\mathcal{D}(\mathcal{S}^{O(\mathrm{Eq}(y),\cdot)}, x, y) = 1] \right| \leq \frac{1}{p(|x|)}.$$

That is, the simulator with oracle access to $O(\mathrm{Eq}(y), \cdot)$ outputs a distribution that is indistinguishable from the output of $\mathcal{A}$ on x.

THEOREM A.3. *Definitions* 2.2, A.1, *and* A.2 *are equivalent.*

Proof. Let $\mathcal{A}$ be an algorithm that is private with respect to a privacy structure $\mathcal{R}$ according to Definition 2.2. Taking the simulator $\mathcal{S}$ to be $\mathcal{A}$ itself shows that $\mathcal{A}$

[8] The choice of randomized polynomial-time is arbitrary, as S can delegate computations to the oracle O.

is private according to Definition A.1 as well. Privacy under Definition A.1 implies privacy under Definition A.2, as any simulator $\mathcal{S}$ in view of Definition A.1 can be transformed into an oracle simulator that asks for a representative of the class and proceeds like $\mathcal{S}$.

Finally, we prove that privacy under Definition A.2 implies privacy under Definition 2.2. Let $\mathcal{A}$ be an algorithm that is private under Definition A.2. Let x and y be inputs such that $x \equiv_{\mathcal{R}} y$, $\mathcal{S}$ be the corresponding simulator, and $\mathcal{D}$ be a distinguishing algorithm. By privacy under Definition A.2, the advantage of $\mathcal{D}$ in distinguishing $(\mathcal{A}(x), x, y)$ and $(\mathcal{S}^{O(\mathrm{Eq}(y),\cdot)}, x, y)$ is negligible. Furthermore, replacing the roles of x and y, the advantage of $\mathcal{D}$ in distinguishing $(\mathcal{A}(y), x, y)$ and $(\mathcal{S}^{O(\mathrm{Eq}(x),\cdot)}, x, y)$ is negligible. Therefore, as $\mathrm{Eq}(x) = \mathrm{Eq}(y)$, we get that the advantage of $\mathcal{D}$ in distinguishing $(\mathcal{A}(x), x, y)$ and $(\mathcal{A}(y), x, y)$ is also negligible, satisfying Definition 2.2. □

Appendix B. Impossibility result for randomized private approximation of vertex cover. In this section, we generalize the inapproximability results of section 5.2 to *randomized* private protocols that leak $O(\epsilon \log n)$ bits. We follow a strategy similar to that in the proof of Lemma 5.10 for deterministic private protocols that leak $O(\epsilon \log n)$ bits. In that proof we checked if $\mathcal{A}(G_t) \cap \{i_{t+1}, j_{2t+1}\} \neq \emptyset$. Here, we execute $\mathcal{A}(G_t)$ many times and estimate the probability that $\mathcal{A}(G_t) \cap \{i_{t+1}, j_{2t+1}\} \neq \emptyset$.

In Figure B.1, we present Algorithm `RelNonCrit—randomized` log n `bits`, which assumes that Algorithm $\mathcal{A}$ is a possibly randomized $n^{1-\epsilon}$-approximation algorithm that leaks at most $\frac{\epsilon}{6} \log n$ bits. Algorithm `RelNonCrit—randomized` log n `bits` is a randomized algorithm which returns a correct answer with probability at least $1 - \delta$ for a given $0 < \delta < 1$. Given a graph G with n vertices, we construct the graphs from Definition 5.2. These graphs have $N = (120n/\delta)^{2/\epsilon}$ vertices, $2m$ copies of G, and a disjoint set of vertices I. We choose the number of copies to be $2m$, where

$$m \stackrel{\mathrm{def}}{=} N^{\epsilon/6} = (120n/\delta)^{1/4}. \tag{B.1}$$

Therefore, the size of the set I is $|I| = N - 2mn$. In the proof of Claim B.4 it will become clear why we made these choices.

Algorithm `RelNonCrit—randomized` log n `bits`

INPUT: A graph $G = \langle V, E \rangle$, a vertex $v \in V$, and a number $0 < \delta < 1$.
OUTPUT: One of the following: (i) The vertex v is relevant for G. (ii) The vertex v is not critical for G. The algorithm errs with probability at most δ.

1. Choose distinct $i_1, \ldots, i_m, j_1, \ldots, j_{2m}$ at random from I.
2. For $t = 0$ to $m - 1$ do:
   (a) Let $G_t = G(\overset{j_1 j_2}{\vee}{}_{i_1} \cdots \overset{j_{2t-1} j_{2t}}{\vee}{}_{i_t} \; \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$.
   (b) Execute n times $\mathcal{A}(G_t)$.
   (c) If one of the vertices i_{t+1}, j_{2t+1} appears in more than $0.3n$ of the n answers of $\mathcal{A}$, then return "Noncritical."
3. RETURN "Relevant."

FIG. B.1. Algorithm `RelNonCrit—randomized` log n `bits`.

In the following we assume that, on graphs with n vertices, $\mathcal{A}$ leaks at most $k(n) = \frac{\epsilon}{6} \log n$ bits. Recall that we execute $\mathcal{A}$ on graphs with N vertices; thus, the approximation ratio is $N^{1-\epsilon}$, and the leakage is bounded by $k(N) = \frac{\epsilon}{6} \log N =$

$\log m$. The next sequence of claims asserts the correctness of Algorithm `RelNonCrit`—`randomized` $\log\ n$ `bits`.

Let $0 \leq p \leq 1$. We say that a vertex v is p-light for a graph H if $\Pr[v \in \mathcal{A}(H)] < p$ and v is p-heavy for H if $\Pr[v \in \mathcal{A}(H)] \geq p$. Let $\mathcal{R}'$ be a privacy structure that is a $k(N)$-refinement of $\mathcal{R}_{\text{minVC}}$ such that $\mathcal{A}$ is private with respect to $\mathcal{R}'$. The next observation follows from Chernoff's inequality.

OBSERVATION B.1. *Assume we execute n times $\mathcal{A}(H)$.*

- *If a vertex u is 0.4-heavy for H, then the probability that u appears in at most $0.3n$ of the n answers of $\mathcal{A}$ is $2^{-O(n)}$.*
- *If a vertex u is 0.2-light for H, then the probability that u appears in more than $0.3n$ of the n answers of $\mathcal{A}$ is $2^{-O(n)}$.*

A distinguishing algorithm can approximate the probability that a vertex is in $\mathcal{A}(H)$. Hence, for every two graphs H_0, H_1 such that $H_0 \equiv_{\mathcal{R}'} H_1$ and for every vertex u the probability that $u \in \mathcal{A}(H_0)$ is approximately the same as the probability that $u \in \mathcal{A}(H_1)$. This is formalized in the following claim.

CLAIM B.2. *Let $0 \leq p < 1$ and $0 < \gamma < 1 - p$ be constants. Then, there exists $n_0 \in \mathbb{N}$ such that if H_0 and H_1 are two graphs with $n \geq n_0$ vertices and there exists a vertex w that is p-light for H_0 and is $(p+\gamma)$-heavy for H_1, then H_0 and H_1 are not equivalent according to $\mathcal{R}'$.*

Proof. Consider the following distinguishing algorithm $\mathcal{D}$.

INPUT: Two graphs Γ_0, Γ_1 and a cover $C = \mathcal{A}(\Gamma_j)$ for some $j \in \{0,1\}$.

1. For $i \in \{0,1\}$, execute n times $\mathcal{A}(\Gamma_i)$.
2. For every $u \in V$ and $i \in \{0,1\}$, let $\tilde{p}_u^i$ be the fraction of executions in which u appears in the answers of $\mathcal{A}(\Gamma_i)$.
3. Choose a vertex u such that $\tilde{p}_u^1 - \tilde{p}_u^0 > 2\gamma/3$ (if no such u exists, return 1).
4. If $u \in C$, return 1; otherwise return 0.

For every $v \in V$ and $i \in \{0,1\}$ define $p_v^i \stackrel{\text{def}}{=} \Pr[v \in \mathcal{A}(\Gamma_i)]$. By Chernoff's inequality, with probability $1 - 2^{-O(n)}$, for every $v \in V$ and $i \in \{0,1\}$,

$$|p_v^i - \tilde{p}_v^i| < \gamma/6. \tag{B.2}$$

Now consider an execution of $\mathcal{D}$ on the graphs H_0 and H_1, and consider the vertex w guaranteed by the claim. Assume that (B.2) holds for every vertex in the graph. In particular,

$$\tilde{p}_w^1 - \tilde{p}_w^0 = p_w^1 - p_w^0 - (p_w^1 - \tilde{p}_w^1) - (\tilde{p}_w^0 - p_w^0) \ \geq \ \gamma - 2\gamma/6 \ = \ 2\gamma/3.$$

Thus, the probability that there is no such u in step 3 is negligible. Furthermore, for the u chosen in step 3, with overwhelming probability $p_u^1 - p_u^0 = \tilde{p}_w^1 - \tilde{p}_w^0 - (\tilde{p}_w^1 - p_w^1) - (p_w^0 - \tilde{p}_w^0) \geq \gamma/3$. Therefore,

$$|\Pr[\mathcal{D}(\mathcal{A}(\Gamma_1), \Gamma_0, \Gamma_1) = 1] - \Pr[\mathcal{D}(\mathcal{A}(\Gamma_0), \Gamma_0, \Gamma_1) = 1]| \geq \gamma/3 - 2^{-O(n)}.$$

This implies that H_0 and H_1 are not equivalent according to $\mathcal{R}'$, since $\mathcal{A}$ is private with respect to $\mathcal{R}'$. ∎

The following claim is analogous to Claim 5.6 for the deterministic case.

CLAIM B.3. *If a vertex v is nonrelevant, then the probability that Algorithm* `RelNonCrit`—`randomized` *log n* `bits` *returns "Relevant" is negligible.*

Proof. Let $G_m = G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2m-1} j_{2m}}{\underset{i_m}{\vee}})$. Since the vertex v is nonrelevant for G, by Claim 5.4, for every $0 \le t' < t \le m$ the graphs $G_{t'}$ and G_t are equivalent according to $\mathcal{R}_{\text{minVC}}$. As $\mathcal{A}$ leaks at most $k(n) = \log m$ bits, every equivalence class of $\mathcal{R}_{\text{minVC}}$ is partitioned into at most m equivalence classes of $\mathcal{R}'$. Since there are $m+1 = 2^k + 1$ graphs $G_0, \ldots, G_m$, there are two indices $0 \le t' < t \le m$, such that $G_{t'}$ and G_t are equivalent according to $\mathcal{R}'$.

As every cover of G_t must cover the edge $(i_{t'+1}, j_{2t'+1})$, at least one of $i_{t'+1}, j_{2t'+1}$ is 0.5-heavy for G_t. Applying Claim B.2 to G_t and $G_{t'}$, at least one of the vertices $i_{t'+1}, j_{2t'+1}$ is 0.4-heavy for $G_{t'}$. Thus, by Observation B.1, with high probability this vertex will appear at least $0.3n$ times in the answers of $\mathcal{A}(G_{t'})$, and the algorithm will return "Noncritical." ∎

To prove that Algorithm `RelNonCrit`—`randomized` log n `bits` is correct when v is critical, we need the following claim.

Claim B.4. *Let $0 \le t < m$. For every $i_1, \ldots, i_t, j_1, \ldots, j_{2t} \in I$, consider the following experiment: choose $i_{t+1}, \ldots, i_m$ and j_{2t+1} at random and with uniform distribution from I and return 1 if at least one of the vertices i_{t+1}, j_{2t+1} is 0.2-heavy for $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$. If v is critical, then the probability that this experiment returns 1 is at most δ/m.*

Proof. We chose N, the number of vertices in the graphs we construct, such that the size of each cover $\mathcal{A}$ returns is at most $|I|/(10\alpha)$ for $\alpha \stackrel{\text{def}}{=} 2m^2/\delta$. To show this, we shall need the following requirement:

$$|I| = N - 2mn \ge N/2. \tag{B.3}$$

That is, we need $N \ge 4mn$. As $m = N^{\epsilon/6} \le N^{1/6}$, it suffices to require $N \ge (4n)^{6/5}$. By the choice of N,

$$N = (120n/\delta)^{2/\epsilon} \ge (120n)^2 \ge (4n)^{6/5}$$

(since $0 < \delta, \epsilon \le 1$); thus (B.3) holds.

We next upper-bound the size of the covers that $\mathcal{A}$ outputs. The size of a minimum vertex cover of these graphs is at most $(2 \cdot n + 1)m \le 3nm$. Since $\mathcal{A}$ is an $N^{1-\epsilon}$-approximation algorithm for vertex cover, the size of its output is at most $3 \cdot nm \cdot N^{1-\epsilon}$. Recall that $N = (\frac{120n}{\delta})^{2/\epsilon}$ and $m = N^{\epsilon/6}$, and thus, using (B.1),

$$120nm^3 = \delta(120n/\delta) \cdot N^{\epsilon/2} = \delta((120n/\delta)^{2/\epsilon})^{\epsilon/2} \cdot N^{\epsilon/2} = \delta N^{\epsilon/2} N^{\epsilon/2} = \delta N^{\epsilon}. \tag{B.4}$$

Therefore, the size of the cover returned by $\mathcal{A}$ is at most

$$3 \cdot nm \cdot N^{1-\epsilon} = \frac{3nmN}{N^{\epsilon}} \le \frac{6nm|I|}{N^{\epsilon}} = \frac{120nm^3}{\delta N^{\epsilon}} \cdot \frac{\delta |I|}{20m^2} = \frac{\delta |I|}{20m^2} = \frac{|I|}{10\alpha},$$

where the inequality above follows from (B.3) and the last two equalities follow from (B.4) and the definition of α. Therefore, for *every* choice of $i_{t+1}, \ldots, i_m, j_{2t+1}, \ldots, j_{2m}$ there are at most $|I|/\alpha$ vertices that are 0.1-heavy for $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$.

We will show that if v is critical, then the number of vertices that are 0.2-heavy for $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$ for at least one choice of $i_{t+1}, \ldots, i_m, j_{2t+1}, \ldots, j_{2m}$ is at most $m|I|/\alpha$.

We first prove an upper bound on the number of 0.2 heavy vertices in a given equivalence class of $\mathcal{R}'$. Fix $i_{t+1}, \ldots, i_m, j_{2t+1}, \ldots, j_{2m}$ and let

$$H_0 = G\left(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge}\right).$$

If a vertex u is 0.2-heavy for some H such that $H \equiv_{\mathcal{R}'} H_0$, then by Claim B.2, it is (at least) 0.1-heavy for H_0. Thus, there are at most $|I|/\alpha$ vertices that are 0.2-heavy for at least one of the graphs in the equivalence class of H_0 with respect to $\equiv_{\mathcal{R}'}$.

We next bound the number of vertices that are 0.2-heavy vertices for at least one graph of the form $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$. If v is critical, then, by Claim 5.3, every two choices of $i_{t+1}, \ldots, i_m$ result in equivalent graphs according to $\mathcal{R}_{\text{minVC}}$. Since $\mathcal{A}$ leaks at most $k(N) = \frac{\epsilon}{6} \log N = \log m$ bits, there are at most $2^{k(N)} = m$ equivalence classes of $\mathcal{R}'$ for the different choices of $i_{t+1}, \ldots, i_m, j_{2t+1}, \ldots, j_{2m}$. Thus, there are at most $m|I|/\alpha$ vertices that are 0.2-heavy for at least one graph $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$ for some $i_{t+1}, \ldots, i_m, j_{2t+1}, \ldots, j_{2m}$.

We now analyze the probability that the experiment returns 1. The probability that at least one of the vertices i_{t+1}, j_{2t+1} is 0.2-heavy for some graph is, by the union bound, at most $\frac{2m}{\alpha}$. Therefore, the probability of the experiment returning 1 is at most $\frac{2m}{\alpha} = \delta/m$, as required. □

CLAIM B.5. *If v is critical, then the probability that Algorithm* `RelNonCrit`—`randomized` log n `bits` *returns "Noncritical" is at most δ.*

Proof. Assume v is critical. By Claim B.4, the probability that we choose $i_1, \ldots, i_m, j_1, \ldots, j_{2m}$ such that for some $0 \le t < m$ at least one of the vertices $i_{t+1}, \ldots, i_m, j_{2t+1}, \ldots, j_{2m}$ is 0.2-heavy for $G(\overset{j_1 j_2}{\underset{i_1}{\vee}} \cdots \overset{j_{2t-1} j_{2t}}{\underset{i_t}{\vee}} \overset{i_{t+1}}{\wedge} \cdots \overset{i_m}{\wedge})$ is at most δ. By Observation B.1, the probability that Algorithm `RelNonCrit`—`randomized` log n `bits` returns "Noncritical" in this case is, therefore, negligible. □

Algorithm `RelNonCrit`—`randomized` log n `bits` executes nm times the polynomial-time algorithm $\mathcal{A}$ on graphs with N vertices, where $N = (120n/\delta)^{2/\epsilon}$. Therefore, we have proved the following claim.

CLAIM B.6. *If $\mathcal{A}$ runs in polynomial time and $0 \le \epsilon < 1$ is a constant, then the running time of Algorithm* `RelNonCrit`—`randomized` log n `bits` *is* poly(n/δ).

We are ready to prove Theorem 5.1 (similarly to the proof of Lemma 5.10).

THEOREM 5.1 *Let $\epsilon > 0$ be a constant. If $\mathcal{RP} \neq \mathcal{NP}$, then there is no $n^{1-\epsilon}$-approximation algorithm for the search problem of* vertex cover *that leaks at most $\frac{\epsilon}{6} \log n$ bits.*

Proof. Algorithm `Greedy Vertex Cover`, which solves vertex cover, executes at most n times Algorithm `RelNonCrit`—`randomized` log n `bits` with graphs of size at most n. We execute these calls with $\delta = \frac{1}{4n}$ (where n is the original number of vertices in G); thus, all together, the error is at most 1/4. By Claim B.6, the running time of Algorithm `Greedy Vertex Cover` is polynomial.

Thus, if there is an $n^{1-\epsilon}$-approximation algorithm for the search problem of vertex cover that leaks at most $k(n) = \frac{\epsilon}{6} \log n$ bits, then there is a polynomial-time randomized algorithm for minimum vertex cover that errs with probability 1/4. This implies that $\mathcal{NP} \subseteq \mathcal{BPP}$.

To contradict $\mathcal{RP} \neq \mathcal{NP}$, this algorithm is transformed into a one-sided error algorithm for the decision problem of vertex cover: Given $\langle G, s \rangle$, decide if G has a vertex cover of size at most s. The transformation is simple; execute the algorithm for the search problem of vertex cover. If this algorithm returns a set that covers G and its size is at most s, return "yes." □

Appendix C. Negative result for private approximation of exact 3SAT. Recall that the privacy structure $\mathcal{R}_{\text{maxE3SAT}}$ contains all pairs of exact 3CNF formulae ϕ_1, ϕ_2 over n variables for which an assignment a satisfies the maximum possible number of clauses in ϕ_1 iff it satisfies the maximum possible number of clauses in ϕ_2.

DEFINITION C.1 (private approximation of maxE3SAT). *An algorithm $\mathcal{A}$ is a* private $c(n)$ approximation *algorithm for* maxE3SAT *if* (i) $\mathcal{A}$ *runs in polynomial time;* (ii) $\mathcal{A}$ *is a* $c(n)$*-approximation algorithm for* maxE3SAT*, that is, for every exact* 3*CNF formula* ϕ *over* n *variables it returns with probability* 1 *an assignment that satisfies at least* $c(n)$ *times the maximum possible number of clauses that are simultaneously satisfiable in* ϕ*;* (iii) $\mathcal{A}$ *is private with respect to the privacy structure* $\mathcal{R}_{\text{maxE3SAT}}$*.*

We now prove Theorem 3.16, part 1.

THEOREM 3.16 (part 1). *Let* $\epsilon > 0$ *be a constant. If* $\mathcal{P} \neq \mathcal{NP}$*, then there is no deterministic private* $1/n^{1-\epsilon}$*-approximation algorithm for the search problem of* maxE3SAT*.*

Sketch of proof. Similarly to the proof of Theorem 3.2, we will assume the existence of a deterministic private $1/n^{1-\epsilon}$-approximation algorithm $\mathcal{A}$, and use $\mathcal{A}$ to construct a deterministic algorithm for deciding the satisfiability of exact 3CNF formulae. We emphasize that we are solving the decision problem of satisfiability and not the optimization problem of maximum satisfiability.

DEFINITION C.2 (relevant assignment to a variable). *Let* ϕ *be an exact* 3*CNF formula over Boolean variables* $x_1, \ldots, x_n$*. We say that a variable* x_i *is* σ*-relevant (where* $\sigma \in \{0,1\}$*) if there exists an assignment* a *that satisfies the maximum possible number of clauses in* ϕ *such that* $a(x_i) = \sigma$*.*

Note that for every satisfiable formula, every variable is 0-relevant or 1-relevant (or both). Furthermore, if a variable is not σ-relevant, then, in each assignment that satisfies the formula, its value is $\neg\sigma$; that is, the variable is "$\neg\sigma$-critical."

We will first assume an algorithm `Relevant Variable-Assignment` that, given an exact 3CNF formula ϕ over variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, outputs an index i and a bit σ such that x_i is σ-relevant. The variables $y_1, \ldots, y_n$ are auxiliary variables that we add to the formula to ensure that the formula remains an exact 3CNF formula. Algorithm `Relevant Variable-Assignment` returns an index of a variable x_i (it never returns a variable y_i). For technical reasons, `Relevant Variable-Assignment` needs that ϕ includes at least 3 variables from $x_1, \ldots, x_n$ (negated or nonnegated). Algorithm `Greedy E3SAT`, described in Figure C.1, uses Algorithm `Relevant Variable-Assignment` to decide E3SAT.

Note that the algorithm terminates in at most $n-2$ iterations, as every iteration reduces the number of x variables in $\hat{\phi}$ by at least one. The final length of $\hat{\phi}$ is at most eight times that of ϕ as the replacement steps 2(c)ii and 2(c)iii are applied only to clauses including x variables, resulting in one less x variable in each of the two replacement clauses. Hence, if Algorithm `Relevant Variable-Assignment` runs in polynomial time, so does `Greedy E3SAT`.

We now prove the correctness of Algorithm `Greedy E3SAT`. Clearly, the only way `Greedy E3SAT` can err is by outputting 0 on a satisfiable formula ϕ; hence we assume from now on that ϕ is satisfiable and show that executing `Greedy E3SAT` results in outputting 1. Before proving the correctness of the algorithm, we will try to give the intuition behind it. In each step of the algorithm we have a partial assignment to ϕ that can be extended to an assignment that satisfies ϕ. On one hand, this partial assignment already satisfies some clauses in ϕ, and therefore we deleted them from $\hat{\phi}$. On the other hand, some literals in various clauses are not satisfied by the partial assignment. We would have liked to delete these literals from the clauses that they appear in, and continue. However, this might result in a 3CNF formula that is not an exact 3CNF formula. We therefore replace the literal ℓ_i that is not satisfied by an auxiliary variable y_i. By replacing each clause $(\ell_i \vee \ell_j \vee \ell_k)$ (where ℓ_j and ℓ_k are

Algorithm `Greedy E3SAT`

INPUT: An exact 3CNF formula ϕ on n variables $x_1, \ldots, x_n$.
OUTPUT: 1 if ϕ is satisfiable and 0 otherwise.

1. Let $\hat{\phi} = \phi$.
2. While $\hat{\phi}$ contains at least three variables in $x_1, \ldots, x_n$ do:
   (a) Execute Algorithm `Relevant Variable-Assignment` on $\hat{\phi}$. Denote the answer $\langle i, \sigma \rangle$.
   (b) Assign $a_i \leftarrow \sigma$.
   (c) Modify $\hat{\phi}$ as follows:
      i. Leave in $\hat{\phi}$ every clause that does not include x_i or $\neg x_i$.
      ii. If $\sigma = 0$, then replace every clause $(x_i \vee \ell_j \vee \ell_k)$ in $\hat{\phi}$ by the clauses $(y_i \vee \ell_j \vee \ell_k)$ and $(\neg y_i \vee \ell_j \vee \ell_k)$. Remove all clauses that include $\neg x_i$.
      iii. If $\sigma = 1$, then replace every clause $(\neg x_i \vee \ell_j \vee \ell_k)$ in $\hat{\phi}$ by the clauses $(y_i \vee \ell_j \vee \ell_k)$ and $(\neg y_i \vee \ell_j \vee \ell_k)$. Remove all clauses that include x_i.
3. For every variable x_i that is not in $\hat{\phi}$ and was not assigned a value in step 2b, assign $a_i \leftarrow 0$.
4. Exhaustively check all four assignments a that agree with the assignments made in step 2b and step 3. If at least one of these assignments satisfies ϕ, return 1; otherwise return 0.

FIG. C.1. *Algorithm* `Greedy E3SAT`.

literals—variables or negations) with two clauses $(y_i \vee \ell_j \vee \ell_k)$ and $(\neg(y_i) \vee \ell_j \vee \ell_k)$, one with y_i and one with $\neg y_i$, we ensure that a satisfying assignment to $\hat{\phi}$ must satisfy $\ell_j \vee \ell_k$.

The formal proof is by induction on the number of iterations in `Greedy E3SAT`: After executing the main iteration in Algorithm `Greedy E3SAT` for k times (i) $\hat{\phi}$ is satisfiable; (ii) k variables are assigned values in step 2(b); (iii) $\hat{\phi}$ does not contain these variables; and (iv) any assignment that extends the k assigned variables satisfies ϕ iff it satisfies $\hat{\phi}$. □

To conclude the proof, we present Algorithm `Relevant Variable-Assignment`. On input ϕ, the algorithm uses a private $1/n^{1-\epsilon}$-approximation algorithm $\mathcal{A}$ to produce an index i and a bit σ such that x_i is σ-relevant.

By our choice of ℓ_1, ℓ_2, ℓ_3, the clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ is not satisfied by the assignment a. Hence, as $\mathcal{A}$ is a $1/n^{1-\epsilon}$-approximation algorithm, at least one of x_1, x_2, x_3 changes assignments between a and a'. Thus, step 5 in Algorithm `Relevant Variable-Assignment` is always possible. Furthermore, as $\mathcal{A}$ respects the privacy structure $\mathcal{R}_{\text{maxE3SAT}}$, we conclude that the formulae ϕ and ϕ' differ on their sets of maximum assignments. The following claim implies the correctness of Algorithm `Relevant Variable-Assignment`.

CLAIM C.3. *If $a'(x_i) \neq a(x_i)$ for some $i \in \{1, 2, 3\}$, then x_i is $a(x_i)$-relevant.*

Proof. Assume the contrary, i.e., that every assignment that satisfies the maximum possible number of clauses of ϕ assigns $a'(x_i)$ to x_i. It is easy to see that every such assignment of ϕ is also an assignment that satisfies the maximum possible number of clauses of ϕ' as it satisfies the added clauses $(\ell_1 \vee \ell_2 \vee \ell_3)$. In the reverse direction, note that every assignment that satisfies the maximum possible number of

Algorithm `Relevant Variable-Assignment`

INPUT: An exact 3CNF formula ϕ over Boolean variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. Without loss of generality, all variables x_1, x_2, x_3 appear in ϕ. Let m denote the number of clauses in ϕ.
OUTPUT: An index $i \in \{1, 2, 3\}$ and a bit σ such that x_i is σ-relevant.

1. Execute $\mathcal{A}$ on ϕ and denote $a = \mathcal{A}(\phi)$.
2. For $i \in \{1, 2, 3\}$, let $\ell_i = x_i$ if $a(x_i) = 0$ and $\ell_i = \neg x_i$ otherwise.
3. Set $\phi' = \phi \wedge (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \cdots \wedge (\ell_1 \vee \ell_2 \vee \ell_3)$, where the clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ is added $n^{1-\epsilon} \cdot (m + 1)$ times.
4. Execute $\mathcal{A}$ on ϕ' and denote $a' = \mathcal{A}(\phi')$.
5. Choose $i \in \{1, 2, 3\}$ such that $a'(x_i) \neq a(x_i)$. Let $\sigma = a(x_i)$.
6. Return $\langle i, \sigma \rangle$.

clauses of ϕ' also satisfies the maximum possible number of ϕ as the assignment to the two other variables from x_1, x_2, x_3 does not affect the satisfiability of the clause $(\ell_1 \vee \ell_2 \vee \ell_3)$. We get that $\langle \phi, \phi' \rangle \in \mathcal{R}_{\text{maxE3SAT}}$, contradicting $\mathcal{A}(\phi) \neq \mathcal{A}(\phi')$. □

Acknowledgments. We thank Yinnon Haviv for helpful discussions, and we thank Robi Krauthgamer for pointing out that we can use almost k-wise independent spaces. Finally, we thank the anonymous reviewer for useful comments.

REFERENCES

[1] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.
[2] N. ALON, O. GOLDREICH, J. HÅSTAD, AND R. PERALTA, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.
[3] R. BAR-YEHUDA, B. CHOR, E. KUSHILEVITZ, AND A. ORLITSKY, *Privacy, additional information, and communication*, IEEE Trans. Inform. Theory, 39 (1993), pp. 1930–1943.
[4] R. BAR-YEHUDA AND S. EVEN, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.
[5] A. BEIMEL, R. HALLAK, AND K. NISSIM, *Private approximation of clustering and vertex cover*, in Proceedings of the 4th Theory of Cryptography Conference (TCC 2007), S. Vadhan, ed., Lecture Notes in Comput. Sci. 4392, Springer-Verlag, 2007, New York, pp. 383–403.
[6] A. BEIMEL, T. MALKIN, K. NISSIM, AND E. WEINREB., *How should we solve search problems privately?*, in Advances in Cryptology (CRYPTO 2007), A. Menezes, ed., Lecture Notes in Comput. Sci. 4622, Springer-Verlag, New York, 2007, pp. 31–49.
[7] M. BELLARE AND E. PETRANK, *Making zero-knowledge provers efficient*, in Proceedings of the 24th ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 711–722.
[8] B. CHOR, J. FRIEDMANN, O. GOLDREICH, J. HASTAD, S. RUDICH, AND R. SMOLANSKY, *The bit extraction problem or t-resilient functions*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1985, pp. 396–407.
[9] I. DINUR AND S. SAFRA, *On the hardness of approximating minimum vertex cover*, Ann. of Math. (2), 162 (2005), pp. 439–485.
[10] J. FEIGENBAUM, Y. ISHAI, T. MALKIN, K. NISSIM, M. J. STRAUSS, AND R. N. WRIGHT, *Secure multiparty computation of approximations*, ACM Trans. Algorithms, 2 (2006), pp. 435–472.
[11] M. J. FREEDMAN, K. NISSIM, AND B. PINKAS, *Efficient private matching and set intersection*, in Advances in Cryptology (EUROCRYPT 2004), C. Cachin and J. Camenisch, eds., Lecture Notes in Comput. Sci. 3027, Springer-Verlag, New York, 2004, pp. 1–19.
[12] O. GOLDREICH, S. MICALI, AND A. WIGDERSON, *How to play any mental game*, in Proceedings of the 19th ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 218–229.

[13] O. Goldreich, R. Ostrovsky, and E. Petrank, *Computational complexity and knowledge complexity*, SIAM J. Comput., 27 (1998), pp. 1116–1141.
[14] O. Goldreich and E. Petrank, *Quantifying knowledge complexity*, Comput. Complexity, 8 (1999), pp. 50–98.
[15] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.
[16] S. Halevi, R. Krauthgamer, E. Kushilevitz, and K. Nissim, *Private approximation of NP-hard functions*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 550–559.
[17] E. Halperin, *Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs*, in Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 329–337.
[18] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
[19] P. Indyk and D. Woodruff, *Polylogarithmic private approximations and efficient matching*, in Proceedings of the 3rd Theory of Cryptography Conference (TCC 2006), S. Halevi and T. Rabin, eds., Lecture Notes in Comput. Sci. 3876, Springer-Verlag, New York, 2006, pp. 245–264.
[20] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.
[21] S. Khot and O. Regev, *Vertex cover might be hard to approximate to within* 2-ϵ, J. Comput. System Sci., 74 (2008), pp. 335–349.
[22] E. Kiltz, G. Leander, and J. Malone-Lee, *Secure computation of the mean and related statistics*, in Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005), J. Kilian, ed., Lecture Notes in Comput. Sci. 3378, Springer-Verlag, New York, 2005, pp. 283–302.
[23] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.
[24] B. Monien and E. Speckenmeyer, *Ramsey numbers and an approximation algorithm for the vertex cover problem*, Acta Inform., 22 (1985), pp. 115–123.
[25] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.
[26] C. H. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the V-C dimension*, J. Comput. System Sci., 53 (1996), pp. 161–170.
[27] E. Petrank and G. Tardos, *On the knowledge complexity of NP*, Combinatorica, 22 (2002), pp. 83–121.
[28] A. C. Yao, *Protocols for secure computations*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1982, pp. 160–164.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1761–1781

APPROXIMATING MINIMUM MAX-STRETCH SPANNING TREES ON UNWEIGHTED GRAPHS*

YUVAL EMEK† AND DAVID PELEG†

Abstract. Given a graph G and a spanning tree T of G, we say that T is a *tree t-spanner* of G if the distance between every pair of vertices in T is at most t times their distance in G. The problem of finding a tree t-spanner minimizing t is referred to as the *Minimum Max-Stretch spanning Tree (MMST)* problem. This paper concerns the MMST problem on unweighted graphs. The problem is known to be NP-hard, and the paper presents an $O(\log n)$-approximation algorithm for it. Furthermore, it is established that unless P = NP, the problem cannot be approximated additively by any $o(n)$ term.

Key words. spanning trees, low stretch, spanners

AMS subject classifications. 05C05, 05C12, 05C85

DOI. 10.1137/060666202

1. Introduction.

1.1. The problem. Consider a connected n-vertex graph G. Let T be a spanning tree of G and let x and y be two vertices in G. The *stretch* of x and y in T, denoted $\mathrm{str}_T(x,y)$, is the ratio of the distance between x and y in T to their distance in G. The *maximum stretch*[1] of T, denoted max-str(T), is defined as the maximum of $\mathrm{str}_T(x,y)$, taken over all pairs of vertices x,y in G. The problem of finding a spanning tree T minimizing max-str(T) is referred to as the *Minimum Max-Stretch spanning Tree (MMST)* problem. In this paper we study the MMST problem on unweighted graphs. This problem is known to be NP-hard [5], and this paper presents the first nontrivial approximation algorithm for it, achieving an approximation ratio of $O(\log n)$. Our algorithm is inspired by the algorithm presented in [16] for the *Minimum Restricted Diameter spanning Tree (MRDT)* problem. We then establish a hardness of approximation result, showing that it is NP-hard to approximate the problem additively by a term of $o(n)$.

The MMST problem finds applications in network design and, in particular, in the context of distributed systems. One such application is the *arrow distributed directory protocol* introduced in [7]. This protocol supports the location of mobile objects in a distributed network. It is implemented over a spanning tree T that spans the network, and, as shown in [19], the worst case overhead ratio of the protocol is proportional to the maximum stretch of T. Therefore, a good candidate for the backbone of the arrow protocol is a spanning tree with low maximum stretch (see also [17]).

1.2. Related work. The notion of stretch can be defined for any spanning subgraph. Formally, given a graph G, a spanning subgraph H of G, and a pair of

*Received by the editors July 27, 2006; accepted for publication (in revised form) September 15, 2008; published electronically December 19, 2008. An extended abstract appeared in [11]. This research was supported in part by a grant from the Israel Science Foundation.
http://www.siam.org/journals/sicomp/38-5/66620.html
†Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel (yuval.emek@weizmann.ac.il, david.peleg@weizmann.ac.il).
[1] In some papers the notion of stretch refers to the maximum taken over all vertex pairs. To avoid misunderstanding, we distinguish between stretch, defined in the current paper for a specific vertex pair, and maximum stretch, defined for the whole tree.

vertices x, y in G, the stretch of x and y in H is defined as the ratio of the distance between x and y in H to their distance in G. A spanning subgraph with maximum stretch t is called a *t-spanner*. Spanners for general graphs were first introduced in [22]. Sparse spanners (namely, spanners with a small number of edges) were first studied in [20], where the problem of determining for a given graph G and a positive integer m whether G has a t-spanner with at most m edges is shown to be NP-complete for $t = 2$ while a polynomial time construction is presented for a $(4t + 1)$-spanner with $O\left(n^{1+1/t}\right)$ edges for every n-vertex graph and $t \geq 1$. Simple algorithms for constructing sparse spanners for arbitrary weighted graphs are presented in [2], including the construction of a $(2t+1)$-spanner with at most $n\left\lceil n^{1/t}\right\rceil$ edges for every n-vertex graph and $t > 0$. For any fixed $t \geq 3$ the problem of determining, for an arbitrary graph G and a positive integer m, whether G has a t-spanner with at most m edges is proved to be NP-complete in [4]. A polynomial time construction for a 3-spanner with $O\left(n^{3/2}\right)$ edges is presented in [9].

Cast in this terminology, the MMST problem can therefore be redefined as the attempt to find a *tree t-spanner* minimizing t. The NP-hardness of the MMST problem, even on unweighted graphs, is established in [5], where it is proved that determining whether an arbitrary weighted (respectively, unweighted) graph has a tree t-spanner is NP-complete for every fixed $t > 1$ (respectively, $t \geq 4$). The same paper also presents a polynomial time algorithm for constructing a tree 1-spanner in a weighted graph (if such a spanner exists) and a polynomial time algorithm for constructing a tree 2-spanner in an unweighted graph (if such a spanner exists).

Low stretch spanning trees in planar graphs were first studied in [12], where it is proved that finding a spanning tree T with minimum max-str(T) is NP-hard even for unweighted planar graphs. Polynomial time algorithms are presented therein for the problem of deciding for a fixed parameter t whether a planar unweighted graph with bounded face length has a tree t-spanner and for the problem of deciding whether an arbitrary unweighted planar graph has a tree 3-spanner. A polynomial time algorithm for the MMST problem on outerplanar graphs is presented in [21].

Hardness of approximation is established in [19], where it is shown that approximating the MMST problem within a factor better than $(1 + \sqrt{5})/2$ is NP-hard. This is improved in [18] by observing that the $2 - \epsilon$ inapproximabilty result established in [14] for the *min-max strictly fundamental cycle basis* problem also holds in the context of the MMST problem on unweighted graphs. A number of papers have studied the related but easier problem of finding a spanning tree with good *average* stretch factor [1, 3, 13, 10].

Given a complete graph $G = (V(G), E(G))$ with edge weights obeying the triangle inequality and a subset $R \subseteq E(G)$ called the *requirements* of G, the MRDT problem is to find a spanning tree T of G that minimizes the *restricted diameter* of T defined as the diameter of T restricted to vertex pairs in R. The MRDT problem is presented and proved to be NP-hard in [16]. The same paper presents an $O(\log n)$-approximation algorithm for this problem. It can be shown that the MMST problem on complete graphs with edge weights arising from the distances in some unweighted graph is a special case of the MRDT problem.

1.3. A brief description of the technique. Our main result is the first nontrivial approximation algorithm for the MMST problem on unweighted graphs. This algorithm relies on a graph decomposition technique that, given an unweighted graph G of size n, breaks it into disjoint connected components, each of size at most $n/2$, by discarding the edges internal to some ball B of radius proportional to the maximum

stretch ρ of an optimal solution to the MMST problem on G. The spanning trees generated by recursive invocations of the algorithm on each such connected component are combined with a single source shortest paths spanning tree of B to produce a spanning tree T of G.

Consider an arbitrary edge (x, y) in G. Our analysis relies on observing that the number of edges added to the unique path between x and y in T on each recursive level is $O(\rho)$. Since there are at most $\log n$ recursive levels (as the size of the graph decreases by a factor of 2 on each recursive level), the stretch of x and y in T is $O(\log n)$.

The technique presented in [16] for the approximation algorithm of the MRDT problem is similar to that described above. The novel approach in this paper is the adaptation of this technique to graphs which are not necessarily complete.

1.4. Outline of the paper. In section 2 we present the basic notation and definitions used throughout this paper. Two lower bounds on the minimum max-stretch of unweighted graphs are established in section 3. Our approximation algorithm, named Algorithm `Construct_Tree`, is presented in section 4. In section 5 we prove that the output of Algorithm `Construct_Tree` is a spanning tree, and in section 6 we analyze the performance guarantee of the algorithm, establishing an $O(\log n)$ upper bound on the approximation ratio. Our analysis is based on the lower bounds of section 3. In section 7 we prove that the analysis of the performance guarantee of the algorithm is tight. The hardness of approximating the MMST problem on unweighted graphs is studied in section 8, proving that unless P = NP, the problem cannot be approximated additively by a term of $o(n)$.

2. Preliminaries. Throughout, we consider a connected unweighted undirected n-vertex graph G. Let $V(G)$ and $E(G)$ denote the vertex and edge sets of G, respectively. The *length* of a path P in the graph is the number of edges in the path, denoted by $\text{len}(P)$. For two vertices u, v in $V(G)$, let $\text{dist}_G(u, v)$ denote the *distance* between them in G, i.e., the length of a shortest path between u and v. The definition of distance is extended to vertex subsets as follows. Let U and W be two subsets of $V(G)$. The *distance* between U and W is the minimum distance between any pair of vertices in U and W, denoted by $\text{dist}_G(U, W) = \min\{\text{dist}_G(u, w) \mid u \in U \text{ and } w \in W\}$.

For a subset $U \subseteq V(G)$, let $G(U)$ denote the subgraph of G *induced* by U, that is, $V(G(U)) = U$ and $E(G(U)) = E(G) \cap (U \times U)$. Denote the set of edges *internal* to U by $E(U) = E(G(U))$.

Although the notion of stretch can be defined for every spanning subgraph, our focus in the current paper is on spanning trees only. Consider some spanning tree T of G. Denote the *stretch* of u and v in T with respect to G by

$$\text{str}_{T,G}(u, v) = \frac{\text{dist}_T(u, v)}{\text{dist}_G(u, v)}.$$

Denote the *maximum stretch* of T with respect to G by

$$\text{max-str}(T, G) = \max_{x,y \in V(G)} \{\text{str}_{T,G}(x, y)\}.$$

When the graph G is clear from the context we may omit it and write simply $\text{str}_T(u, v)$ and $\text{max-str}(T)$. Denote the *minimum max-stretch* of G by

$$\text{max-str}(G) = \min\{\text{max-str}(T, G) \mid T \text{ is a spanning tree of } G\}.$$

Consider two metrics τ and δ over the vertices V. We say that τ *dominates* δ if $\tau(u,v) \geq \delta(u,v)$ for every $u,v \in V$. We say that τ is a *tree metric*[2] if τ is induced by the distances in some weighted tree over V. We extend the definitions of stretch and maximum stretch for tree metrics as follows. For every two vertices $u,v \in V$, denote the *stretch* of u and v in τ with respect to δ by

$$\mathrm{str}_{\tau,\delta}(u,v) = \frac{\tau(u,v)}{\delta(u,v)}.$$

The *maximum stretch* of τ with respect to δ, denoted $\text{max-str}(\tau,\delta)$, is defined accordingly.

Consider some metric δ over the vertices V. Given a vertex set $U \subseteq V$, we define the *diameter* of U with respect to δ as

$$\mathrm{diam}_\delta(U) = \max\{\delta(u,v) \mid u,v \in U\}.$$

Given a vertex $u \in V$ and some positive real ρ, we denote the *ball* centered at u of radius ρ with respect to δ by $B_\delta(u,\rho) = \{v \in V(G) \mid \delta(u,v) \leq \rho\}$.

A *partition* of a set S is a collection $P = \{U_1,\ldots,U_k\}$, where $U_i \cap U_j = \emptyset$ for every $1 \leq i < j \leq k$ and $\bigcup_{1\leq i\leq k} U_i = S$. Unless stated otherwise, we assume that U_i is nonempty for every $1 \leq i \leq k$. If S is the set of vertices of some graph, then the subsets $U_1,\ldots,U_k$ are called the *clusters* of the partition. (Note that our definition does not require a cluster to be connected in G.)

For a partition $P = \{U_1,\ldots,U_k\}$ of $V(G)$, let $E(P)$ denote the set of edges *internal* to the clusters of P, i.e., $E(P) = \bigcup_{U_i\in P} E(U_i)$. Denote the set of edges *external* to P, namely, the edges connecting vertices in two different clusters, by $\overline{E}(P) = E(G) - E(P)$. Every edge set $F \subseteq \overline{E}(P)$ *induces* a logical *cluster graph* on P, obtained by contracting each cluster in P into a single node and replacing each edge $(u,v) \in F$, where $u \in U_i$ and $v \in U_j$, by the edge (U_i,U_j). We say that F *induces a tree* on P if the cluster graph induced by F on P is a tree. For a subset $X \subseteq V(G)$, denote the set of clusters in P that intersect X by $\mathcal{I}(P,X) = \{U_i \in P \mid U_i \cap X \neq \emptyset\}$.

A central tool in our construction is a graph decomposition based on eliminating the edges of some ball of radius ρ. This decomposition is obtained as follows. For a metric δ over $V(G)$, a vertex $u \in V(G)$, and a positive integer ρ, erase from G the internal edges (but not the vertices) of the ball centered at u of radius ρ with respect to δ, $E(B_\delta(u,\rho))$, and let $G^1,\ldots,G^r$ be the connected components in the remaining graph. The collection $\{G^1,\ldots,G^r\}$ is referred to as the (u,ρ,δ)-*decomposition* of G. Figure 1 illustrates the $(u,2,\mathrm{dist}_G(\cdot,\cdot))$-decomposition of a graph G. We say that the vertex u is a ρ-*center* with respect to G and δ if $|V(G^i)| \leq n/2$ for every $1 \leq i \leq r$.

3. Lower bounds on the minimum max-stretch. In this section we establish some general lower bounds on the minimum max-stretch of a graph G. These lower bounds are used in section 6 to yield the performance guarantee of our approximation algorithm.

We begin with correlating the existence of a ρ-center with the minimum max-stretch of the graph.

THEOREM 3.1. *Consider a graph G and a connected vertex induced subgraph H of G, and let δ be the restriction of* $\mathrm{dist}_G(\cdot,\cdot)$ *to the vertices of H. Then H admits a (2κ)-center with respect to δ, where $\kappa = \text{max-str}(G)$.*

[2]Our definition of a tree metric differs from the standard definition, where τ is induced by the distances between pairs of vertices from V in some weighted tree over a superset of V. Clearly, a tree metric according to our definition is also a tree metric according to the standard definition.

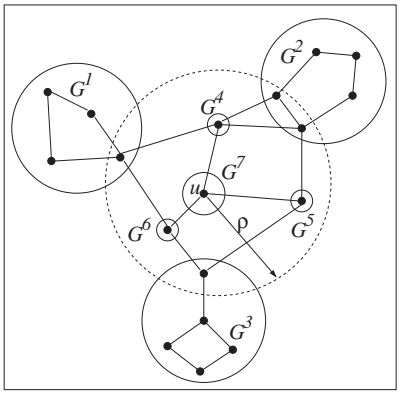


FIG. 1. *The decomposition of G with respect to $\mathrm{dist}_G(\cdot,\cdot)$, u, and $\rho = 2$. The dashed circle represents $B_{\mathrm{dist}_G(\cdot,\cdot)}(u,2)$.*

Proof. Clearly, every spanning tree of G induces a tree metric over $V(G)$ that dominates the distances in G. In particular, there exists such a tree metric τ with $\text{max-str}(\tau, \mathrm{dist}_G(\cdot,\cdot)) = \kappa$. In [15] it is proved that a tree metric τ over the vertex set V can be transformed into a tree metric τ' over an arbitrary vertex subset $U \subseteq V$, such that $\tau(x,y) \leq \tau'(x,y) \leq 8\,\tau(x,y)$ for every two vertices x, y in U. This result is improved in [16] for the special case where τ arises from the distances in an unweighted tree over V so that $\tau(x,y) \leq \tau'(x,y) \leq 4\,\tau(x,y)$. It follows that there exists a tree metric τ' over $V(H)$ that dominates δ such that $\text{max-str}(\tau', \delta) \leq 4\kappa$. We next show that this implies that H admits a (2κ)-center with respect to δ.

Recall that in every n-vertex tree T, there exists a vertex u, named the *centroid* of T, such that the removal of all edges incident with u disconnects T to subtrees of size at most $n/2$ each. Consider the tree T over $V(H)$ that corresponds to τ', and let u be a centroid of T. (Observe that T is not necessarily a spanning tree of H.) Let $T_1, \ldots, T_k$ be the subtrees of T obtained from the removal of all edges incident with u.

Let $\{H^1, \ldots, H^r\}$ be the $(u, 2\kappa, \delta)$-decomposition of H, namely, the connected components of H after erasing the internal edges of $B_\delta(u, 2\kappa)$. We claim that for every $1 \leq i \leq r$, there exists some $1 \leq j \leq k$ such that $V(H^i) \subseteq V(T_j)$; hence u is a (2κ)-center of H and the theorem holds. Assume by way of contradiction that the claim is false, and let H^i be a subgraph of H that falsifies the claim, i.e., such that $V(H^i) \not\subseteq V(T_j)$ for any $1 \leq j \leq k$. Since H^i is connected, it follows that there exists an edge $(x,y) \in E(H^i)$ such that $x \in V(T_j)$ and $y \in V(T_{j'})$ for some $1 \leq j < j' \leq k$. The edge (x,y) was not removed by the $(u, 2\kappa, \delta)$-decomposition of H; thus $\delta(u,x) \geq 2\kappa$ and $\delta(u,y) \geq 2\kappa$, where at least one of these two inequalities is strict. Therefore, $\delta(u,x) + \delta(u,y) > 4\kappa$. Since τ' dominates δ, it follows that $\tau'(u,x) + \tau'(u,y) > 4\kappa$. But x and y are in different subtrees obtained from the removal of the edges incident with u; therefore, $\tau'(x,y) = \tau'(u,x) + \tau'(u,y) > 4\kappa$, in contradiction to the fact that $\text{max-str}(\tau', \delta) \leq 4\kappa$. The theorem follows. □

Consider a graph G, and let H be a connected vertex induced subgraph of G. Let $\bar{H}$ be the subgraph induced on G by $V(G) - V(H)$. Denote the set of edges that cross between H and $\bar{H}$ by $F(H) = \{(x,y) \in E(G) \mid x \in V(H) \text{ and } y \in V(\bar{H})\}$. Let $W(H)$ be the set of endpoints of edges in $F(H)$. We say that H is a *κ-outspread subgraph* of G if $F(H)$ can be partitioned into two *remote parts* F_1 and F_2 with endpoints W_1

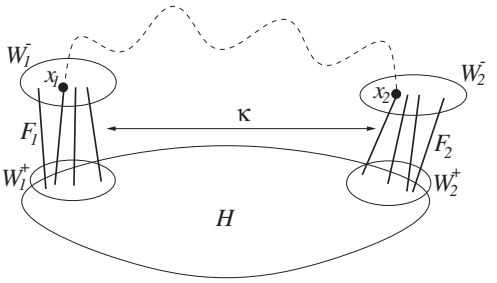


FIG. 2. *A κ-outspread subgraph H. The vertices x_1 and x_2 are connected in $\bar{H}$.*

and W_2, respectively, such that

- $\text{dist}_G(W_1, W_2) \geq \kappa$ and
- there exist two vertices $x_1 \in W_1 \cap V(\bar{H})$ and $x_2 \in W_2 \cap V(\bar{H})$ such that $\bar{H}$ admits a simple path between x_1 and x_2.

Note that $\bar{H}$ is not necessarily connected, but it is assumed to have some "connectivity" between endpoints of F_1 and endpoints of F_2. In what follows, we define $W_i^+ = W_i \cap V(H)$ and $W_i^- = W_i \cap V(\bar{H})$ for $i = 1, 2$. A κ-outspread subgraph is illustrated in Figure 2. The existence of an outspread subgraph in a graph implies a lower bound on its minimum max-stretch, as established in the following theorem.

THEOREM 3.2. *If a graph G admits a κ-outspread subgraph, then* $\text{max-str}(G) > \kappa$.

Proof. Consider a graph G, and let H be a κ-outspread subgraph of G, with remote parts F_1 and F_2 with endpoints W_1 and W_2. Let T be a spanning tree of G and suppose, toward deriving contradiction, that $\text{max-str}(T) \leq \kappa$.

We begin by strengthening the second requirement of a κ-outspread subgraph and proving that W_1^- and W_2^- are connected by a path fully contained in $T \cap \bar{H}$. To show this, we prove that T contains a subtree T' such that

- $V(T') \cap W_1^- \neq \emptyset$;
- $V(T') \cap W_2^- \neq \emptyset$;
- T' lies entirely in $\bar{H}$; and
- T' is maximal; namely, if T'' is a subtree of T and $V(T') \subset V(T'')$, then $V(T'') \cap V(H) \neq \emptyset$.

By definition, since H is a κ-outspread subgraph of G, the subgraph $\bar{H}$ admits a simple path between W_1^- and W_2^-. Let $\psi = (x_1^0, x_1^1, \ldots, x_1^s, x_2^t, x_2^{t-1}, \ldots, x_2^0)$ be the shortest such path, where $x_1^0 \in W_1^-$, $x_2^0 \in W_2^-$, $\kappa \leq \text{len}(\psi) = s + t + 1$, and $\frac{\text{len}(\psi)}{2} - 1 \leq s \leq t \leq \frac{\text{len}(\psi)}{2}$. Since ψ is a shortest path between W_1^- and W_2^-, it follows that $\text{dist}_G(x_i^j, W_i^-) = j$ for $i = 1, 2$ and every vertex x_i^j in ψ. By the definition of a κ-outspread subgraph, we have $\text{dist}_G(x_i^j, W_i^+) = j + 1$ and $\text{dist}_G(x_i^j, W_{3-i}^+) \geq \min\{\kappa + j + 1, s + t + 2 - j\}$ for $i = 1, 2$ and every x_i^j in ψ. Therefore, $\text{dist}_G(x_1^s, W_i^+) + \text{dist}_G(x_2^t, W_i^+) > \kappa$ for $i = 1, 2$. Consequently, the unique path between x_1^s and x_2^t in T does not contain any vertex in $W_1^+ \cup W_2^+$, as, otherwise, it is of length greater than κ, in contradiction to the assumption that $\text{max-str}(T) \leq \kappa$. Moreover, $\text{dist}_G(x_i^j, W_{3-i}^+) + \text{dist}_G(x_i^{j-1}, W_{3-i}^+) > \kappa$ for $i = 1, 2$ and every x_i^j and x_i^{j-1} in ψ; hence the unique path between x_i^j and x_i^{j-1} in T does not contain any vertex in W_{3-i}^+.

Let $\pi_{s,t}$ be the unique path between x_1^s and x_2^t in T. As $\pi_{s,t}$ does not contain any vertex in $W_1^+ \cup W_2^+$, it must lie entirely in $\bar{H}$. We would like to develop $\pi_{s,t}$ into a subtree T' as described above. For $i = 1, 2$, apply the following process to extend

$\pi_{s,t}$ up to a vertex in W_i^- without adding any edge of $F(H)$. Initialize the variable subtree Υ to be the path $\pi_{s,t}$. Initialize the variable integer j^{min} by $j^{min} = \min\{j \mid x_i^j \in V(\Upsilon)\}$ (j^{min} is well defined, as x_1^s and x_2^t are in Υ). If $j^{min} = 0$, then we are done since the subtree Υ now contains a W_i^- vertex and $V(\Upsilon) \subseteq V(\bar{H})$.

Otherwise, apply a "developing step" as follows. Let χ be the unique path between $x_i^{j^{min}}$ and $x_i^{j^{min}-1}$ in T. The path χ does not contain any edge in F_{3-i} since, otherwise, it also contains a vertex in W_{3-i}^+. If χ contains an edge in F_i, then it must contain a vertex in W_i^- right before that edge, in which case the subtree Υ can be extended up to a vertex in W_i^- without adding any $F(H)$ edge, and we are done. Otherwise, the path χ does not contain any $F(H)$ edge at all; thus Υ can be extended so that j^{min} decreases by a positive integral term without adding any edge in $F(H)$. We repeat the "developing step" until $j^{min} = 0$. Applying this process for $i = 1, 2$ yields a subtree T' of T such that T' contains a vertex in W_1^- and a vertex in W_2^- and $E(T') \cap F(H) = \emptyset$; hence T' lies entirely in $\bar{H}$. If T' is not maximal, then extend it by adding adjacent edges of $E(T) \cap E(\bar{H})$ as long as possible.

We now turn to label the vertices of H by their location in T with respect to the subtree T'. Pick an arbitrary vertex $r \in V(T')$ and direct the edges of T toward r. Consider a vertex v in $V(H)$, and let π_v be the unique path from v to r in T. If u is the first vertex on π_v such that $u \in V(T')$, then we say that v is *covered* by u with respect to T'. Since T' is maximal, if v is covered by u, then the edge entering u on π_v is in F and u must lie in W_i^- for some $i \in \{1, 2\}$. For $i = 1, 2$, let

$$U_i = \{v \in V(H) \mid v \text{ is covered by some vertex } u \in W_i^- \text{ with respect to } T'\}.$$

Note that $\{U_1, U_2\}$ is a partition of $V(H)$. For two vertices $x, y \in V(H)$, we say that x and y are *separated* by T' if $x \in U_1$ and $y \in U_2$ (or vice versa). Observe that this implies that the unique path between x and y in (the undirected) T contains an edge in F_1 and an edge in F_2; thus its length is at least $\kappa + 2$. It follows that for an edge (x, y) in $E(H)$, the vertices x and y cannot be separated by T' since, otherwise, we have max-str$(T) > \kappa$. But, by definition, the subgraph H is connected; therefore, $V(H) = U_i$ for some $i \in \{1, 2\}$ and $U_{3-i} = \emptyset$. Without loss of generality, suppose that $V(H) = U_1$.

Let y_2^- be a vertex in $V(T') \cap W_2^-$, and consider a neighbor $y_2^+ \in W_2^+$ of y in G. As y_2^+ is in U_1, it is covered by some vertex $y_1^- \in W_1^-$; hence $\text{dist}_T(y_2^-, y_2^+) = \text{dist}_T(y_2^-, y_1^-) + \text{dist}_T(y_2^+, y_1^-)$. Since $\text{dist}_G(W_1, W_2) \geq \kappa$, and since the distances in T dominate the distances in G, it follows that $\text{dist}_T(y_2^-, y_2^+) \geq 2\kappa$, in contradiction to the assumption that max-str$(T) \leq \kappa$. The theorem follows. ∎

4. The approximation algorithm.

4.1. Overview. The main idea behind our algorithm is that, given an n-vertex graph $\widehat{G}$ with max-str$(\widehat{G}) = \kappa$, there exists (as proved in Theorem 3.1) a vertex $u \in V(\widehat{G})$ with the property that discarding the internal edges of the ball centered at u of radius 2κ decomposes the graph into several connected components, each of size at most $n/2$, with no edges crossing between them. Using this fact, our algorithm is invoked on the integer test values $\rho \in (1, 2n]$, suspected of being 2κ. For every such test value, the algorithm tries to construct the output spanning tree $\widehat{T}$ by looking for a vertex u and a decomposition as above and merging a spanning tree T_{local} of the ball centered at u of radius $3\rho/2$ (a superset of the ball of radius ρ) with the spanning trees returned from the recursive calls made for every connected component of the decomposition. The $\rho/2$ gap between the radius of the ball used for decomposing

the graph and the radius of the ball being spanned by the tree T_{local} guarantees the connectivity of the spanning tree produced by the algorithm. A detailed description of this process is presented in the following sections.

4.2. Algorithm `Construct_Tree`. We present an algorithm named Algorithm `Construct_Tree`, that, given an n-vertex graph $\widehat{G}$ and integral test value ρ, constructs a spanning tree $\widehat{T}$ of $\widehat{G}$ with maximum stretch at most $3\rho \log n$ or reports failure. In section 6 we prove that the algorithm does not fail if $\rho \geq 2\text{max-str}(\widehat{G})$. Since the minimum max-stretch of $\widehat{G}$ is an integer in $[1, n]$, a test value $\rho \leq 2\text{max-str}(\widehat{G})$ on which the algorithm does not fail can be guessed in $O(\log n)$ attempts, to yield a $(6 \log n)$-approximation algorithm.

Algorithm `Construct_Tree` works in a "divide and conquer" (recursive) approach. Throughout the execution of the algorithm, we maintain a forest (i.e., a cycle-free subgraph containing all the vertices) $\mathcal{F}$ of the input graph $\widehat{G}$. The forest is a global data structure shared by all recursive invocations of the algorithm. Initially, the forest $\mathcal{F}$ is empty (does not contain any edge), and, upon termination of the algorithm, $\mathcal{F}$ contains the spanning tree $\widehat{T}$ of $\widehat{G}$. On each recursive invocation, Algorithm `Construct_Tree` gets a vertex induced subgraph G of $\widehat{G}$ as input and adds some edges of G to $\mathcal{F}$. Upon termination of the recursive invocation on G, the vertices of $V(G)$ all belong to a single tree in $\mathcal{F}$ (each connected component in $\mathcal{F}$ is a tree).

Consider a recursive invocation of Algorithm `Construct_Tree` on the vertex induced subgraph G of $\widehat{G}$. Due to some earlier recursive invocations, the forest $\mathcal{F}$ may already contain some edges of G. Let $\mathcal{P}$ be the partition of $V(G)$ that corresponds to the connected components of $\mathcal{F}$; that is, each cluster in $\mathcal{P}$ is a subset of $V(G)$, and two vertices $x, y \in V(G)$ are in the same cluster in $\mathcal{P}$ if and only if $\mathcal{F}$ admits a path between x and y. We refer to the partition $\mathcal{P}$ as the *connectivity partition* of $V(G)$ with respect to $\mathcal{F}$. To prevent the possibility of creating cycles in $\mathcal{F}$, the algorithm will add an edge $e \in E(G)$ to $\mathcal{F}$ only if e is external to $\mathcal{P}$. Let U be an arbitrary cluster in $\mathcal{P}$. Note that the vertex induced subgraph $G(U)$ is not necessarily connected, although every two vertices in U are connected by a path in $\mathcal{F}$ (it may be the case that some of the vertices of this path are missing in G).

Throughout we denote the metric defined by distances in $\widehat{G}$ by $\widehat{\delta}$. Let δ be the restriction of $\widehat{\delta}$ to the vertices of G. Algorithm `Construct_Tree` works as follows. It first finds a ρ-center u with respect to δ and identifies the set of clusters $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$, namely, the clusters of $\mathcal{P}$ that intersect the ball centered at u of radius $3\rho/2$. If a ρ-center cannot be found, then the algorithm halts and reports "failure." Next, a subset of the external edges of the partition $\mathcal{P}$ that induces a tree (referred to as the *local tree* T_{local}) on these clusters is added to $\mathcal{F}$. The choice of the subset of edges that induces the local tree on the clusters $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$ is made by Procedure `Construct_Local_Tree`. The edges internal to the ball centered at u of radius ρ (but not the vertices) are discarded from the graph, and subsequently the graph decomposes[3] into separate connected components $G^1, \ldots, G^r$. A recursive call is then made on the graph[4] $G(V(G^i))$ for every $1 \leq i \leq r$.

Let $\mathcal{F}'$ be the forest $\mathcal{F}$ after the edges of the local tree were added to it, and let $\mathcal{P}'$ be the connectivity partition of $V(G)$ with respect to $\mathcal{F}'$. Observe that the connec-

[3]The connected components of this graph decomposition should not be confused with the clusters of the connectivity partition $\mathcal{P}$.

[4]Observe that $G(V(G^i))$ is not necessarily identical to G^i, as some of the edges in $G(V(G^i))$ may be internal to $B_\delta(u, \rho)$. Such edges were discarded in the decomposition process, and they cannot be found in G^i.

tivity partition $\mathcal{P}'$ is obtained from the connectivity partition $\mathcal{P}$ (that corresponds to $\mathcal{F}$ before the edges of the local tree were added) by merging all clusters that intersect $B_\delta(u, 3\rho/2)$ into a single cluster, referred to as the *kernel cluster* U'. It is easy to verify (a stronger claim is proved later on) that the kernel cluster disintegrates into several connected components of the graph decomposition $G^1, \ldots, G^r$. Consequently, a pair of vertices in the kernel cluster may be *separated* on this recursive invocation. However, it is crucial for the correctness of Algorithm `Construct_Tree` that all other clusters in $\mathcal{P}'$ remain intact (see section 5). In fact, assuming that $\rho \geq 2\text{max-str}(\widehat{G})$, the $\rho/2$ gap between the radius of the ball used for decomposing the graph and that being spanned by the local tree ensures (as proved in section 6.1) that this is indeed the case.

Formally, we say that the connectivity partition $\mathcal{P}'$ is a *hub* with respect to the connected components $G^1, \ldots, G^r$ of the graph decomposition if, for every cluster $U \in \mathcal{P}' - \{U'\}$, there exists a connected component G^i, $1 \leq i \leq r$, such that $U \subseteq V(G^i)$. In other words, there is at most one cluster in $\mathcal{P}'$ that disintegrates into several connected components of the graph decomposition. (Recall that the connectivity partition is defined with respect to the global forest regardless of the connectivity in G; hence the subgraphs induced on G by its clusters are not necessarily connected to begin with. A cluster that induces a nonconnected subgraph on G may be disintegrated into several connected components even if none of its internal edges were discarded in the graph decomposition.) If $\mathcal{P}'$ is not a hub with respect to $G^1, \ldots, G^r$, then the algorithm halts and reports "failure."

The current recursive invocation of the algorithm is said to *succeed* if it manages to avoid the two failures, namely, if it finds a ρ-center and $\mathcal{P}'$ is a hub. Otherwise, we say that the current recursive invocation *fails*, in which case the algorithm should be reinvoked on $\widehat{G}$ with a larger test value ρ. A formal description of Algorithm `Construct_Tree` is given in Table 1.

TABLE 1
Algorithm `Construct_Tree`.

Input: A vertex induced subgraph G of $\widehat{G}$.
Let δ be the restriction of $\widehat{\delta}$ to the vertices $V(G)$.

1. If $|V(G)| = 1$, then return.
2. Find a ρ-center u with respect to G and δ.
   If none exists, then halt and report "failure."
3. $T_{\text{local}} \leftarrow$ `Construct_Local_Tree`(G, u).
4. Set $\mathcal{F} \leftarrow \mathcal{F} \cup T_{\text{local}}$.
5. Let $G^1, \ldots, G^r$ be the connected components of the graph remaining from G after discarding the edges in $E(B_\delta(u, \rho))$.
6. Let $\mathcal{P}'$ be the connectivity partition of $V(G)$ with respect to the (modified) forest $\mathcal{F}$.
7. If $\mathcal{P}'$ is not a hub with respect to $G^1, \ldots, G^r$, then halt and report "failure."
8. For every $1 \leq i \leq r$, invoke `Construct_Tree`$(G(V(G^i)))$.

Since the connected components $G^1, \ldots, G^r$ of the (u, ρ, δ)-decomposition of G are formed by discarding the edges internal to $B_\delta(u, \rho)$, we have the following.

OBSERVATION 4.1. *The vertex set $B_\delta(u, \rho)$ intersects G^i for every $1 \leq i \leq r$.*

4.3. Procedure `Construct_Local_Tree`. Consider a graph G. Let ξ be a metric over $V(G)$, and let $P = \{U_1, \ldots, U_k\}$ be a partition of $V(G)$. We say that the graph $\mathcal{R}$ is the *transitive graph* of G with respect to P and ξ if each cluster in P is completed

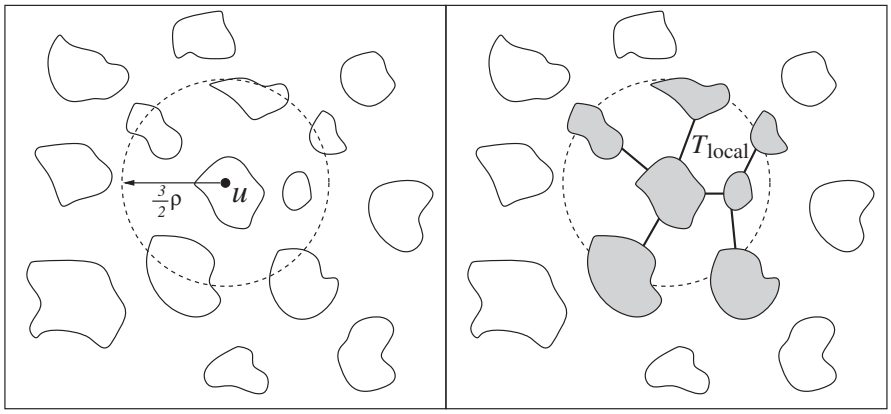


FIG. 3. *The operation of Procedure* `Construct_Local_Tree`. (a) *The clusters of* $\mathcal{P}$ *and the ball centered at* u *of radius* $3\rho/2$. (b) T_{local} *induces a tree on the clusters of* $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$. *The shaded area forms the kernel cluster* U'.

into a clique in $\mathcal{R}$, namely,

- $V(\mathcal{R}) = V(G)$ and
- $E(\mathcal{R}) = \overline{E}(P) \cup \bigcup_{1 \le i \le k} (U_i \times U_i)$.

The graph $\mathcal{R}$ is weighted with edge lengths $\ell(e) = \xi(e)$ for every $e \in E(\mathcal{R})$. (Distances in a weighted graph are defined with respect to the length of the edges.)

Recall that $\mathcal{P}$ is the connectivity partition of $V(G)$ with respect to the forest $\mathcal{F}$ and that δ is the restriction of $\widehat{\delta}$ to the vertices $V(G)$. The purpose of the following procedure, named `Construct_Local_Tree`, is to find a subset T_{local} of external edges from $\overline{E}(\mathcal{P})$ that induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$. Let $\mathcal{P}'$ be the connectivity partition of $V(G)$ with respect to $\mathcal{F} \cup T_{\text{local}}$. Note that by the choice of T_{local}, it follows that $\mathcal{P}' - \mathcal{P}$ contains a single cluster, named the *kernel cluster* U', which is the union of all the clusters in $\mathcal{P}$ that intersect $B_\delta(u, 3\rho/2)$. (See Figure 3.) The subset T_{local} should be chosen so that the diameter of the kernel cluster is not much greater than the sum of diameters of the $\mathcal{P}$-clusters it replaces (this requirement is presented formally in section 6).

In principle, this task can be achieved by using a depth-$(3\rho/2)$ shortest path tree rooted at u on the transitive graph $\mathcal{R}$. However, a naive choice of such a shortest path tree might create cycles in the cluster graph induced on $\mathcal{P}$. Procedure `Construct_Local_Tree` carefully avoids this complication. The procedure begins by constructing the transitive graph $\mathcal{R}$ of G with respect to $\mathcal{P}$ and δ. Next, the vertex set U' is initiated to consist of the cluster of u in $\mathcal{P}$, and the edge set T_{local} is initiated to be empty. The vertices of $\mathcal{R}$ are then processed in increasing order of distances from the vertex u. In section 5 we prove that whenever Procedure `Construct_Local_Tree` is invoked, the distances in $\mathcal{R}$ agree with δ; hence if the procedure processes the vertex x before it processes the vertex y, then $\delta(u, x) \le \delta(u, y)$.

Consider a vertex v when it is processed by the procedure, and let U_i be v's cluster in the partition $\mathcal{P}$. If $v \notin U'$, then the procedure adds the vertices of U_i to U' and adds the edge (w, v) to T_{local}, where w is the predecessor of v in some shortest path from u to v. Note that (w, v) is an edge in $E(G)$. This is justified since v must be the first vertex in U_i to be processed by the procedure (as otherwise, it was already in U'), and since w was processed before v (as $\text{dist}_{\mathcal{R}}(u, w) < \text{dist}_{\mathcal{R}}(u, v)$). The procedure

halts when all vertices at distance at most $3\rho/2$ from u are processed and returns the edge set T_{local}. A formal description of Procedure `Construct_Local_Tree` is given in Table 2. Procedure `Construct_Local_Tree` can be implemented as a simple variant of the well-known Dijkstra algorithm for finding shortest paths from a single source [8, 6].

TABLE 2
Procedure `Construct_Local_Tree`.

Input: A vertex induced subgraph G of $\widehat{G}$ and a vertex $u \in V(G)$.
Output: An edge set $T_{\mathrm{local}} \subseteq \overline{E}(\mathcal{P})$ that induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$.

1. Construct the transitive graph $\mathcal{R}$ of G with respect to $\mathcal{P}$ and δ.
2. Let U be u's cluster in $\mathcal{P}$.
3. Set $U' \leftarrow U$ and $T_{\mathrm{local}} \leftarrow \emptyset$.
4. Let $v_1, \ldots, v_n$ be the vertices of $\mathcal{R}$ in increasing order of distances from u, namely, $v_1 = u$ and $\mathrm{dist}_{\mathcal{R}}(v_i, u) \leq \mathrm{dist}_{\mathcal{R}}(v_{i+1}, u)$ for every $1 \leq i < n$.
5. For $i = 2, \ldots, n$, and as long as $\mathrm{dist}_{\mathcal{R}}(v_i, u) \leq \frac{3\rho}{2}$, do:
   (a) Let U_i be v_i's cluster in $\mathcal{P}$.
   (b) If $v_i \notin U'$, then do:
      i. Set $U' \leftarrow U' \cup U_i$.
      ii. Let π be a shortest path from u to v_i in $\mathcal{R}$. Let w be the predecessor of v_i in π.
      iii. Set $T_{\mathrm{local}} \leftarrow T_{\mathrm{local}} \cup (w, v_i)$.
6. return T_{local}.

OBSERVATION 4.2. *The edge set T_{local} output by the procedure induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, U')$.*

5. Correctness. In this section we prove that Algorithm `Construct_Tree` generates a spanning tree of the given graph. In what follows, $\widehat{G}$ and $\widehat{T}$ stand for the unweighted n-vertex connected graph input to the first invocation of the algorithm and the subgraph stored in $\mathcal{F}$ upon termination of the algorithm, respectively. In order to analyze the recursive algorithm, we shall *label* each recursive invocation with a string in $\mathbb{N}^*$. The labeling is done in an inductive manner. The top recursive invocation (on the graph $\widehat{G}$) is labeled with the empty string ω. Consider a recursive invocation labeled with the string σ on the graph G (which is a vertex induced subgraph of $\widehat{G}$) for some $\sigma \in \mathbb{N}^*$, and let $G^1, \ldots, G^r$ be the connected components of the graph decomposition of G (see line 5 of Algorithm `Construct_Tree`)). Then the recursive invocation on the graph $G(V(G^i))$ is labeled with the string σi for every $1 \leq i \leq r$. We refer to the recursive invocation labeled with the string σ as the *σ-recursive invocation*.

We shall use subscript σ to denote the various ingredients of the σ-recursive invocation. Let G_σ, δ_σ, and u_σ denote the input graph G, the restriction of $\widehat{\delta}$ to the vertices $V(G)$, and the center vertex u, respectively, in the σ-recursive invocation. Observe that the original graph $\widehat{G}$ input to the top recursive invocation is denoted by G_ω. Let $\mathcal{F}_\sigma$ and $\mathcal{F}'_\sigma$ be snapshots of the forest $\mathcal{F}$ at the beginning of the execution of the σ-recursive invocation and after the addition of the local tree, respectively. Let $\mathcal{P}_\sigma$ and $\mathcal{P}'_\sigma$ denote the connectivity partitions of $V(G_\sigma)$ with respect to $\mathcal{F}_\sigma$ and $\mathcal{F}'_\sigma$, respectively. Let U'_σ be the single cluster in $\mathcal{P}'_\sigma - \mathcal{P}_\sigma$ (the kernel cluster). Finally, let $\mathcal{R}_\sigma$ be the transitive graph of G_σ with respect to $\mathcal{P}_\sigma$ and δ_σ.

We begin with establishing a few fundamental facts regarding the execution of our algorithm.

LEMMA 5.1. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$.*

1. *If there exists a path π between some two vertices x and y in the graph $\widehat{G}$ such that* $\text{len}(\pi) > 1$ *and* $V(\pi) \cap V(G_\sigma) = \{x, y\}$*, then both x and y are in the same cluster in $\mathcal{P}_\sigma$. In fact, if $|\sigma| > 0$, then both x and y are in the same cluster in $\mathcal{P}'_{\sigma_{-1}}$, where σ_{-1} is the longest proper prefix of σ.*
2. *The distances in the transitive graph $\mathcal{R}_\sigma$ agree with δ_σ.*

Proof. We prove the two claims simultaneously by induction on the length of the string σ. On the top recursive invocation, which is labeled with the empty string ω, we have the following.

1. The graphs G_ω and $\widehat{G}$ are identical; hence such a path π does not exist and the first claim holds vacuously.
2. The graphs $\mathcal{R}_\omega$ and $\widehat{G}$ are identical and the second claim holds trivially.

Now assume that the claims hold for the σ_{-1}-recursive invocation, where σ_{-1} is the longest proper prefix of σ, and consider the σ-recursive invocation. We first prove claim 1. Consider a path π between the vertices x and y as in the claim. If all internal vertices of the path π (namely, vertices other than the endpoints x and y) were already missing on the previous recursion level, i.e., if $V(\pi) \cap V(G_{\sigma_{-1}}) = \{x, y\}$, then, due to the inductive hypothesis, x and y are in the same tree in $\mathcal{F}_{\sigma_{-1}}$; thus they remain in the same tree in $\mathcal{F}'_{\sigma_{-1}}$ (and in $\mathcal{F}_\sigma$) and the assertion holds. Otherwise, there must be some internal vertices of the path π that have existed in the graph $G_{\sigma_{-1}}$ on the previous recursion level.

We argue that vertex x must be in the kernel cluster $U'_{\sigma_{-1}}$. The same line of reasoning shows that $y \in U'_{\sigma_{-1}}$ as well; hence both x and y are in the same tree in $\mathcal{F}'_{\sigma_{-1}}$ (and they remain in the same tree in $\mathcal{F}_\sigma$) so that the assertion holds. Let w be the internal vertex of π that still existed in $G_{\sigma_{-1}}$ which is closest to x in π; that is, the vertex w satisfies $\text{dist}_\pi(w, x) \leq \text{dist}_\pi(w', x)$ for any vertex $w' \in V(\pi) \cap V(G_{\sigma_{-1}}) - \{x, y\}$. We consider two cases (illustrated in Figure 4).

Case (a). If (x, w) is an edge in $E(\widehat{G})$, then since w does not exist in G_σ, it follows that both w and x were in $B_{\delta_{\sigma_{-1}}}(u_{\sigma_{-1}}, \rho)$; hence $\delta_{\sigma_{-1}}(u_{\sigma_{-1}}, x) \leq \rho$. By the inductive hypothesis on claim 2, the distances in $\mathcal{R}_{\sigma_{-1}}$ agree with $\delta_{\sigma_{-1}}$; thus $\text{dist}_{\mathcal{R}_{\sigma_{-1}}}(u_{\sigma_{-1}}, x) \leq \rho$. Therefore, since Procedure `Construct_Local_Tree` halted at distance $3\rho/2$ from the center vertex $u_{\sigma_{-1}}$ (see line 5 of the procedure), it follows that x is in the kernel cluster $U'_{\sigma_{-1}}$.

Case (b). If (x, w) is not an edge in $E(\widehat{G})$, then let U_w be w's cluster in the partition $\mathcal{P}_{\sigma_{-1}}$. By the inductive hypothesis, the vertex x is in U_w as well (due to the subpath of π that starts at x and ends at w); thus w and x are in the same cluster in $\mathcal{P}_{\sigma_{-1}}$ and they remain in the same cluster $\bar{U}_w$ in $\mathcal{P}'_{\sigma_{-1}}$. Since w is not a vertex in G_σ, it follows that the cluster $\bar{U}_w$ disintegrated into different connected components in the decomposition of the graph $G_{\sigma_{-1}}$. Therefore, the cluster $\bar{U}_w$ must be the kernel cluster $U'_{\sigma_{-1}}$ since, otherwise, the recursive invocation of Algorithm `Construct_Tree` on $G_{\sigma_{-1}}$ would have failed as $\mathcal{P}'_{\sigma_{-1}}$ is not a hub (see line 7 of the algorithm).

We now turn to prove claim 2. Consider the transitive graph $\mathcal{R}_\sigma$ as in the claim. Let x and y be any two vertices in $V(G_\sigma)$, and let π be a shortest path between x and y in $\widehat{G}$. Some segments of the path π (namely, some consecutive sequences of vertices and edges between them) may be missing in the graph G_σ. Consider such a missing segment, and let x' and y' be the vertices in π right before and right after that missing segment, respectively; i.e., the path π consists of a segment between x and x', a segment between x' and y', and a segment between y' and y, where the internal vertices of the segment between x' and y' are all missing in G_σ. By claim 1, the vertices x' and y' must be in the same tree in $\mathcal{F}_\sigma$ (see Figure 5); hence they are in the same cluster in

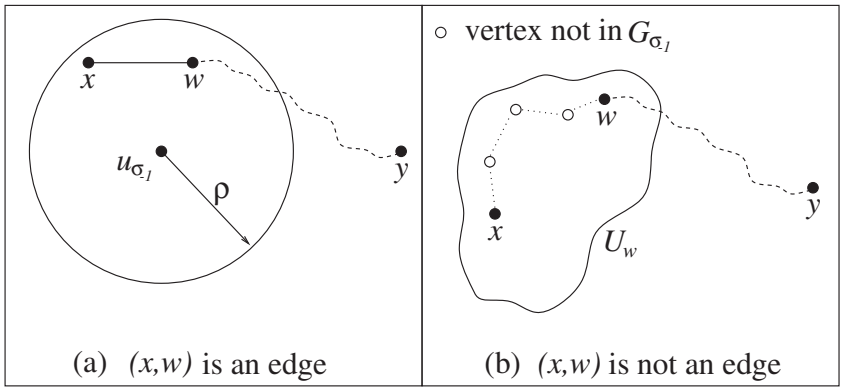


FIG. 4. *The two cases in the proof of Lemma* 5.1*'s claim* 1. *In both cases, the vertex* x *is in the kernel cluster* $U'_{\sigma_{-1}}$.

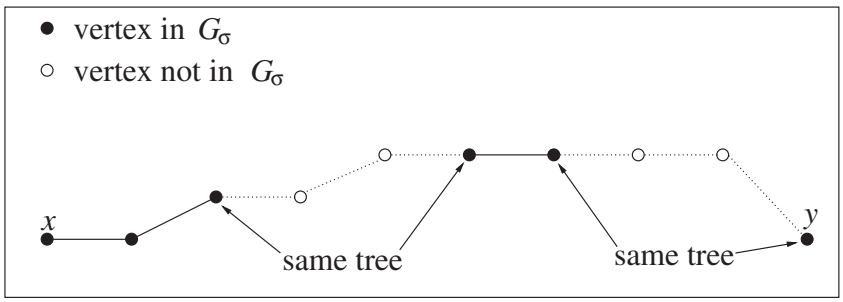


FIG. 5. *Proof of claim* 2 *of Lemma* 5.1. *The path* π *contains some missing segments.*

the partition $\mathcal{P}_\sigma$, and by the definition of the transitive graph $\mathcal{R}_\sigma$, we conclude that $\mathrm{dist}_{\mathcal{R}_\sigma}(x', y') = \delta_\sigma(x', y')$. The assertion follows as $\delta_\sigma(x, y) = \mathrm{len}(\pi)$. □

By claim 2 of the last lemma and since Procedure `Construct_Local_Tree` halts at distance $3\rho/2$ from the source vertex u (see line 5 of Procedure `Construct_Local_Tree`), we have the following.

COROLLARY 5.2. *Consider the* σ*-recursive invocation for some string* σ *in* $\mathbb{N}^*$. *The kernel cluster* U'_σ *satisfies*

$$B_{\delta_\sigma}(u_\sigma, \rho) \subseteq B_{\delta_\sigma}(u_\sigma, 3\rho/2) \subseteq U'_\sigma.$$

Next, we prove that the subgraph $\widehat{T}$ output by the algorithm is indeed a spanning tree of $\widehat{G}$.

PROPOSITION 5.3. *Consider an edge* $(x, y) \in E(\widehat{G})$. *There exists some string* σ *in* $\mathbb{N}^*$ *such that both* x *and* y *are in the kernel cluster* U'_σ.

Proof. Let σ be the longest string in $\mathbb{N}^*$ such that both x and y are in G_σ. This means that x and y are separated in the graph decomposition on the σ-recursive invocation; hence $x, y \in B_{\delta_\sigma}(u_\sigma, \rho)$. The assertion follows as Corollary 5.2 guarantees that $B_{\delta_\sigma}(u_\sigma, \rho) \subseteq U'_\sigma$. □

Since edges are added to $\mathcal{F}$ only if they are external to the connectivity partition $\mathcal{P}$ (see Procedure `Construct_Local_Tree`), it follows that $\widehat{T}$ is cycle-free. To see that $\widehat{T}$ is connected, consider an arbitrary edge $(x, y) \in E(\widehat{G})$. By Proposition 5.3, at some stage during the execution of the algorithm, both x and y belong to the kernel cluster

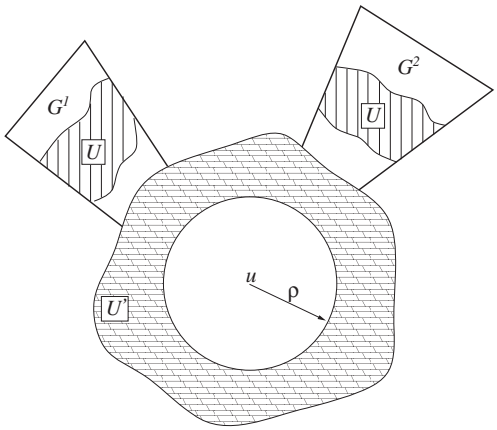


FIG. 6. *The cluster U decomposes into the connected components G^1 and G^2 of the graph decomposition.*

U'; hence the forest $\mathcal{F}$ at that stage admits a path between x and y. Therefore, $\widehat{T}$ admits a path between x and y. As $\widehat{G}$ is connected, we have the following.

THEOREM 5.4. *The graph $\widehat{T}$ output by Algorithm* `Construct_Tree` *is a spanning tree of the input graph $\widehat{G}$.*

6. Analysis. In this section we analyze the performance of our algorithm. In section 6.1 we prove that the recursive invocations of Algorithm `Construct_Tree` on all vertex induced subgraphs of $\widehat{G}$ succeed as long as the test value ρ is at least 2max-str($\widehat{G}$). The approximation ratio guaranteed by our algorithm is then established in section 6.2, where we prove that if the algorithm succeeds to generate a spanning tree $\widehat{T}$ with test value ρ, then max-str($\widehat{T}$) $\leq 3\rho\lceil\log n\rceil$. In section 6.3 we analyze the running time of the algorithm.

6.1. Successful recursive invocation. Consider some invocation of Algorithm `Construct_Tree` on the vertex induced subgraph G of $\widehat{G}$ with test value $\rho \geq$ 2max-str($\widehat{G}$). The proof that a ρ-center can be found (see line 2 of the algorithm) follows directly from Theorem 3.1. Let $G^1, \ldots, G^r$ be the connected components of the graph decomposition on this recursion level. In order to prove that the connectivity partition $\mathcal{P}'$ of $V(G)$ with respect to $\mathcal{F}'$ is a hub (see line 7 of the algorithm), we have to show that the kernel cluster is the only cluster in $\mathcal{P}'$ that decomposes into several connected components.

Suppose toward deriving contradiction that there exists a cluster U in $\mathcal{P}' - \{U'\}$ that decomposes into several connected components of the graph decomposition. (The execution of the algorithm halts at that stage.) Formally, let $X_i = U \cap V(G^i)$, where without loss of generality $X_i \neq \emptyset$ for $1 \leq i \leq t$ and $X_i = \emptyset$ for $t < i \leq r$, and suppose that $t \geq 2$. Figure 6 illustrates a cluster $U \in \mathcal{P} - \{U'\}$ that decomposes into two connected components.

Let δ be the restriction of $\widehat{\delta}$ to the vertices of G, and let u be the current ρ-center. By the definition of the graph decomposition, every edge e in $E(G)$ that crosses between $V(G^i)$ and $V(G^j)$, where $1 \leq i < j \leq r$, satisfies $e \in B_\delta(u,\rho) \times B_\delta(u,\rho)$. Thus, by Observation 4.1 and Corollary 5.2, we have the following.

OBSERVATION 6.1. *Every edge e that crosses between $V(G^i)$ and $V(G^j)$, where $1 \leq i < j \leq r$, is in $U' \times U'$. Furthermore, the kernel cluster U' satisfies $U' \cap V(G^i) \neq \emptyset$*

for every $1 \leq i \leq r$.

Let T be the tree in $\mathcal{F}$ that corresponds to the cluster U, and let H be the subgraph induced on $\widehat{G}$ by $V(T)$. We prove that H is a $\rho/2$-outspread subgraph of $\widehat{G}$ (refer to section 3 for the definition of an outspread subgraph), in contradiction to Theorem 3.2.

Following the notation of section 3, let $\bar{H}$ denote the subgraph induced on $\widehat{G}$ by the vertices in $V(\widehat{G}) - V(T)$. Let $F(H)$ be the set of edges in $E(\widehat{G})$ that cross between H and $\bar{H}$, and let $W(H)$ be the set of endpoints of edges in $F(H)$. Let $F_1 = \{e \in F(H) \mid e \in E(G^1)\}$; namely, the edge set F_1 consists of the edges that cross between U vertices and other vertices in the connected component G^1 of the decomposition of G. Let $F_2 = F(H) - F_1$. By the choice of U and by Observation 6.1, the connected components $G^1, \ldots, G^t$ contain some vertices in U and some vertices not in U; thus F_1 and F_2 are nonempty.

Let W_i be the endpoints of edges in F_i for $i = 1, 2$. Let $W_i^+ = W_i \cap V(H)$ and $W_i^- = W_i \cap V(\bar{H})$. In order to prove that H is a $\rho/2$-outspread subgraph of $\widehat{G}$, we have to show that the distance between W_1 and W_2 in $\widehat{G}$ is large and to establish some connectivity properties of H and $\bar{H}$. We start with the latter.

Clearly, the vertex induced subgraph H is connected (as T is a tree in $\mathcal{F}$). For $\bar{H}$ we have the following proposition.

PROPOSITION 6.2. *There exist two vertices* $x_1 \in W_1^-$ *and* $x_2 \in W_2^-$ *such that* $\bar{H}$ *admits a simple path between* x_1 *and* x_2.

Proof. Let T' be the tree in $\mathcal{F}$ that corresponds to the cluster U'. Consider the subgraph G' induced on $\widehat{G}$ by $V(T') \cup (V(G) - U)$. This subgraph is not necessarily connected; however, the vertices of T' all belong to the same connected component G'' of G' as T' is connected by its own rights. Since $U' \subseteq V(T') \subseteq V(G'')$, it follows due to Observation 6.1 that $V(G'') \cap V(G^j) \neq \emptyset$ for every $1 \leq j \leq t$. Therefore, there exists a vertex x_i in $W_i^- \cap V(G'')$ for $i = 1, 2$. The proposition follows as G'' is a (connected) subgraph of $\bar{H}$. ∎

We now turn to prove that $\mathrm{dist}_{\widehat{G}}(W_1, W_2) \geq \rho/2$. We first argue that if two adjacent vertices were separated in the graph decomposition on some recursive invocation, then on the subsequent recursive invocations, they both lie far away from the boundary of their corresponding clusters.

PROPOSITION 6.3. *Consider the* σ*-recursive invocation for some string* σ *in* $\mathbb{N}^*$. *Let* x *be a vertex in* $V(G_\sigma)$, *and let* U_x *be its cluster in the connectivity partition* $\mathcal{P}_\sigma$. *If* x *admits a neighbor* y *in* $\widehat{G}$ *such that* $y \notin V(G_\sigma)$, *then* $B_{\delta_\sigma}(x, \rho/2) \subseteq U_x$.

Proof. Let α be the longest string in $\mathbb{N}^*$ such that both x and y are in G_α. The vertices x and y must have been separated in the graph decomposition on the α-recursive invocation (α is a proper prefix of σ). As $(x, y) \in E(G_\alpha)$, this could have happened only if both x and y were in $B_{\delta_\alpha}(u_\alpha, \rho)$. By Corollary 5.2, the kernel cluster U'_α satisfies $B_{\delta_\alpha}(u_\alpha, 3\rho/2) \subseteq U'_\alpha$; thus $w \in U'_\alpha$ for every vertex $w \in V(G_\alpha)$ such that $\mathrm{dist}_{\widehat{G}}(w, x) = \delta_\alpha(w, x) \leq \rho/2$, and w remains in x's tree in the forest from that moment on. Therefore, if such a vertex w still exists in the graph G_σ, then it must lie in U_x. ∎

Recall that the vertex set W_1 consists of the endpoints of edges in F_1. Since every edge in F_1 crosses between different clusters of the connectivity partition $\mathcal{P}$, we have the following.

COROLLARY 6.4. *Every vertex* $x \in V(\widehat{G})$ *such that* $\mathrm{dist}_{\widehat{G}}(x, W_1) \leq \rho/2$ *is in* $V(G)$.

Assume by way of contradiction that $\mathrm{dist}_{\widehat{G}}(W_1, W_2) < \rho/2$. Let π be a shortest path in $\widehat{G}$ between any vertex in W_1 and any vertex in W_2. Since F is a cut, it follows that π lies entirely in H or in $\bar{H}$, but it does not cross between them (as, otherwise, π is not shortest). Corollary 6.4 implies that π lies entirely in the graph G as every vertex in π is at distance less than $\rho/2$ from W_1. By Observation 6.1, every path from W_1 to W_2 in G must intersect U'. Since $V(H) \cap V(G) = U$, it follows that π cannot lie entirely in H; thus π connects between W_1^- and W_2^- in the subgraph induced by $V(G) - U$ on G (and, in particular, in $\bar{H}$).

Recall that Algorithm `Construct_Tree` employs the ball centered at u of radius ρ to decompose the graph (see line 5 of the algorithm), while the ball of radius $(3\rho/2)$ is contained in the kernel cluster U' (see Corollary 5.2). Since every vertex in W_i^- has a neighbor outside U' for $i = 1, 2$, we conclude that $\mathrm{dist}_{\widehat{G}}(W_i^-, B_\delta(u, \rho)) \geq \rho/2$. As $\mathrm{len}(\pi) < \rho/2$, π cannot contain any edge internal to $B_\delta(u, \rho)$. But this implies that the all vertices of π should have been in the same connected component of the (u, ρ, δ)-decomposition of G, in contradiction to the fact that π has one endpoint in G^1 and another in G^j for some $1 < j \leq t$. This establishes the following theorem.

THEOREM 6.5. *Given an input graph* $\widehat{G}$ *and a test value* $\rho \geq 2\text{max-str}(\widehat{G})$*, Algorithm* `Construct_Tree` *succeeds on each recursive invocation.*

6.2. Approximation ratio. In this section we prove that if Algorithm `Construct_Tree` succeeds on each recursive invocation when invoked with test value ρ, then the output spanning tree $\widehat{T}$ satisfies $\mathrm{dist}_{\widehat{T}}(x, y) \leq 3\rho\lceil \log n \rceil$ for every edge (x, y) in $E(\widehat{G})$ (recall that n is the number of vertices in the input graph $\widehat{G}$). By Proposition 5.3, we know that at some stage during the execution of the algorithm, the vertices x and y are both in the kernel cluster. Consequently we would like to bound the diameter of the kernel cluster with respect to the distances in the output spanning tree $\widehat{T}$. In an attempt to establish such a bound, we will actually prove a stronger claim, stating that the sum of the diameters of many clusters, the kernel cluster being one of them, is sufficiently small.

Let $\widehat{\tau}$ be the metric defined by the distances in the output tree $\widehat{T}$. For a collection of vertex subsets $\Lambda = \{U_1, \ldots, U_k\}$, let

$$\varphi(\Lambda) \;=\; \sum_{1 \leq i \leq k} \mathrm{diam}_{\widehat{\tau}}(U_i).$$

Our subsequent analysis revolves on bounding the measure $\varphi(\mathcal{P}'_\sigma)$ as a function of the length of the string $\sigma \in \mathbb{N}^*$.

PROPOSITION 6.6. *Consider the* σ*-recursive invocation for some string* σ *in* $\mathbb{N}^*$*. The kernel cluster* U'_σ *satisfies*

$$\mathrm{diam}_{\widehat{\tau}}(U'_\sigma) \;\leq\; 3\rho + \varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)).$$

Proof. The execution of Procedure `Construct_Local_Tree` halts at distance $3\rho/2$ from the source vertex u_σ (see line 5 of the procedure). Therefore, the tree induced by T_{local} on the clusters of $\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)$ is of depth at most $3\rho/2$ (when the cluster containing u_σ is considered to be the root). Consider some two vertices $x, y \in U'_\sigma$, and let π be the unique path between x and y in $\widehat{T}$. The path π contains at most $2 \cdot (3\rho/2) = 3\rho$ edges from the local tree T_{local}. Let $\pi' = \pi - T_{\mathrm{local}}$ be the rest of the path π. Since π' is a collection of segments internal to the $\mathcal{P}_\sigma$-clusters replaced by U'_σ,

it follows that $\text{len}(\pi') \leq \varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma))$. Therefore, $\text{len}(\pi) \leq 3\rho + \varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma))$, and the assertion holds. ∎

The next proposition enables us to bound the sum of the diameters of the clusters in $\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)$.

LEMMA 6.7. *Consider the σ-recursive invocation for some string $\sigma = \alpha i$, where $\alpha \in \mathbb{N}^*$ and $i \in \mathbb{N}$. For every cluster U in $\mathcal{P}_\sigma$, there exists a cluster $\bar{U}$ in $\mathcal{P}'_\alpha$ such that $U \subseteq \bar{U}$.*

Proof. Suppose toward deriving contradiction that there exists a cluster $U \in \mathcal{P}_\sigma$ such that $U \nsubseteq \bar{U}$ for any cluster $\bar{U}$ in $\mathcal{P}'_\alpha$. This implies that $\mathcal{F}_\sigma$ contains a path π between some two vertices $x, y \in U$ that are not in the same cluster in $\mathcal{P}'_\alpha$. Let π be such a path of minimum length. The path π must satisfy $V(\pi) \cap V(G_\sigma) = \{x, y\}$, since, otherwise, there exists some subpath of π with endpoints $x', y' \in U$, where x' and y' are in different clusters in $\mathcal{P}'_\alpha$, in contradiction to π being the shortest such path.

The path π must contain some edges that do not exist in $\mathcal{F}'_\alpha$. Every such edge was added to $\mathcal{F}$ by some $\alpha j \beta$-recursive invocation, where $j \neq i$ and $\beta \in \mathbb{N}^*$. The graph $G_{\alpha j \beta}$ must contain some internal vertices of the path π; thus $\text{len}(\pi) > 1$. Therefore, Lemma 5.1, when applied to the σ-recursive invocation, implies that x and y are in the same cluster in $\mathcal{P}'_\alpha$, which derives a contradiction. The assertion follows. ∎

We are now ready to establish the main lemma of this section.

LEMMA 6.8. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$. The connectivity partition $\mathcal{P}_\sigma$ satisfies*

$$\varphi(\mathcal{P}_\sigma) \;\leq\; 3\rho(|\sigma| + 1).$$

Proof. We prove the assertion by induction on the length of the string σ. The only nonsingleton cluster in the connectivity partition $\mathcal{P}_\omega$ is the kernel cluster U'_ω, whose diameter with respect to the distances in $\widehat{T}$ is at most 3ρ. The diameter of a singleton cluster is 0. Therefore, the sum of the diameters of all clusters in $\mathcal{P}_\omega$ is at most 3ρ, and the assertion holds. Let $\sigma_{-1} \in \mathbb{N}^*$ be the longest proper prefix of σ, and assume that the assertion holds for σ_{-1}. Lemma 6.7 implies that for every cluster $U \in \mathcal{P}'_\sigma - \{U'_\sigma\}$, there exists a cluster $\bar{U} \in \mathcal{P}'_{\sigma_{-1}} - \mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)$ such that $U \subseteq \bar{U}$. Since $\text{diam}_{\widehat{\tau}}(U)$ is monotone under set inclusion, i.e., $U \subseteq \bar{U}$ implies $\text{diam}_{\widehat{\tau}}(U) \leq \text{diam}_{\widehat{\tau}}(\bar{U})$, it follows that

$$\begin{aligned}\varphi(\mathcal{P}'_\sigma) - \text{diam}_{\widehat{\tau}}(U'_\sigma) = \varphi(\mathcal{P}'_\sigma - \{U'_\sigma\}) &\leq \varphi(\mathcal{P}'_{\sigma_{-1}} - \mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)) \\ &= \varphi(\mathcal{P}'_{\sigma_{-1}}) - \varphi(\mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)).\end{aligned}$$

Thus

$$\varphi(\mathcal{P}'_\sigma) \leq \text{diam}_{\widehat{\tau}}(U'_\sigma) - \varphi(\mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)) + \varphi(\mathcal{P}'_{\sigma_{-1}}).$$

Another application of Lemma 6.7 guarantees that for every cluster $U \in \mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)$, there exists a cluster $\bar{U} \in \mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)$ such that $U \subseteq \bar{U}$; hence $\varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)) \leq \varphi(\mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma))$, and we can bound

$$\varphi(\mathcal{P}'_\sigma) \leq \text{diam}_{\widehat{\tau}}(U'_\sigma) - \varphi(\mathcal{I}(\mathcal{P}'_\sigma, U'_\sigma)) + \varphi(\mathcal{P}'_{\sigma_{-1}}).$$

Proposition 6.6 is employed to deduce that

$$\varphi(\mathcal{P}'_\sigma) \leq 3\rho + \varphi(\mathcal{P}'_{\sigma_{-1}}).$$

The assertion follows by the inductive hypothesis as $|\sigma_{-1}| = |\sigma| - 1$. ∎

Let (x, y) be an arbitrary edge in $E(\widehat{G})$. By Proposition 5.3, there exists a string $\sigma \in \mathbb{N}^*$ such that both x and y are in the kernel cluster U'_σ. Therefore,

$$\mathrm{dist}_{\widehat{T}}(x, y) \leq \mathrm{diam}_{\widehat{\tau}}(U'_\sigma) \leq \varphi(\mathcal{P}'_\sigma).$$

By Lemma 6.8, and since the depth of the recursion is at most $\lceil \log n \rceil$ (as the size of each graph input to the recursive algorithm decreases by a factor of at least 2), we conclude that

$$\mathrm{dist}_{\widehat{T}}(x, y) \leq 3\rho(|\sigma| + 1) \leq 3\rho\lceil \log n \rceil.$$

As the maximum stretch of a spanning tree is always obtained on a pair of vertices that form an edge in the original graph [5], Theorems 5.4 and 6.5 imply the following.

THEOREM 6.9. *Given an n-vertex graph $\widehat{G}$, Algorithm* `Construct_Tree` *can be invoked with $O(\log n)$ different test values to generate a spanning tree $\widehat{T}$ of $\widehat{G}$ satisfying* $\text{max-str}(\widehat{T}) = O(\log n) \cdot \text{max-str}(\widehat{G})$.

6.3. Running time. We now turn to analyze the running time of Algorithm `Construct_Tree` when invoked with test value ρ on input graph $\widehat{G}$ with $\widehat{n}$ vertices and $\widehat{m}$ edges. The distance metric $\widehat{\delta}$ is constructed in a preprocessing stage in time $O(\widehat{n}\widehat{m})$ (a trivial implementation); thus in what follows we assume that $\widehat{\delta}$ and its restrictions to subsets of $V(\widehat{G})$ are known.

Consider a recursive invocation of the algorithm on vertex induced subgraph G of $\widehat{G}$, and let δ be the restriction of $\widehat{\delta}$ to the vertices $V(G)$. Denote $n = |V(G)|$ and $m = |E(G)|$. Given a vertex $u \in V(G)$, we can construct the graph G' remaining from G after the edges in $E(B_\delta(u, \rho))$ are discarded in time $O(m)$. Identifying the connected components of G' can be done in time $O(m)$ as well; hence we can decide whether u is a ρ-center with respect to G and δ in time $O(m)$. By repeating this process for every vertex $u \in V(G)$, a ρ-center is found (if one exists) in time $O(nm)$.

Procedure `Construct_Local_Tree` is merely a variant of Dijkstra's single source shortest paths algorithm on the transitive graph $\mathcal{R}$ that has $O(n^2)$ edges; hence it can be implemented to run in time $O(n^2)$. Using a disjoint-set data structure [6], connectivity queries in the forest $\mathcal{F}$ are answered in near-constant time, and the condition that the connectivity partition $\mathcal{P}'$ is a hub is verified in near-linear time. Therefore, the dominant term in the running time of the recursive invocation on G is proportional to nm. Accounting for all recursive invocations, the running time of Algorithm `Construct_Tree` is $O(\min\{\widehat{n}\widehat{m}\log\widehat{n}, \widehat{n}^3\})$.

7. Tightness of the analysis. In this section we prove that the analysis presented in section 6 is tight. Recall that our approximation algorithm invokes Algorithm `Construct_Tree` with different test values $\rho \in (1, 2n]$, finally returning the output of a successful invocation with the smallest ρ. For the sake of the analysis, in this section we assume that the approximation algorithm ignores the spanning trees output by successful invocations with larger test values, although some of these spanning trees may admit smaller maximum stretch.

LEMMA 7.1. *For every $d \in \mathbb{N}_{>0}$ there exists an unweighted graph G_d with $n(d) = \Theta(2^d)$ vertices such that* $\text{max-str}(G_d)$ *is constant while Algorithm* `Construct_Tree`*, when invoked on G_d with test value $\rho = 2$, constructs a spanning tree with maximum stretch $\Omega(\log n)$.*

Proof. Given an integer $d \geq 1$, construct the unweighted graph G_d as follows. Let T and T' be two complete binary trees of depth d, with roots r and r', respectively.

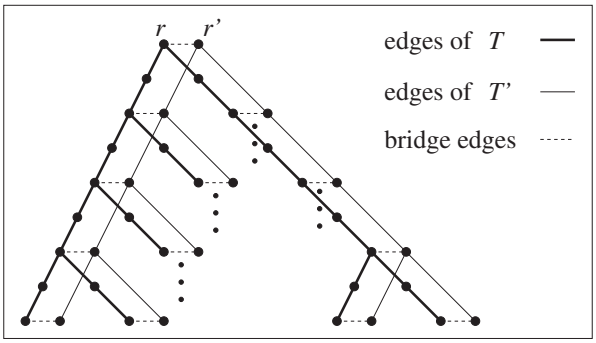


FIG. 7. G_d *for* $d = 4$.

Connect the two trees to each other by adding a *bridge* edge between every pair of corresponding vertices. Split every edge (x, y) in T by adding a new vertex z and replacing (x, y) with new edges (x, z) and (y, z). Figure 7 illustrates the construction of G_4.

Since the number of vertices in a depth d complete binary tree is $2^{d+1} - 1$ and since the number of new vertices added to T is $2^{d+1} - 2$ (equal to the number of edges in a depth d complete binary tree), it follows that

$$n(d) \;=\; |V(G_d)| \;=\; 2\,(2^{d+1} - 1) \,+\, 2^{d+1} - 2 \;=\; 6 \cdot 2^d - 4.$$

It is easy to verify that the spanning tree obtained by removing the edges of T' and remaining with the edges of T plus the bridge edges has maximum stretch 4. On the other hand, when Algorithm `Construct_Tree` is invoked on G_d with test value $\rho = 2$, if the "original" vertices of T (those that existed in T before splitting the edges) are chosen to be the 2-centers on each recursive invocation, then the spanning tree $\widehat{T}$ returned by the algorithm is merely the union of T and T' with their roots connected by an edge, i.e., $E(\widehat{T}) = E(T) \cup E(T') \cup \{(r, r')\}$.

Let l be an arbitrary leaf in T, and let l' be its corresponding leaf in T'. The unique path between l and l' in $\widehat{T}$ goes via the edge (r, r'), and it is of length $2d+1+d = 3d+1$. Therefore, $\widehat{T}$ has maximum stretch $\Omega(d) = \Omega(\log(n(d)))$. □

8. Hardness of approximation. As mentioned in section 1.2, it is NP-hard to decide, for an arbitrary unweighted graph G, whether or not max-str$(G) \leq 4$ [5]. Moreover, since the maximum stretch of a tree on an unweighted graph is always obtained on a pair of vertices that form an edge in the original graph, it follows that max-str(G) must be an integer. Therefore, we have the following.

COROLLARY 8.1. *It is NP-hard to approximate the MMST problem on unweighted graphs by a ratio better than* $5/4$.

We show that unless P = NP, the problem cannot be approximated additively by a term of $o(n)$.

LEMMA 8.2. *It is NP-hard to distinguish between unweighted graphs with minimum max-stretch at most* $5k - 1$ *and those with minimum max-stretch at least* $6k - 1$ *for any positive integer* k.

Proof. The proof is by reduction from the problem of deciding whether an arbitrary unweighted graph has minimum max-stretch at most 4. Consider some positive integer k. Given an arbitrary unweighted graph G, construct the unweighted graph G'_k by replacing every edge $(u, v) \in E(G)$ with a unique path $P_{u,v}$ of length k between u

and v. Observe that every spanning tree T' of G'_k corresponds to a spanning tree T of G where the edge (u,v) is in $E(T)$ if and only if all the edges of $P_{u,v}$ are in $E(T')$. Moreover, if the spanning tree T' of G'_k corresponds to the spanning tree T of G, then, since T' is connected, it follows that

$$|E(T') \cap E(P_{u,v})| \;=\; \begin{cases} k & \text{if } (u,v) \in E(T), \\ k-1 & \text{otherwise} \end{cases}$$

for every $(u,v) \in E(G)$; i.e., the tree T' is missing at most a single edge from every such path $P_{u,v}$. Therefore, $\text{max-str}(T', G'_k) = k \cdot (\text{max-str}(T,G)+1) - 1$. Thus if G has minimum max-stretch at most 4, then G'_k has minimum max-stretch at most $5k-1$. Otherwise, if G has minimum max-stretch at least 5, then G'_k has minimum max-stretch at least $6k-1$. □

Given an approximation algorithm A for the MMST problem on unweighted graphs and an unweighted graph G, let $A(G)$ denote the tree returned by A when invoked on G.

COROLLARY 8.3. *If there exist some real $\delta = o(n)$ and $\epsilon > 0$ and an approximation algorithm A for the MMST problem on unweighted graphs with performance guarantee*

$$\text{max-str}(A(G), G) \;\leq\; \delta + (6/5 - \epsilon) \cdot \text{max-str}(G)$$

for every unweighted graph G, then $P = NP$.

REFERENCES

[1] N. ALON, R. M. KARP, D. PELEG, AND D. WEST, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput., 24 (1995), pp. 78–100.

[2] I. ALTHÖFFER, G. DAS, D. DOBKIN, D. JOSEPH, AND J. SOARES, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[3] Y. BARTAL, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1998, pp. 161–168.

[4] L. CAI, *NP-completeness of minimum spanner problems*, Discrete Appl. Math., 48 (1994), pp. 187–194.

[5] L. CAI AND D. G. CORNEIL, *Tree spanners*, SIAM J. Discrete Math., 8 (1995), pp. 359–387.

[6] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[7] M. J. DEMMER AND M. HERLIHY, *The arrow distributed directory protocol*, in Proceedings of the 12th International Symposium on Distributed Computing (DISC), 1998, Springer-Verlag, New York, pp. 119–133.

[8] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[9] D. DOR, S. HALPERIN, AND U. ZWICK, *All pairs almost shortest paths*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Washington, DC, 1996, pp. 452–461.

[10] M. ELKIN, Y. EMEK, D.A. SPIELMAN, AND S.-H. TENG, *Lower-stretch spanning trees*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2005, pp. 494–503.

[11] Y. EMEK AND D. PELEG, *Approximating minimum max-stretch spanning trees on unweighted graphs*, in Proceedings of the Fifteenth ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2004, pp. 261–270.

[12] S. P. FEKETE AND J. KREMER, *Tree spanners in planar graphs*, in Proceedings of the 24th International Workshop on Graph Theoretic Concepts in Computer Science, Springer-Verlag, London, 1998, pp. 298–309.

[13] J. FAKCHAROENPHOL, S. RAO, AND K. TALWAR, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2003, pp. 448–455.

[14] G. Galbiati, *On min-max cycle bases*, in Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC), Springer-Verlag, London, 2001, pp. 116–123.
[15] A. Gupta, *Steiner points in tree metrics don't (really) help*, in Proceedings of the Twelfth ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2001, pp. 220–227.
[16] R. Hassin and A. Levin, *Minimum restricted diameter spanning trees*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), Springer-Verlag, New York, 2002, pp. 175–184.
[17] M. Herlihy, F. Kuhn, S. Tirthapura, and R. Wattenhofer, *Dynamic analysis of the arrow distributed protocol*, Theory Comput. Syst., 39 (2006), pp. 875–901.
[18] C. Liebchen and G. Wünsch, *The zoo of tree spanner problems*, Discrete Appl. Math., 156 (2008), pp. 569–587.
[19] D. Peleg and E. Reshef, *Low complexity variants of the arrow distributed directory*, J. Comput. System Sci., 63 (2001), pp. 474–485.
[20] D. Peleg and A.A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.
[21] D. Peleg and D. Tendler, *Low Stretch Spanning Trees for Planar Graphs*, Technical report MCS01-14, The Weizmann Institute of Science, Rehovot, Israel, 2001.
[22] D. Peleg and J. D. Ullman, *An optimal synchronizer for the hypercube*, SIAM J. Comput., 18 (1989), pp. 740–747.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1782–1802

THE CSP DICHOTOMY HOLDS FOR DIGRAPHS WITH NO SOURCES AND NO SINKS (A POSITIVE ANSWER TO A CONJECTURE OF BANG-JENSEN AND HELL)*

LIBOR BARTO†, MARCIN KOZIK‡, AND TODD NIVEN†

Abstract. Bang-Jensen and Hell conjectured in 1990 (using the language of graph homomorphisms) a constraint satisfaction problem (CSP) dichotomy for digraphs with no sources or sinks. The conjecture states that the CSP for such a digraph is tractable if each component of its core is a cycle and is NP-complete otherwise. In this paper we prove this conjecture and, as a consequence, a conjecture of Bang-Jensen, Hell, and MacGillivray from 1995 classifying hereditarily hard digraphs. Further, we show that the CSP dichotomy for digraphs with no sources or sinks agrees with the algebraic characterization conjectured by Bulatov, Jeavons, and Krokhin in 2005.

Key words. constraint satisfaction problem, graph homomorphism, smooth digraphs

AMS subject classifications. 68R10, 08A70

DOI. 10.1137/070708093

1. Introduction. The history of the constraint satisfaction problem (CSP) goes back more than thirty years and begins with the work of Montanari [Mon74] and Mackworth [Mac77]. Since that time many combinatorial problems in artificial intelligence and other areas of computer science have been formulated in the language of CSPs. The study of such problems, under this common framework, has applications in database theory [Var00], machine vision recognition [Mon74], temporal and spatial reasoning [SV98], truth maintenance [DD96], technical design [NL], scheduling [LALW98], natural language comprehension [All94], and programming language comprehension [Nad]. Numerous attempts to understand the structure of different CSPs has been undertaken, and a wide variety of tools ranging from statistical physics (e.g., [ANP05, KMRT$^+$07]) to universal algebra (e.g., [JCG97]) has been employed. Methods and results developed in seemingly disconnected branches of mathematics transformed the area. The conjecture proved in this paper resisted the approaches based in combinatorics and theoretical computer science for nearly twenty years. Only recent developments in the structural theory of finite algebras provided tools strong enough to solve this problem.

For the last ten years the study of CSPs has also been a driving force in theoretical computer science. The dichotomy conjecture of Feder and Vardi, published in [FV99], has origins going back to 1993. The conjecture states that a CSP, for any fixed language, is solvable in polynomial time or NP-complete. Therefore the class of CSPs would be a subclass of NP avoiding problems of intermediate difficulty. The logical

*Received by the editors November 14, 2007; accepted for publication (in revised form) August 1, 2008; published electronically January 9, 2009. A part of this article appeared, in a preliminary form, in the Proceedings of the 40th ACM Symposium on Theory of Computing, STOC'08. This work was partly supported by the Eduard Cech Center grant LC505.

http://www.siam.org/journals/sicomp/38-5/70809.html

†Eduard Čech Center, Charles University, 186 75 Praha, Czech Republic (barto@karlin.mff.cuni.cz, niven@karlin.mff.cuni.cz). The first author was supported by the Grant Agency of the Czech Republic under grant 201/06/0664 and by the project of Ministry of Education under grant MSM 0021620839.

‡Eduard Čech Center, Charles University, 186 75 Praha, Czech Republic, and Algorithmics Research Group, Jagiellonian University, 30-072 Kraków, Poland (marcin.kozik@tcs.uj.edu.pl).

characterization of the class of CSPs (see [FV99] and [Kun]) provides arguments in support of the dichotomy; nevertheless the conjecture remains open.

One of the results of [FV99] shows that the CSP dichotomy conjecture is equivalent to the CSP dichotomy conjecture restricted to digraphs. Therefore the CSPs can be defined in terms of the (di)graph homomorphisms studied in graph theory for over forty years (cf. [Sab61, HP64, Lev73]). It adds a new dimension to a well-established problem and shows the importance of solving CSPs for digraphs. The classification of the complexity of the **H**-coloring problems for undirected graphs, discovered by Hell and Nešetřil [HN90], is an important step and provides a starting point towards proving, or refuting, the CSP dichotomy conjecture. There have since appeared many papers on the complexity of digraph coloring problems (see, e.g., [BJH90, BJHM95, Fed01, GWW92, HNZ96a, HNZ96b, HNZ96c, HZZ93, Mac91, Zhu95]), but as yet, no plausible conjecture on a graph theoretical classification has been proposed. Bang-Jensen and Hell [BJH90] did, however, conjecture a classification (implying the dichotomy) for the class of digraphs with no sources or sinks. Their conjecture significantly generalizes the result of Hell and Nešetřil.

In 1995, Bang-Jensen, Hell, and MacGillivray (in [BJHM95]) introduced the notion of hereditarily hard digraphs and conjectured their classification. Surprisingly, they were able to show that this conjecture and the one given in [BJH90] are equivalent. In this paper we prove the conjecture of Bang-Jensen and Hell and, as a consequence, the conjecture of Bang-Jensen, Hell, and MacGillivray.

Our paper relies on the interconnection between the CSP and algebra as first discovered by Jeavons, Cohen, and Gyssens in [JCG97] and refined by Bulatov, Jeavons, and Krokhin in [BJK05]. Using this connection, Bulatov, Jeavons, and Krokhin conjectured a full classification of the NP-complete CSPs. For a small taste of results in the direction of proving this classification, see [BIM$^+$06, Bul06, Dal05, Dal06, KV07]. A particularly interesting example, demonstrating the potency of the algebraic approach, is Bulatov's proof of the result of Hell and Nešetřil (see [Bul05]). A recent, purely algebraic result of Maróti and McKenzie [MM07] is one of the key ingredients in the proof of the conjecture of Bang-Jensen and Hell. This provides further evidence supporting the extremely strong bond between the CSP and universal algebra.

2. Preliminaries. We assume that the reader possesses a basic knowledge of universal algebra and graph theory. For an easy introduction to the notions of universal algebra that are not defined in this paper, we invite the reader to consult the monographs [BS81] and [MMT87]. Further information concerning the structural theory of finite algebras (called tame congruence theory) can be found in [HM88]. For an explanation of the basic terms in graph theory and graph homomorphisms, we recommend [HN04]. Finally, for an introduction to the connections between universal algebra and the CSP, we recommend [BJK05].

Throughout the paper we deviate from the standard definition of the CSP, with respect to a fixed language (found in, e.g., [BKJ00]), in favor of an equivalent definition from [FV99, LZ06]. The definitions of a *relational structure*, a *homomorphism*, or a *polymorphism* are presented further in this section in their full generality as well as in restriction to directed graphs.

A *directed graph* (or *digraph*) is a pair $\mathbf{G} = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. More generally *a relational structure* $\mathcal{T} = (T, \mathcal{R})$ is an ordered pair, where T is a finite nonempty set and $\mathcal{R}$ is a finite set of finitary relations on T indexed by a set J. Let d_j denote the arity of the relation $R_j \in \mathcal{R}$. The indexed set of all the d_j constitutes the *signature* of $\mathcal{T}$.

A vertex of a digraph is called a *source* (resp., a *sink*) if it has no incoming (resp., outgoing) edges. An *oriented walk* is a sequence of vertices $(v_0, \dots, v_{n-1})$ such that $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$ for any $i < n-1$ and the *length* of such a walk is $n-1$. A *walk* is an oriented walk such that $(v_i, v_{i+1}) \in E$ for any $i < n-1$. A *closed walk* is a walk such that $v_0 = v_{n-1}$. Given a digraph $\mathbf{G}$, we sometimes denote the set of vertices of $\mathbf{G}$ by $V(\mathbf{G})$ and similarly the edges of $\mathbf{G}$ by $E(\mathbf{G})$. A graph with n vertices is a *cycle* if its vertices can be ordered (i.e., $V = \{v_0, \dots, v_{n-1}\}$) in such a way that $E = \{(v_i, v_j) | \, j = i+1 \mod n\}$.

A *graph homomorphism* is a function between sets of vertices of two graphs mapping edges to edges. A graph is 3-colorable if and only if it maps homomorphically to the complete graph on three vertices (without loops). The notion of *colorability* is generalized using graph homomorphisms: a digraph, say $\mathbf{G}$, is $\mathbf{H}$-colorable if there exists a homomorphism mapping $\mathbf{G}$ to $\mathbf{H}$. For two relational structures of the same signature, say $\mathcal{T} = (T, \mathcal{R})$ and $\mathcal{U} = (U, \mathcal{S})$, a map $h : T \to U$ is a *homomorphism* if $h(T_j) \subseteq R_j$ for all $j \in J$ (where $h(T_j)$ is computed pointwise).

A *digraph polymorphism* is a homomorphism from a finite Cartesian power of a graph to the graph itself. Precisely, for a digraph $\mathbf{G} = (V, E)$ a function $h : V^n \to V$ is a *polymorphism* of $\mathbf{G}$ if, for any vertices $a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1} \in V$,

$$\text{if } (a_i, b_i) \in E \text{ for all } i < n, \text{ then } (h(a_0, \dots, a_{n-1}), h(b_0, \dots, b_{n-1})) \in E.$$

The notion of a polymorphism is defined for relational structures as well. A *polymorphism* h of $\mathcal{T}$ is an operation $h : T^n \to T$ such that, for all relations $R \in \mathcal{R}$ of arity m, if

$$(a_{i,0}, a_{i,1}, \dots, a_{i,m-1}) \in R \qquad \text{for all } i < n,$$

then

$$(h(a_{0,0}, a_{1,0}, \dots, a_{n-1,0}), \dots, h(a_{0,m-1}, a_{1,m-1}, \dots, a_{n-1,m-1})) \in R.$$

A digraph $\mathbf{G} = (V, E)$ *retracts* to an induced subgraph $\mathbf{H} = (W, F)$ if there is an endomorphism $h : V \to V$ such that $h(V) = W$ and $h(a) = a$ for all $a \in W$. Such a map h is called a *retraction*. A *core* of a digraph is a minimal induced subgraph to which the digraph retracts. The definition of retraction and core clearly generalize to relational structures. It is a trivial fact that, for any digraph $\mathbf{H}$, and for a core of $\mathbf{H}$, say $\mathbf{H}'$, the set of $\mathbf{H}$-colorable digraphs coincides with the set of $\mathbf{H}'$-colorable digraphs.

An *algebra* is a tuple $\mathbf{A} = (A, f_0, \dots)$ consisting of a nonempty set A (called a *universe* of $\mathbf{A}$) and *operations* on A. An *operation* f_i is an n_i-ary function $f_i : A^{n_i} \to A$. With each operation we associate an *operation symbol* and, by an abuse of notation, denote it also by f_i. A set $B \subseteq A$ is a *subuniverse* of an algebra $\mathbf{A}$ if, for any number i, the operation f_i restricted to B^{n_i} has all the results in B. For a nonempty subuniverse B of an algebra $\mathbf{A}$ the algebra $\mathbf{B} = (B, f_0', \dots)$ (where f_i' is a restriction of f_i to B^{n_i}) is a *subalgebra* of $\mathbf{A}$. A *power* of an algebra $\mathbf{A}$ has a universe A^k and the operations f_i'' derived from the operations of $\mathbf{A}$ by computing coordinatewise. A subalgebra of a power of an algebra is often called a *subpower*. A *term function* of an algebra is any function that can be obtained by a composition using the operations of the algebra together with all the projections. A *term* is a formal way of denoting such a composition; i.e., a term function is attached to an algebra, but a term can be computed in a subalgebra or a power as well as in the original algebra. A set $C \subseteq A$

generates a subuniverse B in an algebra $\mathbf{A}$ if B is the smallest subuniverse containing C—such a subuniverse always exists and can be obtained by applying all the term functions of the algebra $\mathbf{A}$ to all the choices of arguments coming from C.

In this paper all relational structures, digraphs, and algebras are assumed to be finite.

3. The main result. For a relational structure $\mathcal{T} = (T, R)$ we define the language $\mathrm{CSP}(\mathcal{T})$, of relational structures with the same signature as $\mathcal{T}$, to be

$$\mathrm{CSP}(\mathcal{T}) = \{\, \mathcal{U} \mid \text{there is a homomorphism from } \mathcal{U} \text{ to } \mathcal{T} \,\}.$$

Alternatively we can view $\mathrm{CSP}(\mathcal{T})$ as a decision problem:

INPUT: a relational structure $\mathcal{U}$ with the same signature as $\mathcal{T}$
QUESTION: does there exist a homomorphism from $\mathcal{U}$ to $\mathcal{T}$?

In either approach we are concerned with the computational complexity (of membership of the language, or of the decision problem, respectively) for a given relational structure. The CSP dichotomy conjecture proposed in [FV99] can be stated as follows.

THE CSP DICHOTOMY CONJECTURE. *For a relational structure $\mathcal{T}$ the problem* $\mathrm{CSP}(\mathcal{T})$ *is solvable in polynomial time or NP-complete.*

The (di)graph coloring problems can be viewed as special cases of the CSP. Although a digraph $\mathbf{H} = (W, F)$ is technically different from a relational structure, the set of $\mathbf{H}$-colorable digraphs is obviously polynomially equivalent to the CSP for an appropriate relational structure, and therefore we denote the class of all $\mathbf{H}$-colorable digraphs by $\mathrm{CSP}(\mathbf{H})$. Due to the reduction presented in [FV99], every CSP is polynomially equivalent to a digraph homomorphism problem. Thus we can restate the CSP dichotomy conjecture in the following way.

THE CSP DICHOTOMY CONJECTURE. *For a fixed digraph $\mathbf{H}$, deciding whether a given digraph is $\mathbf{H}$-colorable is either NP-complete or solvable in polynomial time.*

This brings us to the main problem of the paper, a conjecture nearly ten years older than the CSP dichotomy conjecture, and a special case of it. It deals with digraphs with no sources or sinks and was first formulated by Bang-Jensen and Hell in [BJH90].

THE CONJECTURE OF BANG-JENSEN AND HELL. *Let $\mathbf{H}$ be a digraph without sources or sinks. If each component of the core of $\mathbf{H}$ is a cycle, then* $\mathrm{CSP}(\mathbf{H})$ *is polynomially decidable. Otherwise* $\mathrm{CSP}(\mathbf{H})$ *is NP-complete.*

Note that the above conjecture is a substantial generalization of the $\mathbf{H}$-coloring result of Hell and Nešetřil [HN90].

The notion of hereditarily hard digraphs was introduced by Bang-Jensen, Hell, and MacGillivray in [BJHM95]. A digraph $\mathbf{H}$ is said to be *hereditarily hard* if the $\mathbf{H}'$-coloring problem is NP-complete for all loopless digraphs $\mathbf{H}'$ that contain $\mathbf{H}$ as a subgraph (not necessarily induced). The following conjecture was posed and shown to be equivalent to the Bang-Jensen and Hell conjecture in [BJHM95].

THE CONJECTURE OF BANG-JENSEN, HELL, AND MACGILLIVRAY. *Let $\mathbf{H}$ be a digraph. If the digraph $R(\mathbf{H})$ (which is obtained by iteratively removing the sources and sinks from $\mathbf{H}$ until none remain) does not admit a homomorphism to a cycle of length greater than one, then $\mathbf{H}$ is hereditarily hard. Otherwise there exists a loopless digraph $\mathbf{H}'$ containing $\mathbf{H}$ (as a not necessarily induced subgraph) such that $\mathbf{H}'$-coloring is solvable in a polynomial time.*

In this section we prove the Bang-Jensen and Hell conjecture and therefore the conjecture of Bang-Jensen, Hell, and MacGillivray. In this proof we assume Theorem 3.1, which will be proved in the subsequent sections of the paper. The reasoning

uses weak near unanimity operations[1] and Taylor operations (used only to connect Theorems 3.2 and 3.3, and therefore not defined here [HM88, Tay77, LZ06]).

THEOREM 3.1. *If a digraph without sources or sinks admits a weak near unanimity polymorphism, then it retracts to the disjoint union of cycles.*

It is easy to see that the colorability by digraphs retracting to a disjoint union of cycles is tractable (see, e.g., [BJH90]). It remains to prove the NP-completeness of the digraphs not retracting to such a union. Before we do so, we recall two fundamental results.

It follows from [HM88, Lemma 9.4 and Theorem 9.6] that a part of the result of Máróti and McKenzie [MM07, Theorem 1.1] can be stated as follows.

THEOREM 3.2 (see [MM07]). *A finite relational structure $\mathcal{T}$ admits a Taylor polymorphism if and only if it admits a weak near unanimity polymorphism.*

The following result was originally proved in [BKJ00] and [LZ03] and, as stated below, can be found in [LZ06, Theorem 2.3]. It relies on a connection between relational structures and varieties generated by algebras of their polymorphisms. A lack of a Taylor polymorphism in such an algebra implies an existence of a "trivial" algebra in a variety and NP-completeness of the associated CSP.

THEOREM 3.3 (see [LZ06]). *Let $\mathcal{T}$ be a relational structure which is a core. If $\mathcal{T}$ does not admit a Taylor polymorphism, then* $\mathrm{CSP}(\mathcal{T})$ *is NP-complete.*

If a digraph $\mathbf{H}$ without sources or sinks does not retract to a disjoint union of cycles, then its core $\mathbf{H}'$ also does not. Thus, by Theorem 3.1, it follows that $\mathbf{H}'$ does not admit a weak near unanimity polymorphism, and by Theorems 3.2 and 3.3 it follows that $\mathrm{CSP}(\mathbf{H}')$ is NP-complete, completing the proof of the conjecture of Bang-Jensen and Hell.

The conjecture (posed in [BKJ00]), classifying the CSPs from the algebraic point of view, can be stated as follows (see, e.g., [LZ06]).

THE ALGEBRAIC CSP DICHOTOMY CONJECTURE. *Let $\mathcal{T}$ be a relational structure that is a core. If $\mathcal{T}$ admits a Taylor polymorphism, then* $\mathrm{CSP}(\mathcal{T})$ *is polynomial time solvable. Otherwise* $\mathrm{CSP}(\mathcal{T})$ *is NP-complete.*

Note that the proof of the conjecture of Bang-Jensen and Hell immediately implies that the structure of the NP-complete digraph coloring problems agrees with the algebraic CSP dichotomy conjecture. The remainder of the paper is dedicated to the proof of the Theorem 3.1.

4. Notation. In this section we introduce the notation required throughout the remainder of the paper.

4.1. Neighborhoods in graphs. For a fixed digraph $\mathbf{G} = (V, E)$ we denote $(a, b) \in E$ by $a \to b$, and we use $a \xrightarrow{k} b$ to say that there is a directed walk from a to b of length precisely k. More generally we call a digraph $\mathbf{H}$ a *pattern* if $V(\mathbf{H}) = \{0, \dots, n-1\}$ and $(u, v) \in E$ if and only if $|u - v| = 1$ and $(v, u) \notin E$. We denote patterns by lowercase Greek letters and, for a pattern α, we write $a \xrightarrow{\alpha} b$ if there exists a homomorphism ϕ from α into $\mathbf{G}$ such that $\phi(0) = a$ and $\phi(n-1) = b$. In such a case we say that a and b can be connected via the pattern α. The oriented walk connecting vertices a and b and consisting of the images of elements of α under ϕ is a *realization* of the pattern. For any $W \subseteq V$ we define

$$W^{+n} = \{v \in V \mid (\exists w \in W)\ w \xrightarrow{n} v\}$$

[1] A weak near unanimity operation is a function such that, for any choice of arguments a, b, $w(b, a, \dots, a) = w(a, b, \dots, a) = \cdots = w(a, a, \dots, b)$ and $w(a, \dots, a) = a$. These operations are described in more detail in section 4.

and similarly

$$W^{-n} = \{v \in V \mid (\exists w \in W)\ v \xrightarrow{n} w\}.$$

We define $W^0 = W$, and write a^{+n} (resp., a^{-n}, a^0) instead of $\{a\}^{+n}$ (resp., $\{a\}^{-n}, \{a\}^0$) for any $a \in V$. More generally, for a pattern α, we write

$$W^{\alpha} = \{v \in V \mid (\exists w \in W)\ w \xrightarrow{\alpha} v\}.$$

As before, we use a^{α} for $\{a\}^{\alpha}$. Sometimes, for ease of presentation, we write $a \xrightarrow{k,n} b$ to denote $a \xrightarrow{k} b$ and $a \xrightarrow{n} b$.

4.2. Digraph path powers. Let $\mathbf{G} = (V, E)$ be a digraph and α be a pattern. We define a path power of the digraph $\mathbf{G}$, which we denote by $\mathbf{G}^{\alpha}$, in the following way: the vertices of the power are the vertices of the digraph $\mathbf{G}$, and a pair $(c, d) \in V^2$ is an edge in $\mathbf{G}^{\alpha}$ if and only if $c \xrightarrow{\alpha} d$ in $\mathbf{G}$. Moreover, we set $\mathbf{G}^{+n} = \mathbf{G}^{\alpha}$ for the pattern α consisting of n arrows pointing forward. Note that if $f : V^m \to V$ is a polymorphism of $\mathbf{G}$, then it is also a polymorphism of any path power of this digraph. Path powers are special cases of primitive positive definitions (used in, e.g., [Bul05]) or indicator constructions introduced in [HN90] in order to deal with the colorability problem for undirected graphs.

4.3. Components. A *connected* digraph is a digraph such that there exists an oriented walk, consisting of at least one edge, between every choice of two vertices. A *strongly connected* digraph is a digraph such that, for every choice of two vertices, there is a walk connecting them. By a *component* (resp., *strong component*) of a digraph $\mathbf{G}$, we mean a maximal (under inclusion) induced subgraph that is connected (resp., strongly connected). Note that, according to this definition, a single vertex with the empty set of edges is not connected, and thus not every digraph decomposes into a union of components (or strong components). Given a digraph $\mathbf{G}$ with no sources or sinks, we say that a strong component $\mathbf{H}$ of $\mathbf{G}$ is a *top component* if $V(\mathbf{H})^{+1} = V(\mathbf{H})$. Similarly, we say that a strong component $\mathbf{H}$ of $\mathbf{G}$ is a *bottom component* if $V(\mathbf{H})^{-1} = V(\mathbf{H})$.

4.4. Algebraic length. The following definition is taken from [HNZ96b]. For a pattern α we define the *algebraic length* $al(\alpha)$ to be

$$al(\alpha) = |\{\text{edges going forward in } \alpha\}| - |\{\text{edges going backward in } \alpha\}|.$$

An *algebraic length of an oriented walk* is a shorthand expression for an algebraic length of a pattern which can be realized as such an oriented walk—the pattern is always clear from the context. For a digraph $\mathbf{G} = (V, E)$ we set

$$al(\mathbf{G}) = \min\{i > 0 \mid (\exists v \in V)\ (\exists \text{ a pattern } \alpha)\ v \xrightarrow{\alpha} v \text{ and } al(\alpha) = i\}$$

whenever the set on the right-hand side is nonempty and ∞ otherwise. In case of strongly connected digraphs in section 7 the algebraic length can be equivalently defined (cf. Corollary 5.7) as the greatest common divisor of the lengths of closed walks in a digraph. We note that for digraphs with no sources or sinks (or with a closed walk) the algebraic length of a nonempty digraph is always a natural number. It is folklore (cf. [HN04, Proposition 5.19]) that a connected digraph $\mathbf{G}$ retracts to a cycle if and only if it contains a closed walk of length $al(\mathbf{G})$.

4.5. Algebraic notation. By $\overline{a}$ we denote the tuple $(a, a, \dots, a)$ (the arity will always be clear by the context), and by $\overrightarrow{a}$ we denote the tuple $(a_0, a_1, \dots, a_n)$. Further, we extend the notation $\overline{a}$ to the sets in the following way. For a set W let $\overline{W}$ be an appropriate Cartesian power of W. Thus, for example, given a vertex a of a digraph $\mathbf{G}$, the set $\overline{a^{+n}}$ is the collection of all tuples whose coordinates are vertices reachable by a walk of length n from a.

An idempotent operation on a set A is an operation, say $f : A^n \to A$, such that $f(\overline{a}) = a$ for all $a \in A$. In accordance with [MM07], by a weak near unanimity operation we understand an idempotent operation $w(x_0, \dots, x_{n-1})$ that satisfies

$$w(y, x, \dots, x) = w(x, y, \dots, x) = \cdots = w(x, x, \dots, y),$$

for any choice of x and y in the underlying set. Moreover, for a term t of arity n, we define

$$t^{(i)}(x_0, x_1, \dots, x_{n-1}) = t(x_{n-i}, x_{n-i+1}, \dots, x_0, x_1, \dots, x_{n-i-1}),$$

for each $0 \le i < n$, where addition on the indices is performed modulo n.

5. Preliminary results on digraphs. We start with a number of basic results describing the connection between digraphs and their path powers. The following lemma reveals the behavior of the algebraic lengths of oriented walks in powers of a digraph.

LEMMA 5.1. *Let $\mathbf{G}$ be a digraph without sources or sinks. Let α be a pattern of algebraic length k, and let $a \xrightarrow{\alpha} b$ in $\mathbf{G}$. Then $a \xrightarrow{\beta} b$ in $\mathbf{G}^{+k}$ for some pattern β of algebraic length one.*

Proof. For a fixed, large enough number j, consider all oriented walks in $\mathbf{G}$ of the form $a \xrightarrow{l_1} a_1 \xleftarrow{l_2} a_2 \xrightarrow{l_3} \cdots \leftrightarrows a_{l_j} = b$, where $l_1 - l_2 + \cdots \pm l_j = k$. We will show that at least one of these walks has all the l_i's divisible by k. Let us choose an oriented walk in which k divides all the l_i in a maximal initial segment of the i, and let l_{i_0} be the last element of this segment. If $i_0 + 1 < j$, then (assuming without loss of generality that i_0 is odd) the walk

$$a \xrightarrow{l_1} \cdots \xrightarrow{l_{i_0}} a_{i_0} \xleftarrow{l_{i_0+1}} a_{i_0+1} \xrightarrow{l_{i_0+2}} a_{i_0+2} \cdots$$

can be altered, using the fact that a_{i_0+1} (and possibly other vertices) is not a source, to obtain

$$a \xrightarrow{l_1} \cdots \xrightarrow{l_{i_0}} a_{i_0} \xleftarrow{l'_{i_0+1}} a'_{i_0+1} \xrightarrow{l'_{i_0+2}} a_{i_0+2} \cdots,$$

where l'_{i_0+1} is greater than l_{i_0+1} and is divisible by k. This contradicts the choice of i_0.

If, on the other hand, $i_0 + 1 = j$, the number k divides $l_1 - l_2 + \cdots \pm l_{i_0}$ and, using the fact that $l_1 - l_2 + \cdots \pm l_{i_0} \mp l_{i_0+1} = k$, we infer that k divides l_{i_0+1}, again contradicting the choice of i_0. Thus $i_0 = j$ and we can find an oriented walk $a \xrightarrow{l_1} a_1 \xleftarrow{l_2} a_2 \xrightarrow{l_3} \cdots a_{l_j} = b$ with $l_1 - l_2 + \cdots \pm l_j = k$, where each l_i is divisible by k. This shows that a is connected to b via a pattern of algebraic length one in $\mathbf{G}^{+k}$. ∎

As a consequence we obtain the following fact.

COROLLARY 5.2. *Let $\mathbf{G}$ be a digraph, without sources or sinks, such that $al(\mathbf{G}) = 1$. Then $al(\mathbf{G}^{+k}) = 1$ for any natural number k.*

Proof. Let $a \xrightarrow{\alpha} a$, where α is a pattern of algebraic length one. Then, by following a realization of α k-many times, we obtain $a \xrightarrow{\beta} a$ in $\mathbf{G}$ for a pattern β of algebraic length k. Now the statement follows from the previous lemma. □

Theorem 3.1 is proved in section 7 for strongly connected digraphs first, and therefore we need some preliminary results on such digraphs. The following very simple lemma is needed to prove some of the further corollaries in this section.

LEMMA 5.3. *Let c be a vertex in a strongly connected digraph. Then the greatest common divisor (GCD) of the lengths of the closed walks in this digraph is equal to the GCD of the lengths of the closed walks containing c.*

Proof. Suppose, for contradiction, that the GCD, say n', of the lengths of the closed walks containing c is bigger than the GCD of the lengths of the closed walks for the entire digraph. Then there exists a walk $d \xrightarrow{l} d$ of length l such that n' does not divide l. On the other hand, since the digraph is strongly connected, $c \xrightarrow{l'} d$ and $d \xrightarrow{l''} c$ for some numbers l', l''. The number n', by definition, divides $l' + l''$ and $l' + l + l''$ and thus divides l, a contradiction. □

Moreover, the following easy proposition holds.

PROPOSITION 5.4. *Let $\mathbf{G}$ be a connected digraph $\mathbf{G}$ and α be a pattern. If $a \xrightarrow{\alpha} a$ for a vertex a in $\mathbf{G}$, then the number $al(\mathbf{G})$ divides $al(\alpha)$.*

Proof. Let $\mathbf{G}$ be a connected digraph and, for some vertex a, $a \xrightarrow{\alpha} a$ via a pattern α. Let b be a vertex in $\mathbf{G}$ such that $b \xrightarrow{\beta} b$ for a pattern β satisfying $al(\beta) = al(\mathbf{G})$. Since $\mathbf{G}$ is connected there is a pattern γ such that $b \xrightarrow{\gamma} a$ and thus $b \xrightarrow{\gamma} a \xrightarrow{\alpha} a \xrightarrow{\gamma'} b$ with $al(\gamma') = -al(\gamma)$. Following appropriate walks, we can obtain an oriented walk, from b to b, of algebraic length $al(\alpha) - k \cdot al(\mathbf{G})$, for any number k. The minimality of $al(\mathbf{G})$ implies that $al(\mathbf{G})$ divides $al(\alpha)$. □

The following lemma is heavily used in the proof of Theorem 3.1 for strongly connected digraphs in section 7.

LEMMA 5.5. *If, for a strongly connected digraph $\mathbf{G} = (V, E)$, the GCD of the lengths of the closed walks in $\mathbf{G}$ is equal to one, then*

$$(\exists m)\ (\forall a, b \in V)\ (\forall n)\ \text{if } n \geq m, \text{ then } a \xrightarrow{n} b.$$

Proof. Fix an arbitrary element $c \in V$. By Lemma 5.3 we find some closed walks containing c such that their lengths $k_1, \ldots, k_i$ satisfy $GCD(k_1, \ldots, k_i) = 1$. Thus c is contained in a closed walk of length l whenever l is a linear combination of $k_1, \ldots, k_i$ with nonnegative integer coefficients. It is easy to see that there is a natural number m' such that, for every $n' \geq m'$, n' can be expressed as such a linear combination; hence c is in a closed walk of length n' for each such n'. Now it suffices to set $m = m' + 2|V|$ since, for arbitrary vertices $a, b \in V$, there are walks of length at most $|V|$ from a to c and from c to b. □

The following easy corollary follows.

COROLLARY 5.6. *For a strongly connected digraph $\mathbf{G}$ with GCD of the lengths of the closed walks equal to one, and for any number n, the digraph $\mathbf{G}^{+n}$ is strongly connected.*

For strongly connected digraphs, the GCD of the lengths of the closed walks and the algebraic length of the digraph coincide.

COROLLARY 5.7. *For a strongly connected digraph, the GCD of the lengths of the closed walks is equal to the algebraic length of the digraph.*

Proof. Let us fix a digraph $\mathbf{G} = (V, E)$ and denote by n the GCD of the lengths of the closed walks in $\mathbf{G}$. Since, by Proposition 5.4, the algebraic length of $\mathbf{G}$ divides the length of every closed walk in $\mathbf{G}$, $al(\mathbf{G})$ divides n.

Conversely, let $a = a_0 \xrightarrow{l_0} b_0 \xleftarrow{k_0} a_1 \xrightarrow{l_1} \cdots \xleftarrow{k_{m-1}} a_m = a$ be a realization of a pattern of algebraic length $al(\mathbf{G})$. Let k_i' be such that $b_i \xleftarrow{k_i} a_{i+1} \xleftarrow{k_i'} b_i$ for all i. Note that n divides $k_i + k_i'$ and $\sum_{i<m} l_i + \sum_{i<m} k_i'$. Thus n divides $\sum_{i<m} l_i - \sum_{i<m} k_i = al(\mathbf{G})$, which shows that $n \leq al(\mathbf{G})$, and the lemma is proved. ∎

Finally, we remark that if α is a pattern of algebraic length one and $\mathbf{G}$ has no sources and no sinks, then $E(\mathbf{G}^\alpha) \supseteq E(\mathbf{G})$. In particular, if $al(\mathbf{G}) = 1$, then $al(\mathbf{G}^\alpha) = 1$.

6. A connection between graphs and algebra. In this section we present basic definitions and results concerning the connection between digraphs and algebras. Let $\mathbf{G} = (V, E)$ be a digraph admitting a weak near unanimity polymorphism $w(x_0, x_1, \ldots, x_{h-1})$. We associate with $\mathbf{G}$ an algebra $\mathbf{A} = (V, w)$ and note that E is a subuniverse of $\mathbf{A}^2$. Note that for any subuniverse of $\mathbf{A}$, say W, we can define the digraph $\mathbf{G}_{|W} = (W, E \cap W \times W)$ (or $(W, E_{|W})$) which admits the weak near unanimity polymorphism $w|_{W^h}$, and the algebra $(W, w|_{W^h})$ is a subalgebra of $\mathbf{A}$. For the remainder of this section we assume that $\mathbf{G}$ and $\mathbf{A}$ are as above.

The first lemma describes the influence of the structure of the digraph on the subuniverses of the algebra.

LEMMA 6.1. *For any subuniverse W of $\mathbf{A}$ the sets W^{+1} and W^{-1} are subuniverses of $\mathbf{A}$.*

Proof. Take any elements $a_0, \ldots, a_{h-1}$ from W^{+1} and choose $b_0, \ldots, b_{h-1} \in W$ such that $b_i \to a_i$ for all i. Then $w(b_0, \ldots, b_{h-1}) \to w(a_0, \ldots, a_{h-1})$ showing that $w(a_0, \ldots, a_{h-1}) \in W^{+1}$, and the claim is proved. The proof for W^{-1} is similar. ∎

Since the weak near unanimity operation is idempotent, all the one element subsets of V are subuniverses of $\mathbf{A}$. Using the previous lemma, the following result follows trivially.

COROLLARY 6.2. *For any $a \in V$, any pattern α, and any number n, the sets a^{+n}, a^{-n}, and a^α are subuniverses of $\mathbf{A}$.*

Subuniverses of $\mathbf{A}$ can also be obtained in another way.

LEMMA 6.3. *Let $\mathbf{H}$ be a strong component of $\mathbf{G}$. Assume that the GCD of the lengths of the cycles in $\mathbf{H}$ is equal to one. Then $V(\mathbf{H})$ is a subuniverse of $\mathbf{A}$.*

Proof. Using Lemma 5.5, we find a number m such that there is a walk $b \xrightarrow{m} c$ in $\mathbf{H}$ for all $b, c \in V(\mathbf{H})$. Fix a vertex $a \in V(\mathbf{H})$. There is a walk $a \xrightarrow{m} b$ for all $b \in V(\mathbf{H})$ and a walk $c \xrightarrow{m} a$ for all $c \in V(\mathbf{H})$. Thus, $V(\mathbf{H}) = a^{+m} \cap a^{-m}$ is a subuniverse. ∎

We present a second construction leading to a subuniverse of the algebra.

LEMMA 6.4. *If $\mathbf{H} = (W, F)$ is the largest induced subgraph of $\mathbf{G}$ without sources or sinks, then W is a subuniverse of $\mathbf{A}$.*

Proof. Clearly, the vertices of $\mathbf{H}$ can be described as those having arbitrarily long walks to and from them. Since $\mathbf{G}$ is finite, there exists a natural number k such that

$$W = \{w \mid (\exists v, v' \in V)\ v \xrightarrow{k} w \ \text{ and } \ w \xrightarrow{k} v'\}.$$

Thus $W = V^{+k} \cap V^{-k}$, and we are done, since both sets on the right-hand side are subuniverses. ∎

7. Strongly connected digraphs. In this section we present a proof Theorem 3.1 in the case of strongly connected digraphs.

THEOREM 7.1. *If a strongly connected digraph of algebraic length k admits a weak near unanimity polymorphism, then it contains a closed walk of length k (and thus retracts to a cycle of length k).*

Using Corollary 5.7, the result can be restated in terms of the GCD of the lengths of closed walks in $\mathbf{G}$, and we will freely use this duality. Theorem 7.1 is a consequence of the following result.

THEOREM 7.2. *If a strongly connected digraph $\mathbf{G}$ of algebraic length one admits a weak near unanimity polymorphism, then it contains a loop.*

We present a proof of Theorem 7.1, assuming Theorem 7.2, and devote the remainder of this section to proving Theorem 7.2.

Proof of Theorem 7.1. Fix an arbitrary vertex c in a strongly connected digraph of algebraic length k. Using Lemma 5.3 and Corollary 5.7, we obtain closed walks containing c with the GCD of their lengths equal to k. Thus, in the path power $\mathbf{G}^{+k}$, the GCD of lengths of closed walks containing c is equal to one. Let $\mathbf{H}$ be the strong component of $\mathbf{G}^{+k}$ containing c. Using Lemma 6.3, we infer that $V(\mathbf{H})$ is a subuniverse of the algebra $(V(\mathbf{G}^{+k}), w)$, and thus $\mathbf{H}$ admits a weak near unanimity polymorphism. The algebraic length of $\mathbf{H}$ (again by Corollary 5.7) is one, and therefore by Theorem 7.2 it follows that there is a loop in $\mathbf{G}^{+k}$. This trivially implies a closed walk of length k in $\mathbf{G}$, and the theorem is proved using the folklore proposition from section 4.4. □

The remaining part of this section is devoted to the proof of Theorem 7.2. We start by choosing a digraph $\mathbf{G} = (V, E)$ to be a minimal (with respect to the number of vertices) counterexample to Theorem 7.2. We fix a weak near unanimity polymorphism $w(x_0, \dots, x_{h-1})$ of this digraph and associate with it the algebra $\mathbf{A} = (V, w)$. The proof will proceed by a number of claims.

CLAIM 7.3. *The digraph $\mathbf{G}$ can be chosen to contain a closed walk of length 2.*

Proof. Using Lemma 5.5, we find a minimal k such that a closed walk of length 2^k is contained in $\mathbf{G}$. Consider the path power $\mathbf{G}^{+2^{k-1}}$. It contains a closed walk of length 2 and admits a weak near unanimity polymorphism. Moreover, since k was chosen to be minimal and $\mathbf{G}$ did not contain a loop, the path power $\mathbf{G}^{+2^{k-1}}$ does not contain a loop either. By Corollary 5.6 the path power is strongly connected, and by Corollary 5.2 it has algebraic length equal to one. Thus, the digraph $\mathbf{G}^{+2^{k-1}}$ is also a counterexample to Theorem 7.2 (with the same number of vertices as $\mathbf{G}$), and therefore we can use it as a substitute for $\mathbf{G}$. □

From this point on we assume that $\mathbf{G}$ contains a closed walk of length 2 (an undirected edge). The next claim allows us to choose and fix an undirected edge with special properties.

CLAIM 7.4. *There are vertices $a, b \in V$ forming an undirected edge in $\mathbf{G}$ and a binary term t of $\mathbf{A}$ such that $a = t(w(\overline{a}, b), w(\overline{b}, a))$.*

Proof. Let $M \subseteq V$ be a minimal (under inclusion) subuniverse of $\mathbf{A}$ containing an undirected edge, and let $a, b \in M$ be vertices in such an edge. Since vertices $w(\overline{a}, b)$, $w(\overline{b}, a) \in M$ form an undirected edge in $\mathbf{G}$, the set $\{w(\overline{a}, b), w(\overline{b}, a)\}$ generates, in the algebraic sense, the set M (by the minimality of M). Since every vertex in a subuniverse is a result of an application of some term function to the generators of the subuniverse, there exists a term t such that $t(w(\overline{a}, b), w(\overline{b}, a)) = a$. □

In the following claims we fix vertices a, b and a term $t(x, y)$ such that $a \to b \to a$ and $a = t(w(\overline{a}, b), w(\overline{b}, a))$ (provided by the previous claim). Note that, by the definition of the operation $w(x_0, \dots, x_{h-1})$, for any numbers $i, j < h$, we obtain $a = t(w^{(i)}(\overline{a}, b), w^{(j)}(\overline{b}, a))$.

Using Lemma 5.5, we find and fix a minimal number n such that $a^{+(n+1)} = V$. We put $W = a^{+n}$ and $F = (W \times W) \cap E$ so that $\mathbf{H} = (W, F)$ is an induced subgraph of the digraph $\mathbf{G}$. Using Corollary 6.2, we infer that W is a subuniverse of $\mathbf{A}$ and thus $\mathbf{H}$ admits a weak near unanimity polymorphism. In the following claims we will show that the algebraic length of some strong component of $\mathbf{H}$ is one, which will contradict the minimality of $\mathbf{G}$.

CLAIM 7.5. *For any vertex in W there exists a closed walk in $\mathbf{H}$ and a walk (also in $\mathbf{H}$) connecting the closed walk to this vertex.*

Proof. Let d_0 denote an arbitrary vertex of W. Since $a^{+(n+1)} = W^{+1} = V$ there is $d_1 \in W$ such that $d_1 \to d_0$. Similarly, there exists $d_2 \in W$ such that $d_2 \to d_1$. By repeating this procedure, we get both statements of the claim. □

The next claim will allow us to fix some more vertices necessary for further construction.

CLAIM 7.6. *There exist vertices $c, c' \in W$ and a number k such that*

1. *$c' \to a$,*
2. *$c \xrightarrow{k} c$ in $\mathbf{H}$, and*
3. *$c \xrightarrow{k-n-1} c'$ in $\mathbf{H}$.*

Proof. Since $W^{+1} = V$ there exists $c' \in W$ such that $c' \to a$. Let l be the length of a closed walk provided by Claim 7.5 for $c' \in W$. For a sufficiently large multiple k of l there is a walk in $\mathbf{H}$ of length $k - n - 1$ from some vertex of the closed walk to c'; we call this vertex c. This finishes the proof. □

From this point on we fix vertices c and c' in W and a number k to satisfy the conditions of the last claim. The following claims focus on uncovering the structure of the strong component containing c in $\mathbf{H}$.

CLAIM 7.7. *For any $m \leq n$ either $a^{+m} \subseteq a^{+n}$ or $a^{+m} \subseteq b^{+n}$.*

Proof. Since a is in a closed walk of length 2, we obviously have $a^{+n} \supseteq a^{+(n-2)} \supseteq a^{+(n-4)} \cdots$, which proves the claim for even m's. If, on the other hand, m is odd, we have $b^{+n} \supseteq a^{+(n-1)} \supseteq a^{+(n-3)} \cdots$, completing the proof. □

The next two claims are of major importance for the proof of Theorem 7.2. They are used to show that the algebraic length of the strong component of $\mathbf{H}$ containing c is one.

CLAIM 7.8. *For any $m \leq n$ and for any $0 \leq i, j < h$ the following inclusion holds:*

$$t(w^{(i)}(\overline{a^{+n}}, a^{+m}), w^{(j)}(\overline{a^{+m}}, a^{+n})) \subseteq a^{+n}.$$

Proof. Note that $a = t(w^{(i)}(\overline{a}, b), w^{(j)}(\overline{b}, a))$ and therefore, for any choice of arguments of the term reachable by walks of length n from corresponding arguments of $t(w^{(i)}(\overline{a}, b), w^{(j)}(\overline{b}, a))$, the result is reachable by a walk of the same length from a, i.e.,

$$a^{+n} \supseteq t(w^{(i)}(\overline{a^{+n}}, b^{+n}), w^{(j)}(\overline{b^{+n}}, a^{+n})).$$

By the same token, using $a = t(w^{(i)}(\overline{a}, a), w^{(j)}(\overline{a}, a))$ provided by the idempotency of the terms, we obtain

$$a^{+n} \supseteq t(w^{(i)}(\overline{a^{+n}}, a^{+n}), w^{(j)}(\overline{a^{+n}}, a^{+n})).$$

Now the claim follows directly from Claim 7.7. □

The following technical claim will allow us to find walks in the strong component of $\mathbf{H}$ containing c.

CLAIM 7.9. *The following implication holds in* $\mathbf{H}$ (*i.e., all the walks and vertices lie inside* $\mathbf{H}$). *For any numbers* $0 \leq i, j < h$ *and all* $e, e', f \in W$ *and* $\overrightarrow{d}, \overrightarrow{d'}, \overrightarrow{g} \in \overline{W}$,

$$\text{if} \quad \begin{matrix} e \\ \downarrow k \\ e' \end{matrix} \quad \text{and} \quad \begin{matrix} d_l \\ \downarrow k \\ d'_l \end{matrix} \quad \text{for all } l, \quad \text{then} \quad \begin{matrix} t(w^{(i)}(\overrightarrow{d}, c), w^{(j)}(\overline{c}, e)) \\ \downarrow k \\ t(w^{(i)}(\overrightarrow{d'}, f), w^{(j)}(\overrightarrow{g}, e')). \end{matrix}$$

Proof. Note that, by looking at the tuples of vertices pointwise, we can find the following walks in $\mathbf{G}$:

$$\begin{matrix} \overrightarrow{d} & & c & & \overline{c} & & e \\ \Big\downarrow & & \downarrow k-n-1 & & \downarrow k-n-1 & & \Big\downarrow \\ & & c' & & \overline{c'} & & \\ k & & \downarrow & & \downarrow & & k \\ & & a & & \overline{a} & & \\ & & \downarrow n & & \downarrow n & & \\ \overrightarrow{d'} & & f & & \overrightarrow{g} & & e' \end{matrix}$$

where the walks from c to c' are provided by Claim 7.6 and lie entirely in $\mathbf{H}$. Applying the appropriate term to the consecutive vertices of the walks (rows in the diagram above), we obtain a walk of length k connecting $t(w^{(i)}(\overrightarrow{d}, c), w^{(j)}(\overline{c}, e))$ to $t(w^{(i)}(\overrightarrow{d'}, f), w^{(j)}(\overrightarrow{g}, e'))$. It remains to prove that all the vertices of this walk are in W. The first $k-n-1$ vertices of the walks are in W, since W is a subuniverse and they are results of an application of a term to vertices of the subuniverse. For $m \geq 0$, the $(k-n+m)$th vertex of the walk is a member of $t(w^{(i)}(\overline{a^{+n}}, a^{+m}), w^{(j)}(\overline{a^{+m}}, a^{+n}))$ and thus in W by Claim 7.8. □

We now construct a closed walk in $\mathbf{H}$, that contains c, of length coprime to k.

CLAIM 7.10. *There exists a closed walk* $c \xrightarrow{(h+1)k-1} c$ *in digraph* $\mathbf{H}$.

Proof. In the proof of this claim we use only vertices and walks that lie inside $\mathbf{H}$. Fix $d \in W$ (provided by Claim 7.6) such that $c \to d \xrightarrow{k-1} c$ in $\mathbf{H}$. By repeatedly applying Claim 7.9 we obtain

$$\begin{matrix} t(w(c,\dots,c,c,c), w(c,c\dots,c)) & & \\ \downarrow k & & \\ t(w(c,\dots,c,c,d), w(d,c,\dots,c)) & = & t(w^{(1)}(c,\dots,c,d,c), w^{(1)}(c,\dots,c,d)) \\ & & \downarrow k \\ t(w^{(2)}(c,\dots,d,d,c), w^{(1)}(c,\dots,c,d)) & = & t(w^{(1)}(c,\dots,c,d,d), w^{(1)}(c,\dots,c,d)) \\ \vdots & & \\ & = & t(w^{(h-1)}(d,\dots,d,d,c), w^{(1)}(c,\dots,c,d)) \\ & & \downarrow k \\ & & t(w^{(h-1)}(d,\dots,d,d,d), w^{(1)}(d,\dots,d,d)) \end{matrix}$$

and since the algebra is idempotent, the starting point of this walk is c and the ending point is d. Thus $c \xrightarrow{hk} d$ (for h the arity of the operation $w(x_0,\dots,x_{h-1})$), which immediately gives us the claim. □

By Claims 7.6 and 7.10, the strong component of $\mathbf{H}$ containing c has GCD of the lengths of its closed walks equal to one, and thus, by Lemma 6.3, its vertex set forms a subuniverse of the algebra $\mathbf{A}$. As a digraph it admits a weak near unanimity polymorphism. By Corollary 5.7 it has algebraic length one, and (as an induced subgraph of $\mathbf{G}$) it has no loops. Since $\mathbf{H}$ was chosen to be strictly smaller than $\mathbf{G}$ we obtain a contradiction with the minimality of $\mathbf{G}$, and the proof of Theorem 7.2 is complete.

8. The general case. In this section we prove Theorem 3.1 in its full generality. Nevertheless the majority of this section is devoted to the proof of the following result.

THEOREM 8.1. *If a digraph with no sources or sinks has algebraic length one and admits a weak near unanimity polymorphism, then it contains a loop.*

Using the above result, we prove the core theorem of the paper, Theorem 3.1.

Proof of Theorem 3.1. Let $\mathbf{G}$ be a digraph with no sources or sinks which admits a weak near unanimity polymorphism. Let n be the algebraic length of some component of $\mathbf{G}$. The path power $\mathbf{G}^{+n}$ admits a weak near unanimity polymorphism, has no sources or sinks, and, by Lemma 5.1, has algebraic length equal to one. Thus, Theorem 8.1 applied to $\mathbf{G}^{+n}$ provides a loop in the path power and therefore a closed walk of length n in $\mathbf{G}$.

Let n be minimal, under divisibility, in the set of algebraic lengths of components of $\mathbf{G}$. Since the algebraic length of a component divides (by Proposition 5.4) the length of any closed walk in it, every closed walk of length n (for such a minimal n) forms a subgraph which is a cycle. Moreover, by the same reasoning, cycles obtained for two different minimal n's cannot belong to the same component. Thus each component of $\mathbf{G}$ maps homomorphically to an n-cycle (for any minimal n dividing the algebraic length of this component), and it is not difficult to see that these homomorphisms can be chosen so that their union is a retraction. This proves the theorem. ∎

Therefore the only missing piece of the proof to the conjecture of Bang-Jensen and Hell is Theorem 8.1. We prove this result by way of contradiction. Suppose that $\mathbf{G} = (V, E)$ is a minimal (with respect to the number of vertices) counterexample to Theorem 8.1, and let $\mathbf{A} = (V, w(x_0, \dots, x_{h-1}))$ be the algebra associated with $\mathbf{G}$, in the sense of section 6, for some weak near unanimity polymorphism $w(x_0, \dots, x_{h-1})$.

The first part of the proof is dedicated to finding a particular counterexample satisfying more restrictive conditions than $\mathbf{G}$. To do so we need to define a special family of digraphs called *tambourines*. The n-tambourine is the digraph $(\{d_0, \dots, d_{n-1}, u_0, \dots, u_{n-1}\}, F_n)$ such that

$$F_n = \bigcup_i \{(d_i, d_{i+1}), (d_i, u_i), (d_i, u_{i+1}), (u_i, u_{i+1})\},$$

where the addition on the indices is computed modulo n. The 12-tambourine can be found in Figure 1. We begin the proof of the theorem with the following claim.

CLAIM 8.2. *We can choose a digraph* $\mathbf{G}$ *and a number* n *such that*

1. *the* n*-tambourine maps homomorphically to* $\mathbf{G}$*,*
2. *every vertex of* $\mathbf{G}$ *is in a closed walk of length* n*, and*
3. $\mathbf{G}^{+(mn+1)} = \mathbf{G}$ *for any number* m*.*

To prove this claim, we begin with an easy subclaim and work towards replacing $\mathbf{G}$ with a particular path power of $\mathbf{G}$ which satisfies the additional conditions. Note that, for any pattern α, the path power G^α admits $w(x_0, \dots, x_{h-1})$ as a polymorphism and has no sources or sinks. If such a path power has algebraic length one and does not contain a loop, then it can be taken as a substitute for $\mathbf{G}$.

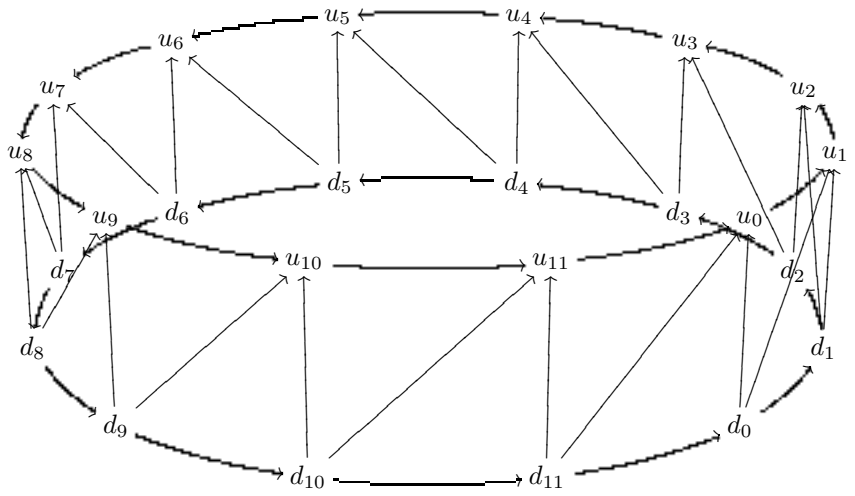


FIG. 1. *The* 12*-tambourine.*

SUBCLAIM 8.2.1. *The digraph* $\mathbf{G}$ *contains vertices* d *and* u *such that* $d \xrightarrow{|V|,|V|+1} u$.

Proof. Let α be the pattern

$$\underbrace{\rightarrow \ \cdots \rightarrow}_{|V|+1} \ \underbrace{\leftarrow \ \cdots \leftarrow}_{|V|}.$$

Using the fact that $al(\alpha) = 1$ and that $\mathbf{G}$ has no sources or sinks, it follows that $E(\mathbf{G}) \subseteq E(\mathbf{G}^\alpha)$. Moreover, let a, b be vertices in $\mathbf{G}$ such that b is contained in a closed walk and $a \xrightarrow{k} b$ for some k. Then $a \xrightarrow{k'} b$ for some $k' \leq |V|$, and choosing b' (from the closed walk containing b) such that $b' \xrightarrow{k'+1} b$, we obtain

$$b' \xrightarrow{k'+1} b \xrightarrow{(|V|+1)-(k'+1)} c \xleftarrow{(|V|+1)-(k'+1)} b \xleftarrow{k'} a \quad \text{for some } c.$$

Thus $b' \xrightarrow{\alpha} a$, and this implies that every component of $\mathbf{G}$ becomes a strong component of $\mathbf{G}^\alpha$.

Let $\mathbf{H} = (W, F)$ be a component of $\mathbf{G}$ with a closed walk realizing a pattern of algebraic length one. Then, for an appropriate F', containing F, the digraph $\mathbf{H}' = (W, F')$ is a strong component of $\mathbf{G}^\alpha$. The digraph $\mathbf{H}'$ contains $\mathbf{H}$ as a subgraph, and therefore its algebraic length is one. The path power $\mathbf{G}^\alpha$ admits $w(x_0, \ldots, x_{h-1})$ as a polymorphism, and thus, by Lemma 6.3, the digraph $\mathbf{H}'$ admits an appropriate restriction of $w(x_0, \ldots, x_{h-1})$. Theorem 7.2 provides a loop in $\mathbf{H}'$, which in turn implies the existence of vertices $d, u \in W$ such that $d \xrightarrow{|V|,|V|+1} u$ in $\mathbf{G}$. ∎

Proof of Claim 8.2. We fix $n = |V|!$ and argue that, for some k, the path power $\mathbf{G}_k = \mathbf{G}^{+(kn+1)}$ satisfies the assertions of the claim and therefore can be taken as a substitute for $\mathbf{G}$. Note that, for any number k, the digraph $\mathbf{G}_k$ admits $w(x_0, \ldots, x_{h-1})$ as a polymorphism, has no sources or sinks, and, by Corollary 5.2, has algebraic length one.

We first prove that, for all k, the digraph $\mathbf{G}_k$ does not contain a loop. If $\mathbf{G}_k$ does contain a loop, then there exists a closed walk of length $kn+1$ in some strong component of $\mathbf{G}$. In the same strong component in $\mathbf{G}$ there exists a closed walk of length smaller than n and thus coprime to $kn+1$; therefore the GCD of the lengths of closed walks in this strong component is one, and, using Corollary 5.7, Lemma 6.3, and Theorem 7.2, we obtain a loop in this strong component and therefore also in $\mathbf{G}$, a contradiction. Thus, to prove the claim, it remains to verify the additional required properties.

We now show that, for the fixed number n, the n-tambourine maps homomorphically to $\mathbf{G}_k$ for $k \geq 4$. Let d, u be vertices of $\mathbf{G}$ provided by Subclaim 8.2.1. Since $\mathbf{G}$ has no sources or sinks, we can find vertices d', u', each contained in a closed walk, such that d' is connected by a walk to d and u is connected by a walk to u'. By following the closed walks containing d' and u' multiple times, we get d'_0, u'_0, each contained in a closed walk, such that $d'_0 \xrightarrow{3n,3n+1} u'_0$. Moreover, again following the closed walks multiple times, we obtain

$$d'_0 \xrightarrow{n} d'_0 \xrightarrow{3n,3n+1} u'_0 \xrightarrow{n} u'_0.$$

Let d'_i denote the ith vertex of the closed walk $d'_0 \xrightarrow{n} d'_0$ and, similarly, u'_i the ith vertex of the closed walk $u'_0 \xrightarrow{n} u'_0$. Then, for any number $k \geq 4$ and any $i < n$, we have $d'_i \xrightarrow{kn+1} u'_i$ and $d'_i \xrightarrow{kn+1} u'_{(i+1) \bmod n}$. On the other hand, $d'_i \xrightarrow{kn+1} d'_{(i+1) \bmod n}$ and $u'_i \xrightarrow{kn+1} u'_{(i+1) \bmod n}$. Thus, for any $k \geq 4$, the map $d_i \mapsto d'_i$, $u_i \mapsto u'_i$ is a homomorphism from the n-tambourine in the path power $\mathbf{G}_k$.

To prove the second assertion of the claim we need to show that if $k \geq 4$, then any vertex of $\mathbf{G}_k$ is in a closed walk of length n. We fix such a number k and let $W \subset V$ be the subuniverse of $\mathbf{A}$ generated by $\{d'_0, \dots, d'_{n-1}, u'_0, \dots, u'_{n-1}\}$. Let $\mathbf{G}'_k$ be the subgraph induced by $\mathbf{G}_k$ on W. The digraph $\mathbf{G}'_k$ obviously admits a restriction of $w(x_0, \dots, x_{h-1})$ and (since the n-tambourine maps homomorphically to it) has algebraic length one. Choose an arbitrary $a \in W$. Then, by the definition of W, we have a term $t(x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1})$ such that $a = t(d'_0, \dots, d'_{n-1}, u'_0, \dots, u'_{n-1})$. Therefore,

$$\left.\begin{array}{c} t(d'_0, \dots, d'_{n-2}, d'_{n-1}, u'_0, \dots, u'_{n-2}, u'_{n-1}) \\ \downarrow \\ t(d'_1, \dots, d'_{n-1}, d'_0, u'_1, \dots, u'_{n-1}, u'_0) \\ \vdots \\ t(d'_{n-1}, \dots, d'_{n-3}, d'_{n-2}, u'_{n-1}, \dots, u'_{n-3}, u'_{n-2}) \\ \downarrow \\ t(d'_0, \dots, d'_{n-2}, d'_{n-1}, u'_0, \dots, u'_{n-2}, u'_{n-1}) \end{array}\right\} n,$$

and thus a is in a closed walk of length n. This proves that $\mathbf{G}'_k$ has no sources and no sinks, and since it cannot be a counterexample smaller than $\mathbf{G}$, we infer that $W = V$. Therefore the second assertion holds for all the digraphs $\mathbf{G}_k$ with $k \geq 4$.

In the digraph $\mathbf{G}_4$ every vertex is in a closed walk of length n, and therefore $E(\mathbf{G}_4{}^{+(nm+1)}) \subseteq E(\mathbf{G}_4{}^{+(n(m+1)+1)})$ for any number m. Thus, there is a number l such that for any $m \geq l$ we have $\mathbf{G}_4{}^{+(nm+1)} = \mathbf{G}_4{}^{+(nl+1)}$. Take

$$\mathbf{G}' = \mathbf{G}_4{}^{+(nl+1)} = \mathbf{G}^{+(4n+1)(nl+1)} = \mathbf{G}_{(4nl+l+4)n+1}$$

and note that, according to the previous paragraphs of this proof, such a digraph satisfies all but the last assertion of the claim. Let m be arbitrary. Then $(\mathbf{G}')^{+(mn+1)} = \mathbf{G}_4{}^{+((mnl+l+m)n+1)} = \mathbf{G}_4{}^{+(nl+1)} = \mathbf{G}'$, and thus $\mathbf{G}'$ can be taken to substitute for $\mathbf{G}$ and the claim is proved. □

From this point on we substitute $\mathbf{G}$ with a digraph provided by the previous claim and fix it together with the number n. For ease of notation we denote the number modulo n using brackets (e.g., $[n+1] = 1$). We already know that the n-tambourine maps homomorphically to $\mathbf{G}$, but we must choose such a homomorphism carefully.

CLAIM 8.3. *The n-tambourine can be mapped homomorphically to $\mathbf{G}$ in such a way that, for some term $t(x_0, \dots, x_{n-1})$ of algebra $\mathbf{A}$,*

$$d'_i = t^{(i)}(w(\overline{d'_0}, d'_1), w(\overline{d'_1}, d'_2), \dots, w(\overline{d'_{n-1}}, d'_0)) \quad \text{for all } i < n,$$

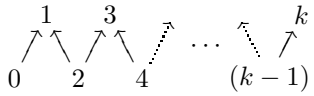
where d'_i is the image of d_i.

Proof. Let $d_i \mapsto d'_i, u_i \mapsto u'_i$ be a homomorphism from the n-tambourine to $\mathbf{G}$. Then, for any i, we have

$$\begin{array}{ccccc} w(\overline{u'_i}, u'_{[i+1]}) & \longrightarrow & w(\overline{u'_{[i+1]}}, u'_{[i+2]}) & \longrightarrow & \cdots \\ \uparrow & \nearrow & \uparrow & \cdots & \\ w(\overline{d'_i}, d'_{[i+1]}) & \longrightarrow & w(\overline{d'_{[i+1]}}, d'_{[i+2]}) & \longrightarrow & \cdots \end{array}$$

and thus $d_i \mapsto w(\overline{d'_i}, d'_{[i+1]}), u_i \mapsto w(\overline{u'_i}, u'_{[i+1]})$ is also a homomorphism from the n-tambourine to $\mathbf{G}$. By repeating this procedure, we obtain an infinite sequence of homomorphisms from the n-tambourine to $\mathbf{G}$, and thus some homomorphism has to appear twice in this sequence. This homomorphism satisfies the claim, since the term $t(x_0, \dots, x_{n-1})$ can be easily obtained as a composition of the polymorphism $w(x_0, \dots, x_{h-1})$ used in the construction of the sequence. □

In the remaining part of the proof we fix vertices $d'_0, \dots, d'_{n-1}, u'_0, \dots, u'_{n-1}$ provided by the previous claim and a term $t(x_0, \dots, x_{n-1})$ associated with them. Let φ_k be the pattern $0 \xrightarrow{\varphi_k} k$



with exactly k edges. (The last edge of the pattern is pointing forward for odd k, as in the above picture, and backward for even k.)

CLAIM 8.4. *The neighborhood $(d'_0)^{\varphi_n}$ contains all vertices of $\mathbf{G}$.*

Proof. Note that, in the n-tambourine, we have

$$(d_0)^{\varphi_n} = \{d_0, \dots, d_{n-1}, u_0, \dots, u_{n-1}\},$$

and thus in the digraph $\mathbf{G}$ we have

$$(d'_0)^{\varphi_n} \supseteq \{d'_0, \dots, d'_{n-1}, u'_0, \dots, u'_{n-1}\}.$$

Let $\mathbf{G}'$ denote the subgraph of $\mathbf{G}$ induced on the set $(d'_0)^{\varphi_n}$. Then, by Corollary 6.2, $\mathbf{G}'$ admits a restriction of $w(x_0, \dots, x_{h-1})$ as a polymorphism and has algebraic length one. Further restricting the digraph $\mathbf{G}'$, denote the largest induced subgraph of $\mathbf{G}'$ without sources or sinks by $\mathbf{G}''$. By Lemma 6.4 $\mathbf{G}''$ admits a weak near unanimity

polymorphism. Moreover, the vertices $\{d'_0, \dots, d'_{n-1}, u'_0, \dots, u'_{n-1}\}$ are among the vertices of $\mathbf{G}''$. Thus $\mathbf{G}''$ is a counterexample to Theorem 8.1 and therefore has to be equal to $\mathbf{G}$. This proves the claim. □

We choose (and fix) k to be a minimal number such that $(d'_0)^{\varphi_{k+1}} = V$. Define $W_i = (d'_i)^{\varphi_k}$, for each $i < n$. We set

$$W = \bigcap_{i<n} W_i,$$

and since W is an intersection of subuniverses of $\mathbf{A}$, by Corollary 6.2, it is itself a subuniverse of $\mathbf{A}$. We denote by $\mathbf{H}$ the subgraph of $\mathbf{G}$ induced by W and prove that $\mathbf{H}$ is a counterexample to Theorem 8.1, contradicting the minimality of $\mathbf{G}$.

The most involved part of the proof deals with constructing a closed realization of a pattern with the algebraic length one in $\mathbf{H}$. Two following claims introduce tools for "projecting" certain walks from $\mathbf{G}$ to $\mathbf{H}$.

CLAIM 8.5. *There exists a term $s(x_0, \dots, x_{p-1})$ of algebra $\mathbf{A}$ such that for every coordinate $q < p$ there exists i such that*

$$s^{(q)}(W_l, W, \dots, W) \subseteq W_{[i-l]} \cap W_{[i-l+1]} \quad \textit{for any } l < n.$$

Proof. Let $p = hn$ and let $s(x_0, \dots, x_{p-1})$ be defined by

$$t\left(w(x_0, \dots, x_{h-1}), w(x_h, \dots, x_{2h-1}), \dots, w(x_{(n-1)h}, \dots, x_{hn-1})\right).$$

For all $q < p$, let i be maximal such that $q = ih + q''$ for some nonnegative q''. Then, for all $l < n$

$$\begin{aligned} s^{(q)}(W_l, \overline{W}) &\subseteq t^{(i)}\left(w^{(q'')}(W_l, \overline{W}), w(\overline{W}), \dots, w(\overline{W})\right) \\ &\subseteq t^{(i)}\left(w^{(q'')}(\overline{W_l}, W_{[l+1]}), w(\overline{W_{[l+1]}}, W_{[l+2]}), \dots, w(\overline{W_{[l+n-1]}}, W_l)\right) \\ &= t^{([i-l])}\left(w(\overline{W_0}, W_1), \dots, w^{(q'')}(\overline{W_l}, W_{[l+1]}), \dots, w(\overline{W_{n-1}}, W_0)\right) \\ &\subseteq W_{[i-l]}, \end{aligned}$$

where the last inclusion follows from Claim 8.3 and the fact that

$$\begin{aligned} d'_{[i-l]} &= t^{([i-l])}(w(\overline{d'_0}, d'_1), \dots, w(\overline{d'_l}, d'_{[l+1]}), \dots, w(\overline{d'_{n-1}}, d'_0)) \\ &= t^{([i-l])}(w(\overline{d'_0}, d'_1), \dots, w^{(q'')}(\overline{d'_l}, d'_{[l+1]}), \dots, w(\overline{d'_{n-1}}, d'_0)). \end{aligned}$$

Similar reasoning shows that

$$\begin{aligned} s^{(q)}(W_l, \overline{W}) &\subseteq t^{(i)}\left(w^{(q'')}(W_l, \overline{W}), w(\overline{W}), \dots, w(\overline{W})\right) \\ &\subseteq t^{(i)}\left(w^{(q'')}(W_l, \overline{W_{[l-1]}}), w(W_{[l+1]}, \overline{W_l}), \dots, w(W_{[l+n-1]}, \overline{W_{[l+n-2]}})\right) \\ &= t^{[i-l+1]}\left(w(W_1, \overline{W_0}), \dots, w^{(q'')}(W_l, \overline{W_{[l-1]}}), \dots, w(W_0, \overline{W_{n-1}})\right) \\ &\subseteq W_{[i-l+1]}, \end{aligned}$$

and the proof is finished. □

Further, using the term constructed in the last claim, we can construct a term satisfying stronger conditions.

CLAIM 8.6. *There exists a term $r(x_0, \dots, x_{m-1})$ of algebra $\mathbf{A}$ such that for every coordinate $q < m$*

$$r^{(q)}\left(\bigcup_{l<n} W_l, W, \dots, W\right) \subseteq W.$$

Proof. Let $s(x_0, \dots, x_{p-1})$ be the p-ary term provided by the previous claim. Note that the term

$$s_2(x_0, x_1, \dots, x_{p^2-1}) = s(s(x_0, \dots, x_{p-1}), \dots, s(x_{p^2-p}, \dots, x_{p^2-1}))$$

has the property that for every coordinate $q < p^2 - 1$ there exists an i such that

$$s_2^{(q)}(W_l, \overline{W}) \subseteq W_{[i-l]} \cap W_{[i-l+1]} \cap W_{[i-l+2]}.$$

To prove a more general statement we recursively define a sequence of terms

- $s_1(x_0, \dots, x_{p-1}) = s(x_0, \dots, x_{p-1})$ and
- $s_{j+1}(x_0, \dots, x_{p^j-1}) = s(s_j(x_0, \dots, x_{p^{j-1}-1}), \dots, s_j(x_{(p-1)p^{j-1}}, \dots, x_{p^j-1}))$.

We claim that for any j, any $q < p^j$, and any $l < n$ there is an i such that

$$s_j^{(q)}\big(W_l, W, \dots, W\big) \subseteq W_{[i-l]} \cap \dots \cap W_{[i-l+j]}.$$

We prove this fact by induction on j. The first step of the induction holds via Claim 8.5. Assume that the fact holds for j; then for any l (setting q' to be the result of integer division of q by p^{j-1}, and q'' the remainder of this division) there exist i and i' such that

$$\begin{aligned} s_{j+1}^{(q)}(W_l, \overline{W}) &\subseteq s^{(q')}\big(s_j^{(q'')}(W_l, \overline{W}), s_j(\overline{W}), \dots, s_j(\overline{W})\big) \\ &\subseteq s^{(q')}\big(W_{[i-l]} \cap \dots \cap W_{[i-l+j]}, \overline{W}\big) \\ &\subseteq W_{[i'+i-l]} \cap \dots \cap W_{[i'+i-l+(j+1)]}, \end{aligned}$$

where the second inclusion follows from the induction step and the last one from Claim 8.5. Setting $r(x_0, \dots, x_{m-1})$ equal to $s_{n-1}(x_0, \dots, x_{p^n-1})$ proves the claim. □

From this point on we fix a term $r(x_0, \dots, x_{m-1})$ (of arity m) provided by the previous claim. To prove additional properties of the set W (e.g., the fact that it is not empty) we require the following easy claim.

CLAIM 8.7. *Let α be a pattern, and let $a_0 \to a_1$ and $b_0 \to b_1$ be edges that belong to closed walks. If $a_0 \xrightarrow{\alpha} b_0$, then $a_1 \xrightarrow{\alpha} b_1$.*

Proof. We prove the claim by induction with respect to the number of edges in α. Let the vertices a_0, a_1, b_0, b_1 be as in the statement of the claim. Assume that $a_0 \to b_0$. If i is the length of the closed walk containing the edge $a_0 \to a_1$, then, following this walk almost n times, $a_1 \xrightarrow{in-1} a_0 \to b_0 \to b_1$ and, by Claim 8.2, $a_1 \to b_1$. The same reasoning can be applied to the case of $a_0 \leftarrow b_0$, and the first step of the induction is proved.

For a pattern α consisting of more than one edge we can assume, without loss of generality, that the last edge is going forward. Then $a_0 \xrightarrow{\alpha'} a_0' \to b_0$ for some vertex a_0' (where α' is the pattern obtained by removing the last edge of α). By Claim 8.2, it follows that a_0' is in a closed walk of length n, and therefore $a_0' \to a_1' \xrightarrow{n-1} a_0'$ for some a_1'. By the induction hypothesis, $a_1 \xrightarrow{\alpha'} a_1'$ and, by the first step of the induction, $a_1' \to b_1$, which proves the claim. □

We recall the definition of the top and bottom components of the graph from subsection 4.3 and prove some basic properties of W.

CLAIM 8.8. *The digraph* $\mathbf{H}$ *has no sources and no sinks and*

1. *if* k *is even, then every bottom component is contained in* W*, and*
2. *if* k *is odd, then every top component is contained in* W*.*

Proof. First we show that, for any vertices a, b such that $a \xrightarrow{i} b \xrightarrow{j} a$ in $\mathbf{G}$ for some i, j,

$$\text{if } a \in W_l, \quad \text{then } b \in W_{[l+i]}.$$

To see this note that if $d'_l \xrightarrow{\varphi_k} a$ and $a \to b \xrightarrow{j} a$, then, using Claim 8.7 and the edge $d'_l \to d'_{[l+1]}$, we infer that $d'_{[l+1]} \xrightarrow{\varphi_k} b$. The same procedure repeated i-many times provides the result for arbitrary i.

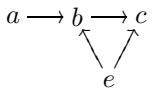
Let $a \in W$ be arbitrary and b be such that $a \xrightarrow{i} b \xrightarrow{j} a$ for some numbers i, j. Since $a \in W$ it follows, using the note above, that $b \in \bigcap_{l<n} W_{[l+i]} = W$, and this implies that W is a union of strong components. Since, by Claim 8.2, every vertex in $\mathbf{G}$ belongs to a closed walk of length n, the digraph $\mathbf{H}$ has no sources or sinks.

Let k be even and let a be a member of a bottom component. Since every vertex of the graph, by Claim 8.2, belongs to a closed walk, there exists b in the bottom component containing a such that $a \to b$. Since $(d'_0)^{\varphi_{k+1}} = V$, we have $d'_0 \xrightarrow{\varphi_{k-1}} c \leftarrow a' \to b$ for some a' and c. The vertex a is in a bottom component, and therefore a' must be a member of the same bottom component. This implies that $a' \to b \xrightarrow{i} a'$, for some i, and following the closed walk containing b almost n times, $a \to b \xrightarrow{n(i+1)-1} a' \to c$. Thus, by Claim 8.2, we have $a \to c$ and $a \in W_0$. Therefore every bottom component is contained in W_0. To see that every a from a bottom component is contained in an arbitrary W_l we find a b satisfying $a \xrightarrow{l} b \xrightarrow{i} a$ for some i and apply the note from the beginning of the proof of the claim. The claim is proved for even k's, and the same reasoning provides a proof for odd k and top components. ∎

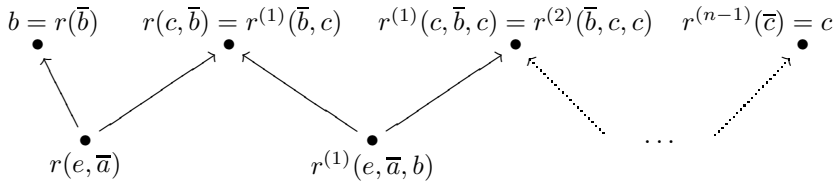
Now we are ready to prove the final claim of this section.

CLAIM 8.9. *The algebraic length of* $\mathbf{H}$ *is one.*

Proof. In the case where k is odd, we want to find $a, b, c \in W$ and $e \in W_0$ such that

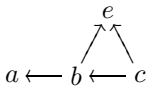


To find such vertices we set $e = d'_1$ and find, using Claim 8.8, $b \in W$ from a top component such that $u'_{[2]} \xrightarrow{in-1} b$ for some i. There exist a and c in the same component (and thus in W by Claim 8.8) such that $a \to b \to c$. Since $d'_1 \xrightarrow{1,2} u'_{[2]}$, we have $e \xrightarrow{in+1} b$ and $e \xrightarrow{in+1} c$, and therefore, by Claim 8.2, the vertices a, b, c, and e satisfy the required properties. Then, using the term $r(x_0, \dots, x_{m-1})$, we produce the following oriented walk:



By Claim 8.6, all the vertices of this walk belong to W. Thus we have constructed an oriented walk in $\mathbf{H}$ realizing a pattern of algebraic length zero connecting b to c. Since $b \to c$ we immediately obtain that the algebraic length of $\mathbf{H}$ is one.

In the case where k is even, we similarly find $a, b, c \in W$ and $e \in W_0$ (using u'_1 for e) such that



The construction of a closed oriented walk realizing a pattern of algebraic length one is the same as it is for odd k, with the exception that the direction of the edges is reversed. □

Thus $\mathbf{H}$ is a digraph without sources or sinks (by Claim 8.8), admitting a weak near unanimity polymorphism and, by the last claim, having algebraic length equal to one. Since, by the definition of W, the number of vertices in $\mathbf{H}$ is strictly smaller than the number of vertices in $\mathbf{G}$, we obtain a contradiction with the minimality of $\mathbf{G}$, and Theorem 8.1 is proved.

REFERENCES

[All94] J. Allen, *Natural Language Understanding*, 2nd ed., Benjamin Cummings, San Francisco, CA, 1994.

[ANP05] D. Achlioptas, A. Naor, and Y. Peres, *Rigorous location of phase transitions in hard optimization problems*, Nature, 435 (2005), pp. 759–764.

[BIM+06] J. Berman, P. Idziak, P. Marković, R. McKenzie, M. Valeriote, and R. Willard, *Varieties with few subalgebras of powers*, Trans. Amer. Math. Soc., to appear.

[BJH90] J. Bang-Jensen and P. Hell, *The effect of two cycles on the complexity of colourings by directed graphs*, Discrete Appl. Math., 26 (1990), pp. 1–23.

[BJHM95] J. Bang-Jensen, P. Hell, and G. MacGillivray, *Hereditarily hard H-colouring problems*, Discrete Math., 138 (1995), pp. 75–92.

[BJK05] A. Bulatov, P. Jeavons, and A. Krokhin, *Classifying the complexity of constraints using finite algebras*, SIAM J. Comput., 34 (2005), pp. 720–742.

[BKJ00] A. A. Bulatov, A. A. Krokhin, and P. Jeavons, *Constraint satisfaction problems and finite algebras*, in Automata, Languages and Programming (Geneva, 2000), Lecture Notes in Comput. Sci. 1853, Springer, Berlin, 2000, pp. 272–282.

[BS81] S. Burris and H. P. Sankappanavar, *A Course in Universal Algebra*, Graduate Texts in Math. 78, Springer-Verlag, New York, 1981.

[Bul05] A. A. Bulatov, *H-coloring dichotomy revisited*, Theoret. Comput. Sci., 349 (2005), pp. 31–39.

[Bul06] A. A. Bulatov, *A dichotomy theorem for constraint satisfaction problems on a 3-element set*, J. ACM, 53 (2006), pp. 66–120.

[Dal05] V. Dalmau, *A new tractable class of constraint satisfaction problems*, Ann. Math. Artif. Intell., 44 (2005), pp. 61–85.

[Dal06] V. Dalmau, *Generalized majority-minority operations are tractable*, Log. Methods Comput. Sci., 2 (2006), pp. 1–15.

[DD96] R. Dechter and A. Dechter, *Structure-driven algorithms for truth maintenance*, Artificial Intelligence, 82 (1996), pp. 1–20.

[Fed01] T. Feder, *Classification of homomorphisms to oriented cycles and of k-partite satisfiability*, SIAM J. Discrete Math., 14 (2001), pp. 471–480.

[FV99] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[GWW92] W. Gutjahr, E. Welzl, and G. Woeginger, *Polynomial graph-colorings*, Discrete Appl. Math., 35 (1992), pp. 29–45.

[HM88] D. Hobby and R. McKenzie, *The Structure of Finite Algebras*, Contemp. Math. 76, American Mathematical Society, Providence, RI, 1988.

[HN90] P. Hell and J. Nešetřil, *On the complexity of H-coloring*, J. Combin. Theory Ser. B, 48 (1990), pp. 92–110.

[HN04] P. Hell and J. Nešetřil, *Graphs and Homomorphisms*, Oxford Lecture Series in Math. Appl. 28, Oxford University Press, Oxford, 2004.

[HNZ96a] P. Hell, J. Nešetřil, and X. Zhu, *Complexity of tree homomorphisms*, Discrete Appl. Math., 70 (1996), pp. 23–36.

[HNZ96b] P. Hell, J. Nešetřil, and X. Zhu, *Duality and polynomial testing of tree homomorphisms*, Trans. Amer. Math. Soc., 348 (1996), pp. 1281–1297.

[HNZ96c] P. Hell, J. Nešetřil, and X. Zhu, *Duality of graph homomorphisms*, in Combinatorics, Paul Erdős is Eighty, Vol. 2 (Keszthely, 1993), Bolyai Soc. Math. Stud. 2, János Bolyai Mathematical Society, Budapest, 1996, pp. 271–282.

[HP64] Z. Hedrlín and A. Pultr, *Relations (graphs) with given finitely generated semigroups*, Monatsh. Math., 68 (1964), pp. 213–217.

[HZZ93] P. Hell, H. S. Zhou, and X. Zhu, *Homomorphisms to oriented cycles*, Combinatorica, 13 (1993), pp. 421–433.

[JCG97] P. Jeavons, D. Cohen, and M. Gyssens, *Closure properties of constraints*, J. ACM, 44 (1997), pp. 527–548.

[KMRT+07] F. Krząkała, A. Montanari, F. Ricci-Tersenghi, G. Semerjian, and L. Zdeborová, *Gibbs states and the set of solutions of random constraint satisfaction problems*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 10318–10323.

[Kun] G. Kun, *Constraints, MMSNP and Expander Relational Structures*, manuscript, 2007; http://arxiv.org/abs/0706.1701.

[KV07] E. Kiss and M. Valeriote, *On tractability and congruence distributivity*, Log. Methods Comput. Sci., 3 (2007), 2:6 (electronic).

[LALW98] D. Lesaint, N. Azarmi, R. Laithwaite, and P. Walker, *Engineering dynamic scheduler for Work Manager*, BT Technology J., 16 (1998), pp. 16–29.

[Lev73] L. A. Levin, *Universal enumeration problems*, Problemy Peredači Informacii, 9 (1973), pp. 115–116.

[LZ03] B. Larose and L. Zádori, *The complexity of the extendability problem for finite posets*, SIAM J. Discrete Math., 17 (2003), pp. 114–121.

[LZ06] B. Larose and L. Zádori, *Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras*, Internat. J. Algebra Comput., 16 (2006), pp. 563–581.

[Mac77] A. Mackworth, *Consistency in networks of relations*, Artificial Intelligence, 8 (1977), pp. 99–118.

[Mac91] G. MacGillivray, *On the complexity of colouring by vertex-transitive and arc-transitive digraphs*, SIAM J. Discrete Math., 4 (1991), pp. 397–408.

[MM07] M. Maróti and R. McKenzie, *Existence theorems for weakly symmetric operations*, Algebra Universalis, to appear.

[MMT87] R. N. McKenzie, G. F. McNulty, and W. F. Taylor, *Algebras, Lattices, Varieties, Vol.* I, The Wadsworth & Brooks/Cole Mathematics Series, Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA, 1987.

[Mon74] U. Montanari, *Networks of constraints: Fundamental properties and applications to picture processing*, Information Sci., 7 (1974), pp. 95–132.

[Nad] B. A. Nadel, *Constraint satisfaction in Prolog: Complexity and theory-based heuristics*, Inform. Sci., 83 (1995), pp. 113–131.

[NL] B. A. Nadel and J. Lin, *Automobile transmission design as a constraint satisfaction problem: Modeling the kinematic level*, Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), 53 (1991), pp. 137–171.

[Sab61] G. Sabidussi, *Graph derivatives*, Math. Z., 76 (1961), pp. 385–401.

[SV98] E. Schwalb and L. Vila, *Temporal constraints: A survey*, Constraints, 3 (1998), pp. 129–149.

[Tay77] W. Taylor, *Varieties obeying homotopy laws*, Canad. J. Math., 29 (1977), pp. 498–527.

[Var00] M. Vardi, *Constraint satisfaction and database theory: A tutorial*, in Proceedings of 19th ACM Symposium on Principles of Database Systems (PODS'00), ACM, New York, 2000, pp. 76–85.

[Zhu95] X. Zhu, *A polynomial algorithm for homomorphisms to oriented cycles*, J. Algorithms, 19 (1995), pp. 333–345.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1803–1820

SPANNERS OF COMPLETE k-PARTITE GEOMETRIC GRAPHS*

PROSENJIT BOSE†, PAZ CARMI†, MATHIEU COUTURE†, ANIL MAHESHWARI†, PAT MORIN†, AND MICHIEL SMID†

Abstract. We address the following problem: Given a complete k-partite geometric graph K whose vertex set is a set of n points in $\mathbb{R}^d$, compute a spanner of K that has a "small" stretch factor and "few" edges. We present two algorithms for this problem. The first algorithm computes a $(5+\epsilon)$-spanner of K with $O(n)$ edges in $O(n \log n)$ time. The second algorithm computes a $(3+\epsilon)$-spanner of K with $O(n \log n)$ edges in $O(n \log n)$ time. The latter result is optimal: We show that for any $2 \leq k \leq n - \Theta(\sqrt{n \log n})$, spanners with $O(n \log n)$ edges and stretch factor less than 3 do not exist for all complete k-partite geometric graphs.

Key words. computational geometry, spanners, k-partite geometric graphs

AMS subject classification. 68U05

DOI. 10.1137/070707130

1. Introduction.

Let S be a set of n points in $\mathbb{R}^d$. A *geometric graph* with vertex set S is an undirected graph H whose edges are line segments $\overline{pq}$ that are weighted by the Euclidean distance $|pq|$ between p and q. For any two points p and q in S, we denote by $\delta_H(p,q)$ the length of a shortest path in H between p and q. For a real number $t \geq 1$, a subgraph G of H is said to be a *t-spanner* of H if $\delta_G(p,q) \leq t \cdot \delta_H(p,q)$ for all points p and q in S. The smallest t for which this property holds is called the *stretch factor* of G. Thus, a subgraph G of H with stretch factor t approximates the $\binom{n}{2}$ pairwise shortest-path lengths in H within a factor of t. If H is the complete geometric graph with vertex set S, then G is also called a t-spanner of the point set S.

Most of the work on constructing spanners has been done for the case when H is the complete graph. It is well known that, for any set S of n points in $\mathbb{R}^d$ and for any real constant $\epsilon > 0$, there exists a $(1+\epsilon)$-spanner of S containing $O(n)$ edges. Moreover, such spanners can be computed in $O(n \log n)$ time; see Salowe [8] and Vaidya [9]. For a detailed overview of results on spanners for point sets, see the book by Narasimhan and Smid [6].

For spanners of arbitrary geometric graphs, much less is known. Althöfer et al. [1] have shown that, for any $t > 1$, every weighted graph H with n vertices contains a subgraph with $O(n^{1+2/(t-1)})$ edges, which is a t-spanner of H. Observe that this result holds for any weighted graph; in particular, it is valid for any geometric graph. For geometric graphs, a lower bound was given by Gudmundsson and Smid [5]: They proved that, for every real number t with $1 < t < \frac{1}{4} \log n$, there exists a geometric graph H with n vertices, such that every t-spanner of H contains $\Omega(n^{1+1/t})$ edges. Thus, if we are looking for spanners with $O(n)$ edges of arbitrary geometric graphs, then the best stretch factor we can obtain is $\Theta(\log n)$.

In this paper, we consider the case when the input graph is a complete k-partite geometric graph. Let S be a set of n points in $\mathbb{R}^d$, and let S be partitioned into

*Received by the editors November 2, 2007; accepted for publication (in revised form) August 7, 2008; published electronically January 9, 2009. This research was partially supported by NSERC, MRI, CFI, and MITACS.

http://www.siam.org/journals/sicomp/38-5/70713.html

†School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada (jit@cg.scs.carleton.ca, carmip@gmail.com, couturem@gmail.com, anil@scs.carleton.ca, morin@scs.carleton.ca, michiel.smid@gmail.com).

subsets $C_1, C_2, \ldots, C_k$. Let $K_{C_1 \ldots C_k}$ denote the *complete k-partite graph on S*. This graph has S as its vertex set, and two points p and q are connected by an edge (of length $|pq|$) if and only if p and q are in different subsets of the partition. The problem we address is formally defined as follows.

PROBLEM 1.1. *Let $k \geq 2$ be an integer, let S be a set of n points in $\mathbb{R}^d$, and let S be partitioned into k subsets $C_1, C_2, \ldots, C_k$. Compute a t-spanner of the complete k-partite graph $K_{C_1 \ldots C_k}$ that has a "small" number of edges and whose stretch factor t is "small."*

The main contribution of this paper is to present an algorithm that computes such a t-spanner with $O(n)$ edges in $O(n \log n)$ time, where $t = 5+\epsilon$ for any constant $\epsilon > 0$. We also show that if one is willing to use $O(n \log n)$ edges, then our algorithm adapts easily to reach a stretch factor of $t = 3 + \epsilon$. Finally, we show that the latter result is optimal: For any k with $2 \leq k \leq n - \Theta(\sqrt{n \log n})$, spanners with $O(n \log n)$ edges and stretch factor less than 3 do not exist for all complete k-partite geometric graphs.

We remark that, in a recent paper, Bose et al. [2] considered the problem of constructing spanners of point sets that have $O(n)$ edges and whose chromatic number is at most k. This problem is different from ours: Bose et al. compute a spanner of the complete graph, and their algorithm can choose a "good" k-partition of the vertices. In our problem, the k-partition is given, and we want to compute a spanner of the complete k-partite graph.

Possible applications of our algorithm are in wireless networks having the property that communicating nodes are partitioned into sets such that two nodes can communicate if and only if they do not belong to the same set. This would be the case, for example, when time division multiplexing (TDMA) is used. Since the wireless medium prohibits simultaneous transmission and reception at one node, two nodes communicating during the same time slots cannot communicate with each other. For more details, we refer to Raman and Chebrolu [7]; see also Bose et al. [2].

The rest of this paper is organized as follows. In section 2, we recall properties of the well-separated pair decomposition (WSPD) that we use in our algorithm. In section 3, we provide an algorithm that solves the problem of constructing a spanner of the complete k-partite graph. In section 4, we show that the spanner constructed by this algorithm has $O(n)$ edges and that its stretch factor is bounded from above by a constant that depends only on the dimension d. In section 5, we show how a simple modification to our algorithm improves the stretch factor to $5 + \epsilon$ while still having $O(n)$ edges. In section 6, we show how to achieve a stretch factor of $3 + \epsilon$ using $O(n \log n)$ edges. We also prove that the latter result is optimal. We conclude in section 7.

2. The well-separated pair decomposition. In this section, we recall crucial properties of the WSPD of Callahan and Kosaraju [4] that we use for our construction. The reader who is familiar with the WSPD may go directly to section 3. Our presentation follows the one in Narasimhan and Smid [6]. Intuitively, a WSPD is a partition of the edges of a complete geometric graph such that all edges that are grouped together are *approximately* equal. To give a formal definition of the WSPD, we first need to define what it means for two sets to be well separated.

DEFINITION 2.1. *Let S be a set of points in $\mathbb{R}^d$. The* bounding box $\beta(S)$ *of S is the smallest axes-parallel hyperrectangle that contains S.*

DEFINITION 2.2. *Let X and Y be two sets of points in $\mathbb{R}^d$ and let $s > 0$ be a real number. We say that X and Y are* well separated *with respect to s if there exist two*

balls B_1 and B_2 such that

1. *B_1 and B_2 have the same radius, say ρ,*
2. *B_1 contains the bounding box of X,*
3. *B_2 contains the bounding box of Y, and*
4. *the distance $\min\{|xy| : x \in B_1, y \in B_2\}$ between B_1 and B_2 is at least $s\rho$.*

DEFINITION 2.3. *Let S be a set of points in $\mathbb{R}^d$ and let $s > 0$ be a real number. A* WSPD of S with separation constant s *is a set of unordered pairs of subsets of S that are well separated with respect to s such that for any two distinct points $p, q \in S$ there is a unique pair $\{X, Y\}$ in the WSPD such that $p \in X$ and $q \in Y$.*

LEMMA 2.4 (Lemma 9.1.2 in [6]). *Let $s > 0$ be a real number and let X and Y be two-point sets that are well separated with respect to s.*

1. *If $p, p', p'' \in X$ and $q \in Y$, then $|p'p''| \leq (2/s)|pq|$.*
2. *If $p, p' \in X$ and $q, q' \in Y$, then $|p'q'| \leq (1 + 4/s)|pq|$.*

The first part of this lemma states that distances within one set are very small compared to distances between pairs of points having one endpoint in each set. The second part states that all pairs of points having one endpoint in each set have approximately the same distance.

Callahan and Kosaraju [3] have shown how to construct a t-spanner of S from a WSPD: All one has to do is pick from each pair $\{X, Y\}$ an arbitrary edge (p, q) with $p \in X$ and $q \in Y$. Using induction on the rank of the length of the edges in the complete graph K_S, it can be shown that, when $s > 4$, this process leads to a $((s+4)/(s-4))$-spanner. Thus, by choosing s to be a sufficiently large constant, the stretch factor can be made arbitrarily close to 1.

In order to compute a spanner of S that has a linear number of edges, one needs a WSPD that has a linear number of pairs. Callahan and Kosaraju [4] showed that a WSPD with a linear number of pairs always exists and can be computed in time $O(n \log n)$. Their algorithm uses a split-tree.

DEFINITION 2.5. *Let S be a nonempty set of points in $\mathbb{R}^d$. The* split-tree *of S is defined as follows: If S contains only one point, then the split-tree is a single node that stores that point. Otherwise, the split-tree has a root that stores the bounding box $\beta(S)$ of S, as well as an arbitrary point of S called the* representative *of S and denoted by rep(S). Split $\beta(S)$ into two hyperrectangles by cutting its longest interval into two equal parts, and let S_1 and S_2 be the subsets of S contained in the two hyperrectangles. The root of the split-tree of S has two subtrees, which are recursively defined split-trees of S_1 and S_2.*

The precise way Callahan and Kosaraju used the split-tree to compute a WSPD with a linear number of pairs is of no importance to us. The only important aspect we need to retain is that each pair is uniquely determined by a pair of nodes in the tree. More precisely, for each pair $\{X, Y\}$ in the WSPD that is output by their algorithm, there are unique internal nodes u and v in the split-tree such that the sets S_u and S_v of points stored at the leaves of the subtrees rooted at u and v are precisely X and Y. Since there is such a unique correspondence, we will denote pairs in the WSPD by $\{S_u, S_v\}$, meaning that u and v are the nodes corresponding to the sets $X = S_u$ and $Y = S_v$. Also, although the WSPD of a point set is not unique, when we talk about *the* WSPD, we mean the WSPD that is computed by the algorithm of Callahan and Kosaraju.

Before we present our algorithm, we give the statement of the following lemmas that we use to analyze our algorithm in section 4. If R is an axes-parallel hyperrectangle in $\mathbb{R}^d$, then we use $L_{\max}(R)$ to denote the length of a longest side of R.

LEMMA 2.6 (Lemma 9.5.3 in [6]). *Let u be a node in the split-tree and let u' be a node in the subtree of u such that the path between them contains at least d edges. Then*

$$L_{\max}(\beta(S_{u'})) \leq \frac{1}{2} \cdot L_{\max}(\beta(S_u)).$$

LEMMA 2.7 (Lemma 11.3.1 in [6]). *Let $\{S_u, S_v\}$ be a pair in the WSPD, let ℓ be the distance between the centers of $\beta(S_u)$ and $\beta(S_v)$, and let $\pi(u)$ be the parent of u in the split-tree. Then*

$$L_{\max}(\beta(S_{\pi(u)})) \geq \frac{2\ell}{\sqrt{d}(s+4)}.$$

3. A first algorithm. We now show how the WSPD can be used to address the problem of computing a spanner of a complete k-partite graph. In this section, we introduce an algorithm that outputs a graph with constant stretch factor and $O(n)$ edges. The analysis of this algorithm is presented in section 4. In section 5, we show how this algorithm can be improved to achieve a stretch factor of $5 + \epsilon$.

The input set $S \subseteq \mathbb{R}^d$ is the disjoint union of k sets $C_1, C_2, \ldots, C_k$. We say that the elements of C_c have "color" c. The graph $K = K_{C_1 \ldots C_k}$ is the complete k-partite geometric graph.

DEFINITION 3.1. *Let T be the split-tree of S that is used to compute the WSPD of S.*

1. *For any node u in T, we denote by S_u the set of all points in the subtree rooted at u.*
2. *We define MWSPD to be the subset of the WSPD obtained by removing all pairs $\{S_u, S_v\}$ for which all points of $S_u \cup S_v$ have the same color.*
3. *A node u in T is called* multichromatic *if there exist points p and q in S_u and a node v in T such that p and q have different colors and $\{S_u, S_v\}$ is in the MWSPD.*
4. *A node u in T is called a c-*node *if all points of S_u have color c and there exists a node v in T such that $\{S_u, S_v\}$ is in the MWSPD.*
5. *A c-node u in T is called a c-*root *if it does not have a proper ancestor that is a c-node in T.*
6. *A c-node u in T is called a c-*leaf *if it does not have another c-node in its subtree.*
7. *A c-node u' in T is called a c-*child *of a c-node u in T if u' is in the subtree rooted at u and there is no c-node on the path strictly between u and u'. In this case, we also say that u is a c-*parent *of u'.*
8. *For each color c and for each c-node u in T, $rep(S_u)$ denotes a fixed arbitrary point in S_u.*
9. *For each multichromatic node u in T, $rep(S_u)$ and $rep'(S_u)$ denote two fixed arbitrary points in S_u that have different colors.*
10. *The* distance *between two sets S_v and S_w, denoted by $dist(S_v, S_w)$, is defined to be the distance between the centers of their bounding boxes.*
11. *Let u be a c-node in T. Consider all pairs $\{S_v, S_w\}$ in the MWSPD, where v is a c-node on the path in T from u to the root (this path includes u). Let $\{S_v, S_w\}$ be such a pair for which $dist(S_v, S_w)$ is minimum. We define $cl(S_u)$ to be the set S_w.*

Algorithm 1 computes a spanner of a complete k-partite geometric graph $K = K_{C_1 \ldots C_k}$. The intuition behind this algorithm is the following. First, the algorithm

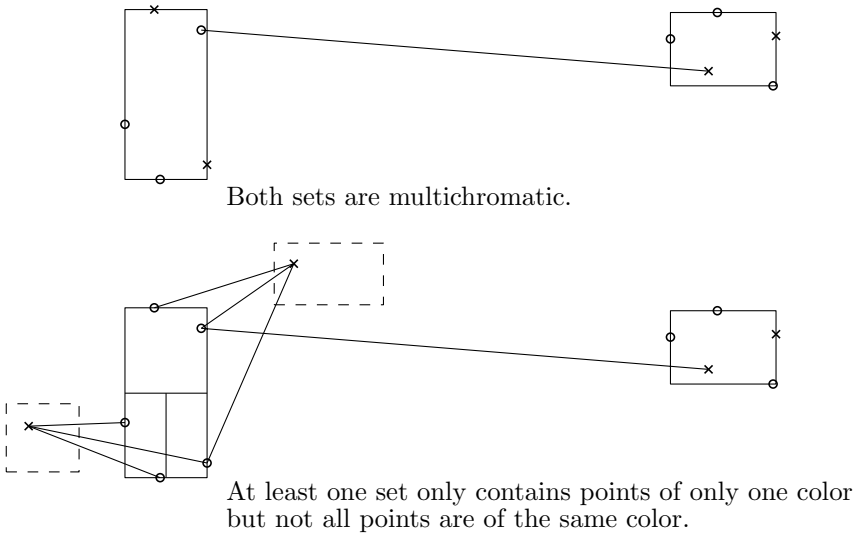


FIG. 1. *The two cases of Algorithm* 1.

computes the MWSPD. Then, it considers each pair $\{S_u, S_v\}$ of the MWSPD and decides whether or not to add an edge between S_u and S_v. The outcome of this decision is based on the following two cases, which are illustrated in Figures 1 and 2.

Case 1. Both S_u and S_v are multichromatic. In this case, Algorithm 1 adds one edge between S_u and S_v to the spanner; see lines 28–29. Observe that the two vertices of this edge do not have the same color. This edge will allow us to approximate each edge (p, q) of K, where $p \in S_u$, $q \in S_v$, and p and q have different colors.

Case 2. All points in S_u are of the same color c. In this case, an edge is added between $\text{rep}(S_u)$ and a representative of S_v whose color is not c; see lines 17–18. In order to approximate each edge of K having one vertex (of color c) in S_u and the other vertex (of a different color) in S_v, more edges have to be added. This is done in such a way that our final graph contains a "short" path between every point p of S_u and the representative $\text{rep}(S_u)$ of S_u. Observe that this path must contain points whose color is not equal to c; thus, these points are not in S_u. One way to achieve this is to add an edge between each point of S_u and a representative of $\text{cl}(S_u)$ whose color is not c; we call this construction a *star*. However, since the subtree rooted at u may contain other c-nodes, many edges may be added for each point in S_u, which could possibly lead to a quadratic number of edges in the final graph. To guarantee that the algorithm does not add too many edges, it introduces a star only if u is a c-leaf; see lines 8–11. If u is a c-node, the algorithm adds only one edge between $\text{rep}(S_u)$ and a representative of $\text{cl}(S_u)$ whose color is not c; see lines 14–15. Then, the algorithm links each c-node u'' that is not a c-root to its c-parent u'. This is done through an edge between $\text{rep}(S_{u''})$ and a representative of $\text{cl}(S_{u'})$ whose color is not c; see lines 21–22. This second case is illustrated in Figure 2.

4. Analysis of Algorithm 1.

LEMMA 4.1. *The graph G computed by Algorithm* 1 *has $O(|S|)$ edges.*

Proof. For each color c and for each c-leaf u', the algorithm adds $|S_{u'}|$ edges to G in lines 9–10. Since the sets $S_{u'}$, where u' ranges over all c-leaves and c ranges over

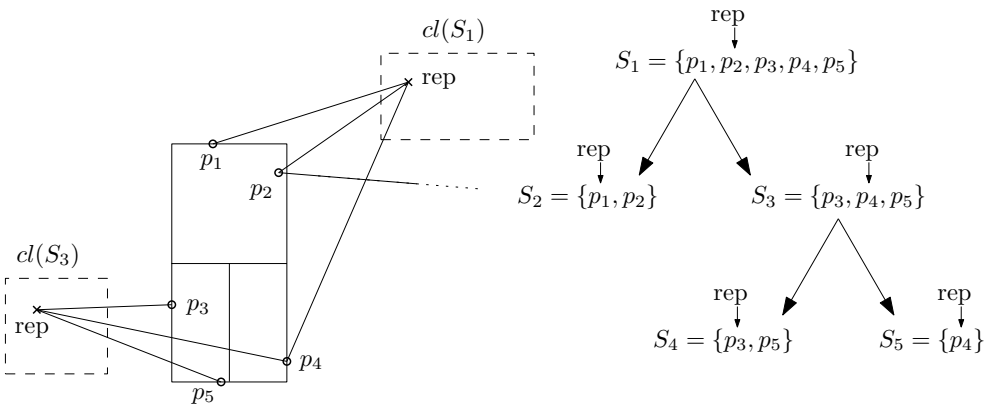


FIG. 2. *Handling a c-node.*

all colors, are pairwise disjoint, the total number of edges that are added in lines 9–10 is $O(|S|)$.

The total number of edges that are added in lines 17–18 and 28–29 is at most the number of pairs in the MWSPD. Since the WSPD contains $O(|S|)$ pairs (see [4]), the same upper bound holds for the number of edges added in lines 17–18 and 28–29.

The total number of edges that are added in lines 14–15 and 21–22 is at most twice the number of nodes in the split-tree, which is $O(|S|)$. □

LEMMA 4.2. *Let G be the graph computed by Algorithm* 1. *Let p and q be two points of S with different colors, and let $\{S_u, S_v\}$ be the pair in the MWSPD for which $p \in S_u$ and $q \in S_v$. Assume that u is a c-node for some color c. Then there is a path in G between p and $rep(S_u)$ whose length is at most $t'|pq|$, where*

$$t' = 4\sqrt{d}(\mu d + 1)(1 + 4/s)^3,$$

$$\mu = \left\lceil \log\left(\sqrt{d}(1 + 4/s)\right)\right\rceil + 1,$$

and s is the separation constant of the WSPD.

Proof. Let w be the c-leaf such that $p \in S_w$, and let $w = w_0, \ldots, w_k = u$ be the sequence of c-nodes that are on the path in T from w to u.

Recall from Definition 3.1 that each set S_{w_i}, $0 \le i \le k$, has a representative $\mathrm{rep}(S_{w_i})$ (of color c) associated with it. Also, recall the definition of the sets $\mathrm{cl}(S_{w_i})$, $0 \le i \le k$; see Definition 3.1. If $\mathrm{cl}(S_{w_i})$ is a c'-node for some color c', then this set has one representative $\mathrm{rep}(\mathrm{cl}(S_{w_i}))$ associated with it. Otherwise, $\mathrm{cl}(S_{w_i})$ is multichromatic, and this set has two representatives $\mathrm{rep}(\mathrm{cl}(S_{w_i}))$ and $\mathrm{rep}'(\mathrm{cl}(S_{w_i}))$ of different colors associated with it. We may assume without loss of generality that, for all $0 \le i \le k$, the color of $\mathrm{rep}(\mathrm{cl}(S_{w_i}))$ is not equal to c.

Let Π be the path

$$\begin{array}{rcccl} p & \to & \mathrm{rep}(\mathrm{cl}(S_{w_0})) & \to & \mathrm{rep}(S_{w_0}) \\ & \to & \mathrm{rep}(\mathrm{cl}(S_{w_1})) & \to & \mathrm{rep}(S_{w_1}) \\ & \vdots & & \vdots & \\ & \to & \mathrm{rep}(\mathrm{cl}(S_{w_k})) & \to & \mathrm{rep}(S_{w_k}) \quad = \quad \mathrm{rep}(S_u). \end{array}$$

Even though we use arrows to define this path, we remark that the graph G and, therefore, the path Π, is undirected.

ALGORITHM 1: Computing a sparse subgraph of $K_{C_1 \ldots C_k}$ whose stretch factor is bounded by a constant.

Input: A set S of points in $\mathbb{R}^d$, which is partitioned into k subsets $C_1, \ldots, C_k$.
Output: A spanner $G = (S, E)$ of the complete k-partite graph $K_{C_1 \ldots C_k}$.

```
1  compute the split-tree T of S;
2  using T, compute the WSPD with respect to a separation constant s > 0;
3  using the WSPD, compute the MWSPD;
4  E ← ∅;
5  for each color c in {1, 2, ..., k} do
6      for each c-root u in T do
7          for each c-leaf u' in the subtree of u do
8              for each p ∈ S_u' do
9                  if rep(cl(S_u')) does not have color c, then add (p, rep(cl(S_u'))) to E;
10                 else add (p, rep'(cl(S_u'))) to E;
11             end
12         end
13         for each c-node u' that is in the subtree of u (including u) do
14             if rep(cl(S_u')) does not have color c, then add (rep(S_u'), rep(cl(S_u'))) to
               E;
15             else add (rep(S_u'), rep'(cl(S_u'))) to E;
16             for each pair {S_u', S_v'} in the MWSPD do
17                 if rep(S_v') does not have color c, then add (rep(S_u'), rep(S_v')) to E;
18                 else add (rep(S_u'), rep'(S_v')) to E;
19             end
20             for each c-child u'' of u' do
21                 if rep(cl(S_u')) does not have color c, then add
                   (rep(S_u''), rep(cl(S_u'))) to E;
22                 else add (rep(S_u''), rep'(cl(S_u'))) to E;
23             end
24         end
25     end
26 end
27 for each {S_u, S_v} in the MWSPD for which both u and v are multichromatic do
28     if rep(S_u) and rep(S_v) have distinct colors, then add (rep(S_u), rep(S_v)) to E;
29     else add (rep(S_u), rep'(S_v)) to E;
30 end
31 return the graph G = (S, E)
```

The first edge on the path Π, i.e., $(p, \mathrm{rep}(\mathrm{cl}(S_{w_0})))$, is added to the graph G in lines 9–10 of the algorithm. The edges $(\mathrm{rep}(\mathrm{cl}(S_{w_i})), \mathrm{rep}(S_{w_i}))$, $0 \le i \le k$, are added to G in lines 14–15. Finally, the edges $(\mathrm{rep}(S_{w_{i-1}}), \mathrm{rep}(\mathrm{cl}(S_{w_i})))$, $1 \le i \le k$, are added to G in lines 21–22. It follows that Π is a path in G between p and $\mathrm{rep}(S_u)$. In the rest of the proof, we will show that the length of Π is at most $t'|pq|$.

Let i be an integer with $0 \le i \le k$. Recall the definition of $\mathrm{cl}(S_{w_i})$; see Definition 3.1: We consider all pairs $\{S_x, S_y\}$ in the MWSPD, where x is a c-node on the path in T from w_i to the root, and pick the pair for which $\mathrm{dist}(S_x, S_y)$ is minimum. We denote the pair picked by (S_{x_i}, S_{y_i}). Thus, x_i is a c-node on the path in T from w_i to the root, $\{S_{x_i}, S_{y_i}\}$ is a pair in the MWSPD, and $\mathrm{cl}(S_{w_i}) = S_{y_i}$. We define

$$\ell_i = \mathrm{dist}(S_{x_i}, S_{y_i}).$$

We start by proving an upper bound on the length of Π in terms of $\ell_0, \ell_1, \ldots, \ell_k$. Consider the first edge $(p, \text{rep}(\text{cl}(S_{w_0})))$ on the path Π. Since $p \in S_{w_0} \subseteq S_{x_0}$ and $\text{rep}(\text{cl}(S_{w_0})) \in S_{y_0}$, it follows from Lemma 2.4 that

$$|p, \text{rep}(\text{cl}(S_{w_0}))| \leq (1 + 4/s) \cdot \text{dist}(S_{x_0}, S_{y_0}) = (1 + 4/s)\ell_0.$$

Let $0 \leq i \leq k$ and consider the edge $(\text{rep}(\text{cl}(S_{w_i})), \text{rep}(S_{w_i}))$ on Π. Since $\text{rep}(S_{w_i}) \in S_{w_i} \subseteq S_{x_i}$ and $\text{rep}(\text{cl}(S_{w_i})) \in S_{y_i}$, it follows from Lemma 2.4 that

$$|\text{rep}(\text{cl}(S_{w_i})), \text{rep}(S_{w_i})| \leq (1 + 4/s) \cdot \text{dist}(S_{x_i}, S_{y_i}) = (1 + 4/s)\ell_i. \tag{4.1}$$

Let $1 \leq i \leq k$ and consider the edge $(\text{rep}(S_{w_{i-1}}), \text{rep}(\text{cl}(S_{w_i})))$ on Π. Since $\text{rep}(S_{w_{i-1}}) \in S_{w_{i-1}} \subseteq S_{x_i}$ and $\text{rep}(\text{cl}(S_{w_i})) \in S_{y_i}$, it follows from Lemma 2.4 that

$$|\text{rep}(S_{w_{i-1}}), \text{rep}(\text{cl}(S_{w_i}))| \leq (1 + 4/s) \cdot \text{dist}(S_{x_i}, S_{y_i}) = (1 + 4/s)\ell_i.$$

Thus, the length of the path Π is at most

$$\sum_{i=0}^{k} 2(1 + 4/s)\ell_i.$$

Therefore, it is sufficient to prove that

$$\sum_{i=0}^{k} \ell_i \leq 2\sqrt{d}(\mu d + 1)(1 + 4/s)^2|pq|.$$

Next, we prove an upper bound on ℓ_k in terms of $|pq|$. As a result, we will obtain an inequality (see (4.3) below) which implies the above inequality.

It follows from the definition of $\text{cl}(S_u) = \text{cl}(S_{w_k})$ that

$$\ell_k = \text{dist}(S_{x_k}, S_{y_k}) \leq \text{dist}(S_u, S_v).$$

Since, by Lemma 2.4, $\text{dist}(S_u, S_v) \leq (1 + 4/s)|pq|$, it follows that

$$\ell_k \leq (1 + 4/s)|pq|. \tag{4.2}$$

Thus, it is sufficient to prove that

$$\sum_{i=0}^{k} \ell_i \leq 2\sqrt{d}(\mu d + 1)(1 + 4/s)\ell_k. \tag{4.3}$$

For each i with $0 \leq i \leq k$, we define

$$a_i = L_{\max}(\beta(S_{w_i}));$$

i.e., a_i is the length of a longest side of the bounding box of S_{w_i}.

We now present an outline of the rest of the proof (which consists of proving (4.3)). As we will see below, Lemma 2.4 implies that (i) $a_i \leq \frac{2}{s}\ell_i$. It follows from Lemma 2.6 that (ii) $a_i \leq \frac{1}{2}a_{i+d}$. Finally, we will show that Lemma 2.7 implies that (iii) $\ell_i \leq \frac{\sqrt{d}(s+4)}{2}a_{i+1}$. By combining (i), (ii), and (iii), we obtain the inequality $\ell_i \leq \frac{1}{2}\ell_{i+1+\mu d}$, where μ is defined in the statement of the lemma. This allows us to split the summation $\sum_{i=0}^{k} \ell_i$ into $\mu d + 1$ geometric series. The final step is then

to prove that the total sum of these geometric series is at most the quantity on the right-hand side in (4.3). This approach makes sense only if k is sufficiently large. For small values of k, we will prove (4.3) by a direct argument.

We now present the details. If $k = 0$, then (4.3) obviously holds. Assume from now on that $k \geq 1$. Let $0 \leq i \leq k$. It follows from Lemma 2.4 that

$$L_{\max}(\beta(S_{x_i})) \leq \frac{2}{s}\ell_i.$$

Since w_i is in the subtree of x_i, we have $L_{\max}(\beta(S_{w_i})) \leq L_{\max}(\beta(S_{x_i}))$. Thus, we have

$$a_i \leq \frac{2}{s}\ell_i \quad \text{for } 0 \leq i \leq k. \tag{4.4}$$

Lemma 2.6 states that

$$a_i \leq \frac{1}{2}a_{i+d} \quad \text{for } 0 \leq i \leq k-d. \tag{4.5}$$

Let $0 \leq i \leq k-1$. Since w_i is a c-node, there is a node w_i' such that $\{S_{w_i}, S_{w_i'}\}$ is a pair in the MWSPD. Then it follows from the definition of $\mathrm{cl}(S_{w_i})$ that

$$\ell_i = \mathrm{dist}(S_{x_i}, S_{y_i}) \leq \mathrm{dist}(S_{w_i}, S_{w_i'}).$$

By applying Lemma 2.7, we obtain

$$\begin{aligned}\mathrm{dist}(S_{w_i}, S_{w_i'}) &\leq \frac{\sqrt{d}(s+4)}{2}\, L_{\max}(\beta(S_{\pi(w_i)})) \\ &\leq \frac{\sqrt{d}(s+4)}{2}\, L_{\max}(\beta(S_{w_{i+1}})) \\ &= \frac{\sqrt{d}(s+4)}{2}\, a_{i+1}.\end{aligned}$$

Thus, we have

$$\ell_i \leq \frac{\sqrt{d}(s+4)}{2}\, a_{i+1} \quad \text{for } 0 \leq i \leq k-1. \tag{4.6}$$

Recall the integer μ as defined in the statement of the lemma. First assume that $1 \leq k \leq \mu d$. Let $0 \leq i \leq k-1$. By using (4.6), the fact that the sequence $a_0, a_1, \ldots, a_k$ is nondecreasing, and (4.4), we obtain

$$\ell_i \leq \frac{\sqrt{d}(s+4)}{2}\, a_{i+1} \leq \frac{\sqrt{d}(s+4)}{2}\, a_k \leq \sqrt{d}\left(1 + \frac{4}{s}\right)\ell_k.$$

Therefore,

$$\sum_{i=0}^{k} \ell_i \leq k\sqrt{d}(1+4/s)\ell_k + \ell_k \leq (k+1)\sqrt{d}(1+4/s)\ell_k \leq (\mu d+1)\sqrt{d}(1+4/s)\ell_k,$$

which is less than the right-hand side in (4.3).

It remains to consider the case when $k > \mu d$. Let $i \geq 0$ and $j \geq 0$ be integers such that $i + 1 + jd \leq k$. By applying (4.6) once, (4.5) j times, and (4.4) once, we obtain

$$\ell_i \leq \frac{\sqrt{d}(s+4)}{2}\, a_{i+1} \leq \frac{\sqrt{d}(s+4)}{2} \left(\frac{1}{2}\right)^j a_{i+1+jd} \leq \sqrt{d}\left(1+\frac{4}{s}\right)\left(\frac{1}{2}\right)^j \ell_{i+1+jd}.$$

For $j = \mu = \lceil \log(\sqrt{d}(1+4/s)) \rceil + 1$, this implies that, for $0 \leq i \leq k-1-\mu d$,

$$\ell_i \leq \frac{1}{2}\ell_{i+1+\mu d}. \tag{4.7}$$

By rearranging the terms in the summation in (4.3), we obtain

$$\sum_{i=0}^{k} \ell_i = \sum_{h=0}^{\mu d} \sum_{j=0}^{\lfloor (k-h)/(\mu d+1) \rfloor} \ell_{k-h-j(\mu d+1)}.$$

Let j be such that $0 \leq j \leq \lfloor (k-h)/(\mu d+1) \rfloor$. By applying (4.7) j times, we obtain

$$\ell_{k-h-j(\mu d+1)} \leq \left(\frac{1}{2}\right)^j \ell_{k-h}.$$

It follows that

$$\sum_{j=0}^{\lfloor (k-h)/(\mu d+1) \rfloor} \ell_{k-h-j(\mu d+1)} \leq \sum_{j=0}^{\infty} \left(\frac{1}{2}\right)^j \ell_{k-h} = 2\ell_{k-h}.$$

Thus, we have

$$\sum_{i=0}^{k} \ell_i \leq 2 \sum_{h=0}^{\mu d} \ell_{k-h}.$$

By applying (4.6) and the fact that the sequence $a_0, a_1, \ldots, a_k$ is nondecreasing, followed by (4.4), we obtain, for $0 \leq i \leq k-1$ and $1 \leq j \leq k-i$,

$$\ell_i \leq \frac{\sqrt{d}(s+4)}{2}\, a_{i+1} \leq \frac{\sqrt{d}(s+4)}{2}\, a_{i+j} \leq \sqrt{d}\left(1+\frac{4}{s}\right)\ell_{i+j}.$$

Obviously, the inequality $\ell_i \leq \sqrt{d}(1+4/s)\ell_{i+j}$ also holds for $j = 0$. Thus, for $i = k-h$ and $j = h$, we get

$$\ell_{k-h} \leq \sqrt{d}(1+4/s)\ell_k \quad \text{for } 0 \leq h \leq \mu d.$$

It follows that

$$\sum_{i=0}^{k} \ell_i \leq 2 \sum_{h=0}^{\mu d} \sqrt{d}(1+4/s)\ell_k = 2\sqrt{d}(\mu d+1)(1+4/s)\ell_k,$$

completing the proof that (4.3) holds. □

LEMMA 4.3. *Assuming that the separation constant s of the WSPD is chosen sufficiently large, the graph G computed by Algorithm 1 is a t-spanner of the complete k-partite graph $K_{C_1 \ldots C_k}$, where $t = 2t' + 1 + 4/s$ and t' is as in Lemma 4.2.*

Proof. We denote the graph $K_{C_1 \ldots C_k}$ by K. It suffices to show that for each edge (p, q) of K, the graph G contains a path between p and q of length at most $t|pq|$. We will prove this by induction on the lengths of the edges in K.

Let p and q be two points of S with different colors, and let $\{S_u, S_v\}$ be the pair in the MWSPD for which $p \in S_u$ and $q \in S_v$.

The base case is when (p, q) is a shortest edge in K. Since $s > 2$, it follows from Lemma 2.4 that u is a c-node and v is a c'-node, for some colors c and c' with $c \neq c'$. In line 17 of Algorithm 1, the edge $(\mathrm{rep}(S_u), \mathrm{rep}(S_v))$ is added to G. By Lemma 2.4, the length of this edge is at most $(1 + 4/s)|pq|$. The claim follows from two applications of Lemma 4.2 to get from p to $\mathrm{rep}(S_u)$ and from $\mathrm{rep}(S_v)$ to q.

In the induction step, we distinguish four cases.

Case 1. u is a c-node and v is a c'-node, for some colors c and c' with $c \neq c'$.

This case is identical to the base case.

Case 2. u is a c-node for some color c and v is a multichromatic node.

In lines 17–18, the edge $(\mathrm{rep}(S_u), \mathrm{rep}(S_v))$ or $(\mathrm{rep}(S_u), \mathrm{rep}(S'_v))$ is added to G. We may assume without loss of generality that $(\mathrm{rep}(S_u), \mathrm{rep}(S_v))$ is added. By Lemma 2.4, the length of this edge is at most $(1 + 4/s)|pq|$.

By Lemma 4.2, there is a path in G between p and $\mathrm{rep}(S_u)$ whose length is at most $t'|pq|$.

First assume that q and $\mathrm{rep}(S_v)$ have the same color. Let r be a point in S_v that has a color different from q's color. Since $s > 2$, it follows from Lemma 2.4 that $|qr| < |pq|$. Thus, by induction, there is a path in G between q and r whose length is at most $t|qr|$, which, by Lemma 2.4, is at most $(2t/s)|pq|$. By a similar argument, since $|r, \mathrm{rep}(S_v)| < |pq|$, there is a path in G between r and $\mathrm{rep}(S_v)$ whose length is at most $(2t/s)|pq|$. Thus, G contains a path between q and $\mathrm{rep}(S_v)$ of length at most $(4t/s)|pq|$. If q and $\mathrm{rep}(S_v)$ have different colors, then, by induction, there is a path in G between q and $\mathrm{rep}(S_v)$ whose length is at most $(2t/s)|pq| < (4t/s)|pq|$.

Thus, the graph G contains a path between q and $\mathrm{rep}(S_v)$ of length at most $(4t/s)|pq|$.

We have shown that there is a path in G between p and q whose length is at most

$$(t' + (1 + 4/s) + 4t/s)\,|pq|. \tag{4.8}$$

By choosing s sufficiently large, this quantity is at most $t|pq|$.

Case 3. u is a multichromatic node and v is a c-node for some color c.

This case is symmetric to Case 2.

Case 4. Both u and v are multichromatic nodes.

In lines 28–29, the edge $(\mathrm{rep}(S_u), \mathrm{rep}(S_v))$ or $(\mathrm{rep}(S_u), \mathrm{rep}(S'_v))$ is added to G. We may assume without loss of generality that $(\mathrm{rep}(S_u), \mathrm{rep}(S_v))$ is added. By Lemma 2.4, the length of this edge is at most $(1 + 4/s)|pq|$.

As in Case 2, the graph G contains a path between p and $\mathrm{rep}(S_u)$ of length at most $(4t/s)|pq|$ and a path between q and $\mathrm{rep}(S_v)$ of length at most $(4t/s)|pq|$.

It follows that there is a path in G between p and q whose length is at most

$$((1 + 4/s) + 8t/s)\,|pq|. \tag{4.9}$$

By choosing s sufficiently large, this quantity is at most $t|pq|$. □

LEMMA 4.4. *The running time of Algorithm* 1 *is* $O(n \log n)$, *where* $n = |S|$.

Proof. Using the results of Callahan and Kosaraju [4], the split-tree T and the WSPD can be computed in $O(n \log n)$ time. The representatives of all sets S_u and all sets $\mathrm{cl}(S_u)$ can be computed in $O(n)$ time by traversing the split-tree in postorder

and preorder, respectively. The time for the rest of the algorithm, i.e., lines 3–31, is proportional to the sum of the size of T, the number of pairs in the WSPD, and the number of edges in the graph G. Thus, the rest of the algorithm takes $O(n)$ time. □

To summarize, we have shown the following: For any complete k-partite geometric graph K whose vertex set has size n, Algorithm 1 computes a t-spanner of K having $O(n)$ edges, where t is given in Lemma 4.3. The running time of this algorithm is $O(n \log n)$. By choosing the separation constant s sufficiently large, the stretch factor t converges to

$$8\sqrt{d}\left(d\left\lceil\frac{1}{2}\log d\right\rceil + d + 1\right) + 1.$$

In the next section, we show how to modify the algorithm so that the bound in Lemma 4.2 is reduced, thus improving the stretch factor. The price to pay is in the number of edges in G; however, it is still $O(n)$.

5. An improved algorithm. As before, we are given a set S of n points in $\mathbb{R}^d$ which is partitioned into k subsets $C_1, C_2, \ldots, C_k$. Intuitively, the way to improve the bound of Lemma 4.2 is by adding shortcuts along the path from each c-leaf to the c-root above it. More precisely, from (4.7) in the proof of Lemma 4.2, we know that if we go $1 + \mu d$ levels up in the split-tree T, then the length of the edge along the path doubles. Thus, for each c-node in T, we will add edges to all $2\delta(1 + \mu d)$ c-nodes above it in T. Here, δ is an integer constant that is chosen such that the best result is obtained in the improved bound.

DEFINITION 5.1. *Let $c \in \{1, 2, \ldots, k\}$, and let u and u' be c-nodes in the split-tree T such that u' is in the subtree rooted at u. For any integer $\zeta \geq 1$, we say that u is* ζ levels above *u' if there are exactly $\zeta - 1$ c-nodes on the path strictly between u and u'. We say that u' is a* ζ-level c-child *of u if u is at most ζ levels above u'.*

The improved algorithm is given as Algorithm 2. The following lemma generalizes Lemma 4.2.

LEMMA 5.2. *Let G be the graph computed by Algorithm 2. Let p and q be two points of S with different colors, and let $\{S_u, S_v\}$ be the pair in the MWSPD for which $p \in S_u$ and $q \in S_v$. Assume that u is a c-node for some color c. Then there is a path in G between p and $\mathrm{rep}(S_u)$ whose length is at most $(2 + \epsilon/3)|pq|$.*

Proof. Let w be the c-leaf such that $p \in S_w$, and let $w = w_0, w_1, \ldots, w_k = u$ be the sequence of c-nodes that are on the path in T from w to u. As in the proof of Lemma 4.2, we assume without loss of generality that, for all $0 \leq i \leq k$, the color of $\mathrm{rep}(\mathrm{cl}(S_{w_i}))$ is not equal to c.

Throughout the proof, we will use the variables x_i, y_i, ℓ_i, and a_i, for $0 \leq i \leq k$, that were introduced in the proof of Lemma 4.2.

We first assume that $0 \leq k \leq 2\delta(\mu d + 1)$. Let Π be the path

$$p \to \mathrm{rep}(\mathrm{cl}(S_w)) \to \mathrm{rep}(S_u).$$

It follows from Algorithm 2 that Π is a path in G. We have to prove that the length of Π is at most $(2 + \epsilon/3)|pq|$.

Since $p \in S_w = S_{w_0} \subseteq S_{x_0}$ and $\mathrm{rep}(\mathrm{cl}(S_w)) = \mathrm{rep}(\mathrm{cl}(S_{w_0})) \in S_{y_0}$, it follows from Lemma 2.4 that

$$|p, \mathrm{rep}(\mathrm{cl}(S_w))| \leq (1 + 4/s) \cdot \mathrm{dist}(S_{x_0}, S_{y_0}) = (1 + 4/s)\ell_0. \tag{5.1}$$

ALGORITHM 2: Computing a sparse $(5+\epsilon)$-spanner of $K_{C_1 \ldots C_k}$.

Input: A set S of points in $\mathbb{R}^d$, which is partitioned into k subsets $C_1, \ldots, C_k$, and a real constant $0 < \epsilon < 1$.

Output: A $(5+\epsilon)$-spanner $G = (S, E)$ of the complete k-partite graph $K_{C_1 \ldots C_k}$.

Choose a separation constant s such that $s \geq 12/\epsilon$ and $(1+4/s)^2 \leq 1+\epsilon/36$ and choose an integer constant δ such that $\frac{2^\delta}{2^\delta - 1} \leq 1 + \epsilon/36$.
The rest of the algorithm is the same as Algorithm 1, except for lines 20–23, which are replaced by the following:

$\zeta \leftarrow 2\delta(\mu d + 1)$;
for each ζ-level c-child u'' of u' **do**
  if $\text{rep}(\text{cl}(S_{u'}))$ does not have color c, **then** add $(\text{rep}(S_{u''}), \text{rep}(\text{cl}(S_{u'})))$ to E;
  else add $(\text{rep}(S_{u''}), \text{rep}'(\text{cl}(S_{u'})))$ to E;
  if $\text{rep}(\text{cl}(S_{u''}))$ does not have color c, **then** add $(\text{rep}(\text{cl}(S_{u''})), \text{rep}(S_{u'}))$ to E;
  else add $(\text{rep}'(\text{cl}(S_{u''})), \text{rep}(S_{u'}))$ to E;
end

Since $\{S_u, S_v\}$ is one of the pairs that is considered in the definition of $\text{cl}(S_{w_0})$, we have $\text{dist}(S_{x_0}, S_{y_0}) \leq \text{dist}(S_u, S_v)$. Again by Lemma 2.4, we have $\text{dist}(S_u, S_v) \leq (1+4/s)|pq|$. Thus, we have shown that

$$|p, \text{rep}(\text{cl}(S_w))| \leq (1+4/s)^2 |pq|.$$

By the triangle inequality, we have

$$|\text{rep}(\text{cl}(S_w)), \text{rep}(S_u)| \leq |\text{rep}(\text{cl}(S_w)), p| + |p, \text{rep}(S_u)|.$$

Since p and $\text{rep}(S_u)$ are both contained in S_u, it follows from Lemma 2.4 that $|p, \text{rep}(S_u)| \leq (2/s)|pq|$. Thus, we have

$$|\text{rep}(\text{cl}(S_w)), \text{rep}(S_u)| \leq (1+4/s)^2 |pq| + (2/s)|pq|.$$

We have shown that the length of the path Π is at most

$$\left(2(1+4/s)^2 + 2/s\right)|pq|,$$

which is at most $(2+\epsilon/3)|pq|$ by our choice of s in Algorithm 2.

In the rest of the proof, we assume that $k > 2\delta(\mu d + 1)$. We define

$$m = k \bmod (\delta(\mu d + 1))$$

and

$$m' = \frac{k - m}{\delta(\mu d + 1)}.$$

In the proof of Lemma 4.2, we defined the path Π between p and $\text{rep}(S_u)$ by using all c-nodes $w = w_0, w_1, \ldots, w_k = u$. Since Algorithm 2 adds shortcuts, it suffices to define Π using only the sequence

$$w = w_0, w_{\delta(\mu d+1)+m}, w_{2\delta(\mu d+1)+m}, w_{3\delta(\mu d+1)+m}, \ldots, w_k = u$$

of c-nodes. We define Π to be the path

$$\begin{array}{rcccccl} p & \to & \mathrm{rep}(\mathrm{cl}(S_{w_0})) & \to & \mathrm{rep}(S_{w_{\delta(\mu d+1)+m}}) & & \\ & \to & \mathrm{rep}(\mathrm{cl}(S_{w_{2\delta(\mu d+1)+m}})) & \to & \mathrm{rep}(S_{w_{2\delta(\mu d+1)+m}}) & & \\ & \to & \mathrm{rep}(\mathrm{cl}(S_{w_{3\delta(\mu d+1)+m}})) & \to & \mathrm{rep}(S_{w_{3\delta(\mu d+1)+m}}) & & \\ & \vdots & & \vdots & & & \\ & \to & \mathrm{rep}(\mathrm{cl}(S_{w_k})) & \to & \mathrm{rep}(S_{w_k}) & = & \mathrm{rep}(S_u). \end{array}$$

It follows from Algorithm 2 that Π is a path in G. Thus, it remains to show that the length of this path is at most $(2+\epsilon/3)|pq|$.

We start by proving an upper bound on the length of Π in terms of $|pq|$, $\ell_{\delta(\mu d+1)+m}$, $\ell_{2\delta(\mu d+1)+m}$, $\ell_{3\delta(\mu d+1)+m}, \ldots, \ell_k$.

We have already shown (see (5.1)) that the length of the first edge on Π satisfies

$$|p, \mathrm{rep}(\mathrm{cl}(S_{w_0}))| \le (1+4/s)\ell_0.$$

The length of the second edge satisfies

$$\begin{aligned} |\mathrm{rep}(\mathrm{cl}(S_{w_0})), \mathrm{rep}(S_{w_{\delta(\mu d+1)+m}})| &\le |\mathrm{rep}(\mathrm{cl}(S_{w_0})), p| + |p, \mathrm{rep}(S_{w_{\delta(\mu d+1)+m}})| \\ &\le (1+4/s)\ell_0 + |p, \mathrm{rep}(S_{w_{\delta(\mu d+1)+m}})|. \end{aligned}$$

Since p and $\mathrm{rep}(S_{w_{\delta(\mu d+1)+m}})$ are both contained in S_u, it follows from Lemma 2.4 that

$$|p, \mathrm{rep}(S_{w_{\delta(\mu d+1)+m}})| \le (2/s)|pq|.$$

Thus, the length of the second edge on Π satisfies

$$|\mathrm{rep}(\mathrm{cl}(S_{w_0})), \mathrm{rep}(S_{w_{\delta(\mu d+1)+m}})| \le (1+4/s)\ell_0 + (2/s)|pq|.$$

Let $2 \le j \le m'$. We have seen in (4.1) in the proof of Lemma 4.2 that the length of the edge

$$(\mathrm{rep}(\mathrm{cl}(S_{w_{j\delta(\mu d+1)+m}})), \mathrm{rep}(S_{w_{j\delta(\mu d+1)+m}}))$$

satisfies

$$|\mathrm{rep}(\mathrm{cl}(S_{w_{j\delta(\mu d+1)+m}})), \mathrm{rep}(S_{w_{j\delta(\mu d+1)+m}})| \le (1+4/s)\ell_{j\delta(\mu d+1)+m}.$$

Again, let $2 \le j \le m'$. Since

$$\mathrm{rep}(S_{w_{(j-1)\delta(\mu d+1)+m}}) \in S_{w_{j\delta(\mu d+1)+m}} \subseteq S_{x_{j\delta(\mu d+1)+m}}$$

and

$$\mathrm{rep}(\mathrm{cl}(S_{w_{j\delta(\mu d+1)+m}})) \in S_{y_{j\delta(\mu d+1)+m}},$$

it follows from Lemma 2.4 that the length of the edge

$$(\mathrm{rep}(S_{w_{(j-1)\delta(\mu d+1)+m}}), \mathrm{rep}(\mathrm{cl}(S_{w_{j\delta(\mu d+1)+m}})))$$

satisfies

$$|\mathrm{rep}(S_{w_{(j-1)\delta(\mu d+1)+m}}), \mathrm{rep}(\mathrm{cl}(S_{w_{j\delta(\mu d+1)+m}}))| \le (1+4/s)\ell_{j\delta(\mu d+1)+m}.$$

We have shown that the length of Π is at most

$$(2/s)|pq| + 2(1 + 4/s)\left(\ell_0 + \sum_{j=2}^{m'} \ell_{j\delta(\mu d+1)+m}\right).$$

The definition of $\ell_0, \ell_1, \ldots, \ell_k$ implies that this sequence is nondecreasing. Thus, $\ell_0 \leq \ell_{\delta(\mu d+1)+m}$, and it follows that the length of Π is at most

$$(2/s)|pq| + 2(1 + 4/s)\sum_{j=1}^{m'} \ell_{j\delta(\mu d+1)+m}.$$

Thus, it suffices to show that the above expression is at most $(2 + \epsilon/3)|pq|$. In the rest of the proof, we repeatedly apply inequality (4.7) in the proof of Lemma 4.2. As we will see, this allows us to estimate the summation in the above expression by a geometric series which evaluates to $\frac{2^\delta}{2^\delta - 1}\ell_k$. We then apply inequality (4.2) in the proof of Lemma 4.2, which estimates ℓ_k in terms of $|pq|$. By putting all these estimates together, it then follows that the length of Π is at most $(2 + \epsilon/3)|pq|$.

We now present the details. Recall inequality (4.7) in the proof of Lemma 4.2, which states that

$$\ell_i \leq \frac{1}{2}\ell_{i+\mu d+1}.$$

By applying this inequality δ times, we obtain

$$\ell_i \leq \left(\frac{1}{2}\right)^\delta \ell_{i+\delta(\mu d+1)}.$$

For $i = j\delta(\mu d + 1) + m$, this becomes

$$\ell_{j\delta(\mu d+1)+m} \leq \left(\frac{1}{2}\right)^\delta \ell_{(j+1)\delta(\mu d+1)+m}.$$

By repeatedly applying this inequality, we obtain, for $h \geq j$,

$$\ell_{j\delta(\mu d+1)+m} \leq \left(\frac{1}{2}\right)^{(h-j)\delta} \ell_{h\delta(\mu d+1)+m}.$$

For $h = m'$, the latter inequality becomes

$$\ell_{j\delta(\mu d+1)+m} \leq \left(\frac{1}{2}\right)^{(m'-j)\delta} \ell_k.$$

It follows that

$$\begin{aligned}
\sum_{j=1}^{m'} \ell_{j\delta(\mu d+1)+m} &\leq \sum_{j=1}^{m'} \left(\frac{1}{2}\right)^{(m'-j)\delta} \ell_k \\
&= \sum_{i=0}^{m'-1} \left(\frac{1}{2}\right)^{i\delta} \ell_k \\
&\leq \sum_{i=0}^{\infty} \left(\frac{1}{2^\delta}\right)^i \ell_k \\
&= \frac{2^\delta}{2^\delta - 1}\ell_k.
\end{aligned}$$

According to (4.2) in the proof of Lemma 4.2, we have

$$\ell_k \leq (1 + 4/s)|pq|.$$

We have shown that the length of the path Π is at most

$$\left(\frac{2}{s} + 2\left(1 + \frac{4}{s}\right)^2 \frac{2^\delta}{2^\delta - 1}\right)|pq|.$$

Our choices of s and δ (see Algorithm 2) imply that $2/s \leq \epsilon/6$, $(1+4/s)^2 \leq 1+\epsilon/36$, and $\frac{2^\delta}{2^\delta-1} \leq 1+\epsilon/36$. Therefore, the length of Π is at most

$$\left(\epsilon/6 + 2(1+\epsilon/36)^2\right)|pq| \leq (2+\epsilon/3)|pq|,$$

where the latter inequality follows from our assumption that $0 < \epsilon < 1$. This completes the proof. □

LEMMA 5.3. *Let $n = |S|$. The graph G computed by Algorithm 2 is a $(5+\epsilon)$-spanner of the complete k-partite graph $K_{C_1 \ldots C_k}$, and the number of edges of this graph is $O(n)$. The running time of Algorithm 2 is $O(n \log n)$.*

Proof. The proof for the upper bound on the stretch factor is similar to the one of Lemma 4.3. The difference is that instead of the value t' that was used in the proof of Lemma 4.3, we now use the value $t' = 2 + \epsilon/3$ of Lemma 5.2. Thus, the stretch factor for the base case of the induction and for Case 1 is at most

$$(1 + 4/s) + 2t' = 5 + 4/s + 2\epsilon/3,$$

which is at most $5+\epsilon$, because of our choice for s in Algorithm 2. For Cases 2 and 3, the stretch factor is at most (see (4.8) in the proof of Lemma 4.3, where $t = 5+\epsilon$)

$$t' + (1 + 4/s) + 4t/s = 3 + \epsilon/3 + (4/s)(6+\epsilon),$$

which is at most $5+\epsilon$, again because of our choice for s. Finally, the stretch factor for Case 4 is at most (see (4.9) in the proof of Lemma 4.3, where $t = 5+\epsilon$)

$$(1 + 4/s) + 8t/s = 1 + (4/s)(11 + 2\epsilon),$$

which is at most $5+\epsilon$, because of our choice for s.

The analysis for the number of edges is the same as in Lemma 4.1, except that the number of edges that are added to each c-node in the modified for-loop is $2\delta(\mu d + 1)$ instead of 1 as in Algorithm 1. Finally, the analysis of the running time is the same as in Lemma 4.4. □

We have proved the following result.

THEOREM 5.4. *Let $k \geq 2$ be an integer, let S be a set of n points in $\mathbb{R}^d$ which is partitioned into k subsets $C_1, C_2, \ldots, C_k$, and let $0 < \epsilon < 1$ be a real constant. In $O(n \log n)$ time, we can compute a $(5+\epsilon)$-spanner of the complete k-partite graph $K_{C_1 \ldots C_k}$ having $O(n)$ edges.*

6. Improving the stretch factor. We have shown how to compute a $(5+\epsilon)$-spanner with $O(n)$ edges of any complete k-partite graph. In this section, we show that if we are willing to use $O(n \log n)$ edges, the stretch factor can be reduced to $3+\epsilon$. We start by showing that a stretch factor less than 3, while using $O(n \log n)$ edges, is not possible.

THEOREM 6.1. *Let $c > 0$ be a constant and let n and k be positive integers with $2 \leq k \leq n - 2c\sqrt{n \log n}$. For every real number $0 < \epsilon < 1$, there exists a complete k-partite geometric graph K with n vertices such that the following is true: If G is any subgraph of K with at most $c^2 n \log n$ edges, then the stretch factor of G is at least $3 - \epsilon$.*

Proof. Let D_1, D_2, and D_3 be three disks of radius $\epsilon/12$ centered at the points $(0, 0)$, $(1 + \epsilon/6, 0)$, and $(2 + \epsilon/3, 0)$, respectively. We place $(n - k + 2)/2$ red points inside D_1 and $(n - k + 2)/2$ blue points inside D_2. The remaining $k - 2$ points are placed inside D_3, and each of these points has a distinct color (which is neither red nor blue). Let K be the complete k-partite geometric graph defined by these n points. We claim that K satisfies the claim in the theorem.

Let G be an arbitrary subgraph of K and assume that G contains at most $c^2 n \log n$ edges. We will show that the stretch factor of G is at least $3 - \epsilon$.

Assume that G contains all red-blue edges. Then the number of edges in G is at least $((n-k+2)/2)^2$. Since $k \leq n - 2c\sqrt{n \log n}$, this quantity is larger than $c^2 n \log n$. Thus, there is a red point r and a blue point b, such that (r, b) is not an edge in G. The length of a shortest path in G between r and b is at least 3. Since $|rb| \leq 1 + \epsilon/3$, it follows that the stretch factor of G is at least $\frac{3}{1+\epsilon/3}$, which is at least $3 - \epsilon$. ∎

THEOREM 6.2. *Let $k \geq 2$ be an integer, let S be a set of n points in $\mathbb{R}^d$ which is partitioned into k subsets $C_1, C_2, \ldots, C_k$, and let $0 < \epsilon < 1$ be a real constant. In $O(n \log n)$ time, we can compute a $(3 + \epsilon)$-spanner of the complete k-partite graph $K_{C_1 \ldots C_k}$ having $O(n \log n)$ edges.*

Proof. Consider the following variant of the WSPD. For every pair $\{X, Y\}$ in the standard WSPD, where $|X| \leq |Y|$, we replace this pair by the $|X|$ pairs $\{\{x\}, Y\}$, where x ranges over all points of X. Thus, in this new WSPD, each pair contains at least one singleton set. Callahan and Kosaraju [4] showed that this new WSPD consists of $O(n \log n)$ pairs.

We run Algorithm 2 on the set S, using this new WSPD. Let G be the graph that is computed by this algorithm. Observe that Lemma 5.2 still holds for G. In the proof of Lemma 5.3 of the upper bound on the stretch factor of G, we have to apply Lemma 5.2 only once. Therefore, the stretch factor of G is at most $3 + \epsilon$. ∎

7. Conclusion. We have shown that, for every complete k-partite geometric graph K in $\mathbb{R}^d$ with n vertices and for every constant $\epsilon > 0$,

1. a $(5+\epsilon)$-spanner of K having $O(n)$ edges can be computed in $O(n \log n)$ time,
2. a $(3+\epsilon)$-spanner of K having $O(n \log n)$ edges can be computed in $O(n \log n)$ time.

The latter result is optimal for $2 \leq k \leq n - \Theta(\sqrt{n \log n})$, because a spanner of K having stretch factor less than 3 and having $O(n \log n)$ edges does not exist for all complete k-partite geometric graphs.

We leave open the problem of determining the best stretch factor that can be obtained by using $O(n)$ edges.

Future work may include verifying other properties that are known for the general geometric spanner problem. For example, is there a spanner of a complete k-partite geometric graph that has bounded degree? Is there a spanner of a complete k-partite geometric graph that is planar? From a more general point of view, it seems that little is known about geometric spanners of graphs other than the complete graph. The unit disk graph has received much attention, but there is a large family of other graphs that also deserve attention.

REFERENCES

[1] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[2] P. Bose, P. Carmi, M. Couture, A. Maheshwari, M. Smid, and N. Zeh, *Geometric spanners with small chromatic number*, in Proceedings of the 5th Workshop on Approximation and Online Algorithms (WAOA 2007), Lecture Notes in Comput. Sci. 4927, Springer-Verlag, Berlin, 2008, pp. 75–88.

[3] P. B. Callahan and S. R. Kosaraju, *Faster algorithms for some geometric graph problems in higher dimensions*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 291–300.

[4] P. B. Callahan and S. R. Kosaraju, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.

[5] J. Gudmundsson and M. Smid, *On spanners of geometric graphs*, in Proceedings of the 10th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 4059, Springer-Verlag, Berlin, 2006, pp. 388–399.

[6] G. Narasimhan and M. Smid, *Geometric Spanner Networks*, Cambridge University Press, New York, 2007.

[7] B. Raman and K. Chebrolu, *Design and evaluation of a new MAC protocol for long-distance* 802.11 *mesh networks*, in Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MobiCom 2005), ACM Press, New York, 2005, pp. 156–169.

[8] J. S. Salowe, *Constructing multidimensional spanner graphs*, Internat. J. Comput. Geom. Appl., 1 (1991), pp. 99–107.

[9] P. M. Vaidya, *A sparse graph almost as good as the complete graph on points in k dimensions*, Discrete Comput. Geom., 6 (1991), pp. 369–381.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1821–1880

PROFILES OF TRIES*

GAHYUN PARK†, HSIEN-KUEI HWANG‡, PIERRE NICODÈME§, AND WOJCIECH SZPANKOWSKI¶

Abstract. Tries (from re*trie*val) are one of the most popular data structures on words. They are pertinent to the (internal) structure of stored words and several splitting procedures used in diverse contexts. The *profile* of a trie is a parameter that represents the number of nodes (either internal or external) with the same distance from the root. It is a function of the number of strings stored in a trie *and* the distance from the root. Several, if not all, trie parameters such as height, size, depth, shortest path, and fill-up level can be uniformly analyzed through the (external and internal) profiles. Although profiles represent one of the most fundamental parameters of tries, they have hardly been studied in the past. The analysis of profiles is surprisingly arduous, but once it is carried out it reveals unusually intriguing and interesting behavior. We present a detailed study of the distribution of the profiles in a trie built over random strings generated by a memoryless source. We first derive recurrences satisfied by the expected profiles and solve them asymptotically for all possible ranges of the distance from the root. It appears that profiles of tries exhibit several fascinating phenomena. When moving from the root to the leaves of a trie, the growth of the expected profiles varies. Near the root, the external profiles tend to zero at an exponential rate, and then the rate gradually rises to being logarithmic; the external profiles then abruptly tend to infinity, first logarithmically and then polynomially; they then tend polynomially to zero again. Furthermore, the expected profiles of asymmetric tries are oscillating in a range where profiles grow polynomially, while symmetric tries are nonoscillating, in contrast to most shape parameters of random tries studied previously. Such a periodic behavior for asymmetric tries implies that the depth satisfies a central limit theorem but not a local limit theorem of the usual form. Also the widest levels in symmetric tries contain a linear number of nodes, differing from the order $n/\sqrt{\log n}$ for asymmetric tries, n being the size of the trees. Finally, it is observed that profiles satisfy central limit theorems when the variance goes unbounded, while near the height they are distributed according to Poisson laws. As a consequence of these results we find typical behaviors of the height, shortest path, fill-up level, and depth. These results are derived here by methods of analytic algorithmics such as generating functions, Mellin transform, Poissonization and de-Poissonization, the saddle-point method, singularity analysis, and uniform asymptotic analysis.

Key words. digital trees, tries, profile, depth, height, shortest path, fill-up level, analytic Poissonization, Mellin transform, saddle-point method, singularity analysis

AMS subject classifications. Primary, 68W40; Secondary, 68P5, 68P10, 05C05, 60F05

DOI. 10.1137/070685531

1. Introduction.

Tries are prototype data structures useful for many indexing and retrieval purposes. They were first proposed by de la Briandais [9] in the late 1950s for information processing; Fredkin [27] suggested the current name as it is

*Received by the editors March 19, 2007; accepted for publication (in revised form) August 19, 2008; published electronically January 9, 2009.
http://www.siam.org/journals/sicomp/38-5/68553.html
†Department of Computer Science, State University of New York at Geneseo, Geneseo, NY 14454 (park@geneseo.edu).
‡Institute of Statistical Science, Academia Sinica, 11529 Taipei, Taiwan (hkhwang@stat.sinica.edu.tw). This author's work was partially supported by a grant from the National Science Council of Taiwan.
§Laboratory LIX, École Polytechnique, 91128 Palaiseau Cedex, France (nicodeme@lix.polytechnique.fr).
¶Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, IN 47907-2066 (spa@cs.purdue.edu). This author's work was supported in part by NSF grants CCF-0513636, DMS-0503742, DMS-0800568, and CCF-0830140; NIH grant R01 GM068959-01; NSA grant H98230-08-1-0092; and AFOSR grant FA8655-08-1-3018.

part of re*trie*val. Tries are multiway trees whose nodes are vectors of characters or digits. Due to their simplicity and efficiency, tries found widespread use in diverse applications including document taxonomy, IP address lookup, data compression, dynamic hashing, partial-match queries, speech recognition, leader election algorithms, and distributed hashing tables (see [29, 50, 53, 79]). In this paper, we are concerned with probabilistic properties of the profiles of tries, where the *profile* of a tree is the sequence of numbers each corresponding to the number of nodes with the same distance from the root. We discover several new phenomena in the profiles of tries built over strings generated by a random memoryless source, and develop asymptotic tools to describe them.

Structure and usefulness of tries. Tries are a natural choice of data structure when the input records involve a notion of alphabets or digits. They are often used to store such data so that future retrieval can be made efficient. Given a sequence of n words over the alphabet $\{a_1, \ldots, a_m\}$, $m \geq 2$, we can construct a trie as follows. If $n = 0$, then the trie is empty. If $n = 1$, then a single (external) node holding the word is allocated. If $n \geq 1$, then the trie consists of a root (internal) node directing words to the m subtrees according to the first alphabet of each word, and words directed to the same subtree are themselves tries (see [50, 53, 79] for more details). For simplicity, we deal only with binary tries in this paper. Unlike other search trees such as digital search trees and binary search trees where records or keys are stored at the internal nodes, the *internal nodes* in tries are branching nodes used merely to direct records to each subtrie, with all records stored in *external nodes* that are leaves of such tries. A trie has more internal nodes than external nodes (fixed to be n throughout this paper), differing from almost all other search trees. In Figure 1 we plot a binary trie of five strings.

The simple organizing procedure used to construct tries and the general efficiency they achieve make tries one of the most popular digital search trees. Since their invention, tries have found frequent use in many computer science applications. For example, tries are widely used in algorithms for automatically correcting words in texts (see [51]) and in algorithms for taxonomies and toolkits of regular language (see the Ph.D. thesis [80]); they are also used to represent the event history in datarace detection for multithreaded object-oriented programs (see [6]); another example is the Internet IP address lookup problem (see [60, 74]), where the search time for the IP address problem is directly related to the distribution of the fill-up level (see below for a more precise definition) and other trie parameters. For applications to other problems in searching, sorting, dynamic hashing, coding, polynomial factorization, Lempel–Ziv compression schemes, and molecular biology, see [29, 79].

The structure of tries also has a close connection to several splitting procedures using coin-flipping; these include algorithms for resolving collisions in multiaccess (or broadcast) communication models, algorithms for loser selection or leader election, etc.; see [45]. Thus most shape parameters in tries have direct interpretations in terms of other related objects.

Random tries under the Bernoulli model. Throughout the paper, we write $B_{n,k}$ to denote the number of external nodes (leaves) at distance k from the root; the number of internal nodes at distance k from the root is denoted by $I_{n,k}$. For simplicity, we will refer to $B_{n,k}$ as the *external profile* and $I_{n,k}$ as the *internal profile*. Figure 1 shows a trie and its profiles.

In this paper we study the profiles of a trie built over n binary strings generated by a memoryless source. More precisely, we assume that the input is a sequence of n independent and identically distributed random variables, each being composed of an

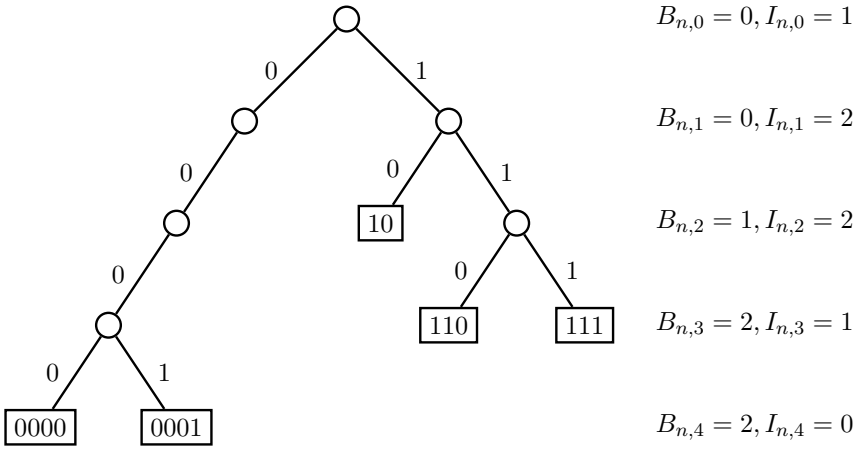


FIG. 1. *A trie of $n = 5$ records and its profiles: the circles represent internal nodes, and rectangles holding the records are external nodes.*

infinite sequence of Bernoulli random variables with mean p, where $0 < p < 1$ is the probability of a "1" and $q := 1 - p$ is the probability of a "0." The corresponding trie constructed from these n binary strings is called a *random trie*. This simple model may seem too idealized for practical purposes; however, the typical behaviors under such a model often hold under more general models such as Markovian or dynamical sources, although the technicalities are usually more involved (see, for example, [8, 12, 15, 35]).

The motivation of studying the profiles is multifold. First, they are fine shape measures closely connected to many other cost measures on tries; some of them are indicated below. Second, they are also asymptotically close to the profiles of suffix trees, which in turn have a direct combinatorial interpretation in terms of words; see [36, 59, 78, 79] for more information and another interpretation in terms of urn models. Third, not only are the analytic problems mathematically challenging, but the diverse new phenomena they exhibit are highly interesting and unusual. Fourth, our findings imply several new results on other shape parameters (see section 8). Finally, most properties of random tries have also a prototype character and are expected to hold for other varieties of digital search trees (and under more general random models), although the proofs are generally more complicated.

Major cost measures on random tries. Due to the usefulness of tries, many cost measures, discussed below, on random tries have been studied in the literature since the early 1970s, and most of these measures can be expressed and analyzed through the profiles studied in this paper:

- *depth*: the distance from the root to a randomly selected node; its distribution is given by the expected external profile divided by n; see [10, 12, 13, 20, 24, 33, 36, 42, 46, 52, 65, 69, 71, 75, 76];
- *total path length*: the sum of distances between nodes and the root, or, equivalently, $\sum_j jI_{n,j}$; see [8, 11, 24, 44, 58, 57, 70, 71, 72, 73, 75];
- *size*: the total number of internal nodes, or $\sum_j I_{n,j}$; see [8, 24, 34, 36, 40, 41, 43, 47, 50, 57, 67, 68, 69, 70, 71, 72, 73];
- *height*: the length of the longest path from the root, or $\max\{j : B_{n,j} > 0\}$; see [8, 11, 12, 13, 14, 22, 26, 33, 48, 64, 65, 77];
- *shortest path*: the length of the shortest path from the root to an external node, or $\min\{j : B_{n,j} > 0\}$; see [64, 65];

- *fill-up (or saturation) level*: the largest full level, or $\max\{j : I_{n,j} = 2^j\}$, where the *levels* of a tree denote the sets of nodes with the same distance from the root; see [49];
- *Horton–Strahler number and stack-size*: certain notions of heights related to the traversal of tries; see [4, 17, 54, 55, 56];
- *distance of two randomly chosen nodes*; see [1, 7];
- *pattern occurrences in tries (including page usage or b-tries)*; see [22, 42, 43, 58, 71, 76];
- *one-sided height (or leader election or loser selection)*; see [21, 39, 66, 81, 82].

The reader is referred to the book [79] and the papers [15, 37, 71] for a systematic treatment of several of these quantities.

The general analytic context. The major difference between most previous study and the current paper is that we are dealing with the asymptotics of bivariate recurrence, in contrast to univariate recurrences (with or without maximization or minimization) addressed in the literature.

To be more precise, we observe that, by assumption of the model, the probability generating function $P_{n,k}(y) := \mathbb{E}(y^{B_{n,k}})$ of the external profile satisfies the recurrence

$$P_{n,k}(y) = \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j} P_{j,k-1}(y) P_{n-j,k-1}(y) \qquad (n \ge 2;\ k \ge 1) \tag{1}$$

with the initial conditions $P_{n,k}(y) = 1 + \delta_{n,1}\delta_{k,0}(y-1)$ when either $n \le 1$ and $k \ge 0$ or $k = 0$ and $n \ge 0$, where $\delta_{a,b}$ is the Kronecker symbol. Observe that this recurrence depends on two parameters n and k, which makes the analysis quite challenging, as we will demonstrate in this paper. The probability generating functions of the internal profile satisfy the same recurrence (1) but with different initial conditions; see section 6.

From (1), the moments of $B_{n,k}$ and $I_{n,k}$ (centered or not) are seen to satisfy a recurrence of the form

$$x_{n,k} = a_{n,k} + \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j} \left(x_{j,k-1} + x_{n-j,k-1}\right)$$

with suitable initial conditions, where $a_{n,k}$ are known (either explicitly or inductively). A standard approach is to consider the Poisson generating function $\tilde{f}_k(z) := e^{-z}\sum_n x_{n,k} z^n/n!$, which in turn satisfies the functional equation

$$\tilde{f}_k(z) = \tilde{g}_k(z) + \tilde{f}_{k-1}(pz) + \tilde{f}_{k-1}(qz)$$

with a suitable $\tilde{g}_k(z)$. This equation can be solved explicitly by a simple iteration argument and asymptotically by using the Mellin transform (see [23, 79]). The final step is to invert from the asymptotics of the Poisson generating function $\tilde{f}_k(z)$ to recover the asymptotics of $x_{n,k}$. This last step is guided by the *Poisson heuristic*, which roughly states that

(2) *if a sequence $\{x_n\}_n$ is "smooth enough," then $x_n \sim e^{-n}\sum_{j\ge 0} x_j n^j/j!$,*

where $x_n \sim y_n$ if $\lim_{n\to\infty} x_n/y_n = 1$. Such a Poisson heuristic has appeared in diverse contexts under different forms such as Borel summability and Tauberian theorems; it dates back at least to Ramanujan's Notebooks; see the book by Berndt [3, pp. 57–66] for more details. It is known as *analytic de-Poissonization* when justified

by complex analysis and the saddle-point method, and was the subject of intensive analysis, resulting in a robust solution presented in [37].

By means of the Poisson heuristic (2), we expect that $\mu_{n,k} \sim e^{-n}\sum_{j\geq 0}\mu_{j,k}n^j/j!$. However, as we will see, such a heuristic holds in our case when $q^{2k}n \to 0$ but fails otherwise. The reason is that $\mu_{n,k}$ is too small in this range. Also it should be mentioned that the asymptotic analysis of the above functional equation is in general more intricate because we have an additional parameter k to be taken into account and we need uniformity for our asymptotic approximations in k (varying with n) and in z (in some region in the complex plane) in order to invert the results to obtain $x_{n,k}$ by suitable complex analysis.

Known results for profiles. As far as probabilistic properties of the profiles of random tries are concerned, very little is known in the literature. Since the distribution of the depth D_n in random tries is given by $\mathbb{P}(D_n = k) = \mu_{n,k}/n$, where $\mu_{n,k} := \mathbb{E}(B_{n,k})$, the asymptotics of the expected profile $\mu_{n,k}$ for $n \to \infty$ and varying $k = k(n)$ can be regarded as local limit theorems for D_n. Although many papers have addressed the limiting behaviors of the depth, none has dealt with the local limit theorem of D_n and the asymptotics of $\mu_{n,k}$ for varying k. We will see in section 8 that our result implies an unusual type of local limit theorem for D_n. However, it should be mentioned that the central limit theorem for the depth was developed in [13, 34, 35].

On the other hand, Pittel [65] showed that the distribution of the number of pairs of input-strings having a common prefix of length at least k is asymptotically Poisson when k is close to the height. Devroye [14] showed that

$$\text{if} \quad \frac{\mathbb{E}(B_{n,k})}{\sqrt{n}} \to \infty, \quad \text{then} \quad \frac{B_{n,k}}{\mathbb{E}(B_{n,k})} \to 1 \quad \text{in probability;}$$

$$\text{if} \quad \mathbb{E}(I_{n,k}) \to \infty, \quad \text{then} \quad \frac{I_{n,k}}{\mathbb{E}(I_{n,k})} \to 1 \quad \text{in probability,}$$

under very general assumptions on the underlying models (see also [15] for further refinements). References [65] and [14] represent known results concerning profiles. We will see that convergence in probability in the two "if statements" holds as long as the variance tends to infinity.

Sketch of the major phenomena. In the next section we present an in-depth discussion of our results. Here, we briefly summarize our main findings. We focus mostly on the profiles of asymmetric tries (when $p \neq q$) since the symmetric tries (when $p = q = 1/2$) are comparatively easier. We will first derive asymptotic approximations to the average external profile $\mu_{n,k}$ for all ranges of k.

Our results show inter alia that for $k \leq (1-\varepsilon)\log n/\log(1/q)$ the average profile $\mu_{n,k}$ is exponentially small where $\varepsilon > 0$ is small. When k increases and lies in the range $(\log n - \log\log\log n + O(1))/\log(1/q)$, then $\mu_{n,k}$ decays to zero logarithmically until $k > k^*$ for a specific threshold k^* in this range beyond which $\mu_{n,k}$ suddenly grows unbounded at a logarithmic rate. The rate becomes polynomial $\Theta(n^\upsilon)$ for some $0 < \upsilon \leq 1$ when

$$\frac{1}{\log(1/q)}(1+\varepsilon)\log n \leq k \leq \frac{2}{\log(1/(p^2+q^2))}(1-\varepsilon)\log n.$$

Surprisingly enough, for this range of k an oscillating factor emerges in the expected profile behavior; that is, $\mathbb{E}(B_{n,k}) \approx G(\log_{p/q} p^k n)n^v/\sqrt{\log n}$, where G is a bounded periodic function. Such behavior is a consequence of an infinite number of saddle-points appearing in the integrand of the associated Mellin integral transform. This

was first observed by Nicodème [59]. For larger values of k, these oscillations disappear since the behavior of the expected profile is dominated by a polar singularity.

Analogous results also hold for the internal profile. In addition, we prove that the variances of both profiles are asymptotically of the same order as their expected values. This suggests a central limit theorem for both external and internal profiles for a wide range of k. We show that this is indeed true; furthermore, we also show that for k near the height the limiting distribution of the profiles becomes Poisson. Some of these results were already anticipated in [63] and constitute the Ph.D. thesis of the first author [62].

Profiles of digital and nondigital log trees. In passing, we observe that most random trees in the discrete probability literature fall into two major categories according to their expected height being of order $\sqrt{n}$ (referred to as *square-root trees* for brevity) or of order $\log n$ (referred to as *log trees*), where n is the tree size. While most random square-root trees were introduced in combinatorics and probability, the majority of log trees arise from data structures and computer algorithms.

We can further classify log trees into "digital-type" and "nondigital-type" log trees, according to the nature of construction (or search) of the tree. Profiles of nondigital-type search trees of logarithmic height for which *binary search trees* are representative have received much recent attention and are shown to exhibit several interesting phenomena such as bimodality of the variance and multifaceted behaviors of the limiting distributions; see [5, 18, 19, 28, 31] for more information. In contrast, profiles of digital-type search trees have not been addressed as much, and most properties remain unknown; see [14, 15, 65] for tries and [2, 38] for digital search trees. We will show that the limiting behaviors of the profiles are very different from those of nondigital search trees. In particular, while in no range will the normalized profiles in random binary search trees lead to asymptotic normality (in the sense of convergence in distribution), profiles of random tries, when properly centered and normalized, all converge to the standard normal law when the variance goes unbounded in the limit. As is often the case for proving asymptotic normality, we need more precise asymptotic approximation to the variance, rendering our analysis more complicated.

Organization of the paper. The paper is organized as follows. In the next section, we present (rather informally) a more detailed summary of our main findings. This section is to help the reader to comprehend the richness of our results in their fullness but without resorting to rather abstruse mathematical formulations. Sections 3–8 are devoted to precise formulations of our results. This paper contains two major parts: The first, section 3, develops the asymptotic tools we need for deriving the diverse asymptotic approximations to the expected external profile $\mu_{n,k}$. Most proofs of the second part (sections 4–8) are then sketched because they extend the same methods of proof as in the first part. Except for sections 7 and 8, we assume $p \neq q$ throughout this paper. Among these sections, section 4 derives the asymptotics of the variance of $B_{n,k}$, the corresponding results of convergence in distribution being given in section 5. The internal profiles are addressed in section 6, and results for symmetric tries are given in section 7. Consequences of our findings are discussed in section 8, where we establish typical behaviors of the height, the width, the shortest path, the fill-up level, and the right-profile, as well as a rather atypical local limit theorem for the depth.

2. Summary of main results. In this section we discuss informally our main results. We focus here on describing the major phenomena arising in the analysis of

profiles rather than presenting the precise and complicated results to which we devote the remaining sections of this paper.

Crucial to our analysis of the profiles is the asymptotics of the expected profiles. Not only are the results fundamental and highly interesting, but the analytic methods we used are also of certain generality.

From (1), we see that the expected external profile $\mu_{n,k} := \mathbb{E}(B_{n,k})$ satisfies the following recurrence:

$$\mu_{n,k} = \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j} (\mu_{j,k-1} + \mu_{n-j,k-1}) \tag{3}$$

for $n \ge 2$ and $k \ge 1$ with the initial values $\mu_{n,0} = 0$ for all $n \ne 1$ and 1 for $n = 1$. Furthermore, $\mu_{0,k} = 0,\ k \ge 0$, and $\mu_{1,k} = 0$ for $k \ge 1$ and is equal to 1 when $k = 0$. Throughout we assume that $p > q = 1 - p$ unless stated otherwise.

The polynomial growth of $\mu_{n,k}$. In section 3, we solve asymptotically (3) for various ranges of k when $p \ne q$; a crude description of the asymptotics of $\mu_{n,k}$ is as follows:

$$\frac{\log \mu_{n,k}}{\log n} \to \begin{cases} 0 & \text{if } \alpha \le \alpha_1, \\ -\rho + \alpha \log(p^{-\rho} + q^{-\rho}) & \text{if } \alpha_1 \le \alpha \le \alpha_2, \\ 2 + \alpha \log(p^2 + q^2) & \text{if } \alpha_2 \le \alpha \le \alpha_3, \\ 0 & \text{if } \alpha \ge \alpha_3, \end{cases} \tag{4}$$

where

$$\alpha_1 := \frac{1}{\log(1/q)}, \quad \alpha_2 := \frac{p^2 + q^2}{p^2 \log(1/p) + q^2 \log(1/q)}, \quad \text{and} \quad \alpha_3 := \frac{2}{\log(1/(p^2 + q^2))} \tag{5}$$

are delimiters of $\alpha := \lim_n k/\log n$ $(k = k(n))$, and

$$\rho := \frac{1}{\log(p/q)} \log\left(\frac{1 - \alpha \log(1/p)}{\alpha \log(1/q) - 1}\right).$$

Note that $\alpha_1 \le \alpha_2$; see Figure 2. The limiting estimate (4) gives a rough picture of $\mu_{n,k}$ as follows: $\mu_{n,k}$ is of polynomial growth rate when $\alpha_1 + \varepsilon \le \alpha \le \alpha_3 - \varepsilon$ and is smaller than any polynomial powers when $0 \le \alpha \le \alpha_1 - \varepsilon$ and $\alpha \ge \alpha_3 + \varepsilon$. Near the two boundaries α_1 and α_3, the behaviors of $\mu_{n,k}$ will undergo phase changes from being subpolynomial to being polynomial or the other way around.

More refined asymptotics. To derive more precise asymptotics of $\mu_{n,k}$ than the phase transitions (4) of the polynomial order of $\mu_{n,k}$, we divide all possible values of k into four overlapping ranges.

(I) *Elementary range*: $1 \le k \le \alpha_1(\log n - \log\log\log n + O(1))$.
(II) *Saddle-point range*: $\alpha_1(\log n - \log\log\log n + K_n) \le k \le \alpha_2(\log n - K_n\sqrt{\log n})$.
(III) *Gaussian transitional range*: $k = \alpha_2 \log n + o((\log n)^{2/3})$.
(IV) *Polar singularity range*: $k \ge \alpha_2 \log n + K_n\sqrt{\log n}$,

where, throughout this paper, $K_n \ge 1$ represents a (generic) sequence tending to infinity.

More precisely, in Theorem 1 we prove that for k lying in range (I) the expected external profile $\mu_{n,k}$ first decays exponentially fast (asymptotic to $q^k n(1 - q^k)^{n-1}$).

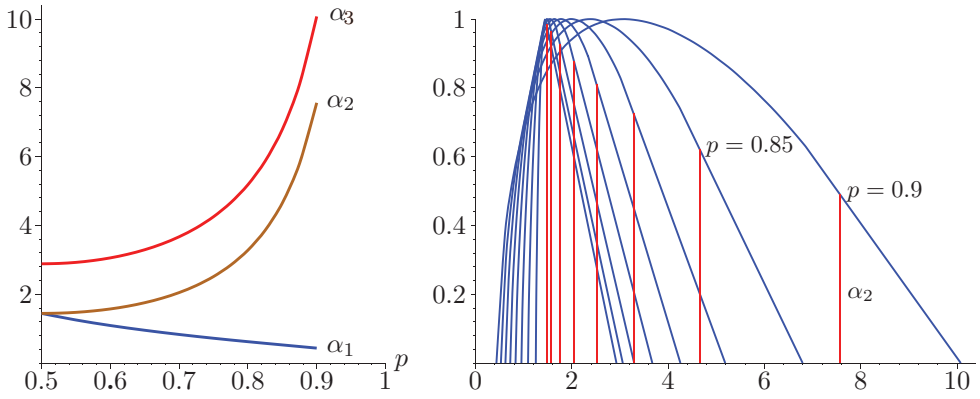


FIG. 2. *Left: a plot of α_1, α_2, and α_3 (defined in (5)) as functions of p. Right: the (nonzero) limiting order of $\log \mu_{n,k}/\log n$ plotted against $\alpha = \lim_n k/\log n$ for $p = 0.55, 0.6, \ldots, 0.9$ (the spans of the curves increase as p grows). The vertical lines represent the positions of α_2 (to the right of which the curves are straight lines); see (4).*

Then, when k is around $\alpha_1(\log n - \log\log\log n + \log(p/q - 1) + m\log(p/q))$ for some integer $m \geq 0$,

$$\mu_{n,k} \sim \frac{k^m}{m!} p^m q^{k-m} n e^{-np^m q^{k-m}},$$

which is of order

$$\mu_{n,k} = O\left(\frac{\log\log n}{\log^{\xi-m} n}\right)$$

for some ξ. Thus, for $m < \xi$ the expected external profile decays only logarithmically, but for $m \geq \xi$ it increases logarithmically.

The behavior of $\mu_{n,k}$ in range (II) is described in Theorem 2. The situation becomes highly nontrivial and interesting. More precisely, for $\alpha_1(1+\varepsilon)\log n \leq k \leq \alpha_2(1-\varepsilon)\log n$, we find that

$$\mu_{n,k} \sim G_1\left(\rho; \log_{p/q} p^k n\right) \frac{p^\rho q^\rho (p^{-\rho} + q^{-\rho})}{\sqrt{2\pi\alpha_{n,k}}\,\log(p/q)} \cdot \frac{n^{\upsilon_1}}{\sqrt{\log n}},$$

where ($\alpha_{n,k} := k/\log n$)

$$\upsilon_1 = -\rho + \alpha_{n,k}\log(p^{-\rho} + q^{-\rho}),$$
$$\rho = -\frac{1}{\log(p/q)} \log\left(\frac{-1 - \alpha_{n,k}\log q}{1 + \alpha_{n,k}\log p}\right),$$

and $G_1(\rho; x)$ is a periodic function. We plot in Figures 3 and 4 the periodic parts of $G_1(-1, x)$ for a few values of p and ρ, respectively. *Analytically*, these oscillations are consequences of an infinite number of saddle-points appearing in the integrand of the associated Mellin transform of the expected profile, but *visually* they look like certain sine waves due to the fact that the corresponding Fourier expansions involve a

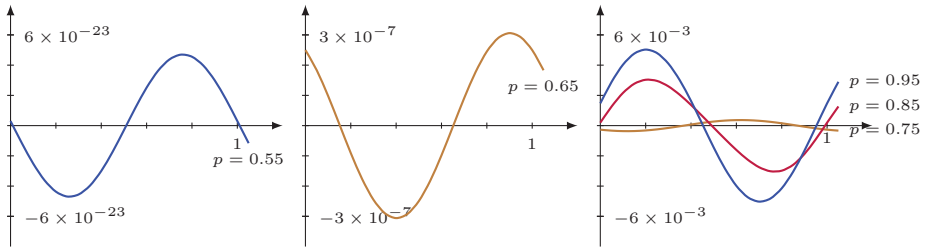


FIG. 3. *The fluctuating part around the mean of the periodic function* $G_1(-1;x)$ *for* $p = 0.55, 0.65, \ldots, 0.95$ *and for* x *in the unit interval; its amplitude tends to zero when* $p \to 0.5^+$.

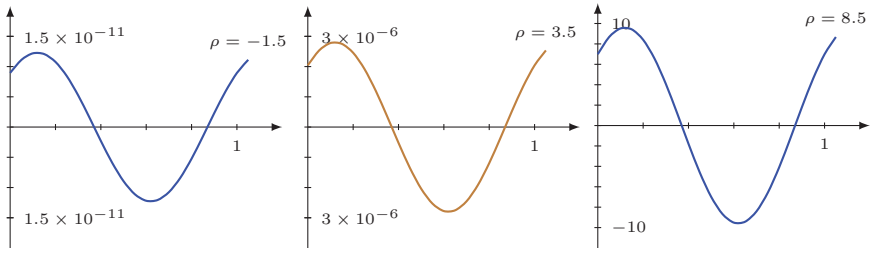


FIG. 4. *The fluctuating part around the mean of the periodic function* $G_1(\rho;x)$ *for* $\rho \in \{-1.5, 3.5, 8.5\}$ *and* $x \in [0,1]$. *The amplitude increases as* ρ *grows.*

Gamma function with increasing parameters, which decreases very fast along a fixed vertical line for an increasing imaginary part, so that only a few terms dominate.

Finally, in Theorem 3 we prove that for k in range (IV)

$$\mu_{n,k} \sim 2pqn^2(p^2+q^2)^{k-1} = \frac{2pq}{p^2+q^2}\, n^{\upsilon_2},$$

where $\upsilon_2 = 2 + \alpha_{n,k}\log(p^2+q^2)$, and the periodic function disappears. In this region, the asymptotic behavior of the expected profile is dictated by the expected number of pairs (of input-strings) having common prefixes of length at least k. This property is analytically reflected by a polar singularity in the associated Mellin transform. The asymptotics of $\mu_{n,k}$ in range (III) for $k = \alpha_2 \log n + o(\log^{2/3} n)$ are presented in Theorem 4. In this transitional range, the saddle-point coalesces with the polar singularity, so we use the Gaussian integral to describe the behavior of $\mu_{n,k}$.

In summary, our results roughly state that $\mu_{n,k} \to 0$ when $1 \le k \le k^*$ for some k^* close to $\alpha_1(\log n - \log\log\log n + O(1))$, and then $\mu_{n,k}$ tends *abruptly* to infinity at a logarithmic rate when $k > k^*$. Such an abrupt change has already been observed in the literature for the shortest path and the fill-up level (see [49, 65]), but not much is known for $\mu_{n,k}$ beyond that. Then we show that $\mu_{n,k}$ grows polynomially when k lies in the range $\alpha_1(1+\varepsilon)\log n \le k \le \alpha_3(1-\varepsilon)\log n$, reaching the peak where it is of order $n/\sqrt{\log n}$; it decays at a slower rate afterwards until it tends to zero again when $k \ge \alpha_3(\log n + K_n)$. A salient feature here is the presence of an oscillating function in the asymptotic approximation when $p \ne q$.[1] In Figure 5, a plot of the rough silhouettes of $\mu_{n,k}$ is presented.

[1] The expected values of many shape characteristics of random tries often exhibit the asymptotic pattern: $\sim F(\log_c n)n$ if $\log p/\log q$ is rational for some periodic function F and constant c expressible in terms of p, and $\sim Cn$ if $\log p/\log q$ is irrational; see [37, 71, 79].

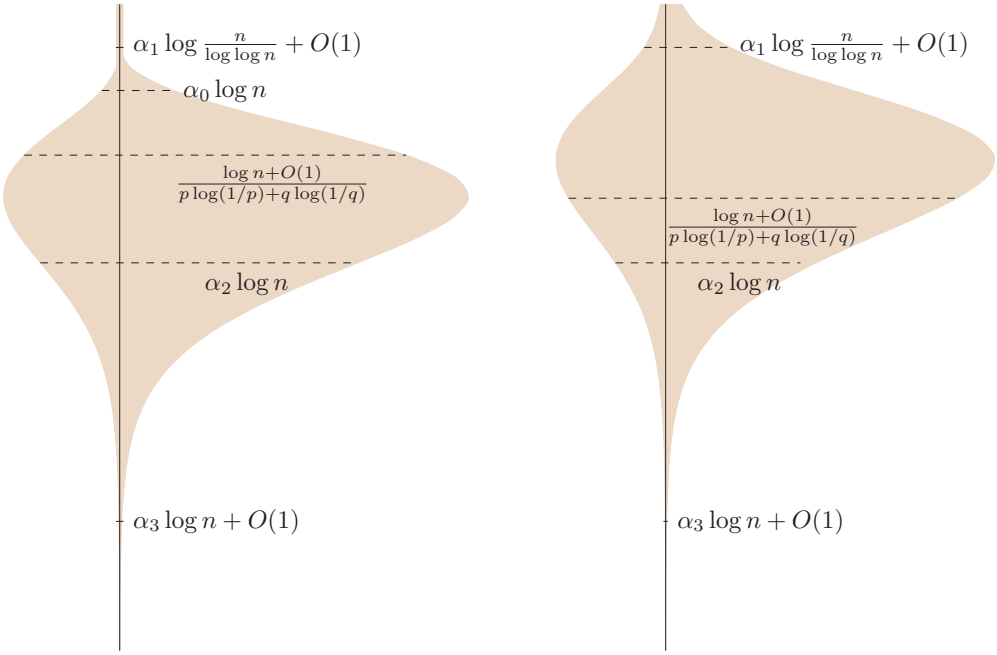


FIG. 5. *The silhouettes of the expected external (left) and internal (right) profiles of an asymmetric trie ($p = 0.75$). Note that the right subtrees of the asymmetric trie have more nodes than their left siblings since $p > 1/2$. Also, the first few levels contain almost no external nodes but are almost full of internal nodes.*

Asymptotics of the expected internal profile. The expected value of the internal profile $\mathbb{E}(I_{n,k})$ is discussed in section 6. In particular, the expected internal profile is asymptotically equivalent to 2^k for $k \le \alpha_0(\log n - K_n\sqrt{\log n})$, where $\alpha_0 := 2/(\log(1/p) + \log(1/q))$. When $k \ge \alpha_2(\log n + K_n\sqrt{\log n})$, then $\mathbb{E}(I_{n,k}) \sim (p^2+q^2)\mathbb{E}(B_{n,k})/pq$. Between these two ranges, it is again the infinite number of saddle-points that yield the dominant asymptotic approximation. Unlike $\mu_{n,k}$, an additional phase transition appears in the asymptotics of the $\mathbb{E}(I_{n,k})$ when $k = \alpha_0 \log n + O(\sqrt{\log n})$, reflecting the structural change of the internal nodes from being asymptotically full to being of the same order as the number of external nodes. The silhouettes of the expected internal profiles for a symmetric trie and an asymmetric ($p = 0.75$) trie are presented in Figure 6.

Variance and limiting distributions. In section 4 we deal with the variance of the profile. In particular, in Theorem 7 we derive asymptotic approximations to the variance of the profile, which asymptotically turns out to be of the same order as the expected value for all ranges of $k \ge 1$; namely, $\mathbb{V}(B_{n,k}) = \Theta(\mathbb{E}(B_{n,k}))$. In fact, we show that $\mathbb{V}(B_{n,k}) \sim \mathbb{E}(B_{n,k})$ in range (I), for range (IV) $\mathbb{V}(B_{n,k}) \sim 2\mathbb{E}(B_{n,k})$, and in range (II) (polynomial growth) the variance and the expected profile differ only by the oscillating functions. The variance of the internal profile behaves almost identically to the variance of the external profile; roughly, $\mathbb{V}(I_{n,k}) = \Theta(\mathbb{V}(B_{n,k}))$ for all k. The methods used to derive these results are the same as those used in section 3.

We then prove, in section 5, that both internal and external profiles, after proper normalization, are asymptotically normally distributed iff the variance tends to infinity (see Theorems 8 and 9). The limiting distribution is Poisson when the variance

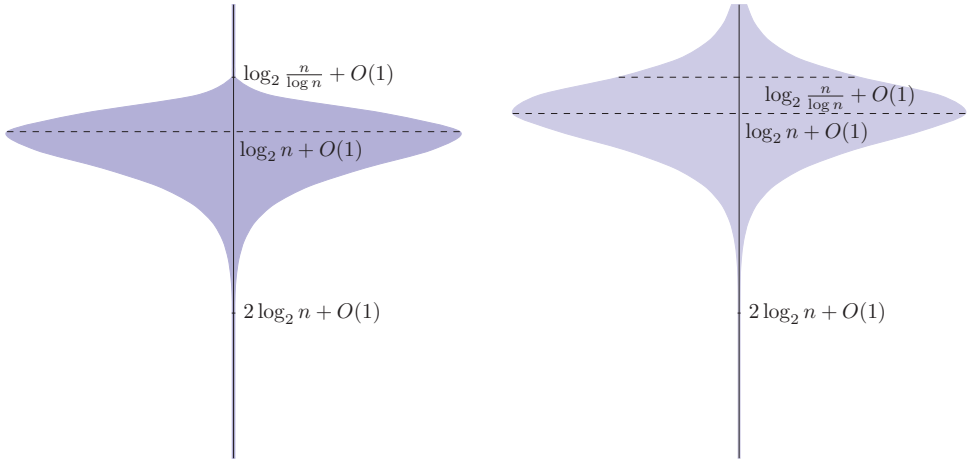


FIG. 6. *The silhouettes of the expected external and internal profiles of a symmetric; compare Figure* 5.

remains bounded away from zero and infinity. In particular, we will prove that when $\mathbb{V}(B_{n,k}) = \Theta(1)$,

$$\mathbb{P}\left(B_{n,k} = 2m\right) = \frac{\lambda_0^m}{m!}\, e^{-\lambda_0} + o(1) \quad \text{and} \quad \mathbb{P}\left(B_{n,k} = 2m+1\right) = o(1),$$

where $\lambda_0 := pqn^2(p^2+q^2)^{k-1}$, while for $\mathbb{V}(I_{n,k}) = \Theta(1)$, we find

$$\mathbb{P}(I_{n,k} = m) = \frac{\lambda_1^m}{m!}\, e^{-\lambda_1} + o(1) \qquad (m = 0, 1, \dots),$$

where $\lambda_1 := n^2(p^2+q^2)^k/2$. These results hold for both symmetric and asymmetric tries, but the ranges where the variances become unbounded are different.

Symmetric tries. For the symmetric case, we have $\alpha_1 = \alpha_2 = 1/\log 2$. This means that the two ranges separated by α_2 coalesce into one for symmetric tries; see Figure 2. The analysis then becomes simpler as shown in section 7. An interesting property is that unlike for asymmetric tries, the fattest levels of profiles of symmetric tries contain a linear number of nodes. The global picture of a random symmetric trie is roughly as follows ($\alpha_1 = 1/\log 2$):

- When $1 \le k \le \alpha_1(\log n - \log\log n + O((\log n)^{-1}))$, each level is almost full of internal nodes ($I_{n,k} \approx 2^k$), the number of external nodes tending to zero; in particular, the variances of both profiles tend to zero.
- When $\alpha_1(\log n - \log\log n + K_n/\log n) \le k \le 2\alpha_1(\log n - K_n)$, where K_n is any sequence tending to infinity, the variances of both profiles tend to infinity, and we prove the asymptotic normality of both profiles.
- When $k = 2\alpha_1(\log n + O(1))$, both profiles are asymptotically Poisson distributed, but $B_{n,k}$ assumes only even values.
- When $k \ge \alpha_1(\log n + K_n)$, then nodes appear very unlikely.

Section 8 describes some consequences of our main results. In particular, we point out a rather unusual form of the local limit theorem for the depth due to the oscillating factor in the expected profile. Then we apply our results to rederive typical behavior for the height, the shortest path, and the fill-up level. Also, the width and

the right-profile (counting only right branches and neglecting the left ones) are briefly discussed.

This completes the summary of our main results. Precise formulations and proofs are presented in the next five sections. Enjoy the reading!

3. Expected external profile. We derive asymptotic approximations to the expected external profile $\mu_{n,k}$ in this section, starting with a few useful expressions for $\mu_{n,k}$.

Notation. Throughout this paper, $p \in [1/2, 1)$ is fixed and $q = 1-p$. Let $k = k(n)$ and $\alpha := \lim_n k/\log n$, whenever the limit exists. The constants α_1, α_2, and α_3 are defined in (5). For convenience, we also write

$$L_n := \log n, \quad LL_n := \log\log n, \quad LLL_n := \log\log\log n.$$

The generic symbol ε is always used to represent a suitably small constant whose value may vary from one occurrence to another, and K_n denotes any sequence tending to infinity. The symbol $f(n) = \Theta(g(n))$ means that there are positive constants C and C' such that $C|g(n)| \le |f(n)| \le C'|g(n)|$.

3.1. Exact expressions and integral representations. Denote by $M_k(z)$ the probability generating function $\sum_{n\ge 0} \mu_{n,k} z^n/n!$ of $\mu_{n,k}$ and by $\tilde{M}_k(z)$ the corresponding Poisson generating function $\tilde{M}_k(z) := e^{-z} M_k(z)$.

LEMMA 1. *The Poisson generating function $\tilde{M}_k(z)$ satisfies the integral representation*

$$\tilde{M}_k(z) = \frac{1}{2\pi i} \int_{(\rho)} z^{-s} \Gamma(s+1) g(s) \left(p^{-s} + q^{-s}\right)^k \mathrm{d}s \tag{6}$$

for $k \ge 1$ and $\Re(z) > 0$, where Γ denotes the Gamma function, $g(s) := 1 - 1/(p^{-s} + q^{-s})$, and $\int_{(\rho)}$ stands for the integral $\int_{\rho-i\infty}^{\rho+i\infty}$. The integral with $\rho > -2$ is absolutely convergent for $\Re(z) > 0$.

Proof. By taking the derivative with respect to y on both sides of (1) and then substituting $y = 1$, we see that $\mu_{n,k}$ satisfies the recurrence (3) with the initial conditions $\mu_{n,k} = \delta_{n,1}\delta_{k,0}$ when either $n \le 1$ and $k \ge 0$ or $k = 0$ and $n \ge 0$. Note that

$$\mu_{n,1} = n\left(pq^{n-1} + qp^{n-1}\right) \qquad (n \ge 2).$$

It follows that

$$M_k(z) = e^{qz} M_{k-1}(pz) + e^{pz} M_{k-1}(qz) \qquad (k \ge 2),$$

with $M_1(z) = z(pe^{qz} + qe^{pz} - 1)$. Thus $\tilde{M}_k(z)$ satisfies

$$\tilde{M}_k(z) = \tilde{M}_{k-1}(pz) + \tilde{M}_{k-1}(qz). \tag{7}$$

Iterating this equation yields

$$\tilde{M}_k(z) = \sum_{0\le j<k} \binom{k-1}{j} \tilde{M}_1(p^j q^{k-1-j} z), \tag{8}$$

from which (6) follows since the Mellin transform

$$M_1^*(s) = \int_0^\infty z^{s-1} \tilde{M}_1(z) dz$$

of $\tilde{M}_1(z) = pze^{-pz} + qze^{-qz} - ze^{-z}$ equals

$$M_1^*(s) = \Gamma(s+1)(p^{-s} + q^{-s} - 1),$$

where $\Re(s) > -2$, and the Mellin transform of $M_1(p^j q^{k-1-j} z)$ is $p^{-sj} q^{-s(k-1-j)} M_1^*(s)$ (see [23, 79]).

To justify the absolute convergence of the integral, we apply the Stirling formula for the Gamma function (with complex parameter)

$$\Gamma(s+1) = \sqrt{2\pi s}\left(\frac{s}{e}\right)^s \left(1 + O\left(|s|^{-1}\right)\right),$$

uniformly as $|s| \to \infty$ and $|\arg s| \le \pi - \varepsilon$, which implies that

$$|\Gamma(\rho + it)| = \Theta(|t|^{\rho - 1/2} e^{-\pi|t|/2}), \tag{9}$$

uniformly for $|t| \to \infty$ and $\rho = o(|t|^{2/3})$.

The integrand in (6) is analytic for $\Re(s) > -2$ and bounded above by

$$\begin{aligned} &z^{-\rho-it}\Gamma(\rho+1+it)g(\rho+it)\left(p^{-\rho-it} + q^{-\rho-it}\right)^k \\ &\quad = O\left(|z|^{-\rho}|t|^{\rho+1/2}e^{-\pi|t|/2+\arg(z)t}(p^{-\rho} + q^{-\rho})^k\right) \end{aligned}$$

for large $|t|$. This completes the proof of the lemma. □

COROLLARY 1. *The expected external profile $\mu_{n,k}$ satisfies, for $n, k \ge 1$,*

$$\begin{aligned} \mu_{n,k} = &\sum_{0\le j\le k} \binom{k}{j} p^j q^{k-j} n \left(1 - p^j q^{k-j}\right)^{n-1} \\ &- \sum_{0\le j<k} \binom{k-1}{j} p^j q^{k-1-j} n \left(1 - p^j q^{k-1-j}\right)^{n-1} \end{aligned} \tag{10}$$

and the integral representation

$$\mu_{n,k} = \frac{1}{2\pi i}\int_{(\rho)} \frac{\Gamma(n+1)\Gamma(s+1)}{\Gamma(n+1+s)}\, g(s)\left(p^{-s} + q^{-s}\right)^k \mathrm{d}s \qquad (\rho > -2), \tag{11}$$

where $g(s) = 1 - 1/(p^{-s} + q^{-s})$.

Proof. By definition and (8)

$$M_k(z) = \sum_{0\le j<k} \binom{k-1}{j} p^j q^{k-1-j} z \left(pe^{(1-p^{j+1}q^{k-1-j})z} + qe^{(1-p^j q^{k-j})z} - e^{(1-p^j q^{k-1-j})z}\right).$$

Thus (the symbol $[z^n]f(z)$ denoting the coefficient of z^n in the Taylor expansion of $f(z)$),

$$\begin{aligned} \mu_{n,k} &= n![z^n]M_k(z) \\ &= \sum_{0\le j<k} \binom{k-1}{j}\left(p^{j+1}q^{k-1-j} n\left(1 - p^{j+1}q^{k-1-j}\right)^{n-1} + p^j q^{k-j} n\left(1 - p^j q^{k-j}\right)^{n-1}\right) \\ &\quad - \sum_{0\le j<k} \binom{k-1}{j} p^j q^{k-1-j} n\left(1 - p^j q^{k-1-j}\right)^{n-1}. \end{aligned}$$

Rearranging the indices of the first sum, we obtain (10).

On the other hand, it also follows from (7) that, denoting by $\tilde{\mu}_{n,k} := n![z^n]\tilde{M}_k(z)$,

$$\tilde{\mu}_{n,k} = (p^n + q^n)\tilde{\mu}_{n,k-1} \qquad (k \geq 2),$$

with

$$\tilde{\mu}_{n,1} = \sum_{2\leq j\leq n} \binom{n}{j} (-1)^{n-j}\mu_{j,1} = (-1)^n n\,(1 - p^n - q^n) \qquad (n \geq 1).$$

Iterating this recurrence yields

$$\tilde{\mu}_{n,k} = (-1)^n n\,(1 - p^n - q^n)\,(p^n + q^n)^{k-1} \qquad (n, k \geq 1).$$

By definition, we have for $n \geq 2$

$$\mu_{n,k} = \sum_{0\leq j\leq n} \binom{n}{j} \tilde{\mu}_{j,k} = \sum_{2\leq j\leq n} \binom{n}{j} (-1)^j j\left(1 - p^j - q^j\right)\left(p^j + q^j\right)^{k-1}. \tag{12}$$

The last sum falls under the so-called Rice integral representation for finite differences (see [25, 79]) from which we conclude

$$\mu_{n,k} = \frac{1}{2\pi i}\int_{(\rho)} \frac{\Gamma(n+1)\Gamma(-s)}{\Gamma(n+1-s)}\, s\,(1 - p^s - q^s)\,(p^s + q^s)^{k-1}\,\mathrm{d}s.$$

This gives (11). Absolute convergence of the integral in (11) when $\Re(s) > -2$ is justified as above. Note that $g(-1) = 0$. □

Remarks. The integral representation (11) follows formally from interchanging the Cauchy and Mellin integrals as shown below:

$$\begin{aligned}
\mu_{n,k} &= \frac{n!}{2\pi i}\int z^{-n-1}e^z \tilde{M}_k(z)\mathrm{d}z \\
&= \frac{n!}{2\pi i}\int z^{-n-1}e^z \left(\frac{1}{2\pi i}\int z^{-s}\Gamma(s+1)g(s)\left(p^{-s} + q^{-s}\right)^k \mathrm{d}s\right)\mathrm{d}z \\
&= \frac{n!}{2\pi i}\int \Gamma(s+1)g(s)\left(p^{-s} + q^{-s}\right)^k \left(\frac{1}{2\pi i}\int z^{-n-1-s}e^z \mathrm{d}z\right)\mathrm{d}s \\
&= \frac{n!}{2\pi i}\int \frac{\Gamma(s+1)}{\Gamma(n+s+1)} g(s)\left(p^{-s} + q^{-s}\right)^k \mathrm{d}s.
\end{aligned} \tag{13}$$

Although all steps here can be justified by analytic properties of the functions involved (which are essentially the estimates needed by the saddle-point method), the way we proved (11), based solely on finite differences, does not rely on any analytic properties.

Note that since the Mellin transform of $x(1-x)^{n-1}$, $x \in (0,1)$, equals $\Gamma(n)\Gamma(s+1)/\Gamma(n+1+s)$, the exact expression (10) also follows from (11) by expanding $(p^{-s} + q^{-s})^k$ and then integrating term by term. For numerical purposes, the expression (10) is preferable to (12), especially when k is not too large.

On the other hand, the closed-form expression (10) can also be proved directly by either a direct combinatorial argument (see [65] for similar details) or an urn model argument (see [59]).

3.2. Road map of the proof through de-Poissonization. From the preceding analysis, we have a choice of two different integral representations: the Rice integral (11) and the Cauchy integral (13). The approach via the Rice integral (11) is simpler than that via the Cauchy and Mellin integrals (13), but the latter can be easily amended for computing the variance and limiting distribution as will be evident from sections 4 and 5. It is for this reason that we use here the route via the Cauchy and Mellin integrals.

By the Poisson heuristic (2), we anticipate the asymptotic equivalence $\mu_{n,k} \sim \tilde{M}_k(n)$. We will see that this holds when $q^{2k}n \to 0$ but requires suitable modification when $q^{2k}n \not\to 0$.

A simple analytic de-Poissonization result. Define a sequence of (Charlier) polynomials $\tau_\ell(n)$ by

$$\tau_\ell(n) := n![z^n]e^z(z-n)^\ell = \ell![z^\ell](1+z)^n e^{-nz} \qquad (\ell = 0, 1, \dots).$$

Then $\tau_0(n) = 1$, $\tau_1(n) = 0$, $\tau_2(n) = -n$, $\tau_3(n) = 2n$, and $\tau_4(n) = 3n^2 - 6n$. Note that $\tau_\ell(n)$ is a polynomial in n of degree $\lfloor \ell/2 \rfloor$.

PROPOSITION 1. *Let $f(z) := \sum_n a_n z^n/n!$ be an entire function, where a_n is a given sequence, and let $\tilde{f}(z) := e^{-z}f(z)$. Write $z = re^{i\theta}$. If*

$$|f(z)| \le f(r)e^{-cr\theta^2} \tag{14}$$

holds uniformly for $r \ge 0$, $c > 0$, and $|\theta| \le \pi$, where $f(r) \ge 0$, and

$$\tilde{f}^{(\ell)}(ne^{i\theta}) = O\left(\delta(n)^\ell \tilde{f}(n)\right) \qquad (\ell = 0, 1, \dots) \tag{15}$$

holds uniformly for $|\theta| \le \theta_1$, where $\theta_1 \ge n^{-1/2+\varepsilon}$ and $\delta(n) = o(n^{-1/2})$, then for any $\ell_0 \ge 2$

$$a_n = \sum_{0 \le \ell < \ell_0} \frac{\tilde{f}^{(\ell)}(n)}{\ell!}\tau_\ell(n) + O\left(n^{\ell_0/2}\delta(n)^{\ell_0}\tilde{f}(n)\right). \tag{16}$$

Proof. By the Cauchy formula and the condition (14), we have

$$a_n = \frac{n!}{2\pi i}\int_{\substack{|z|=n \\ |\arg(z)|\le\theta_0}} z^{-n-1}e^z\tilde{f}(z)\mathrm{d}z + O\left(n!n^{-n}f(n)\int_{\theta_0}^{\infty} e^{-cn\theta^2}\mathrm{d}\theta\right), \tag{17}$$

where $\theta_0 = n^{-2/5}$. By Stirling's formula, we see that the O-term in (17) is bounded above by

$$O\left(n^{1/2}\tilde{f}(n)n^{-1/2}e^{-cn^{1/5}}\right) = O\left(e^{-cn^{1/5}}\tilde{f}(n)\right), \tag{18}$$

which is negligible in comparison to the main term $\tilde{f}(n)$. It remains to evaluate the first term in (17). To that purpose, we expand $\tilde{f}(z)$ at $z = n$ and then integrate term by term, the error term introduced being of the form

$$\frac{n!}{2\pi i(\ell_0-1)!}\int_{\substack{|z|=n \\ |\arg(z)|\le n^{-2/5}}} z^{-n-1}e^z(z-n)^{\ell_0}\int_0^1 (1-t)^{\ell_0-1}\tilde{f}^{(\ell_0)}(n+(z-n)t)\mathrm{d}t\mathrm{d}z,$$

for any $\ell_0 \geq 1$, which is easily seen, by (15), to be bounded above by the O-term in (16); see [30] for similar details. Since $\delta(n) = o(n^{-1/2})$, this proves the asymptotic nature of (16). □

Remark. In particular, we have

$$a_n = \tilde{f}(n) + O(n\delta^2(n)\tilde{f}(n)),$$
$$a_n = \tilde{f}(n) - \frac{n}{2}\tilde{f}''(n) + O\left(n^2\delta^4(n)\tilde{f}(n)\right) \tag{19}$$

for large n.

The theorem indicates that, when the *regularity condition* (14) and the *smoothness condition* (15) both hold for $\tilde{M}_k(z)$, the asymptotics of $\mu_{n,k}$ are reduced to those of their Poisson generating function $\tilde{M}_k(z)$ for large z near the real axis. Our effort in this section is mostly devoted to finding the uniform bounds for justifying the de-Poissonization result (16), which holds for $\mu_{n,k}$ when $q^{2k}n \to 0$. Note that although the condition (14) may seem too strong for our purposes, it can be checked rather systematically in the cases studied in this paper; see [37] for weaker conditions.

On the other hand, we show that when (16) fails (which is the case when $q^{2k}n \not\to 0$), the same proof given above through the Cauchy integral (17) can be appropriately amended because (18) also holds in this case. Thus when deriving our asymptotic estimates for $\mu_{n,k}$, we will either follow the de-Poissonization route through Proposition 1 or evaluate the integral (13) directly using (17).

3.3. Range (I): An elementary analysis. We show in this section that when $1 \leq k \leq \alpha_1(L_n - LLL_n + O(1))$, the asymptotics of $\mu_{n,k}$ are dictated by one or two terms in the first sum of (10). Although the asymptotics of $\mu_{n,k}$ in this range can be easily derived by (10) using only elementary arguments, we will use a lengthier analytic approach based on Cauchy's integral representation since this approach is readily amended later for the asymptotics of the variance. Define

$$k_m := \alpha_1\left(L_n - LLL_n + \log\left(\frac{p}{q} - 1\right) + m\log\frac{p}{q}\right) \qquad (m \geq 0), \tag{20}$$
$$S_{n,k,j} := \binom{k}{j} p^j q^{k-j} n\left(1 - p^j q^{k-j}\right)^{n-1} \qquad (0 \leq j \leq k).$$

For convenience, define $k_{-1} = 0$.

Our first result says that $\mu_{n,k}$ is asymptotic to $S_{n,k,m}$ when $k_{m-1} < k < k_m$ except when k is close to the boundaries, where the corresponding neighboring term (either $S_{n,k,m-1}$ or $S_{n,k,m+1}$) is of the same order.

THEOREM 1 (asymptotics of $\mu_{n,k}$ in range (I)). *Assume $m \geq 0$. If*

$$k_{m-1} + \frac{\alpha_1 K_n}{LL_n} \leq k \leq k_m - \frac{\alpha_1 K_n}{LL_n}, \tag{21}$$

then

$$\mu_{n,k} = S_{n,k,m}\left(1 + O((m+1)e^{-K_n})\right). \tag{22}$$

If $k = k_m + \alpha_1 x/LL_n$, where $x = o(\sqrt{LL_n})$, then

$$\mu_{n,k} = S_{n,k,m}\left(1 + \frac{p\alpha_1 e^x}{q(m+1)}\right)\left(1 + O\left(x^2 LL_n^{-1} + (m+1)L_n^{-(1-q/p)}\right)\right). \tag{23}$$

Remark. Since $\log(p/q) < 1$ for $p \in (1/2, e/(e+1))$, the interval (21) may contain no integer.

By Theorem 1, the proofs of the following special cases are straightforward.

COROLLARY 2. *If $k \geq 1$ and $q^k n \to \infty$, then*

$$\mu_{n,k} \sim q^k n (1-q^k)^{n-1};$$

if $q^{2k} n \to 0$ and $k \leq \alpha_1(L_n - LLL_n + K_n)$, then

$$S_{n,k,m} \sim \frac{k^m}{m!} p^m q^{k-m} n e^{-p^m q^{k-m} n} \qquad (m \geq 0).$$

On the other hand, the estimate

$$\mu_{n,k} = \Theta(S_{n,k,m}) \tag{24}$$

holds uniformly for $k_{m-1} \leq k \leq k_m$, $m \geq 0$.

The proof of Theorem 1 is based on evaluating the Cauchy integral (13) along the circle $|z| = n$ by the same arguments used in the proof of Proposition 1 (see (18)). Observe that

$$\mu_{n,k} = \frac{n!}{2\pi i} \int_{\substack{|z|=n \\ |\arg(z)| \leq \theta_0}} z^{-n-1} e^z \tilde{M}_k(z) \mathrm{d}z + O\left(e^{-cn^{1/5}} \tilde{M}_k(n)\right), \tag{25}$$

where the O-term is justified by applying the following estimate for $M_k(z)$.

LEMMA 2. *Uniformly for $r \geq 0$ and $|\theta| \leq \pi$*

$$|M_k(re^{i\theta})| \leq M_k(r) e^{-cr\theta^2} \qquad (r > 0; |\theta| \leq \pi) \tag{26}$$

for all $k = k(n) \geq 1$ and some constant $c > 0$.

The proof of (26) follows directly from the next proposition in view of (8) and $[z^n] M_1(z) \geq 0$.

PROPOSITION 2. *Let $f(z)$ be an entire function and let $z = re^{i\theta}$, where $r \geq 0$ and $|\theta| \leq \pi$. If*

$$|e^z f(z)| \leq e^r f(r) \qquad (r \geq 0; |\theta| \leq \pi), \tag{27}$$

where $f(r) \geq 0$, then the sum $f_k(z) := \sum_{0 \leq j \leq k} \binom{k}{j} f(p^j q^{k-j} z)$ satisfies

$$|e^z f_k(z)| \leq e^r f_k(r) e^{-cr\theta^2}, \tag{28}$$

uniformly for $k \geq 0$, $r \geq 0$, and $|\theta| \leq \pi$, where $c > 0$ is independent of z and k.

Proof. By (27) and the elementary inequality

$$1 - \cos\theta \geq \frac{2}{\pi^2} \theta^2 \qquad (|\theta| \leq \pi), \tag{29}$$

we obtain

$$\begin{aligned} |e^z f_k(z)| &\leq \sum_{0 \leq j \leq k} \binom{k}{j} e^{(1-p^i q^{k-j}) r \cos\theta} e^{p^j q^{k-j} r} f(p^j q^{k-j} r) \\ &\leq \sum_{0 \leq j \leq k} \binom{k}{j} e^{(1-p^i q^{k-j}) r (1 - 2\theta^2/\pi^2)} e^{p^j q^{k-j} r} f(p^j q^{k-j} r) \\ &\leq e^{-2r\theta^2 (1-p^k)/\pi^2} e^r f_k(r). \end{aligned}$$

This proves (28) with, say, $c = 2(1-p)/\pi^2$. ∎

Proof of (22) *in Theorem* 1. We next evaluate $\tilde{M}_k(z)$ more precisely in the following lemma whose proof is presented in Appendix A.

Let

$$S_{k,m}(z) := \binom{k-1}{m} p^m q^{k-m} z e^{-p^m q^{k-m} z}.$$

LEMMA 3. (i) $(m = 0)$. *If* $1 \le k \le k_0 - \alpha_1 K_n / LL_n$, *then*

$$\tilde{M}_k(z) = q^k z e^{-q^k z} \left(1 + O(e^{-K_n})\right), \tag{30}$$

uniformly for $|z| = n$ *and* $\arg(z) = o(LL_n^{-1/2})$.

(ii) $(m \ge 1)$. *If*

$$k = \alpha_1 \left(L_n - LLL_n + \log(p/q - 1) + m \log(p/q) - \eta\right), \tag{31}$$

where $m \ge 1$ *and*

$$\frac{K_n}{LL_n} \le \eta \le \log(p/q) - \frac{K_n}{LL_n},$$

then

$$\tilde{M}_k(z) = S_{k,m}(z) \left(1 + O(me^{-K_n})\right), \tag{32}$$

uniformly for $|z| = n$ *and* $\arg(z) = o(LL_n^{-1/2})$.

Using the above lemma, we now prove Theorem 1. It remains to evaluate the integral in (25). We first consider the case $m = 0$. By substituting (30) into the integral in (25), and by completing the arc $|\arg(z)| \le \theta_0$ to a full circle, we see that

$$\begin{aligned}\frac{n!}{2\pi i} \int_{\substack{|z|=n \\ |\arg(z)|\le\theta_0}} z^{-n-1} e^z \tilde{M}_k(z) \mathrm{d}z &= \frac{q^k n!}{2\pi i} \int_{\substack{|z|=n \\ |\arg(z)|\le\theta_0}} z^{-n} e^{(1-q^k)z} \mathrm{d}z + O(E_1) \\ &= q^k n! [z^{n-1}] e^{(1-q^k)z} + O(E_2) + O(E_1),\end{aligned}$$

where

$$\begin{aligned}E_1 &:= e^{-K_n} n! n^{-n} q^k n \int_{-\theta_0}^{\theta_0} e^{(1-q^k) n \cos\theta} \mathrm{d}\theta, \\ E_2 &:= q^k n! n^{1-n} \int_{\theta_0}^{\pi} e^{(1-q^k) n \cos\theta} \mathrm{d}\theta.\end{aligned}$$

By inequality (29), we have

$$\begin{aligned}E_1 &= O\left(e^{-K_n} n^{1/2} q^k n e^{-q^k n} \int_{-\infty}^{\infty} e^{-2n(1-q^k)\theta^2/\pi^2} \mathrm{d}\theta\right) \\ &= O\left(e^{-K_n} q^k n e^{-q^k n}\right).\end{aligned}$$

Similarly,

$$E_2 = O\left(q^k n e^{-q^k n} n^{-1/10} e^{-2n^{1/5}/\pi^2}\right).$$

This completes the proof of (22) when $m = 0$. For $m \ge 1$, we proceed in a similar manner but using part (ii) of Lemma 3.

Proof of (23) *in Theorem* 1. We now consider the remaining gaps when k is of the form (31) with $\eta = x/LL_n$, where $x = o(\sqrt{LL_n})$. In this case, the same analysis as above shows that both terms $S_{k,m}(z)$ and $S_{k,m+1}(z)$ are asymptotically close so that

$$\tilde{M}_k(z) = (S_{k,m}(z) + S_{k,m+1}(z))\,(1 + O(E_3))\,, \tag{33}$$

where the error E_3 introduced is bounded above by

$$\begin{aligned} E_3 &= O\left(\sum_{0\le j<m}\left|\frac{S_{k,j}(z)}{S_{k,m}(z)}\right| + \sum_{m+2\le j\le k}\left|\frac{S_{k,j}(z)}{S_{k,m}(z)}\right|\right)\\ &= O\left((m+1)L_n^{-(1-qe^{\eta}\cos\theta/p)}\right) + O\left(m!\sum_{j\ge 2}\frac{(p\alpha_1/q)^j}{(j+m)!}L_n^{j-\frac{(p/q)^j-1}{p/q-1}e^{\eta}\cos\theta}\right)\\ &= O\left((m+1)L_n^{-(1-q/p)} + (m+1)^{-1}L_n^{-(p/q-1)}\right)\\ &= O\left((m+1)L_n^{-(1-q/p)}\right), \end{aligned}$$

since $1 - q/p \le p/q - 1$, where we used the inequality

$$\frac{t^j - 1}{t-1} \ge \frac{t+1}{2}\,j \qquad (t > 1;\ j \ge 2),$$

and $\theta = o(LL_n^{-1/2})$. Thus the same analysis as above gives

$$\mu_{n,k} = \frac{k^m}{m!}p^m q^{k-m} n e^{-p^m q^{k-m} n}\left(1 + \frac{pL_n^{1-e^{\eta}}}{q(m+1)\log(1/q)}\right)\left(1 + O\left((m+1)L_n^{-(1-q/p)}\right)\right),$$

which implies (23). □

3.4. Range (II): A saddle-point analysis. We now assume that

$$\alpha_1\,(L_n - LLL_n + K_n) \le k \le \alpha_2(L_n - K_n\sqrt{L_n}), \tag{34}$$

and proceed by the saddle-point method (see [79, 83]) to derive the following main result of this subsection.

THEOREM 2 (asymptotics of $\mu_{n,k}$ in range (II)). *If k satisfies* (34), *then*

$$\mu_{n,k} = G_1\left(\rho; \log_{p/q} p^k n\right)\frac{n^{-\rho}\,(p^{-\rho} + q^{-\rho})^k}{\sqrt{2\pi\beta_2(\rho)k}}\left(1 + O\left(\frac{1}{k(p/q)^{\rho}} + \frac{1}{k(\rho+2)^2}\right)\right), \tag{35}$$

where $\rho = \rho(n,k) > -2$ is chosen to satisfy the saddle-point equation

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}\rho}\left(\rho^{\rho}e^{-\rho}n^{-\rho}(p^{-\rho} + q^{-\rho})^k\right) = 0 & \text{if } \rho \ge 1,\\[2ex] \dfrac{\mathrm{d}}{\mathrm{d}\rho}\left(n^{-\rho}(p^{-\rho} + q^{-\rho})^k\right) = 0 & \text{if } \rho \le 1, \end{cases} \tag{36}$$

and

$$\beta_2(\rho) := \frac{p^{-\rho}q^{-\rho}\log(p/q)^2}{(p^{-\rho} + q^{-\rho})^2}, \tag{37}$$

$$G_1(\rho; x) = \sum_{j\in\mathbb{Z}} g(\rho + it_j)\Gamma(\rho + 1 + it_j)e^{-2j\pi i x} \qquad (t_j := 2j\pi/\log(p/q))$$

with $g(s) = 1 - 1/(p^{-s} + q^{-s})$*, and* $G_1(\rho, x)$ *is a* 1*-periodic function (see Figures* 3 *and* 4*).*

We devote the rest of this subsection to the proof of Theorem 2.

3.4.1. Two-step saddle-point method. Here we outline the main steps of the proof of Theorem 2. The approach may be called a two-step saddle-point method since the saddle-point method will be applied twice. First, we start from the Mellin integral (6) and apply the saddle-point method to obtain precise asymptotics of $\tilde{M}_k(re^{i\theta})$ for small θ (i.e., around the *real axis*) and large r. The proof here is complicated by the fact that

$$\left|p^{-\rho-it} + q^{-\rho-it}\right| = p^{-\rho} + q^{-\rho} \tag{38}$$

when $t = t_j = 2\pi j/\log(p/q)$, $j \in \mathbb{Z}$, which implies that the number of saddle-points with the same real part is *infinite*, yielding the 1-periodic function $G_1(\rho; x)$.

This first application of the saddle-point method yields a good approximation to $\tilde{M}_k(z)$ for z large and near the real axis; then we *de-Poissonize* $\tilde{M}_k(z)$ by another application of the saddle-point method and establish that $\mu_{n,k} \sim \tilde{M}_k(n)$. Ultimately, we will use the de-Poissonization result of Proposition 1; however, in the first approximation we do de-Poissonization by "bare hands" by applying the argument already used in the proof of Proposition 1, namely, (17) and (18). Thus we focus on the evaluation of the Cauchy integral (13) but with $|\theta| \le n^{-2/5}$ (the first integral of (25)).

3.4.2. Location of saddle-points. The integrand $z^{-s}\Gamma(s+1)g(s)\left(p^{-s} + q^{-s}\right)^k$ of the integral in (6) has simple poles at $s = -j$, $j = 2, 3, \ldots$, the rightmost (dominant) one being at $s = -2$; it also has saddle-points, which are the zeros of the equation

$$\frac{\mathrm{d}}{\mathrm{d}s}\left(\Gamma(s+1)n^{-s}(p^{-s} + q^{-s})^k\right) = 0 \tag{39}$$

(note that $g(s)$ is uniformly bounded for all s). In view of (38), there are infinitely many saddle-points of the form $\rho + it_j/\log(p/q)$ ($j = 0, \pm 1, \ldots$), where the real part ρ satisfies (39). Also it is easy to see that

$$\begin{cases} \rho \to +\infty & \text{if } \frac{k}{L_n} \downarrow \frac{1}{\log(1/q)}, \\ \rho \to -\infty & \text{if } \frac{k}{L_n} \uparrow \frac{1}{\log(1/p)}. \end{cases}$$

We distinguish between two cases $\rho \ge 1$ and $-2 < \rho < 1$. In the former, the saddle-points are asymptotically determined, by Stirling's formula for the Gamma function, by the first equation in (36), which is simpler than (39), while in the latter case they are asymptotically determined by the second equation of (36) since $\Gamma(\rho+1)$ is uniformly bounded and thus does not contribute significantly to the saddle-point location.

More precisely, first consider the case when $\rho \ge 1$ (the choice of 1 being arbitrary). In this case, by (36), we obtain

$$\frac{k}{L_n - \log\rho} = \frac{p^{-\rho} + q^{-\rho}}{p^{-\rho}\log(1/p) + q^{-\rho}\log(1/q)},$$

which can be written in the form

$$\rho = \frac{1}{\log(p/q)} \log\left(\frac{L_n - \log\rho - k\log(1/p)}{k\log(1/q) - L_n + \log\rho}\right),$$

whenever $L_n - \log \rho < k \log(1/q)$, which will be seen to be the case when k satisfies (34).

On the other hand, when $\rho \le 1$, we consider the second equation in (36) or

$$\frac{k}{L_n} = \frac{p^{-\rho} + q^{-\rho}}{p^{-\rho}\log(1/p) + q^{-\rho}\log(1/q)},$$

which is solved to be

$$\rho = \frac{1}{\log(p/q)} \log\left(\frac{L_n - k\log(1/p)}{k\log(1/q) - L_n}\right). \tag{40}$$

It follows that if k satisfies (34), then

$$\rho \le \frac{1}{\log(p/q)} \left(LL_n - \log K_n + \log \frac{\log(p/q)}{\log(1/q)} + o(1)\right), \tag{41}$$

implying, in particular, that $\rho = O(LL_n)$. Also, if $k = \alpha_1(L_n - LLL_n + \log\log(p/q) + K_n)$, then

$$\rho = \frac{1}{\log(p/q)} \left(LL_n - \log K_n + \log \frac{\log(p/q)}{\log(1/q)} + O(K_n^{-1})\right).$$

However, if k is close to the right boundary of (34), more precisely, $k = \alpha_2(1-\varepsilon_n)L_n$, where $\varepsilon_n = o(1)$, then

$$\rho = -2 + \frac{\varepsilon_n}{\alpha_2\beta_2(-2)} + O(\varepsilon_n^2).$$

Thus $\rho = O(1)$.

From (41), we see that if $\rho \ge 1$ and k satisfies (34), then $k\beta_2(\rho) = \Theta(k(p/q)^\rho)$ and

$$k(p/q)^\rho \ge \frac{K_n}{\log(p/q)} + o(1);$$

on the other hand, if $\rho \ge -2 + K_n L_n^{-1/2}$, then $k(\rho+2)^2 \ge K_n^2$. Thus the O-term in (35) is small if we choose K_n sufficiently large.

3.4.3. More transparent behaviors of $\mu_{n,k}$. Before we present a formal proof of Theorem 2, we first discuss more transparent behaviors of $\mu_{n,k}$ in some specified ranges.

The central range: $\alpha \in [\alpha_1 + \varepsilon, \alpha_2 - \varepsilon]$. In this case, G_1 is bounded and $G_1(\rho; x) \sim G_1(\rho'; x)$, where

$$\rho' := \frac{1}{\log(p/q)} \log\left(\frac{1 - \alpha\log(1/p)}{\alpha\log(1/q) - 1}\right); \tag{42}$$

also $\beta_2(\rho) \sim \beta_2(\rho')$. Note that $g(\rho + it_j) = 1 - p^{it_j}/(p^{-\rho} + q^{-\rho})$ and

$$G_1\left(\rho; \log_{p/q} p^k n\right) = G_1\left(\rho; \log_{p/q} q^k n\right).$$

More precisely, if $k = \alpha(L_n + x\sqrt{\alpha\beta_2(\rho')L_n})$, where $\alpha \in [\alpha_1+\varepsilon, \alpha_2-\varepsilon]$ and $x = o(L_n^{1/6})$, then

$$\mu_{n,k} = G_1\left(\rho'; \log_{p/q} p^k n\right) \frac{n^{-\rho'}\left(p^{-\rho'}+q^{-\rho'}\right)^k}{\sqrt{2\pi\alpha\beta_2(\rho')L_n}}\, e^{-x^2/2}\left(1+O\left(\frac{1+|x|^3}{\sqrt{L_n}}\right)\right),$$

uniformly in x. In particular, when $\alpha = 1/\mathrm{h}$, where $\mathrm{h} := p\log(1/p) + q\log(1/q)$ is the entropy of the Bernoulli variate, then $\rho' = -1$, and it follows that

$$\mu_{n,k} = \frac{\sqrt{\mathrm{h}}\, G_1\left(-1; \log_{p/q} p^k n\right)}{\log(p/q)\sqrt{2\pi pq}} \cdot \frac{n}{\sqrt{L_n}}\, e^{-x^2/2}\left(1+O\left(\frac{1+|x|^3}{\sqrt{L_n}}\right)\right), \tag{43}$$

uniformly for $x = o(L_n^{1/6})$. Other approximations can be derived for $L_n^{1/6} \ll x = o(\sqrt{L_n})$. Thus $\mu_{n,k}$ reaches the maximum for k near $L_n/\mathrm{h} + O(1)$; also, $\mu_{n,k}$ increases with k when $\alpha < 1/\mathrm{h}$ and decreases with k when $\alpha > 1/\mathrm{h}$; see Figure 2. See also Figure 3 for a plot of $G_1(-1; x)$ for a few p's.

The left boundary: $\rho \to -2^+$ *and* $\rho + 2 \gg L_n^{-1/2}$. In this case, the dominant periodicity vanishes because

$$G_1(\rho; x) \sim \frac{|g(-2)|}{\rho+2} = \frac{2pq}{(p^2+q^2)(\rho+2)};$$

thus

$$\mu_{n,k} \sim \frac{2}{\sqrt{2\pi}\,\log(p/q)(\rho+2)}\, k^{-1/2} n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k. \tag{44}$$

The right boundary: $k/L_n \to 1/\log(1/q)^+$. In this case, $\rho \to \infty$ and $\rho = O(LL_n)$. The periodicity in the leading term of (35) does not vanish because we have

$$G_1(\rho; x) \sim \sum_{j\in\mathbb{Z}} \Gamma(\rho+1+it_j) e^{-2j\pi i x},$$

and G_1 is not bounded. Indeed, the periodicity becomes more pronounced for increasing ρ since

$$\left|\frac{\Gamma(\rho+1+it)}{\Gamma(\rho+1)}\right| = O\left(e^{-t^2/(2\rho)+O(t^4/\rho^3)}\right)$$

for large ρ and $t = o(\rho)$; see Figure 4. This estimate also implies that

$$G_1(\rho; x) = O\left(\sum_{j\in\mathbb{Z}} |\Gamma(\rho+1+it_j)|\right) = O\left(e^{-\rho}\rho^{\rho+1}\right) = O\left(\rho^{1/2}\Gamma(\rho+1)\right).$$

The order is tight. This means that even if we normalize $G_1(\rho; x)$ by $\Gamma(\rho+1)$, $|G_1(\rho; x)|$ still goes to infinity with ρ.

3.4.4. Proof of Theorem 2. In view of (25) (more generally, de-Poissonization Proposition 1), we need only evaluate $\tilde{M}_k(n)$ and obtain precise local expansions for $\tilde{M}_k(ne^{i\theta})$ when $|\theta| \le \theta_0$ in order to estimate the first integral of (25). We first focus on estimating $\tilde{M}_k(n)$ and then extend the same approach to derive the asymptotics of $\tilde{M}_k(ne^{i\theta})$. This suffices to prove that $\mu_{n,k} \sim \tilde{M}_k(n)$. Later in subsection 3.8 we refine this analysis to obtain a better error term.

In order to evaluate $\tilde{M}_k(n)$ by the inverse Mellin transform, first we move the line of integration of (6) to $\Re(s) = \rho$ so that

$$\tilde{M}_k(n) = \frac{1}{2\pi} \int_{-\infty}^{\infty} J_k(n; \rho + it) \mathrm{d}t, \tag{45}$$

where $\rho > -2$ is the saddle-point chosen according to (36) and $J_k(n; s) := n^{-s}\Gamma(s + 1)g(s)(p^{-s}+q^{-s})^k$. We now show that the above integral with $|t| \ge \sqrt{L_n}$ is asymptotically smaller than the dominant term in (35) and then assess the main contribution of saddle-points falling into the range $|t| \le \sqrt{L_n}$.

Estimate of the integral when $|t| \ge \sqrt{L_n}$. Assume from now on that ρ is chosen as described above in (36).

Since our $\rho > -2$ satisfies (40), we have, by (9),

$$\begin{aligned}
\frac{1}{2\pi} \int_{|t|\ge\sqrt{L_n}} J_k(n; \rho + it) \mathrm{d}t &= O\left(n^{-\rho}(p^{-\rho} + q^{-\rho})^k \int_{\sqrt{L_n}}^{\infty} |\Gamma(\rho + 1 + it)| \mathrm{d}t \right) \\
&= O\left(n^{-\rho}(p^{-\rho} + q^{-\rho})^k \int_{\sqrt{L_n}}^{\infty} t^{\rho+1/2} e^{-\pi t/2} \mathrm{d}t \right) \\
&= O\left(L_n^{\rho/2+1/4} e^{-\pi\sqrt{L_n}/2} n^{-\rho}(p^{-\rho} + q^{-\rho})^k \right).
\end{aligned}$$

On the other hand, since $\rho = O(LL_n)$ and $\rho \ge -2 + K_n L_n^{-1/2}$, we then obtain

$$L_n^{\rho/2+1/4} e^{-\pi\sqrt{L_n}/2} = O\left(e^{-\pi\sqrt{L_n}/2 + O(LL_n^2)} \right) = O\left(\Gamma(\rho + 2) e^{-\sqrt{L_n}} \right)$$

for large enough n; the last O-term holds uniformly for $\rho \ge -2 + K_n L_n^{-1/2}$ and ρ satisfying (41).

Contribution from each saddle-point. Let j_0 be the largest integer j for which $2j\pi/\log(p/q) \le \sqrt{L_n}$. Then we can split the integral over $\int_{|t|\le\sqrt{L_n}}$ as follows:

$$\begin{aligned}
\int_{|t|\le\sqrt{L_n}} J_k(n; \rho + it) \mathrm{d}t = &\sum_{|j|<j_0} \int_{|t-t_j|\le\pi/\log(p/q)} J_k(n; \rho + it) \mathrm{d}t \\
&+ \int_{t_{j_0}\le|t|\le\sqrt{L_n}} J_k(n; \rho + it) \mathrm{d}t.
\end{aligned}$$

The last integral is bounded above by

$$O\left(\Gamma(\rho + 2) n^{-\rho}(p^{-\rho} + q^{-\rho})^k e^{-\sqrt{L_n}} \right),$$

by the same argument used above. It remains to evaluate the integrals

$$T_j := \frac{1}{2\pi} \int_{|t-t_j|\le\pi/\log(p/q)} J_k(n; \rho + it) \mathrm{d}t$$

for $|j| < j_0$.

We first derive a uniform bound for $|p^{-\rho-it}+q^{-\rho-it}|$. By the elementary inequalities (29) and

$$\sqrt{1-x} \le 1-\frac{x}{2} \qquad (x\in[0,1]),$$

we have

$$\begin{aligned}
\left|p^{-\rho-it}+q^{-\rho-it}\right| &= \left(p^{-\rho}+q^{-\rho}\right)\sqrt{1-\frac{2p^{-\rho}q^{-\rho}}{(p^{-\rho}+q^{-\rho})^2}\left(1-\cos\left(t\log\left(\frac{p}{q}\right)\right)\right)} \\
&\le \left(p^{-\rho}+q^{-\rho}\right)\left(1-\frac{p^{-\rho}q^{-\rho}}{(p^{-\rho}+q^{-\rho})^2}\left(1-\cos\left((t-t_j)\log\left(\frac{p}{q}\right)\right)\right)\right) \\
&\le \left(p^{-\rho}+q^{-\rho}\right)\left(1-\frac{2p^{-\rho}q^{-\rho}}{\pi^2(p^{-\rho}+q^{-\rho})^2}(t-t_j)^2\log\left(\frac{p}{q}\right)^2\right) \\
&\le \left(p^{-\rho}+q^{-\rho}\right)e^{-c_0(t-t_j)^2}, \qquad (46)
\end{aligned}$$

uniformly for $|t-t_j|\le \pi/\log(p/q)$, where

$$c_0 = c_0(\rho) := \frac{2p^{-\rho}q^{-\rho}\log(p/q)^2}{\pi^2(p^{-\rho}+q^{-\rho})^2} = \frac{2}{\pi^2}\,\beta_2(\rho).$$

We now take

$$v_0 := \begin{cases} k^{-2/5} & \text{if } -2<\rho\le 1, \\ (c_0k)^{-2/5} & \text{if } \rho\ge 1, \end{cases}$$

and split the integration range into two parts: $|t-t_j|\le v_0$ and $v_0<|t-t_j|\le \pi/\log(p/q)$. (We assume that k is so large that $v_0<\pi/\log(p/q)$.)

First consider the case when $-2<\rho\le 1$. From the inequality (46), it follows that

$$\begin{aligned}
T_j'' &:= \frac{1}{2\pi}\int_{v_0\le|t-t_j|\le\pi/\log(p/q)} J_k(n;\rho+it)\mathrm{d}t \qquad (47)\\
&= O\left(|\Gamma(\rho+2+it_j)|n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k\int_{k^{-2/5}}^{\infty}e^{-c_0kv^2}\mathrm{d}v\right) \\
&= O\left(n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k k^{-3/5}e^{-c_0k^{1/5}}\times\begin{cases}|\Gamma(\rho+1+it_j)| & \text{if } j\ne 0\\ 1 & \text{if } j=0\end{cases}\right)
\end{aligned}$$

for each $|j|\le j_0$.

When $\rho\ge 1$ and satisfies (34), we have

$$\begin{aligned}
T_j'' &= O\left(|\Gamma(\rho+1+it_j)|n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k\int_{(c_0k)^{-2/5}}^{\infty}e^{-c_0kv^2}\mathrm{d}v\right) \\
&= O\left(|\Gamma(\rho+1+it_j)|n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k(c_0k)^{-3/5}e^{-(c_0k)^{1/5}}\right)
\end{aligned}$$

for $|j|\le j_0$.

The dominant terms. It remains to evaluate the integrals T_j for t in the range $|t-t_j| \le v_0$. Note that, by our choice of t_j,

$$p^{-\rho-it_j} + q^{-\rho-it_j} = p^{-it_j}\left(p^{-\rho}+q^{-\rho}\right) = q^{-it_j}\left(p^{-\rho}+q^{-\rho}\right),$$

so that

$$\begin{aligned}\frac{p^{-\rho-it}+q^{-\rho-it}}{p^{-\rho-it_j}+q^{-\rho-it_j}} &= 1+\sum_{\ell\ge 1}\frac{i^\ell(t-t_j)^\ell}{\ell!}\cdot\frac{p^{-\rho-it_j}\log(1/p)^\ell + q^{-\rho-it_j}\log(1/q)^\ell}{p^{-\rho-it_j}+q^{-\rho-it_j}}\\ &= 1+\sum_{\ell\ge 1}\frac{i^\ell(t-t_j)^\ell}{\ell!}\cdot\frac{p^{-\rho}\log(1/p)^\ell + q^{-\rho}\log(1/q)^\ell}{p^{-\rho}+q^{-\rho}},\end{aligned}$$

where we recall that $t_j = 2\pi j/\log(p/q)$.

It follows that

$$\log\left(p^{-\rho-it}+q^{-\rho-it}\right) = \log\left(p^{-\rho-it_j}+q^{-\rho-it_j}\right) + \sum_{\ell\ge 1}\frac{\beta_\ell(\rho)}{\ell!}\,i^\ell(t-t_j)^\ell,$$

where, in particular,

$$\beta_1(\rho) = \frac{p^{-\rho}\log(1/p)+q^{-\rho}\log(1/q)}{p^{-\rho}+q^{-\rho}}.$$

The remaining manipulation by using the saddle-point method is then straightforward. We use the local expansions

$$\left(\frac{p^{-\rho-it}+q^{-\rho-it}}{p^{-\rho-it_j}+q^{-\rho-it_j}}\right)^k = \exp\left(k\sum_{1\le\ell\le 3}\frac{\beta_\ell(\rho)}{\ell!}i^\ell(t-t_j)^\ell + O(k|\beta_4(\rho)||t-t_j|^4)\right)$$

and

$$\begin{aligned}&\Gamma(\rho+1+it)g(\rho+it)\\ &\quad= \begin{cases} C_0 + C_1 i(t-t_j) + O\left(\dfrac{(t-t_j)^2}{(\rho+2)^2}\right) & \text{if } -2<\rho\le 1,\\ \Gamma(\rho+1+it_j)e^{(\log\rho)i(t-t_j)}\left(1+C_2 i(t-t_j)+O(|C_2|^3|t-t_j|^2)\right) & \\ \quad\times\left(g(\rho+it_j)+g'(\rho+it_j)i(t-t_j)+O\left(|t-t_j|^2\right)\right) & \text{if } \rho\ge 1,\end{cases}\end{aligned}$$

where

$$\begin{cases} C_0 := \Gamma(\rho+1+it_j)g(\rho+it_j),\\ C_1 := g(\rho+it_j)\Gamma(\rho+1+it_j)\psi(\rho+1+it_j) + g'(\rho+it_j)\Gamma(\rho+1+it_j),\end{cases}$$

$\psi(s) = \Gamma'(s)/\Gamma(s)$ being the logarithmic derivative of the Gamma function, and

$$C_2 := \psi(\rho+1+it_j) - \log\rho \qquad (\rho\ge 1).$$

Here C_0 and C_1 are defined to be their limits when $\rho=-1$ and $j=0$, namely,

$$\begin{cases} C_0 := p\log(1/p)+q\log(1/q),\\ C_1 := -\frac{2p-1}{2}\left(p\log(p)^2 - q\log(q)^2\right) - C_0\gamma - 2pq\log(p)\log(q).\end{cases}$$

Note that $\psi(\rho+1+it_j)-\log\rho = O(\log(1+|t_j|))$. It follows that for $|j|<j_0$

$$T_j = \frac{g(\rho+it_j)}{\sqrt{2\pi\beta_2(\rho)k}}\,\Gamma(\rho+1+it_j)n^{-\rho-it_j}\left(p^{-\rho}+q^{-\rho}\right)^k p^{-ikt_j} \times\left(1+O\left(\frac{1}{k\beta_2(\rho)}+\frac{1}{k(\rho+2)^2}\right)\right).$$

Summing over all $|j|<j_0$ and collecting all estimates, we obtain

$$\tilde{M}_k(n) = \frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}}\sum_{|j|<j_0} g(\rho+it_j)\Gamma(\rho+1+it_j)(p^k n)^{-it_j} \times\left(1+O\left(\frac{1}{k(p/q)^\rho}+\frac{1}{k(\rho+2)^2}\right)\right).$$

An asymptotic approximation to $\tilde{M}_k(z)$. To complete the de-Poissonization, we need a more precise expansion of $\tilde{M}_k(ne^{i\theta})$ for small θ. The above proof by the saddle-point method can be easily extended mutatis mutandis to $\tilde{M}_k(z)$ for complex values of z lying in the right half-plane since we can write (7) as

$$\tilde{M}_k(ne^{i\theta}) = \frac{1}{2\pi i}\int_{(\rho)} n^{-s}e^{-i\theta s}\Gamma(s+1)g(s)\left(p^{-s}+q^{-s}\right)^k \mathrm{d}s,$$

where $\rho>-2$ and $|\theta|\le\pi/2-\varepsilon$. The result is

$$\tilde{M}_k(ne^{i\theta}) = \frac{(p^{-\sigma}+q^{-\sigma})^k}{\sqrt{2\pi\beta_2(\rho)k}}\sum_{|j|<j_0} g(\sigma+it_j)\Gamma(\sigma+1+it_j)(ne^{i\theta})^{-\rho-it_j}p^{-ikt_j} \times\left(1+O\left(\frac{1}{k(p/q)^\rho}+\frac{1}{k(\rho+2)^2}\right)\right), \tag{48}$$

uniformly for $|\theta|\le\pi/2-\varepsilon$ and k lying in the range (34). Note that the index of the sum can be extended to infinity, but it is easier to manipulate a finite sum than an infinite series since we substitute the right-hand side into the Cauchy integral (13) and then integrate term by term. This completes the proof of (35).

3.5. Range (IV): A singularity analysis. We consider range (IV) first, leaving to the next subsection the analysis in the transitional range when $k=\alpha_2 L_n + o(L_n^{2/3})$.

We show that, for $k\ge\alpha_2 L_n + K_n\sqrt{L_n}$, the asymptotics of the expected profile $\tilde{M}_k(n)$ are dictated by the simple pole at $s=-2$ in (6) or, structurally, by the number of pairs of input-strings sharing the same prefixes of length at least k.

THEOREM 3. *If*

$$k\ge\alpha_2\left(L_n+K_n\sqrt{\alpha_2\beta_2(-2)L_n}\right), \tag{49}$$

where β_2 *is defined in* (37)*, then*

$$\mu_{n,k} = 2pqn^2(p^2+q^2)^{k-1}\left(1+O\left(K_n^{-1}e^{-K_n^2/2+O(K_n^3/\sqrt{L_n})}\right)\right), \tag{50}$$

uniformly for $1\ll K_n = o(\sqrt{L_n})$.

Proof. To prove (50), we move the line of integration (by absolute convergence of the integral) of the integral in (6) to $\Re(s) = \varrho$, where

$$\varrho := -2 - \frac{K_n}{\sqrt{\alpha_2\beta_2(-2)L_n}}.$$

Thus $\tilde{M}_k(ne^{i\theta})$ equals the residue of the integrand at $s = -2$ (the dominant term in (50)) plus the integral along $\Re(s) = \varrho$:

$$\tilde{M}_k(ne^{i\theta}) = |g(-2)|n^2e^{2i\theta}(p^2+q^2)^k + \frac{1}{2\pi}\int_{-\infty}^{\infty} J_k(ne^{i\theta}; \varrho + it)\mathrm{d}t,$$

where $|g(-2)| = 2pq/(p^2+q^2)$. It remains only to estimate the last integral. By the same analysis used for T_j'' (see (47)) and the inequality (46), we have

$$\begin{aligned}
&\frac{1}{2\pi}\left(\int_{|t|\le \pi/\log(p/q)} + \sum_{|j|\ge 1}\int_{|t-t_j|\le \pi/\log(p/q)}\right) J_k(ne^{i\theta}; \varrho + it)\mathrm{d}t \\
&\quad = O\left(|\Gamma(\varrho+1)|n^{-\varrho}\left(p^{-\varrho}+q^{-\varrho}\right)^k \int_{|t|\le \pi/\log(p/q)} e^{-c_0kt^2}\mathrm{d}t\right) \\
&\qquad + O\left(n^{-\varrho}\left(p^{-\varrho}+q^{-\varrho}\right)^k \sum_{|j|\ge 1}\left|\Gamma\left(\varrho+1+\frac{2|j|-1}{\log(p/q)}\pi i\right)\right| e^{(2|j|+1)\pi|\theta|/\log(p/q)}\right. \\
&\qquad\qquad \left.\times \int_{|t-t_j|\le \pi/\log(p/q)} e^{-c_0k(t-t_j)^2}\mathrm{d}t\right) \\
&\quad = O\left(\frac{k^{-1/2}}{|\varrho+2|}\, n^{-\varrho}\left(p^{-\varrho}+q^{-\varrho}\right)^k\right) \\
&\quad = O\left(K_n^{-1}n^{-\varrho}\left(p^{-\varrho}+q^{-\varrho}\right)^k\right),
\end{aligned}$$

where we used (9) to bound the sum

$$\begin{aligned}
&\sum_{|j|\ge 1}\left|\Gamma\left(\varrho+1+\frac{2|j|-1}{\log(p/q)}\pi i\right)\right| e^{(2|j|+1)\pi|\theta|/\log(p/q)} \\
&\quad = O\left(\sum_{|j|\ge 1}(2|j|-1)^{\varrho+1/2}\exp\left(-\frac{\pi^2(2|j|-1)}{2\log(p/q)} + \frac{(2|j|+1)\pi|\theta|}{\log(p/q)}\right)\right) \\
&\quad = O(1),
\end{aligned}$$

uniformly for $|\theta| \le \pi/2 - \varepsilon$.

By our choice of ϱ and by straightforward expansion, we have

$$\begin{aligned}
\frac{K_n^{-1}n^{-\varrho}\left(p^{-\varrho}+q^{-\varrho}\right)^k}{n^2\left(p^2+q^2\right)^k} &= O\left(K_n^{-1}e^{-L_n(\varrho+2)+\frac{k}{\alpha_2}(\varrho+2)+\frac{k}{2}\beta_2(-2)(\varrho+2)^2+O\left(k|\varrho+2|^3\right)}\right) \\
&= O\left(K_n^{-1}e^{-K_n^2/2+O(K_n^3/\sqrt{L_n})}\right).
\end{aligned}$$

Thus

$$\tilde{M}_k(ne^{i\theta}) = |g(-2)|(ne^{i\theta})^2(p^2+q^2)^k\left(1+O\left(K_n^{-1}e^{-K_n^2/2+O(K_n^3/\sqrt{L_n})}\right)\right), \tag{51}$$

uniformly for $|\theta| \le \pi/2 - \varepsilon$. Substituting this into (25), we deduce the desired result (50). □

Remarks. (i) When $K_n \ge \varepsilon\sqrt{L_n}$, we can either take $K_n = \varepsilon\sqrt{L_n}$ or refine the analysis to give a better error term.

(ii) The asymptotic approximation (50) can also be derived from the exact expression (10) by using only elementary arguments.

(iii) Also note that the range (49) implies that the saddle-point ρ satisfies $\rho \le -2 - K_n/\sqrt{L_n}$, but the contribution from this saddle-point is asymptotically negligible (compared to the polar singularity).

3.6. Range (III): A uniform analysis. We consider in this subsection the transitional range $k = \alpha_2 L_n + o(L_n^{2/3})$ and show that the transitional behavior of $\mu_{n,k}$ in this range is well described by a Gaussian distribution function.

THEOREM 4. *If*

$$k = \alpha_2 \left(L_n + \xi \sqrt{\alpha_2 \beta_2(-2) L_n} \right), \tag{52}$$

where $\xi = \xi_{n,k} = o(L_n^{1/6})$, *then*

$$\mu_{n,k} = |g(-2)| \Phi(\xi) n^2 \left(p^2 + q^2\right)^k \left(1 + O\left(\frac{1+|\xi|^3}{\sqrt{L_n}}\right)\right), \tag{53}$$

uniformly in ξ, *where* $\Phi(\xi) = (2\pi)^{-1/2} \int_{-\infty}^{\xi} e^{-t^2/2} \mathrm{d}t$.

Proof. We assume first that k satisfies (52) and $k < \alpha_2 L$ (or $\xi < 0$). We move the line of integration of the integral in (11) to $\Re(s) = \rho$, where ρ is taken to be of the same form as in (40); asymptotically

$$\rho = -2 - \frac{\xi}{\sqrt{\alpha_2 \beta_2(-2) L_n}} + O\left(\xi^2 L_n^{-1}\right). \tag{54}$$

By a similar analysis as the proof of Theorem 3, we obtain

$$\begin{aligned} \tilde{M}_k(ne^{i\theta}) = {} & \frac{1}{2\pi} \int_{|t| \le L_n^{-2/5}} J_k(ne^{i\theta}; \rho + it) \mathrm{d}t \\ & + O\left(|\Gamma(\rho + 1 + iL_n^{-2/5})| n^{-\rho} \left(p^{-\rho} + q^{-\rho}\right)^k e^{-c_0 L_n^{1/5}} \right) \\ & + O\left(k^{-1/2} n^{-\rho} \left(p^{-\rho} + q^{-\rho}\right)^k \right), \end{aligned}$$

where $|\theta| < \pi/2$. By (54), we have

$$|\Gamma(\rho + 1 + iL_n^{-2/5})| = O\left(\frac{1}{|\xi| L_n^{-1/2} + L_n^{-2/5}} \right) = O(L_n^{2/5}).$$

It follows that

$$\tilde{M}_k(ne^{i\theta}) = \frac{1}{2\pi} \int_{|t| \le L_n^{-2/5}} J_k(ne^{i\theta}; \rho + it) \mathrm{d}t + O\left(k^{-1/2} n^{-\rho} \left(p^{-\rho} + q^{-\rho}\right)^k \right).$$

Note that since $s \mapsto \Gamma(s+1) + 1/(s+2)$ is analytic for $|s+2| \le 1 - \varepsilon$, we have

$$\begin{aligned} \tilde{M}_k(ne^{i\theta}) = {} & \frac{|g(-2)|}{2\pi} \int_{|t| \le L_n^{-2/5}} \frac{n^{-\rho - it} e^{-i\theta(\rho + it)}}{\rho + 2 + it} \left(p^{-\rho - it} + q^{-\rho - it}\right)^k \mathrm{d}t \\ & + O\left(k^{-1/2} n^{-\rho} \left(p^{-\rho} + q^{-\rho}\right)^k \right). \end{aligned}$$

The integral on the right-hand side is evaluated as follows:

$$\begin{aligned}
&\frac{|g(-2)|}{2\pi}\int_{|t|\le L_n^{-2/5}} \frac{n^{-\rho-it}e^{-i\theta(\rho+it)}}{\rho+2+it}\left(p^{-\rho-it}+q^{-\rho-it}\right)^k \mathrm{d}t \\
&\quad = \frac{|g(-2)|}{2\pi}\, n^{-\rho}e^{-i\theta\rho}\left(p^{-\rho}+q^{-\rho}\right)^k \int_{|t|\le L_n^{-2/5}} \frac{e^{\theta t-\beta_2(\rho)kt^2/2+O(k|t|^3)}}{\rho+2+it}\,\mathrm{d}t \\
&\quad = \frac{|g(-2)|}{2\pi}\, n^{-\rho}e^{-i\theta\rho}\left(p^{-\rho}+q^{-\rho}\right)^k \int_{-\infty}^{\infty} \frac{e^{-t^2/2}}{\xi_0+it}\left(1+O\left(\frac{|t|+|t|^3}{\sqrt{L_n}}\right)\right)\mathrm{d}t, \qquad (55)
\end{aligned}$$

where

$$\xi_0 := (\rho+2)\sqrt{\beta_2(\rho)k} > 0.$$

Note that $\xi_0 = -\xi + O\left(\xi^2 L_n^{-1/2}\right)$ by (52) and (54). Since $\xi_0 > 0$, we have

$$\begin{aligned}
\frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{e^{-t^2/2}}{\xi_0+it}\,\mathrm{d}t &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-t^2/2}\int_0^{\infty} e^{-v(\xi_0+it)}\mathrm{d}v\mathrm{d}t \\
&= \frac{1}{2\pi}\int_0^{\infty} e^{-v\xi_0}\int_{-\infty}^{\infty} e^{-t^2/2-itv}\mathrm{d}t\mathrm{d}v \\
&= \frac{1}{\sqrt{2\pi}}\int_0^{\infty} e^{-v^2/2-v\xi_0}\mathrm{d}v \\
&= e^{\xi_0^2/2}\Phi(-\xi_0).
\end{aligned}$$

The error term in (55) is estimated similarly and satisfies

$$L_n^{-1/2}\int_{-\infty}^{\infty}\frac{(|t|+|t|^3)e^{-t^2/2}}{|(\rho+2)\sqrt{\beta_2(\rho)k}+it|}\,\mathrm{d}t = O\left(L_n^{-1/2}\int_0^{\infty}(v+v^3)e^{-v^2/2-v\xi_0}\mathrm{d}v\right).$$

Observe that

$$e^{x^2/2}\Phi(-x) = \begin{cases} O\left(x^{-1}\right) & \text{if } x\to\infty, \\ O\left(e^{x^2/2}\right) & \text{if } x\to-\infty. \end{cases} \qquad (56)$$

Also

$$\int_0^{\infty}(v+v^3)e^{-v^2/2-vx}\mathrm{d}v = \begin{cases} O\left(x^{-2}\right) & \text{if } x\to\infty, \\ O\left(|x|^3e^{x^2/2}\right) & \text{if } x\to-\infty, \end{cases}$$

so that

$$\int_0^{\infty} v^3e^{-v^2/2-vx}\mathrm{d}v = O\left(e^{x^2/2}\Phi(-x)(1+|x|^3)\right).$$

Thus

$$\begin{aligned}
\tilde{M}_k(ne^{i\theta}) &= |g(-2)|(ne^{i\theta})^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k e^{\xi_0^2/2}\Phi(-\xi_0)\left(1+O\left(\frac{1+|\xi|^3}{\sqrt{L_n}}\right)\right) \\
&\quad + O\left(k^{-1/2}n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k\right), \qquad (57)
\end{aligned}$$

uniformly for $|\theta| \le \pi/2 - \varepsilon$. Substituting this in (25) and using the expansions

$$n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k = n^2\left(p^2+q^2\right)^k e^{-\xi^2/2+O(|\xi|^3 L_n^{-1/2})},$$
$$e^{\xi_0^2/2}\Phi(\xi_0) = e^{\xi^2/2}\Phi(\xi)\left(1+O\left(|\xi|^3 L_n^{-1/2}\right)\right),$$

we deduce (53) when $\xi < 0$.

The restriction that $\xi < 0$ can now be removed by continuity (when $\xi_0 = 0$ the integral path has to be properly indented) or by a similar analysis. This proves (53).

One can easily check, by (56), that the asymptotic estimate (53) coincides with the two estimates (44) and (50) when $\xi \to -\infty$ and $\xi \to \infty$, respectively. □

Remark. The appearance of the normal distribution function is typical when a saddle-point coalesces with a simple pole; see [83]. Also, the polynomial order (4) of $\mu_{n,k}$ now follows from (35), (50), and (53).

3.7. The range where the expected profile grows unbounded. An important consequence of the preceding results is the following characterization of the range where $\mu_{n,k} \to \infty$, which also will be seen to be the range where $B_{n,k}$ is asymptotically normally distributed.

THEOREM 5. *Define*

$$m_0 := \left\lceil \frac{1}{p/q-1} \right\rceil \quad \textit{and} \quad \alpha_3 := \frac{2}{\log\frac{1}{p^2+q^2}}.$$

Then $\mu_{n,k} \to \infty$ *iff*

$$\alpha_1\left(L_n - LLL_n - \log m_0 + m_0 \log\left(\frac{p}{q}\right) - \frac{LLL_n - K_n}{m_0 LL_n}\right) \le k \le \alpha_3\left(L_n - K_n\right)$$

as $n \to \infty$.

Proof. Consider first the upper bound. If $k \le \alpha_3 L_n - x$, then

$$n^2(p^2+q^2)^k \ge (p^2+q^2)^{-x},$$

which tends to infinity if $x \to \infty$; on the other hand, if $k \ge \alpha_3 L_n - x$, then the reverse inequality holds and the right-hand side remains bounded if x is less than a positive constant.

For the lower bound, we use the estimate (24). First, if $k \le k_0 = \alpha - 1$ $(L_n - LLL_n + \log(p/q-1))$ (see (20)), then

$$\mu_{n,k} = \Theta(q^k n e^{-q^k n}) = o(1).$$

Next, if $k_{m-1} \le k \le k_m$, $m \ge 1$, then by (24)

$$\mu_{n,k} = \Theta(S_{n,k,m}) = \Theta(L_n^{m-e^\eta/(p/q-1)} LL_n),$$

where k is written in the form (31). Since $\eta \in [0, \log(p/q)]$, we have

$$m - \frac{p}{p-q} \le m - \frac{e^\eta}{p/q-1} \le m - \frac{q}{p-q}.$$

Also, by the definition of m_0, we have the inequalities

$$m_0 - 1 < \frac{q}{p-q} = \frac{1}{p/q-1} \le m_0.$$

Thus if $m \le m_0 - 1$, then

$$m - \frac{e^\eta}{p/q-1} \le m - \frac{q}{p-q} < m - m_0 + 1 \le 0,$$

implying that $\mu_{n,k} \to 0$ for $k \le k_{m_0-1}$. Similarly, since

$$m_0 \le \frac{p}{p-q} < m_0 + 1,$$

we have

$$m - \frac{e^\eta}{p/q-1} \ge m - \frac{p}{p-q} > m - m_0 - 1 \ge 0,$$

when $m \ge m_0 + 1$. Therefore, $\mu_{n,k} \to \infty$ if $k \ge k_{m_0}$ (and remains in the range $k \le \alpha_1(L_n - LLL_n + K_n)$).

The remaining range is $k_{m_0-1} \le k \le k_{m_0}$ in which $\mu_{n,k} = \Theta(L_n^{m_0 - e^\eta/(p/q-1)} LL_n)$, where

$$k = \alpha_1(L_n - LLL_n + \log(p/q-1) + m_0 \log(p/q) - \eta).$$

We distinguish three cases: (i) if

$$\eta \ge \log m_0 + \log(p/q-1) + \frac{LLL_n + K_n}{m_0 LL_n},$$

then $\mu_{n,k} = \Theta(L_n^{m - e^\eta/(p/q-1)} LL_n)$ and

$$L_n^{m - e^\eta/(p/q-1)} LL_n \le e^{-K_n} \to 0;$$

(ii) if

$$\eta = \log m_0 + \log(p/q-1) + \frac{LLL_n + x}{m_0 LL_n},$$

then

$$\mu_{n,k} \sim S_{n,k,m_0} \sim \frac{\alpha_1^{m_0}}{(m_0-1)!}\, e^{-x},$$

uniformly for $x = O(1)$; and

(iii) if

$$\eta \le \log m_0 + \log(p/q-1) + \frac{LLL_n - K_n}{m_0 LL_n},$$

then $\mu_{n,k} = \Theta(L_n^{m - e^\eta/(p/q-1)} LL_n)$ and

$$L_n^{m - e^\eta/(p/q-1)} LL_n \ge e^{K_n} \to \infty.$$

Thus $\mu_{n,k}$ is bounded away from zero and infinity in the second case.

This proves the theorem when k lies in ranges (I) and (IV). The remaining cases follow easily from (35) and (53). □

Let $\{x\}$ denote the fractional part of x. The lower bound can be further refined as follows.

COROLLARY 3. *Let*

$$\hat{k} := \alpha_1 \left(L_n - LLL_n - \log m_0 + m_0 \log(p/q) - \frac{LLL_n}{m_0 LL_n} \right), \tag{58}$$

where $m_0 = \lceil 1/(p/q-1) \rceil$. *Then* (i) $\mu_{n,k} \to \infty$ *for* $\lceil \hat{k} \rceil \le k \le \alpha_3(L_n - K_n)$; (ii) $\mu_{n,k} \to 0$ *for* $k \le \lceil \hat{k} \rceil - 2$; *and* (iii)

$$\mu_{n,\lceil \hat{k} \rceil - 1} \begin{cases} = \Theta(1) & \text{if } \{\hat{k}\} = O(LL_n^{-1}), \\ \to 0 & \text{otherwise.} \end{cases}$$

Proof. The proof is similar to that of Theorem 5. We consider only the last case. First write

$$\lceil \hat{k} \rceil - 1 = \hat{k} - \{\hat{k}\} = \alpha_1 \left(L_n - LLL_n + \log(p/q - 1) + m_0 \log(p/q) - \eta' \right),$$

where

$$\eta' = \log m_0 + \log \left(\frac{p}{q} - 1 \right) + \{\hat{k}\}/\alpha_1 + \frac{LLL_n}{m_0 LL_n}.$$

(We assume that $\hat{k}$ is not an integer.) Then we follow the same proof as above by distinguishing three cases. In particular, the case when $\hat{k}$ is an integer is also covered by the bounded case. □

The result is to be compared with Pittel's result in [65], which says that the probability that the shortest path equals either $\langle \kappa_n \rangle$ or $\langle \kappa_n \rangle + 1$ tends to 1, where $\langle x \rangle$ denotes the nearest integer to x and

$$\kappa_n = \alpha_1 \left(L_n - LLL_n - \log \max_{j \ge 1} j(q/p)^j \right).$$

Note that

$$-\log \max_{j \ge 1} j(q/p)^j = -\log m_0 + m_0 \log(p/q).$$

Our result is slightly more precise; see section 8.

3.8. Refinement of $\mu_{n,k}$ by de-Poissonization. All expansions for $\mu_{n,k}$ that we have derived so far are in terms of slowly decreasing powers of L_n^{-1} or LL_n^{-1}, which will turn out to be insufficient for the asymptotics of the variance because of cancellation of dominant terms. Thus in this section we derive a more effective expansion for $\mu_{n,k}$ in terms of $\tilde{M}_k(n)$ and its higher derivatives; namely, we derive an expression of the form (19). The major difference here is that we do not substitute the asymptotic expansions for $\tilde{M}_k(n)$ into the Cauchy integral representation for $\mu_{n,k}$, resulting in a less explicit asymptotic approximation to $\mu_{n,k}$ but with a much better error term.

We start with a lemma in which we again use $k_0 = \alpha_1(L_n - LLL_n + \log(p/q-1))$.

LEMMA 4. *Define*

$$\rho_0 := \begin{cases} q^k n & \text{if } 1 \le k \le k_0, \\ \rho & \text{if } \rho \ge 1 \text{ and } k \ge k_0, \\ 1 & \text{if } \rho \le 1, \end{cases} \tag{59}$$

where ρ is given by (36). *Then*

$$\tilde{M}_k^{(\ell)}(ne^{i\theta}) = O\left(\rho_0^\ell n^{-\ell}\tilde{M}_k(n)\right), \tag{60}$$

uniformly for $\theta = o(LL_n^{-1/2})$.

Proof. If $\ell \geq 1$, then, by (8),

$$\begin{aligned}\tilde{M}_k^{(\ell)}(z) &= \sum_{0\leq j<k}\binom{k-1}{j}(p^j q^{k-1-j})^\ell \tilde{M}_1^{(\ell)}(z)\\ &= \sum_{0\leq j<k}\binom{k-1}{j}(p^j q^{k-j})^{\ell+1} z e^{-p^j q^{k-j}z}\left(1+O\left(|z|^{-1}\right)\right)\end{aligned}$$

as $|z|\to\infty$ and $\Re(z)>0$. If $1\leq k\leq k_0$ (see (20)), then a proof similar to (and simpler than) that of (30) shows that

$$\tilde{M}_k^{(\ell)}(ne^{i\theta}) = O\left(q^{k(\ell+1)} n e^{-q^k n\cos\theta}\right) = O\left(\rho_0^\ell n^{-\ell}\tilde{M}_k(n)\right),$$

uniformly for $\theta = o(LL_n^{-1/2})$. If $k_{m-1}\leq k\leq k_m$, where $m\geq 1$, then the proof of (30) is also easily amended and we obtain

$$\tilde{M}_k^{(\ell)}(ne^{i\theta}) = O\left(k^m (p^m q^{k-m})^{\ell+1} n e^{-p^m q^{k-m} n\cos\theta}\right) = O\left(LL_n^\ell n^{-\ell}\tilde{M}_k(n)\right),$$

uniformly for $\theta = o(LL_n^{-1/2})$. Note that $\rho = O(LL_n)$ when $k_{m-1}\leq k\leq k_m$, $m\geq 1$. For the proof of (60) in the remaining ranges of k, we use the integral representation

$$\tilde{M}_k^{(\ell)}(z) = \frac{(-1)^\ell}{2\pi i}\int_{(\rho)} s(s+1)\cdots(s+\ell-1) z^{-s-\ell}\Gamma(s+1) g(s)\left(p^{-s}+q^{-s}\right)^k \mathrm{d}s$$

and a simpler analysis than that given above for $\tilde{M}_k(z)$. In particular, when k lies in the saddle-point range (II) and $\rho\geq 1$, we have, by the same analysis used for (46),

$$\begin{aligned}&\tilde{M}_k^{(\ell)}(ne^{i\theta})\\ &= O\left(n^{-\rho-\ell}\sum_{j\in\mathbb{Z}}|\rho+it_j|^\ell\,|\Gamma(\rho+1+it_j)|\int_{|t-t_j|\leq \pi/\log(p/q)}\left|(p^{-\rho-it}+q^{-\rho-it})^k\right|\mathrm{d}t\right)\\ &= O\left(k^{-1/2}(q/p)^{\rho/2}(p^{-\rho}+q^{-\rho})^k n^{-\rho-\ell}\rho^\ell\sum_{j\in\mathbb{Z}}|1+it_j|^\ell|\Gamma(\rho+1+it_j)|\right)\\ &= O\left(k^{-1/2}(q/p)^{\rho/2}(p^{-\rho}+q^{-\rho})^k n^{-\rho-\ell}\rho^\ell\right)\\ &= O\left(\rho^\ell n^{-\ell}\tilde{M}_k(n)\right),\end{aligned}$$

uniformly for $|\theta|\leq \pi/2-\varepsilon$. The other cases are treated similarly. Alternatively, we can use the estimates (48), (51), and (57) for $\tilde{M}_k(ne^{i\theta})$ and the integral formula

$$\tilde{M}_k^{(\ell)}(z) = \frac{\ell!}{2\pi i}\int_{|w-z|\leq\varepsilon|z|/\rho_0}\frac{\tilde{M}_k(w)}{(w-z)^{\ell+1}}\,\mathrm{d}w,$$

following a standard analysis (referred to as Ritt's theorem in [61, pp. 9–10]). □

An application of Proposition 1 (analytic de-Poissonization) and the above lemma leads to our refinement.

THEOREM 6. *If* $q^{2k}n \to 0$, *then*

$$\mu_{n,k} = \tilde{M}_k(n) - \frac{n}{2}\,\tilde{M}_k''(n) + O\left(\rho_0^4 n^{-2}\tilde{M}_k(n)\right), \tag{61}$$

where ρ_0 *is given in* (59).

Proof. By (26) and (60), we can take $\delta(n) = \rho_0/n$, which is $o(n^{-1/2})$ if $q^{2k}n \to 0$. □

Remark. The condition that $q^{2k}n \to 0$ is also necessary for $\mu_{n,k} \sim \tilde{M}_k(n)$ because otherwise $\mu_{n,k} \sim q^k n(1-q^k)^{n-1}$, which is not asymptotically equivalent to $\tilde{M}_k(n)$. Note also that (61) and (60) imply that $\mu_{n,k} = \tilde{M}_k(n)\left(1 + O(\rho_0^2 n^{-1})\right)$.

4. Variance of the external profile. Asymptotic approximations to $\sigma_{n,k}^2 := \mathbb{V}(B_{n,k})$ are derived in this section. It turns out that the variance is of the same order as the mean in all ranges, implying that the standard deviation is small; therefore we expect asymptotic normality when the variance tends to infinity with n. The calculations here are more involved due to the cancellation of dominant orders of $\mu_{n,k}^2$. The key idea is a suitable manipulation of the corresponding de-Poissonized approximations for the mean and the second moments.

4.1. Recurrence and generating functions of the second moment. Let $\nu_{n,k} := \mathbb{E}(B_{n,k}^2)$ denote the second moment of $B_{n,k}$. By (1), we have the recurrence

$$\nu_{n,k} = \sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j}\left(\nu_{j,k-1} + \nu_{n-j,k-1}\right) + \omega_{n,k}$$

for $n, k \ge 1$ with $\nu_{n,0} = \delta_{n,1}$, where

$$\omega_{n,k} := 2\sum_{0\le j\le n} \binom{n}{j} p^j q^{n-j}\mu_{j,k-1}\mu_{n-j,k-1}.$$

Generating functions. Let $N_k(z) := \sum_n \nu_{n,k} z^n/n!$. Then $N_k(z)$ satisfies

$$N_k(z) = e^{qz}N_{k-1}(pz) + e^{pz}N_{k-1}(qz) + \omega_k(z) \qquad (k \ge 2)$$

with $N_1(z) = 2pqz^2 + M_1(z)$, where $\omega_k(z) := 2M_{k-1}(pz)M_{k-1}(qz)$. It follows that the Poisson generating function $\tilde{N}_k(z) := e^{-z}N_k(z)$ satisfies

$$\tilde{N}_k(z) = \tilde{N}_{k-1}(pz) + \tilde{N}_{k-1}(qz) + \tilde{\omega}_k(z),$$

where $\tilde{\omega}_k(z) = 2\tilde{M}_{k-1}(pz)\tilde{M}_{k-1}(qz)$. By iterating this functional equation, we obtain

$$\tilde{N}_k(z) = \sum_{0\le \ell<k} \binom{k-1}{\ell} \tilde{N}_1(p^\ell q^{k-1-\ell}z) + \sum_{0\le m\le k-2}\ \sum_{0\le \ell\le m} \binom{m}{\ell}\tilde{\omega}_{k-m}(p^\ell q^{m-\ell}z)$$

for $k \ge 2$.

Regularity of $N_k(z)$. The following estimate is useful in justifying the application of the saddle-point method and the de-Poissonization procedure.

LEMMA 5. *Let $z = re^{i\theta}$, where $r \geq 0$ and $|\theta| \leq \pi$. Then the estimate*

$$|N_k(z)| \leq N_k(r)e^{-cr\theta^2} \tag{62}$$

holds for $r \geq 0$ and $|\theta| \leq \pi$ for some constant c independent of r, k, and θ.

Proof. We start from

$$N_k(z) = e^z \sum_{0\leq \ell<k} \binom{k-1}{\ell} \tilde{N}_1(p^\ell q^{k-1-\ell} z) + \omega_k(z) + e^z \sum_{1\leq m\leq k-2} \sum_{0\leq \ell\leq m} \binom{m}{\ell} \tilde{\omega}_{k-m}(p^\ell q^{m-\ell} z)$$

and apply Lemma 2 to the first sum. For the second term, we observe first that, by (26),

$$|\omega_k(z)| \leq 2M_{k-1}(pr)M_{k-1}(qr)e^{-cr\theta^2} = \omega_k(r)e^{-cr\theta^2},$$

uniformly for $r \geq 0$ and $|\theta| \leq \pi$. It remains to estimate the last sum

$$\left| e^z \sum_{1\leq m\leq k-2} \sum_{0\leq \ell\leq m} \binom{m}{\ell} \tilde{\omega}_{k-m}(p^\ell q^{m-\ell} z) \right|,$$

for which we apply the same argument as that used in the proof of Lemma 2, yielding an estimate of the type (28). Collecting the three parts gives (62). ∎

An auxiliary function for asymptotic variance. As a good approximation to $\mathbb{V}(B_{n,k})$, define $\tilde{V}_k(z) := \tilde{N}_k(z) - \tilde{M}_k^2(z)$. Then $\tilde{V}_k(z)$ satisfies

$$\tilde{V}_k(z) = \tilde{V}_{k-1}(pz) + \tilde{V}_{k-1}(qz) \qquad (k \geq 2),$$

which, by iteration, yields

$$\tilde{V}_k(z) = \sum_{0\leq j<k} \binom{k-1}{j} \tilde{V}_1(p^j q^{k-1-j} z), \tag{63}$$

where

$$\tilde{V}_1(z) = \tilde{M}_1(z) + 2pqz^2e^{-z} - \tilde{M}_1^2(z).$$

It follows that

$$\tilde{V}_k(z) = \frac{1}{2\pi i} \int_{(\rho)} z^{-s}\Gamma(s+1)h(s)\left(p^{-s} + q^{-s}\right)^k \mathrm{d}s, \tag{64}$$

where $\rho > -2$ and

$$h(s) := 1 - \frac{1}{p^{-s} + q^{-s}} - \frac{s+1}{p^{-s} + q^{-s}} \left(\frac{p^{-s} + q^{-s} + 1}{2^{s+2}} - \frac{2p}{(1+p)^{s+2}} - \frac{2q}{(1+q)^{s+2}} \right);$$

compare (6).

4.2. Asymptotics of $\sigma^2_{n,k}$. In this section we show that the variance $\sigma^2_{n,k} := \mathbb{V}(B_{n,k})$ is asymptotically equivalent to $\mu_{n,k}$ when k lies in range (I) and $2\mu_{n,k}$ when k lies in ranges (III) and (IV), and is of the same order as $\mu_{n,k}$ in the central range (II).

THEOREM 7. (i) *If* $1 \le k \le \alpha_1(1+o(1))L_n$, *then*

$$\sigma^2_{n,k} \sim \mu_{n,k}. \tag{65}$$

(ii) *If* $\alpha_1(L_n - LLL_n + K_n) \le k \le \alpha_2(L_n - K_n\sqrt{L_n})$, *then*

$$\sigma^2_{n,k} = G_2\left(\rho; \log_{p/q} p^k n\right) \frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}} \left(1 + O\left(\frac{1}{k(p/q)^\rho} + \frac{1}{k(\rho+2)^2}\right)\right), \tag{66}$$

where $\rho = \rho(n,k) > -2$ *is given in* (36) *and*

$$G_2(\rho; x) = \sum_{j\in\mathbb{Z}} h(\rho + it_j)\Gamma(\rho + 1 + it_j)e^{-2j\pi i x} \qquad (t_j := 2j\pi/\log(p/q)).$$

(iii) *If* $k \ge \alpha_2(1-o(1))L_n$, *then*

$$\sigma^2_{n,k} \sim 2\mu_{n,k}. \tag{67}$$

Proof. Since most details are similar to those for $\mu_{n,k}$, only the key differences will be highlighted. We separate the analysis into two overlapping cases: $1 \le k \le k_0 = \alpha_1(L_n - LLL_n + \log(p/q-1))$ and $q^{2k}n \to 0$.

Consider the first case when $1 \le k \le k_0$. In this range, $\tilde{M}_k(ne^{i\theta}) \to 0$ for $\theta = o(LL_n^{-1/2})$ by (30) and (33). By (63) and the same proof of (30), we have

$$\tilde{V}_k(ne^{i\theta}) = \tilde{M}_k(ne^{i\theta})\left(1 + O\left(q^k n e^{-q^k n\cos\theta}\right)\right),$$

uniformly for $\theta = o(LL_n^{-1/2})$. Then since

$$\nu_{n,k} = n![z^n]N_k(z) = n![z^n]e^z\left(\tilde{V}_k(z) + \tilde{M}_k^2(z)\right),$$

and $\mu_{n,k} \to 0$, it is straightforward to show, by (30), (62), and the proof of (22), that $\sigma^2_{n,k} \sim \mu_{n,k}$ in this case, which establishes (65).

We now consider the range $q^{2k}n \to 0$ that will cover the other two cases. We will show that $\mathbb{V}(B_{n,k}) \sim \tilde{V}_k(n)$, which in turn will imply (66) and (67) (indeed, $|h(-2)| = 2|g(-2)| = 4pq/(p^2+q^2)$).

In this case, by the integral representation (64) and the same method of proof for $\tilde{M}_k(z)$, we have

$$\tilde{V}_k^{(\ell)}(ne^{i\theta}) = O\left(\rho_0^\ell n^{-\ell}\tilde{V}_k(n)\right), \tag{68}$$

uniformly for $\theta = o(LL_n^{-1/2})$ whenever $q^{2k}n \to 0$. On the other hand, since $\tilde{M}_k(z)$ satisfies the estimate (60), we have

$$\frac{\mathrm{d}^\ell}{\mathrm{d}z^\ell}\tilde{M}_k^2(z)\Big|_{z=ne^{i\theta}} = O\left(\rho_0^\ell n^{-\ell}\tilde{M}_k^2(n)\right) \qquad (\ell = 0, 1, \dots),$$

uniformly for $\theta = o(LL_n^{-1/2})$. Thus $\tilde{N}_k(z) = \tilde{V}_k(z) + \tilde{M}_k^2(z)$ also satisfies condition (15) of Proposition 1. Therefore, by (19) of Proposition 1 we have

$$\nu_{n,k} = \tilde{N}_k(n) - \frac{n}{2}\,\tilde{N}_k''(n) + O\left(\rho_0^4 n^{-2}\tilde{N}_k(n)\right)$$

for $k \geq k_0$. Note that $\tilde{N}_k(n) = \Theta(\mu_{n,k}^2)$ when $\mu_{n,k} \to \infty$, but $\tilde{N}_k(n) = \Theta(\mu_{n,k})$ when $\mu_{n,k} \to 0$.

On the other hand, by (61),

$$\mu_{n,k}^2 = \tilde{M}_k^2(n) - n\tilde{M}_k(n)\tilde{M}_k''(n) + O\left(\rho_0^4 n^{-2}\tilde{M}_k^2(n)\right).$$

Therefore

$$\sigma_{n,k}^2 = \tilde{V}_k(n)\left(1 + O(\rho_0^2 n^{-1}\mu_{n,k})\right),$$

whenever $q^{2k}n \to 0$. Note that the O-term is at most of order $LL_n^2 L_n^{-1/2}$. In fact, a further refinement (see (16 or [37]) shows that

$$\sigma_{n,k}^2 = \tilde{V}_k(n) - n\tilde{M}_k'(n)^2 - \frac{n}{2}\,\tilde{V}_k''(n) + O\left(\rho_0^4 n^{-2}\tilde{N}_k(n)\right).$$

It remains to derive asymptotic approximations to $\tilde{V}_k(n)$, which follow the same methods of proof used for $\tilde{M}_k(n)$, the only difference being changing all occurrences of $g(s)$ to $h(s)$. In particular, $G_2(\rho; x) \sim G_1(\rho; x)$ when $\rho \to \infty$, which corresponds to $k \leq \alpha_1(1 + o(1))L_n$; also $|h(-2)| = 2|g(-2)| = 4pq/(p^2 + q^2)$. This proves (66) and (67). □

We conclude this section with two corollaries.

COROLLARY 4. *As $n \to \infty$, the variance $\sigma_{n,k}^2 \to \infty$ iff the mean $\mu_{n,k} \to \infty$.*

COROLLARY 5. *If $\mu_{n,k} \to \infty$, then $B_{n,k}/\mu_{n,k} \to 1$ in probability.*

Proof. The proof follows from Theorem 7 and Chebyshev's inequality. □

5. Limiting distribution. We prove in this section that the limiting distribution of $B_{n,k}$ is normal if $\sigma_{n,k} \to \infty$ and is Poisson if the variance remains bounded. Since the mean and the variance are asymptotically of the same order, the conditions can also be stated by replacing $\sigma_{n,k} \to \infty$ by $\mu_{n,k} \to \infty$. These results cover the range when $k \geq \alpha_1(L_n - LL_n + O(1))$ and $k \leq \alpha_2(L_n + O(1))$. Outside this range, $\mu_{n,k} \to 0$, so $B_{n,k} \to 0$ in probability.

THEOREM 8. (i) *If $\sigma_{n,k} \to \infty$, then*

$$\frac{B_{n,k} - \mu_{n,k}}{\sigma_{n,k}} \xrightarrow{d} \mathcal{N}(0,1), \tag{69}$$

where $\mathcal{N}(0,1)$ denotes a standard normal random variable and $\xrightarrow{d}$ stands for convergence in distribution.

(ii) *If $\sigma_{n,k} = \Theta(1)$, then*

$$\begin{cases} \mathbb{P}\left(B_{n,k} = 2m\right) = \dfrac{\lambda_0^m}{m!}\,e^{-\lambda_0} + o(1), \\ \mathbb{P}\left(B_{n,k} = 2m+1\right) = o(1), \end{cases} \tag{70}$$

uniformly for (finite) $m \geq 0$, where $\lambda_0 := pqn^2(p^2+q^2)^{k-1}$.

Note that (70) implies that $B_{n,k}$ takes asymptotically only even numbers when the mean is bounded. This indeed holds in the wider range when

$$k \geq \frac{3}{\log(1/(p^3+q^3))}(L_n + K_n).$$

Intuitively, this is the range where $\binom{n}{j}(p^j+q^j)^k \to 0$ for all $j \geq 3$, where $\binom{n}{j}(p^j+q^j)^k$ is the expected number of groups of j input-strings having common prefixes of length at least k; since $\sum_{j\geq 3}\binom{n}{j}(p^j+q^j)^k \to 0$, all nodes appearing at levels $\geq k$ are most likely only in pairs (see [32] for more precise local limit theorems for $B_{n,k}$).

Let $\tilde{\sigma}_{n,k} := \sqrt{\tilde{V}_k(n) - n\tilde{M}_k'(n)^2}$ (see Theorem 7). We will prove, by extending the above de-Poissonization procedure, that

$$\mathbb{E}\exp\left(\frac{B_{n,k} - \tilde{M}_k(n)}{\tilde{\sigma}_{n,k}}\, i\varphi\right) = e^{-\varphi^2/2}\left(1 + O\left(\frac{1+|\varphi|^3}{\sigma_{n,k}}\right)\right), \tag{71}$$

uniformly for $\varphi = o(\sigma_{n,k}^{1/5})$, which implies (69) by Lévy's continuity theorem since $\mu_{n,k} \sim \tilde{M}_k(n)$ and $\sigma_{n,k} \sim \tilde{\sigma}_{n,k}$ when $\mu_{n,k} \to \infty$. Note that as far as the central limit theorem is concerned, the validity of (71) in the range $\varphi = O(1)$ suffices; observe also that centering $B_{n,k}$ by the exact mean or normalizing $B_{n,k}$ by the exact variance will result in a poorer error term.

Our method of proof of (71) is roughly as follows. We start by deriving a closed-form expression for the bivariate generating function $\mathcal{P}_k(z,y) = \sum_{n\geq 0} P_{n,k}(y)z^n/n!$ by using the recurrence (1). We then will apply the Cauchy integral representation to prove (71), for which we need, as in the analytic de-Poissonization used above, a crude estimate for $|\mathcal{P}_k(ne^{i\theta}, e^{i\varphi})|$ for $|\theta|$ away from zero, as well as a more precise local expansion when $|\theta|$ is very close to zero. The proof for the Poisson limit law (70) is similar.

5.1. An exact expression for $\mathcal{P}_k(z,y)$. By (1), we have the functional equation

$$\mathcal{P}_k(z,y) = \mathcal{P}_{k-1}(pz,y)\mathcal{P}_{k-1}(qz,y) \qquad (k \geq 2)$$

with

$$\mathcal{P}_1(z,y) = e^z + (y-1)z\left(p(e^{qz}-1) + q(e^{pz}-1)\right) + pq(y-1)^2z^2.$$

By iterating this functional equation, we obtain

$$\mathcal{P}_k(z,y) = \prod_{0\leq j<k} \mathcal{P}_1(p^jq^{k-1-j}z, y)^{\binom{k-1}{j}} \qquad (k \geq 1). \tag{72}$$

This expression, although explicit, is less transparent from an asymptotic viewpoint.

5.2. A uniform estimate for $|\mathcal{P}_k(re^{i\theta}, y)|$. We first prove a uniform bound on $|\mathcal{P}_k(re^{i\theta}, y)|$ that is necessary for the proof of Theorem 8.

PROPOSITION 3. *Uniformly for $k \geq 1$, $r \geq 0$, $|\theta| \leq \pi$ and $|y| = 1$,*

$$|\mathcal{P}_k(re^{i\theta}, y)| \leq e^{r - cr\theta^2} \tag{73}$$

for some constant $c > 0$ independent of k, r, and θ.

In order to prove the above proposition we need a lemma.

LEMMA 6. *If $z = re^{i\theta}$, where $r \geq 0$ and $|\theta| \leq \pi$, then*

$$|e^z - 1 - z| \leq (e^r - 1 - r)e^{-c_1 r\theta^2}, \tag{74}$$

where $c_1 := 2/(3\pi^2)$. On the other hand, if $r \geq r_0$, where $r_0 \approx 1.37$ solves the equation $e^r - r = e^{r/3} + 1$, then

$$|e^z - z| \leq (e^r - r)e^{-c_1 r\theta^2/2} \qquad (|\theta| \leq \pi). \tag{75}$$

Proof. The first inequality is a special case of Pittel's inequality (see [65])

$$\left| e^z - \sum_{0\leq j<m} \frac{z^j}{j!} \right| \leq \left(e^r - \sum_{0\leq j<m} \frac{r^j}{j!} \right) e^{-2r\theta^2/(\pi^2(m+1))} \qquad (r \geq 0; |\theta| \leq \pi).$$

A simple proof of (74) (following [65]) is as follows:

$$\begin{aligned} |e^z - 1 - z| &= |e^{z/3}| \left| e^{2z/3} - (1+z)e^{-z/3} \right| \\ &= e^{r\cos(\theta)/3} \left| \sum_{j\geq 2} \frac{z^j}{j!3^j} \left(2^j + (-1)^j(3j-1)\right) \right| \\ &\leq e^{r\cos(\theta)/3}(e^{2r/3} - (1+r)e^{-r/3}), \end{aligned}$$

since $2^j + (-1)^j(3j-1) \geq 0$ for $j \geq 2$. Thus (74) follows from (29).

For the proof of inequality (75), we have

$$\begin{aligned} |e^z - z| &\leq |e^z - 1 - z| + 1 \\ &\leq (e^r - 1 - r)e^{-c_1 r\theta^2} + 1 \\ &\leq (e^r - r)e^{-c_1 r\theta^2/2}, \end{aligned}$$

since the last inequality is equivalent to

$$1 - e^{-c_1 r\theta^2} \leq (e^r - r)e^{-c_1 r\theta^2/2} \left(1 - e^{-c_1 r\theta^2/2}\right),$$

or $e^{c_1 r\theta^2/2} + 1 \leq e^r - r$, which follows from our choice of r in view of the inequalities $e^{c_1 r\theta^2/2} + 1 \leq e^{r/3} + 1 \leq e^r - r$. □

Proof of Proposition 3. We separate the proof into two cases: $r \leq r_0$ and $r \geq r_0$, where we recall that $r_0 \approx 1.37$ solves the equation $e^r - r = e^{r/3} + 1$. In the first case, we use the expansion

$$\mathcal{P}_1(z, y) = 1 + z + \frac{z^2}{2}\left(1 - 2pq(1-y^2)\right) + \sum_{j\geq 2} \frac{z^j}{j!}\left(1 - j(pq^{j-1} + qp^{j-1})(1-y)\right),$$

which yields

$$\begin{aligned} |\mathcal{P}_1(re^{i\theta}, e^{i\varphi})| &\leq |1 + re^{i\theta}| + \sum_{j\geq 2} \frac{r^j}{j!} \\ &\leq e^r - \frac{2r\theta^2}{\pi^2(1+r)} \\ &\leq e^{r - c_2 r\theta^2}, \end{aligned} \tag{76}$$

uniformly for $0 \le r \le r_0$ and $|\theta| \le \pi$, where we used again (29) and $c_2 := 2/(\pi^2(1+r_0)^2 e^{r_0})$.

Assume now $r \ge r_0$. We can write $\mathcal{P}_1(z,y)$ as follows:

$$\mathcal{P}_1(z,y) = a_1(pz)a_1(qz) + z + (qza_2(pz) + pza_2(qz))y + pqz^2y^2,$$

where $a_1(z) := e^z - z$ and $a_2(z) := e^z - 1 - z$. Note that $\mathcal{P}_1(z,1) = e^z$. By applying the two inequalities (74) and (75), we have

$$\begin{aligned}\left|\mathcal{P}_1(re^{i\theta}, e^{i\varphi})\right| &\le a_1(pr)a_1(qr)e^{-c_1 r\theta^2/2} + r + qra_2(pr)e^{-c_1 pr\theta^2} \\ &\quad + pra_2(qr)e^{-c_1 qr\theta^2} + pqr^2 \\ &\le \left(e^r - r - pqr^2\right) e^{-c_1 qr\theta^2} + r + pqr^2 \\ &\le e^{r - c_1 qr\theta^2/2}, \qquad (77)\end{aligned}$$

the last inequality following from an argument similar to the proof of (75). Indeed, it is equivalent to

$$(r + pqr^2)(e^{c_1 qr\theta^2/2} + 1) \le e^r,$$

but the left-hand side is less than $(r + r^2/4)(e^{r/6} + 1)$, which is in turn less than e^r for $r \ge 0.99$.

Collecting the two inequalities (76) and (77), we obtain

$$|\mathcal{P}_1(re^{i\theta}, e^{i\varphi})| \le e^{r - cr\theta^2} \qquad (c = \min\{c_1, c_2\}),$$

uniformly for $r \ge 0$ and $|\theta| \le \pi$. This implies (73) by (72). ∎

5.3. Local expansion of $\mathcal{P}_k(re^{i\theta}, e^{i\varphi})$. Recall that $\theta_0 := n^{-2/5}$ and ρ_0 is defined in (59).

PROPOSITION 4. *Assume that $\mu_{n,k} \to \infty$. Then uniformly for $|\theta| \le \theta_0$ and $\varphi = o(\sigma_{n,k}^{-2/3})$*

$$\mathcal{P}_k(ne^{i\theta}, e^{i\varphi}) = \exp\left(n - \frac{n}{2}\theta^2 + \tilde{M}_k(n)i\varphi - n\tilde{M}_k'(n)\varphi\theta - \frac{\tilde{V}_k(n)}{2}\varphi^2 + O(E_4)\right), \qquad (78)$$

where

$$E_4 := n|\theta|^3 + \rho_0^2\sigma_{n,k}^2|\varphi|\theta^2 + \rho_0\sigma_{n,k}^2\varphi^2|\theta| + \sigma_{n,k}^2|\varphi|^3.$$

Proof. Define

$$Q(z,y) := \log e^{-z}\mathcal{P}_1(z,y) = \log\left(1 + a_3(z)(y-1) + a_4(z)(y-1)^2\right),$$

where $a_3(z) := pze^{-pz} + qze^{-qz} - ze^{-z}$ and $a_4(z) := pqz^2e^{-z}$. Let

$$Q_k(z,y) := \sum_{0 \le j < k} \binom{k-1}{j} Q(p^j q^{k-1-j} z, y) = \log e^{-z}\mathcal{P}_k(z,y).$$

First, we prove in Lemma 7 of Appendix B that $\mathcal{P}_1(re^{i\theta}, e^{i\varphi})$ is away from zero for $r \ge 0$ and $|\theta| \le \varepsilon$, implying that $Q_k(z,y)$ is well defined when $|\arg(z)| \le \varepsilon$.

Then since $\mu_{n,k} \to \infty$, we need only consider $k \geq k_0$. To that purpose, we start from the expansion

$$Q(z,y) = \begin{cases} pq(y^2-1)z^2 + O(|y-1||z|^3) & \text{as } z \to 0, \\ q(y-1)ze^{-qz}\left(1+O(e^{-(p-q)\Re(z)})\right) & \text{as } z \to \infty, |\arg(z)| \leq \varepsilon. \end{cases} \tag{79}$$

By (79), we have

$$Q_k(z,y) = \frac{1}{2\pi i}\int_{(\rho)} z^{-s} Q^*(s,y)(p^{-s}+q^{-s})^{k-1}\mathrm{d}s, \tag{80}$$

where $\rho > -2$ and $Q^*(s,y) := \int_0^\infty z^{s-1}Q(z,y)\mathrm{d}z$ is well defined for $\Re(s) > -2$. Note that

$$Q(z,y) = a_3(z)(y-1) + \frac{2a_4(z)-a_3(z)^2}{2}(y-1)^2 + \bar{Q}(z,y)(y-1)^3,$$

where by Taylor's remainder formula

$$\bar{Q}(z,y) := \int_0^1 (1-t)^2(a_3(z)+2a_4(z)(y-1)t) \times \frac{(a_3(z)^2 - 3a_4(z) + a_3(z)a_4(z)(y-1)t + a_4(z)^2(y-1)^2t^2)}{(1+a_3(z)(y-1)t+a_4(z)(y-1)^2t^2)^3}\,\mathrm{d}t.$$

The exact form is of less importance here; we need instead the estimates $\bar{Q}(z,y) = O(|z|^4) = O(|z|^2)$ as $z \to 0$ and

$$\bar{Q}(z,y) = O\left(|z|^3 e^{-3q\Re(z)}\right) = O\left(|z|e^{-q\Re(z)}\right)$$

as $z \to \infty$ in the sector $\{z : |\arg(z)| \leq \varepsilon\}$. This expansion gives

$$Q_k(z,y) = \tilde{M}_k(z)(y-1) + \frac{\tilde{V}_k(z)-\tilde{M}_k(z)}{2}(y-1)^2 + \bar{Q}_k(z,y)(y-1)^3,$$

where

$$\bar{Q}_k(z,y) := \sum_{0\leq j<k}\binom{k-1}{j}\bar{Q}(p^j q^{k-1-j}z, y).$$

An application of Lemma 8 presented in Appendix C yields, with $z = ne^{i\theta}$,

$$Q_k(z,y) = \tilde{M}_k(z)(y-1) + \frac{\tilde{V}_k(z)-\tilde{M}_k(z)}{2}(y-1)^2 + O\left(|y-1|^3|\tilde{M}_k(ne^{i\theta})|\right),$$

where the O-term holds uniformly for $|\theta| \leq \varepsilon$ and $|y-1| = o(1)$. Since $\sigma_{n,k}^2 = \Theta(\mu_{n,k}) \to \infty$, this leads to (78) by expansions of $\tilde{M}_k(ne^{i\theta})$ and $\tilde{V}_k(ne^{i\theta})$ at $\theta = 0$, using the estimates (60) and (68). This completes the proof of Proposition 4. □

5.4. Proof of Theorem 8. We are now ready to prove Theorem 8.

Proof of the central limit theorem (69). By Cauchy's integral formula and the two estimates (73) and (78), we have, similar to (25),

$$\begin{aligned}\mathbb{E}\left(e^{B_{n,k}i\varphi}\right) &= \frac{n!}{2\pi i}\int_{|z|=n} z^{-n-1}\mathcal{P}_k(z,y)\mathrm{d}z\\ &= \frac{n!n^{-n}}{2\pi}\, e^{n+\tilde{M}_k(n)i\varphi-\tilde{V}_k(n)\varphi^2/2}\\ &\quad\times\int_{-\theta_0}^{\theta_0} e^{-n\theta^2/2-n\tilde{M}'_k(n)\varphi\theta}\left(1+O(E_4)\right)\mathrm{d}\theta + O\left(n^{-1/10}e^{-cn^{1/5}}\right),\end{aligned}$$

since $E_4\to 0$ in the range of integration and when $\varphi = o(\sigma_{n,k}^{-4/5})$. Applying Stirling's formula, extending the integration limits to $\pm\infty$, and making the change of variables $\theta\mapsto\theta n^{-1/2}$, we obtain

$$\begin{aligned}\mathbb{E}\left(e^{B_{n,k}i\varphi}\right) &= \frac{e^{\tilde{M}_k(n)i\varphi-\tilde{\sigma}^2_{n,k}\varphi^2/2}}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-(\theta+\sqrt{n}\,\tilde{M}'_k(n)\varphi)^2/2}\\ &\quad\times\left(1+O\left(\frac{1+|\theta|^3}{\sqrt{n}}+\frac{\theta^2}{n}\,\rho_0^2\sigma^2_{n,k}|\varphi|+\frac{|\theta|}{\sqrt{n}}\,\rho_0\sigma^2_{n,k}\varphi^2+\sigma^2_{n,k}|\varphi|^3\right)\right)\mathrm{d}\theta,\end{aligned}$$

uniformly in φ. A straightforward evaluation of the integral gives (71). This completes the proof of (69).

Proof of the Poisson limit theorem (70). The proof of (70) is similar to the previous proof but proceeds slightly differently. We first show that

$$Q_k(ne^{i\theta},y) := \log e^{-z}\mathcal{P}_k(z,y) = \lambda_0(y^2-1)e^{2i\theta} + O\left(|y-1|n^{-e(p)}\right), \tag{81}$$

uniformly for $|y|=1$ and $|\theta|\le\varepsilon$, where $e(p) := 2\log(p^3+q^3)/\log(p^2+q^2)-3\in(0,1)$ for $p\in(1/2,1)$.

This follows from the Mellin inversion integral (80) since the Mellin transform $Q^*(s,y)$ has a simple pole at $s=-2$ with residue $pq(y^2-1)$ and can be meromorphically continued into the whole s-plane. Indeed, by moving the line of integration of the integral in (80) to $\Re(s)=-3-\varepsilon$, we obtain

$$Q_k(ne^{i\theta},y) = \lambda_0(y^2-1)e^{2i\theta} + O\left(|y-1|n^3(p^3+q^3)^k\right),$$

whenever $|\theta|\le\pi/2-\varepsilon$ and

$$k\ge\frac{L_n+K_n}{\log((p^2+q^2)/(p^3+q^3))}.$$

Since $\mu_{n,k}=\Theta(1)$, we know that $k=\alpha_3L_n+O(1)$, and for such values of k, we have $n^3(p^3+q^3)^k=\Theta(n^{-e(p)})$. Thus (81) follows.

By (81) and the same choice of θ_0 and (73), we then deduce that

$$\begin{aligned}\mathbb{E}\left(e^{B_{n,k}i\varphi}\right) &= \frac{e^{\lambda_0(e^{2i\varphi}-1)}}{\sqrt{2\pi}}\int_{-n^{1/10}}^{n^{1/10}} e^{-\theta^2/2}\left(1+O\left(n^{-e(p)}|\varphi|\right)\right)\\ &\quad\times\left(1+\frac{12\lambda_0(e^{2i\varphi}-1)i\theta-i\theta^3}{6\sqrt{n}}+O\left(\frac{1+\lambda_0|\varphi|\theta^2+\theta^6}{n}\right)\right)\mathrm{d}\theta\\ &= e^{\lambda_0(e^{2i\varphi}-1)}\left(1+O(E_5)\right),\end{aligned}$$

uniformly for $|\varphi| \le \pi$, where $E_5 := |\varphi| n^{-e(p)} + (1+\lambda_0|\varphi|)n^{-1}$. Thus

$$\mathbb{P}\left(\frac{B_{n,k}}{2} = m\right) = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-im\varphi+\lambda_0(e^{i\varphi}-1)}\,(1+O(E_5))\,\mathrm{d}\varphi,$$

from which the even case in (70) follows since, by (50), $\mu_{n,k} \sim 2\lambda_0$. The odd case is similar. □

Remark. Note that λ_0 is periodic in nature since $k = \alpha_3 L_n + O(1) \in \mathbb{Z}$; indeed, we can write

$$k = \lfloor \alpha_3 L_n \rfloor + \ell = \alpha_3 L_n + \ell - \{\alpha_3 L_n\} \qquad (\ell \in \mathbb{Z}),$$

so that

$$n^2(p^2+q^2)^k = \exp\left(-\frac{2}{\alpha_3}\,(\ell - \{\alpha_3 L_n\})\right).$$

This is why we did not state the Poisson convergence (70) in the usual form: if $\lambda_0 \to \lambda < \infty$, then $B_{n,k}$ converges in distribution to $2\mathrm{Po}(\lambda)$, where $\mathrm{Po}(\lambda)$ denotes a Poisson variate with parameter λ.

6. The internal profile. We consider the internal profile in this section. All asymptotic results follow the same footsteps as in the proof we used for $B_{n,k}$; details will thus be omitted. The main differences are that $\mathbb{E}(I_{n,k})$ and $\mathbb{V}(I_{n,k})$ are not of the same order for all ranges of k, and $I_{n,k}$ assumes both odd and even values when $k = \alpha_3 L_n + O(1)$. These are intuitively clear since most levels close to the root are full and internal nodes do not necessarily appear in pairs near the fringes of the tree.

Let $P^{[I]}_{n,k}(y) = \mathbb{E}(y^{I_{n,k}})$ be the probability generating function of $I_{n,k}$. Then

$$P^{[I]}_{n,k}(y) = \sum_{0\le j\le n}\binom{n}{j} p^j q^{n-j} P^{[I]}_{j,k-1}(y) P^{[I]}_{n-j,k-1}(y) \qquad (n \ge 2; k \ge 1), \tag{82}$$

with the initial conditions

$$P^{[I]}_{n,k}(y) = \begin{cases} y & \text{if } n \ge 2 \text{ and } k = 0,\\ 1 & \text{if } n \le 1 \text{ and } k \ge 0.\end{cases}$$

From this, we obtain, defining $\mathcal{P}^{[I]}_k(z,y) := \sum_n P^{[I]}_{n,k}(y) z^n/n!$,

$$\mathcal{P}^{[I]}_k(z,y) = \prod_{0\le j\le k} \mathcal{P}^{[I]}_0(p^j q^{k-j} z, y)^{\binom{k}{j}} \qquad (k \ge 0), \tag{83}$$

with $\mathcal{P}^{[I]}_0(z,y) = ye^z + (1-y)(1+z)$. This suggests that we consider

$$\bar{I}_{n,k} := 2^k - I_{n,k},$$

so that the bivariate generating function $\mathcal{P}^{[\bar{I}]}_k(z,y) := \sum_n P^{[\bar{I}]}_{n,k}(y) z^n/n!$ is given by

$$\mathcal{P}^{[\bar{I}]}_k(z,y) = \prod_{0\le j\le k} \mathcal{P}^{[\bar{I}]}_0(p^j q^{k-j} z, y)^{\binom{k}{j}} \qquad (k \ge 0),$$

with $\mathcal{P}^{[\bar{I}]}_0(z,y) = e^z + (y^{-1}-1)(1+z)$.

6.1. Expected internal profile. We state without proof the asymptotics of $\mathbb{E}(I_{n,k})$ in this subsection. By (82) or (83), we deduce that the Poisson generating function

$$\tilde{M}_k^{[I]}(z) := e^{-z} \sum_{n\geq 0} \frac{\mathbb{E}(I_{n,k})}{n!} z^n$$

satisfies

$$\tilde{M}_k^{[I]}(z) = 2^k - \sum_{0\leq j\leq k} \binom{k}{j} \tilde{M}_0^{[I]}(p^j q^{k-j} z) \tag{84}$$

$$= 2^k - \frac{1}{2\pi i} \int_{(\rho)} z^{-s}(s+1)\Gamma(s)(p^{-s}+q^{-s})^k \mathrm{d}s, \tag{85}$$

where $\rho > 0$ and $\tilde{M}_0^{[I]}(z) = (1+z)e^{-z}$. Thus, in particular,

$$\mathbb{E}(I_{n,k}) = 2^k - \sum_{0\leq j\leq k} \binom{k}{j} \left(1 + p^j q^{k-j}(n-1)\right) \left(1 - p^j q^{k-j}\right)^{n-1}.$$

Due to the presence of 2^k or the simple pole at $s = 0$ in (85), we have an additional phase transition for $\mathbb{E}(I_{n,k})$ at $\rho = 0$ or, equivalently, at $k \sim \alpha_0 L_n$, where

$$\alpha_0 := \frac{2}{\log(1/p) + \log(1/q)}.$$

We now list asymptotic approximations of $\mathbb{E}(I_{n,k})$ for various ranges of k (without proofs since they follow the same lines as the derivations presented above for the external profile).

Asymptotics of $\mathbb{E}(I_{n,k})$ *when* $1 \leq k \leq \alpha_1(1+o(1))L_n$. Since

$$\tilde{M}_1^{[I]}(z) = 2^k - (1+pz)e^{-pz} + (1+qz)e^{-qz} = 2^k - qze^{-qz}\left(1 + O(|z|^{-1})\right)$$

as $|z| \to \infty$ in the sector $|\arg(z)| \leq \pi/2 - \varepsilon$, we see immediately that in this range

$$\mathbb{E}(I_{n,k}) = 2^k - \mathbb{E}(B_{n,k})(1+o(1)), \tag{86}$$

uniformly in k.

Asymptotics of $\mathbb{E}(I_{n,k})$ *when* $\alpha_1(L_n - LLL_n + K_n) \leq k \leq \alpha_0(L_n - K_n\sqrt{L_n})$. By applying the saddle-point method to the Mellin inversion integral in (85) and then de-Poissonizing, we deduce that in this range

$$\mathbb{E}(I_{n,k}) = 2^k - G_3\left(\rho; \log_{p/q} p^k n\right) \frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}} \left(1 + O\left(\frac{1}{k(p/q)^\rho} + \frac{1}{k\rho^2}\right)\right),$$

where $\rho = \rho(n,k) > 0$ satisfies the saddle-point equation (36), $\beta_2(\rho)$ is the same as in (37), and

$$G_3(\rho; x) = \sum_{j\in\mathbb{Z}} (\rho + 1 + it_j)\Gamma(\rho + it_j)e^{-2j\pi i x},$$

where $t_j := 2j\pi/\log(p/q)$.

Asymptotics of $\mathbb{E}(I_{n,k})$ *when* $k = \alpha_0(L_n + o(L_n^{2/3}))$. In this range, we write

$$k = \alpha_0(L_n + \xi\sqrt{\alpha_0\beta_2(0)L_n}),$$

where $\alpha_0\beta_2(0) = 2(\log(1/p)+\log(1/q))/\log(p/q)^2$ and $\xi = o(L_n^{1/6})$. The same uniform asymptotic analysis we used for proving (53) gives

$$\mathbb{E}(I_{n,k}) = 2^k\Phi(-\xi)\left(1 + O\left(\frac{1+|\xi|^3}{\sqrt{L_n}}\right)\right),$$

uniformly in ξ, where $\Phi(x)$ denotes the standard normal distribution function.

Asymptotics of $\mathbb{E}(I_{n,k})$ *when* $\alpha_0(L_n + K_n\sqrt{L_n}) \le k \le \alpha_2(L_n - K_n\sqrt{L_n})$. The same saddle-point method and de-Poissonization procedure yield

(87)
$$\mathbb{E}(I_{n,k}) = G_3\left(\rho; \log_{p/q} p^k n\right)\frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}}\left(1 + O\left(\frac{1}{k(p/q)^\rho} + \frac{1}{k(\rho+2)^2}\right)\right),$$

with ρ, $\beta_2(\rho)$, and G_3 as defined above.

Asymptotics of $\mathbb{E}(I_{n,k})$ *when* $k = \alpha_2(L_n + o(L_n^{2/3}))$. In this case, we write

$$k = \alpha_0(L_n + \xi\sqrt{\alpha_2\beta_2(-2)L_n}),$$

and we have

$$\mathbb{E}(I_{n,k}) = \frac{1}{2}\Phi(\xi)n^2(p^2+q^2)^k\left(1 + O\left(\frac{1+|\xi|^3}{\sqrt{L_n}}\right)\right),$$

uniformly for $\xi = o(L_n^{1/6})$.

Asymptotics of $\mathbb{E}(I_{n,k})$ *when* $k \ge \alpha_2(L_n + K_n\sqrt{L_n})$. In this case, the simple pole at $s = -2$ in the integrand of (85) dominates, and we have

$$\mathbb{E}(I_{n,k}) = \frac{1}{2}n^2(p^2+q^2)^k\left(1 + O\left(K_n^{-1}e^{-K_n^2/2+O(K_n^3L_n^{-1/2})}\right)\right)$$

as $n \to \infty$.

6.2. Asymptotics of $\mathbb{V}(I_{n,k})$. Since $\mathbb{V}(I_{n,k}) = \mathbb{V}(\bar{I}_{n,k})$, we can apply the same analysis used for proving Theorem 7 to derive asymptotic approximations to $\mathbb{V}(I_{n,k})$. The auxiliary function we need is

$$\tilde{V}_k^{[I]}(z) := e^{-z}\sum_{n\ge0}\frac{\mathbb{E}(\bar{I}_{n,k}^2)}{n!}z^n - \left(e^{-z}\sum_{n\ge0}\frac{\mathbb{E}(\bar{I}_{n,k})}{n!}z^n\right)^2,$$

which satisfies

$$\tilde{V}_k^{[I]}(z) = \sum_{0\le j\le k}\binom{k}{j}\tilde{V}_0^{[I]}(p^jq^{k-j}z) \qquad (k \ge 0), \tag{88}$$

where $\tilde{V}_0^{[I]}(z) = (1+z)e^{-z}(1-(1+z)e^{-z})$. Thus we have

$$\tilde{V}_k^{[I]}(z) = \frac{1}{2\pi i}\int_{(\rho)} z^{-s}(s+1)\Gamma(s)\left(1-2^{-s}-s2^{-s-2}\right)\left(p^{-s}+q^{-s}\right)^k\mathrm{d}s,$$

where $\rho > -2$.

Asymptotics of $\mathbb{V}(I_{n,k})$ *when* $1 \le k \le \alpha_1(1+o(1))L_n$. In this range, we have

$$\mathbb{V}(I_{n,k}) \sim \mathbb{V}(B_{n,k}) \sim \mathbb{E}(B_{n,k}).$$

Asymptotics of $\mathbb{V}(I_{n,k})$ *when* $\alpha_1(L_n - LLL_n + K_n) \le k \le \alpha_2(L_n - K_n\sqrt{L_n})$. We have

$$\mathbb{V}(I_{n,k}) = G_4\left(\rho; \log_{p/q} p^k n\right) \frac{n^{-\rho}\left(p^{-\rho}+q^{-\rho}\right)^k}{\sqrt{2\pi\beta_2(\rho)k}} \left(1 + O\left(\frac{1}{k(p/q)^\rho} + \frac{1}{k(\rho+2)^2}\right)\right),$$

where $\rho = \rho(n,k) > -2$ satisfies the saddle-point equation (36) and

$$G_4(\rho; x) = \sum_{j\in\mathbb{Z}} (\rho + 1 + it_j)\Gamma(\rho + it_j)\left(1 - 2^{-\rho-it_j} - (\rho+it_j)2^{-\rho-2-it_j}\right) e^{-2j\pi i x}.$$

Asymptotics of $\mathbb{V}(I_{n,k})$ *when* $k \ge \alpha_2(L_n + K_n\sqrt{L_n})$. In this case, the simple pole at $s = -2$ again dominates, and we have

$$\mathbb{V}(I_{n,k}) \sim \mathbb{E}(I_{n,k}).$$

Observe that, unlike for the external profile, the variance of the internal profile is asymptotically equivalent to the mean of the internal profile near the height of a trie.

From these asymptotic estimates and from Chebyshev's inequality, we see that $I_{n,k}/\mathbb{E}(I_{n,k}) \to 1$ in probability if $\mathbb{E}(I_{n,k}) \to \infty$; see [15].

6.3. Limiting distributions. The same limiting Gaussian–Poisson behavior for $B_{n,k}$ holds for $I_{n,k}$. We state formally our main result for the internal profile in the following theorem. The proof is indeed simpler than that for Theorem 8 since the base function $\mathcal{P}_0^{[I]}(z,y)$ has a simpler form than $\mathcal{P}_0(z,y)$.

THEOREM 9. (i) *If* $\mathbb{V}(I_{n,k}) \to \infty$, *then*

$$\frac{I_{n,k} - \mathbb{E}(I_{n,k})}{\sqrt{\mathbb{V}(I_{n,k})}} \xrightarrow{d} \mathcal{N}(0,1).$$

(ii) *If* $\mathbb{V}(I_{n,k}) = \Theta(1)$, *then, with* $\lambda_1 := n^2(p^2+q^2)/2$,

$$\mathbb{P}(I_{n,k} = m) = \frac{\lambda_1^m}{m!} e^{-\lambda_1} + o(1) \tag{89}$$

for all $m \ge 0$.

The theorem states that asymptotic normality (in the sense of convergence in distribution) holds as long as

$$\lceil \hat{k} \rceil \le k \le \alpha_3 L_n - K_n$$

for any sequence $K_n \to \infty$, where $\hat{k}$ is defined in (58).

On the other hand, $I_{n,k}$ is asymptotically Poisson distributed when $k = \alpha_3 L_n + O(1)$. A result related to (89) was given in [65] by a method of moments, as a key step in deriving the asymptotic distribution of the height.

7. Profiles under the unbiased Bernoulli model. All exact expressions we have derived up to now, as well as most asymptotic approximations, also hold when $p = q = 1/2$. The major difference is reflected by the fact that $\alpha_1 = \alpha_2$ (see Figure 2), so that the saddle-point range between α_1 and α_2 does not exist, and most of the analysis we give above becomes much simpler. For simplicity of presentation, we omit all error terms in our asymptotic estimates.

Expected external profile. By (8), the Poisson generating function of $\mathbb{E}(B_{n,k})$ is given exactly by

$$\tilde{M}_k(z) = z\left(e^{-z/2^k} - e^{-z/2^{k-1}}\right) \qquad (k \geq 1). \tag{90}$$

From this we deduce, by our de-Poissonization procedures, that

$$\mathbb{E}(B_{n,k}) \sim \begin{cases} n\left(1-2^{-k}\right)^{n-1} & \text{if } 2^{-k}n \to \infty, \\ \tilde{M}_k(n) & \text{if } 4^{-k}n \to 0, \end{cases} \tag{91}$$

where the condition $4^{-k}n \to 0$ is due to the properties that

$$\tilde{M}_k^{(\ell)}(z) = O\left(2^{-k\ell}|\tilde{M}_k(z)|\right) \qquad (|\arg(z)| \leq \pi/2 - \varepsilon)$$

and $2^{-k} = o(n^{-1/2})$; see Proposition 1 and compare with (61). In particular,

$$\mathbb{E}(B_{n,k}) \sim \begin{cases} ne^{-t}\left(1-e^{-t}\right) & \text{if } 2^{-k}n \to t \in (0,\infty), \\ 2^{-k}n^2 & \text{if } 2^{-k}n \to 0. \end{cases}$$

Note that these approximations can also be easily derived by the exact formula

$$\mathbb{E}(B_{n,k}) = n\left(1-2^{-k}\right)^{n-1} - n\left(1-2^{1-k}\right)^{n-1}, \tag{92}$$

by (90) or (10). But such an elementary approach becomes messier for the calculation of the variance. Also

$$\max_k \mathbb{E}(B_{n,k}) \sim \frac{n}{4},$$

which is reached when $k \sim \log_2 n - 1$.

Expected internal profile. In a similar manner, we have, by (84),

$$\tilde{M}_k^{[I]}(z) = 2^k - (2^k + z)e^{-z/2^k} \qquad (k \geq 0).$$

Therefore, the expected internal profiles satisfy

$$\mathbb{E}(I_{n,k}) \sim \begin{cases} 2^k - n\left(1-2^{-k}\right)^{n-1} & \text{if } 2^{-k}n \to \infty, \\ \tilde{M}_k^{[I]}(n) & \text{if } 4^{-k}n \to 0. \end{cases}$$

Consequently,

$$\mathbb{E}(I_{n,k}) \sim \begin{cases} 2^k\left(1-(1+t)e^{-t}\right) & \text{if } 2^{-k}n \to t \in (0,\infty), \\ 2^{-k-1}n^2 & \text{if } 2^{-k}n \to 0. \end{cases}$$

Note that

$$\mathbb{E}(I_{n,k}) = 2^k\left(1-\left(1-2^{-k}\right)^n\right) - n\left(1-2^{-k}\right)^{n-1}$$

and

$$\max_k \mathbb{E}(I_{n,k}) \sim c_3 n, \tag{93}$$

where $c_3 \approx 0.298$ denotes the maximum value achieved by the function $(1-(1+x)e^{-x})/x$ for $x \in \mathbb{R}^+$.

Asymptotics of the variances. Similarly, by (63) and (88), we have

$$\tilde{V}_k(z) = z\left(e^{-z/2^k} - e^{-z/2^{k-1}}\right) + 2^{-k}z^2e^{-z/2^{k-1}} - 2^{1-k}z^2\left(e^{-z/2^k} - e^{-z/2^{k-1}}\right)^2,$$
$$\tilde{V}_k^{[I]}(z) = (2^k+z)e^{-z/2^k} - 2^k\left(1+2^{-k}\right)^2 e^{-z/2^{k-1}};$$

accordingly, if $n/2^k \to \infty$, then

$$\mathbb{V}(B_{n,k}) \sim \mathbb{V}(I_{n,k}) \sim \mathbb{E}(B_{n,k}) \sim n\left(1-2^{-k}\right)^{n-1},$$

and if $n/4^k \to 0$, then

$$\mathbb{V}(B_{n,k}) \sim \tilde{V}_k(n) \quad \text{and} \quad \mathbb{V}(I_{n,k}) \sim \tilde{V}_k^{[I]}(n),$$

uniformly in k. These approximations imply that

$$\mathbb{V}(B_{n,k}) \sim \begin{cases} ne^{-t}\left(1-(1+t)e^{-t}+2te^{-2t}(2-e^{-t})\right) & \text{if } 2^{-k}n \to t \in (0,\infty), \\ 2\mathbb{E}(B_{n,k}) \sim 2^{1-k}n^2 & \text{if } 2^{-k}n \to 0 \end{cases} \tag{94}$$

and

$$\mathbb{V}(I_{n,k}) \sim \begin{cases} 2^k(1+t)e^{-t}\left(1-(1+t)e^{-t}\right) & \text{if } 2^{-k}n \to t \in (0,\infty), \\ \mathbb{E}(I_{n,k}) \sim 2^{-k-1}n^2 & \text{if } 2^{-k}n \to 0. \end{cases}$$

Limiting distributions. Both Theorems 8 and 9 (asymptotic normality of $B_{n,k}$ and $I_{n,k}$, respectively) hold when $p = q = 1/2$ by the same method of proof. Note that both bivariate generating functions become simpler (see (72) and (83)):

$$\mathcal{P}_k(z,y) = \left(e^{z/2^{k-1}} + (y-1)\frac{z}{2^{k-1}}\left(e^{z/2^k}-1\right) + (y-1)^2\frac{z^2}{4^k}\right)^{2^{k-1}},$$
$$\mathcal{P}_k^{[I]}(z,y) = \left(ye^{z/2^k} + (1-y)\left(1+\frac{z}{2^k}\right)\right)^{2^k}.$$

Observe that, as $n \to \infty$, $\mathbb{E}(B_{n,k}) \to \infty$ iff $\mathbb{V}(B_{n,k}) \to \infty$ iff $\mathbb{V}(I_{n,k}) \to \infty$ iff

$$\frac{1}{\log 2}\left(L_n - LL_n + \frac{K_n}{L_n}\right) \le k \le \frac{2}{\log 2}(L_n - K_n) \tag{95}$$

for any sequence $K_n \to \infty$ with n; compare Theorem 5 for the asymmetric case.

THEOREM 10. (i) *If k lies in the range* (95)*, then*

$$\frac{B_{n,k} - \mathbb{E}(B_{n,k})}{\sqrt{\mathbb{V}(B_{n,k})}} \xrightarrow{d} \mathcal{N}(0,1), \quad \frac{I_{n,k} - \mathbb{E}(I_{n,k})}{\sqrt{\mathbb{V}(I_{n,k})}} \xrightarrow{d} \mathcal{N}(0,1).$$

(ii) *If $k = 2(L_n + O(1))/\log 2$, then, with $\lambda_2 := 2^{-k-1}n^2$,*

$$\begin{cases} \mathbb{P}(B_{n,k} = 2m) = \dfrac{\lambda_2^m}{m!}e^{-\lambda_2} + o(1), \quad \mathbb{P}(B_{n,k} = 2m+1) = o(1), \\ \mathbb{P}(I_{n,k} = m) = \dfrac{\lambda_2^m}{m!}e^{-\lambda_2} + o(1), \end{cases}$$

uniformly for $m \ge 0$.

Note that when $p = q$, $\lambda_0 = \lambda_1 = \lambda_2$.

8. Applications of results. In this section, we briefly discuss a few properties of some shape characteristics of random tries, as implied either by our results or by our approaches. We consider only depth, height, shortest path, fill-up level, width, and right-profile.

Depth. The distribution of the depth D_n is given by $\mathbb{P}(D_n = k) = \mu_{n,k}/n$. Our asymptotic approximations for $\mu_{n,k}$ give very precise results for the distribution of D_n. First consider the case when $p \neq q$. By definition, we see that the result (4) for the limiting behaviors of $\log \mu_{n,k}/\log n$ also describes those of $-1 + \log \mathbb{P}(D_n = k)/\log n$, or essentially the large deviations of the distribution of D_n.

Furthermore, (43) can be regarded as a local limit theorem for D_n. Indeed, we have, for $k = \mathrm{h}^{-1}(L_n + x\sqrt{\mathrm{h}^{-1}\beta_2(-1)L_n})$, where $\mathrm{h} := p\log(1/p) + q\log(1/q)$ is the entropy rate,

$$\mathbb{P}(D_n = k) = G_1\left(-1; \log_{p/q} p^k n\right) \frac{e^{-x^2/2}}{\sqrt{2\pi \mathbb{V}(D_n)}} \left(1 + O\left(\frac{1+|x|^3}{\sqrt{L_n}}\right)\right), \tag{96}$$

uniformly for $x = o(L_n^{1/6})$, where $\mathbb{V}(D_n) \sim (\mathrm{h}_2 - \mathrm{h}^2)/\mathrm{h}^3 \log n$, with $\mathrm{h}_2 := p\log^2 p + q\log^2 q$ (see [33, 76]), is also rederived below in (97). Because of the appearance of the uncommon periodic function G_1, we see that D_n *satisfies a central limit theorem but not a local limit theorem (of the usual form).* It can be shown that the right-hand side indeed sums (over all k) asymptotically to 1. The result (96) is new.

If $p = q$, then, by the exact formula (92), we have

$$\mathbb{P}(D_n = k) = \left(1 - 2^{-k}\right)^{n-1} - \left(1 - 2^{1-k}\right)^{n-1},$$

which implies that

$$\mathbb{P}(D_n = \lfloor \log_2 n \rfloor + \ell) = \left(e^{-2^{-\ell+\{\log_2 n\}}} - e^{-2^{1-\ell+\{\log_2 n\}}}\right)\left(1 + O\left(n^{-1}2^{-\ell}\right)\right),$$

uniformly for $\ell \in \mathbb{Z}$, where $\{x\}$ denotes the fractional part of x.

On the other hand, if one is interested in the cumulative distribution functions or tail probabilities, then, by (6) and by partial summation,

$$\mathbb{P}(D_n \leq k) = (n-1)![z^n]\frac{e^z}{2\pi i}\int_{(\rho)} z^{-s}\Gamma(s+1)\left(p^{-s} + q^{-s}\right)^k \mathrm{d}s$$

for $k \geq 1$, where $\rho > -1$. Equivalently, by (11), we have (see [35])

$$\mathbb{P}(D_n \leq k) = \frac{1}{2\pi i}\int_{(\rho)} \frac{\Gamma(n)\Gamma(s+1)}{\Gamma(n+1+s)}\left(p^{-s} + q^{-s}\right)^k \mathrm{d}s,$$

where $\rho > -1$. Asymptotics of such integrals can be treated by our approaches, which give not only the central limit theorem of D_n with convergence rate (since there is a simple pole at $s = -1$) but also precise estimates for tail probabilities. Indeed, we have

$$\mathbb{P}\left(D_n \leq \mathrm{h}^{-1}(L_n + x\sqrt{\mathrm{h}^{-1}\beta_2(-1)L_n})\right) = \Phi(x)\left(1 + O\left(\frac{1+|x|^3}{\sqrt{L_n}}\right)\right),$$

uniformly for $x = o(L_n^{1/6})$, as already shown in [33, 35] (but without rate). Furthermore,

$$-\frac{\log \mathbb{P}(D_n \le \alpha L_n)}{\log n} \to \rho' + 1 - \alpha \log(p^{-\rho'} + q^{-\rho'}) \qquad (\alpha_1 \le \alpha \le \mathrm{h}^{-1}),$$
$$-\frac{\log \mathbb{P}(D_n \ge \alpha L_n)}{\log n} \to \begin{cases} \rho' + 1 - \alpha \log(p^{-\rho'} + q^{-\rho'}) & \text{if } \mathrm{h}^{-1} \le \alpha \le \alpha_2, \\ -1 - \alpha \log(p^2 + q^2) & \text{if } \alpha_2 \le \alpha \le \alpha_3, \end{cases}$$

both tails being asymptotic to -1 for smaller and larger α, respectively, where ρ' is given in (42). These results imply, in particular, that $\mathbb{E}(D_n) \sim L_n/\mathrm{h}$ and

$$\mathbb{V}(D_n) \sim \beta_2(-1)\mathrm{h}^{-3} L_n = \frac{pq \log^2(p/q)}{(p\log(1/p) + q\log(1/q))^3} L_n = \frac{\mathrm{h}_2 - \mathrm{h}^2}{\mathrm{h}^3} L_n, \tag{97}$$

where $\mathrm{h}_2 := p\log^2 p + q \log^2 q$; see [13, 76]. Note that the constant on the right-hand side becomes zero when $p = q$.

Width. The width of tries W_n is defined to be $W_n := \max_k I_{n,k}$, or the size of the most abundant level(s). As a natural lower bound for $\mathbb{E}(W_n)$, we consider $\max_k \mathbb{E}(I_{n,k})$. By (87) and a similar analysis for (43), we have, when $p \ne q$,

$$\mathbb{E}(I_{n,k}) = \frac{\sqrt{\mathrm{h}}\, G_3\left(-1; \log_{p/q} p^k n\right)}{\log(p/q)\sqrt{2\pi pq}} \times \frac{n}{\sqrt{L_n}} \left(1 + O(L_n^{-1/2})\right),$$

uniformly for $k = L_n/\mathrm{h} + O(1)$. This approximation, together with the estimates for $\mathbb{E}(I_{n,k})$ in other ranges given in section 6.1, yields

$$\mathbb{E}(W_n) \ge \max_k \mathbb{E}(I_{n,k}) = \Theta(nL_n^{-1/2}),$$

when $p \ne q$. Indeed, we have

$$\mathbb{E}(W_n) = \Theta(nL_n^{-1/2}).$$

The upper bound can be proved by applying the arguments used in [16], which start from the inequality

$$\mathbb{E}(W_n) \le \mathcal{M}_n + \sum_{|k - L_n/\mathrm{h}| \le \varepsilon L_n^{2/3}} \frac{\mathbb{V}(I_{n,k})}{\mathcal{M}_n - \mathbb{E}(I_{n,k})} + \sum_{|k - L_n/\mathrm{h}| > \varepsilon L_n^{2/3}} \mathbb{E}(I_{n,k}),$$

where $\mathcal{M}_n$: $\max_k \mathbb{E}(I_{n,k})$, and then using the asymptotics of $\mathbb{E}(I_{n,k})$ and $\mathbb{V}(I_{n,k})$ given in section 6.1. Details are omitted here. Finer results for $\mathbb{E}(W_n)$ can be derived, but the proof is more involved due to the presence of the periodic function G_3 (whose parameter involves k).

For symmetric tries, we easily have $\mathbb{E}(W_n) = \Theta(n)$, by (93) and the trivial bound $\mathbb{E}(W_n) \le n$. Thus *random symmetric tries are "fatter," and most nodes lie near the most abundant levels* $k = \log_2 n + O(1)$.

Height. We next derive an estimate for the height of random tries, as a consequence of our estimates for the external profiles together with the use of the first and second moment methods (see [79]).

COROLLARY 6 (height of a trie). *Let* $H_n := \max\{k : B_{n,k} > 0\}$ *be the height of a random trie. Then* $H_n/\log n \to \alpha_3$ *in probability.*

Proof. Let $k_H := \alpha_3 L_n$. First we derive an upper bound for H_n as follows:

$$\begin{aligned}\mathbb{P}(H_n > (1+\varepsilon)k_H) &\le \mathbb{P}(B_{n,k} \ge 1) \text{ (for some } k \ge (1+\varepsilon)k_H)\\ &\le \mathbb{E}(B_{n,k}) \to 0,\end{aligned}$$

where the last inequality follows from Theorem 3 when $p \neq q$ and (91) when $p = q$. For the lower bound, we use the second moment method (see [79]) to find

$$\begin{aligned}\mathbb{P}(H_n < (1-\varepsilon)k_H) &\le \mathbb{P}(B_{n,\lceil(1-\varepsilon)k_H\rceil} = 0)\\ &\le \frac{\mathbb{V}(B_{n,\lceil(1-\varepsilon)k_H\rceil})}{(\mathbb{E}(B_{n,\lceil(1-\varepsilon)k_H\rceil}))^2}\\ &= O\left(\frac{1}{\mathbb{E}(B_{n,\lceil(1-\varepsilon)k_H\rceil})}\right) \to 0\end{aligned}$$

by Theorems 3 and 7 and (94). Combining the two estimates, we obtain the required result. □

Corollary 6 is not new and has already been derived in Devroye [12], Pittel [64, 65], and Szpankowski [77].

Shortest path. The shortest path $S_n := \min\{j : B_{n,j} > 0\}$ of a random trie, discussed next, has attracted much less attention than the height (see [79]) in the literature. It is closely related to the behaviors of the external profile in range (I) near $k = \alpha_1(L_n - LLL_n + O(1))$ as discussed in Theorem 1 and its refinement in Corollary 3.

Define

$$\hat{k} := \begin{cases} \alpha_1\left(L_n - LLL_n - \log m_0 + m_0 \log(p/q) - \dfrac{LLL_n}{m_0 LL_n}\right) & \text{if } p \neq q,\\ \alpha_1(L_n - LL_n) & \text{if } p = q,\end{cases}$$

where $m_0 := \lceil 1/(p/q-1)\rceil$, and

$$k_S := \begin{cases} \lceil \hat{k} \rceil & \text{if } p \neq q,\\ \lfloor \hat{k} \rfloor & \text{if } p = q.\end{cases}$$

Corollary 7 (shortest path length of tries). *If $p \neq q$, then*

$$S_n = \begin{cases} k_S & \text{if } \{\hat{k}\}LL_n \to \infty,\\ k_S \text{ or } k_S - 1 & \text{if } \{\hat{k}\}LL_n = O(1)\end{cases}$$

with high probability;[2] *if $p = q = 1/2$, then*

$$S_n = \begin{cases} k_S + 1 & \text{if } \{\hat{k}\}L_n \to \infty,\\ k_S \text{ or } k_S + 1 & \text{if } \{\hat{k}\}L_n = O(1)\end{cases}$$

with high probability.

Proof. Assume $p \neq q$. Consider first the case $\{\hat{k}\}LL_n \to \infty$. In this case we have, by Corollary 3,

$$\begin{cases} \mu_{n,k_S} \to \infty,\\ \mu_{n,k} \to 0 \quad \text{for } \ k \le k_S - 1.\end{cases}$$

[2]We say that an event holds *with high probability* if it holds with probability tending to 1 as $n \to \infty$.

Thus, again by the second moment method,

$$\mathbb{P}(S_n > k_S) \le \mathbb{P}(B_{n,k_S} = 0) \le \frac{\mathbb{V}(B_{n,k_S})}{(\mathbb{E}(B_{n,k_S}))^2} = O\left(\frac{1}{\mu_{n,k_S}}\right) \to 0.$$

On the other hand, by using the first moment method, we have

$$\begin{aligned}\mathbb{P}(S_n < k_S) &\le \mathbb{P}(B_{n,k} \ge 1) \text{ (for some } k < k_S)\\ &\le \mu_{n,k} \to 0.\end{aligned}$$

These two estimates imply that $\mathbb{P}(S_n = k_S) \to 1$.

Now if $\{\hat{k}\}LL_n = O(1)$, then, again by Corollary 3,

$$\begin{cases} \mu_{n,k_S} \to \infty, \\ \mu_{n,k_S-1} = \Theta(1), \\ \mu_{n,k} \to 0 \text{ for } k \le k_S - 2. \end{cases}$$

Thus applying mutatis mutandis the same proof gives

$$\mathbb{P}(S_n = k_S) + \mathbb{P}(S_n = k_S - 1) \to 1.$$

The proof for the symmetric case is similar, because $\mu_{n,k} \to \infty$ when k lies in the range (95), and from this result we deduce that $\mu_{n,k_S+1} \to \infty$, $\mu_{n,k_S-1} \to 0$, and

$$\mu_{n,k_S} \begin{cases} \to 0 & \text{if } \{\hat{k}\}L_n \to \infty, \\ = \Theta(1) & \text{if } \{\hat{k}\}L_n = O(1). \end{cases}$$

This completes the proof. □

Fill-up level. We now consider the fill-up level $F_n = \max\{k : I_{n,k} = 2^k\}$ of a random trie, which was also analyzed previously by Devroye [12], Pittel [64, 65], and Knessl and Szpankowski [49].

COROLLARY 8 (fill-up level of a trie). *If $p \ne q$, then*

$$F_n = \begin{cases} k_S - 1 & \text{if } \{\hat{k}\}LL_n \to \infty, \\ k_S - 2 \text{ or } k_S - 1 & \text{if } \{\hat{k}\}LL_n = O(1) \end{cases}$$

with high probability; if $p = q = 1/2$, then

$$F_n = \begin{cases} k_S & \text{if } \{\hat{k}\}L_n \to \infty, \\ k_S \text{ or } k_S - 1 & \text{if } \{\hat{k}\}L_n = O(1). \end{cases}$$

Proof. Observe that

$$F_n = \max\{k : \bar{I}_{n,k} = 0\} = \min\{k : \bar{I}_{n,k} > 0\} - 1.$$

By (86), we have $\mathbb{E}(\bar{I}_{n,k}) \sim \mu_{n,k}$ when $k \le \alpha_1(1 + o(1))L_n$. Thus the proof of Corollary 7 applies with little modification. □

Profile enumerating only right branches. We consider the random variable $R_{n,k}$, which denotes the number of external nodes in random tries that are away from the root by k right branches. Since a right branch means a "1" in the input-string, $R_{n,k}$ enumerates the number of strings with exactly k 1's; it also has other concrete interpretations in splitting processes and conflict resolution algorithms. All of our

tools can be extended to $R_{n,k}$, although $R_{n,k}$ exhibits very different behaviors. For example, unlike $B_{n,k}$ or $I_{n,k}$, there is no need to distinguish between symmetric and asymmetric tries, all results being uniform in p; also, the Poisson heuristic holds for all $k \geq 0$. This example further reveals the power of our approaches.

The probability generating function $F_{n,k}(y) := \mathbb{E}(y^{R_{n,k}})$ of $R_{n,k}$ satisfies the recurrence

$$F_{n,k}(y) = \sum_{0\leq j\leq n} \binom{n}{j} p^j q^{n-j} F_{j,k-1}(y) F_{n-j,k}(y) \qquad (n \geq 2; k \geq 0),$$

with the initial conditions $F_{n,k}(y) = 1$ for $n \leq 1$ or $k < 0$ and $F_{2,1}(y) = y$. Thus the bivariate generating function $\mathcal{F}_k(z, y) := \sum_n F_{n,k}(y) z^n/n!$ satisfies

$$\mathcal{F}_k(z, y) = \mathcal{F}_k(qz, y)\mathcal{F}_{k-1}(pz, y) = \prod_{j\geq 0} \mathcal{F}_0(p^k q^j z, y)^{\binom{k+j-1}{j}},$$

where

$$\mathcal{F}_0(z, y) = e^{pz}\mathcal{F}_0(qz, y) + p(1 - p/2)(y - 1)z^2,$$

which is further solved to be

$$\mathcal{F}_0(z, y) = e^z + p(1 - p/2)(y - 1) \sum_{j\geq 0} (q^j z)^2 e^{(1-q^j)z}. \tag{98}$$

From this we deduce that the expected right-profile is given by

$$\mathbb{E}(R_{n,k}) = p(1 - p/2)n![z^n]\frac{e^z}{2\pi i} \int_{(\rho)} z^{-s}\Gamma(s + 2)\frac{p^{-ks}}{(1 - q^{-s})^{k+1}}\mathrm{d}s,$$

where $-2 < \rho < 0$. The integral is not of the same type as (6) but is similar, and our methods of proof easily extend. It has simple poles at $s = -2, -3, \ldots$ and poles of order $k+1$ at $s = 2j\pi i/\log(1/q)$, $j \in \mathbb{Z}$. Thus the asymptotics of $\mathbb{E}(R_{n,k})$ are divided into four overlapping ranges.

- If $0 \leq k = o(\log n)$, then the residues of the poles on the imaginary lines are dominant, and we have

$$\mathbb{E}(R_{n,k}) \sim p(1 - p/2)\frac{(\log p^k n)^k}{k!(\log(1/q))^{k+1}} \left(1 + \sum_{j\neq 0} \Gamma(1 + \chi_j)(p^k n)^{-\chi_j}\right),$$

  uniformly in k, where $\chi_j := 2j\pi i/\log(1/q)$.

- If $k \to \infty$ and $k \leq \alpha^*(L_n - K_n\sqrt{L_n})$, where $K_n \to \infty$ and

$$\alpha^* := \frac{1 - q^2}{(1 - q^2)\log(1/p) - q^2\log(1/q)},$$

  then by the saddle-point method

$$\mathbb{E}(R_{n,k}) \sim \frac{p(1 - p/2)q^{\rho/2}}{\sqrt{2\pi k}\log(1/q)} (p^k n)^{-\rho}(1 - q^{-\rho})^{-k} \sum_{j\in\mathbb{Z}} \Gamma(\rho + 1 + \chi_j)(p^k n)^{-\chi_j},$$

  uniformly in k, where

$$\rho = \log_{1/q} \frac{\log(p^k n)}{\log(p^k n/q^k)}.$$

- If $k = \alpha^* L_n + x\sqrt{\alpha^*(1+\alpha^*\log(p/q))(1+\alpha^*\log p)L_n}$, then

$$\mathbb{E}(R_{n,k}) \sim \frac{1}{2}\Phi(x)(p^k n)^2(1-q^2)^{-k},$$

  uniformly for $x = o(L_n^{1/6})$.
- If $k \geq \alpha^* L_n + K_n\sqrt{\alpha^*(1+\alpha^*\log(p/q))(1+\alpha\log p)L_n}$, then

$$\mathbb{E}(R_{n,k}) \sim \frac{1}{2}(p^k n)^2(1-q^2)^{-k}.$$

These results imply that, as $n \to \infty$, $\mathbb{E}(R_{n,k}) \to \infty$ iff

$$1 \leq k \leq \frac{2}{\log\frac{2-p}{p}}\, L_n - K_n,$$

where $K_n \to \infty$ with n. Note that

$$\log e^{-z}\mathcal{F}_0(z,y) = \log(1+(y-1)\tau(z)),$$

where $\tau(z) := p(1-p/2)\sum_{j\geq 0}(q^j z)^2 e^{-q^j z}$ satisfies $\tau(z) = O(|z|^2)$ as $z \to 0$, and, by Mellin transform, $\tau(z) = O(1)$ as $|z| \to \infty$ in a small sector containing the real axis. This yields, by a straightforward modification of our approaches, that $\mathbb{V}(R_{n,k}) = \Theta(\mathbb{E}(R_{n,k}))$ for all $k = k(n) \geq 0$ and that

$$\frac{R_{n,k} - \mathbb{E}(R_{n,k})}{\sqrt{\mathbb{V}(R_{n,k})}} \xrightarrow{d} \mathcal{N}(0,1)$$

whenever $\mathbb{E}(R_{n,k})$ or $\mathbb{V}(R_{n,k}) \to \infty$. Two remaining cases are $k = 0$ and $k = 2L_n/\log\frac{2-p}{p} + O(1)$. In the first case, $R_{n,0}$ by (98) is Bernoulli distributed with mean equal to $\tau(n)$, which is asymptotic to the periodic function

$$\frac{1}{\log(1/q)}\left(1 + \sum_{j\neq 0}\Gamma(2-\chi_j)n^{-\chi_j}\right),$$

and in the second case,

$$\mathbb{P}(R_{n,k} = m) = \frac{\lambda_3^m}{m!}\, e^{-\lambda_3} + o(1),$$

where $\lambda_3 := (p^k n)^2(1-q^2)^{-k}/2$.

Appendix A: Proof of Lemma 3. In this appendix, we prove Lemma 3. For part (i) let $z = ne^{i\theta}$, where $\theta = o(LL_n^{-1/2})$. By (8)

$$\begin{aligned}
\tilde{M}_k(z) &= \sum_{0\leq j<k}\binom{k-1}{j}p^j q^{k-j} z e^{-p^j q^{k-j} z}\left(1 + O\left(e^{-(p-q)p^j q^{k-1-j} n\cos\theta}\right)\right)\\
&= \sum_{0\leq j<k}\binom{k-1}{j}p^j q^{k-j} z e^{-p^j q^{k-j} z}\left(1 + O\left(e^{-(p-q)q^{k-1} n\cos\theta}\right)\right) \qquad (99)\\
&= q^k z e^{-q^k z}\left(1 + O(E_6)\right),
\end{aligned}$$

where

$$E_6 := \sum_{1\le j<k} \frac{(p\alpha_1/q)^j}{j!} L_n^j e^{-q^k n((p/q)^j-1)\cos\theta}.$$

Since $1 \le k \le k_0 - \alpha_1 K_n/LL_n$, we have

$$q^k n \ge \frac{LL_n}{p/q-1}\, e^{K_n/LL_n}.$$

It follows, by using the inequality

$$\frac{t^j-1}{t-1} \ge j \qquad (t>1; j\ge 1),$$

that

$$\begin{aligned} E_6 &\le \sum_{j\ge 1} \frac{(p\alpha_1/q)^j}{j!} L_n^{j-\frac{(p/q)^j-1}{p/q-1}\cos(\theta)e^{K_n/LL_n}} \\ &\le \sum_{j\ge 1} \frac{(p\alpha_1/q)^j}{j!} L_n^{-j(\cos(\theta)e^{K_n/LL_n}-1)} \\ &\le \sum_{j\ge 1} \frac{(p\alpha_1/q)^j}{j!} e^{-jK+O(j\theta^2 LL_n)} \\ &= O(e^{-K_n}), \end{aligned}$$

since $\theta = o(LL_n^{-1/2})$. This proves (30).

For part (ii), by (99),

$$\tilde{M}_k(z) = S_{k,m}(z)\left(1+O\left(e^{-(p-q)p^m q^{k-1-m} n\cos\theta} + E_7 + E_8\right)\right),$$

where

$$\begin{aligned} E_7 &:= \sum_{0\le j<m} \left|\frac{S_{k,j}(z)}{S_{k,m}(z)}\right| \left(1+O\left(e^{-(p-q)p^j q^{k-1-j} n\cos\theta}\right)\right), \\ E_8 &:= \sum_{m<j<k} \left|\frac{S_{k,j}(z)}{S_{k,m}(z)}\right| \left(1+O\left(e^{-(p-q)p^j q^{k-1-j} n\cos\theta}\right)\right). \end{aligned}$$

By (31), $p^m q^{k-m} n = e^\eta LL_n/(p/q-1)$. It follows, by changing j to $m-j$, that

$$\begin{aligned} E_7 &= O\left(m! \sum_{1\le j\le m} \frac{(q/p)^j}{(m-j)!}\, k^{-j} \exp\left(p^m q^{k-m} n\cos(\theta)\left(1-(q/p)^j\right)\right)\right) \\ &= O\left(m! \sum_{1\le j\le m} \frac{(q/p\alpha_1)^j}{(m-j)!} L_n^{-j+\frac{1-(q/p)^j}{p/q-1}\, e^\eta\cos\theta}\right) \\ &= O\left(m! \sum_{1\le j\le m} \frac{(q/p\alpha_1)^j}{(m-j)!} L_n^{-j(1-qe^\eta\cos\theta/p)}\right) \\ &= O\left(m_n^{-(1-qe^\eta\cos\theta/p)}\right) = O\left(me^{-K_n}\right), \end{aligned}$$

since $\eta \le \log(p/q) - K_n/LL_n$ and $\theta = o(LL_n^{-1/2})$.

Similarly,

$$\begin{aligned}
E_8 &= O\left(m!\sum_{j>m}\frac{(p/q)^{j-m}}{j!}\,k^{j-m}\exp\left(-p^m q^{k-m} n\cos(\theta)\left((p/q)^{j-m}-1\right)\right)\right)\\
&= O\left(m!\sum_{j\ge 1}\frac{(p\alpha_1/q)^j}{(j+m)!}L_n^{j-\frac{(p/q)^j-1}{p/q-1}\,e^\eta\cos\theta}\right)\\
&= O\left(m!\sum_{j\ge 1}\frac{(p\alpha_1/q)^j}{(j+m)!}L_n^{-j(e^\eta\cos\theta-1)}\right)\\
&= O\left(m^{-1}L_n^{-(e^\eta\cos\theta-1)}\right) = O\left(m^{-1}e^{-K_n}\right),
\end{aligned}$$

since $\eta \ge K_n/LL_n$. This completes the proof of (32). □

Appendix B: Well-definedness of $Q_k(z, y)$. We prove here the following lemma that is needed for the proof of Proposition 4.

LEMMA 7. *The function $Q_k(re^{i\theta}, y)$ is well defined for $r \ge 0$, $|\theta| \le \varepsilon$, and $|y| = 1$.*

Proof. We first show that

$$|a_3(r)(e^{i\varphi}-1) + a_4(r)(e^{i\varphi}-1)^2| < 1 \tag{100}$$

for $r \ge 0$ and $|y| = 1$. By direct calculation, we have

$$|a_3(r)(e^{i\varphi}-1) + a_4(r)(e^{i\varphi}-1)^2|^2 = a_3(r)^2\upsilon - a_4(r)(a_3(r)-a_4(r))\upsilon^2,$$

where $\upsilon := 2(1-\cos\varphi)$. Since

$$a_3(r) - a_4(r) \ge a_3(r) - 2a_4(r) = e^{-r}\left(pr(e^{qr}-1-qr) + qr(e^{pr}-1-pr)\right) \ge 0,$$

we have

$$|a_3(r)(e^{i\varphi}-1) + a_4(r)(e^{i\varphi}-1)^2| \le \sqrt{2}\,a_3(r).$$

By simple calculus, we have $a_3(r) < 2^{-1/2}$, which implies (100). Indeed, the inequality $a_3(r) < 2^{-1/2}$ is equivalent to

$$pre^{-pr} + qre^{-qr} - re^{-r} < 2^{-1/2} \qquad (r \ge 0),$$

and we have

$$pre^{-pr} + qre^{-qr} - re^{-r} \le \max_{r\ge 0} re^{-r}(e^{r/2}-1) \approx 0.52 < 2^{-1/2}.$$

This proves the lemma when $z = r$; then the assertion of the lemma follows from analyticity. □

Appendix C: A useful approximation. In the proof of Proposition 4 we need the following lemma.

LEMMA 8. *Let $f(z)$ be an entire function satisfying*

$$f(z) = \begin{cases} O(|z|^2) & \text{as } z \to 0,\\ O(|z|e^{-q\Re(z)}) & \text{as } z\to\infty,\ |\arg(z)| \le \bar{\varepsilon}, \end{cases} \tag{101}$$

where $\bar{\varepsilon} > 0$. *Then, uniformly for all* $k = k(n) \geq 1$ *and* $z = ne^{i\theta}$, $|\theta| \leq \bar{\varepsilon}$,

$$f_k(z) := \sum_{0 \leq j < k} \binom{k-1}{j} f(p^j q^{k-1-j} z) = \Theta(|\tilde{M}_k(z)|).$$

Proof. If $1 \leq k \leq k_0$, then it is easy to see that $f_k(z) = \Theta(|\tilde{M}_k(z)|)$ for $|\theta| \leq \bar{\varepsilon}$. When $k \geq k_0$, let $f^*(s) := \int_0^\infty x^{s-1} f(x) \mathrm{d}x$. Then $f^*(s)$ is well defined in the half-plane $\Re(s) > -2$ by (101). By the estimates in (101) and the same argument used in [23, Proposition 5], we have, assuming $\rho \geq 1$ and $t > 0$,

$$\begin{aligned} f^*(\rho + it) &= \int_0^{e^{i\bar{\varepsilon}}\infty} x^{\rho+it-1} f(x) \mathrm{d}x \\ &= e^{i\bar{\varepsilon}(\rho+it)} \int_0^\infty x^{\rho+it-1} f(xe^{i\bar{\varepsilon}}) \mathrm{d}x \\ &= O(e^{-\bar{\varepsilon}t} \int_0^1 x^{\rho+1} \mathrm{d}x) + O\left(e^{-\bar{\varepsilon}t} \int_1^\infty x^\rho e^{-qx\cos\bar{\varepsilon}} \mathrm{d}x \right) \\ &= O\left(e^{-\bar{\varepsilon}t} \rho^{-1} + e^{-\bar{\varepsilon}t} q^{-\rho} \rho^{1/2} (\rho/e)^\rho \right), \end{aligned}$$

uniformly in ρ and t. If $t < 0$, then changing $e^{i\bar{\varepsilon}}$ to $e^{-i\bar{\varepsilon}}$ gives

$$f^*(\rho + it) = O\left(e^{-\bar{\varepsilon}|t|} \rho^{-1} + e^{-\bar{\varepsilon}|t|} q^{-\rho} \rho^{1/2} (\rho/e)^\rho \right).$$

When $-2 < \rho \leq 1$, $f^*(\rho + it) = O(e^{-\bar{\varepsilon}|t|})$ for large $|t|$ by the same argument. On the other hand, by the first estimate in (101), we also have

$$f^*(s) = O\left(|s+2|^{-1} \right) \qquad (s \to 2).$$

With these estimates and the Mellin inversion integral

$$f_k(z) = \frac{1}{2\pi i} \int_{(\rho)} z^{-s} f^*(s) (p^{-s} + q^{-s})^{k-1} \mathrm{d}s,$$

we can apply the arguments used for $\tilde{M}_k(z)$ and prove that $f_k(z) = \Theta(|\tilde{M}_k(z)|)$ for $|\arg(z)| \leq \bar{\varepsilon}$. □

REFERENCES

[1] R. Aguech, N. Lasmar, and H. Mahmoud, *Distribution of inter-node distances in digital trees*, in Proceedings of the 2005 International Conference on Analysis of Algorithms, C. Martínez, ed., Discrete Mathematics and Theoretical Computer Science, Nancy, France, 2005, pp. 1–10.

[2] D. Aldous and P. Shields, *A diffusion limit for a class of randomly-growing binary trees*, Probab. Theory Related Fields, 79 (1988), pp. 509–542.

[3] B. C. Berndt, *Ramanujan's Notebooks. Part* I, Springer-Verlag, New York, 1985.

[4] J. Bourdon, M. Nebel, and B. Valleé, *On the stack-size of general tries*, Theor. Inform. Appl., 35 (2001), pp. 163–185.

[5] B. Chauvin, M. Drmota, and J. Jabbour-Hattab, *The profile of binary search trees*, Ann. Appl. Probab., 11 (2001), pp. 1042–1062.

[6] J.-D. Choi, K. Lee, A. Loginov, R. O'Callahan, V. Sarkar, and M. Sridharan, *Efficient and precise datarace detection for multithreaded object-oriented programs*, in Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2002, pp. 258–269.

[7] C. A. Christophi and H. M. Mahmoud, *The oscillatory distribution of distances in random tries*, Ann. Appl. Probab., 15 (2005), pp. 1536–1564.
[8] J. Clément, P. Flajolet, and B. Vallée, *Dynamical sources in information theory: A general analysis of trie structures*, Algorithmica, 29 (2001), pp. 307–369.
[9] R. de la Briandais, *File searching using variable length keys*, in Proceedings of the AFIPS Spring Joint Computer Conference, AFIPS Press, Reston, VA, 1959, pp. 295–298.
[10] L. Devroye, *A note on the average depth of tries*, Computing, 28 (1982), pp. 367–371.
[11] L. Devroye, *A probabilistic analysis of the height of tries and of the complexity of triesort*, Acta Inform., 21 (1984), pp. 229–237.
[12] L. Devroye, *A study of trie-like structures under the density model*, Ann. Appl. Probab., 2 (1992), pp. 402–434.
[13] L. Devroye, *Universal limit laws for depths in random trees*, SIAM J. Comput., 28 (1998), pp. 409–432.
[14] L. Devroye, *Laws of large numbers and tail inequalities for random tries and PATRICIA trees*, J. Comput. Appl. Math., 142 (2002), pp. 27–37.
[15] L. Devroye, *Universal asymptotics for random tries and PATRICIA trees*, Algorithmica, 42 (2005), pp. 11–29.
[16] L. Devroye and H.-K. Hwang, *Width and mode of the profile for some random trees of logarithmic height*, Ann. Appl. Probab., 16 (2006), pp. 886–918.
[17] L. Devroye and P. Kruszewski, *On the Horton-Strahler number for random tries*, RAIRO Inform. Théor. Appl., 30 (1996), pp. 443–456.
[18] M. Drmota and H.-K. Hwang, *Bimodality and phase transitions in the profile variance of random binary search trees*, SIAM J. Discrete Math., 19 (2005), pp. 19–45.
[19] M. Drmota and H.-K. Hwang, *Profile of random trees: Correlation and width of random recursive trees and binary search trees*, Adv. in Appl. Probab., 37 (2005), pp. 321–341.
[20] J. Fayolle and M. D. Ward, *Analysis of the average depth in a suffix tree under a Markov model*, in Proceedings of the 2005 International Conference on Analysis of Algorithms, C. Martínez, ed., Discrete Mathematics and Theoretical Computer Science, Nancy, France, 2005, pp. 95–104.
[21] J. A. Fill, H. M. Mahmoud, and W. Szpankowski, *On the distribution for the duration of a randomized leader election algorithm*, Ann. Appl. Probab., 6 (1996), pp. 1260–1283.
[22] P. Flajolet, *On the performance evaluation of extendible hashing and trie searching*, Acta Inform., 20 (1983), pp. 345–369.
[23] P. Flajolet, X. Gourdon, and P. Dumas, *Mellin transforms and asymptotics: Harmonic sums*, Theoret. Comput. Sci., 144 (1995), pp. 3–58.
[24] P. Flajolet, M. Régnier, and D. Sotteau, *Algebraic methods for trie statistics*, in Analysis and Design of Algorithms for Combinatorial Problems (Udine, 1982), Math. Stud. 109, North-Holland, Amsterdam, 1985, pp. 145–188.
[25] P. Flajolet and R. Sedgewick, *Mellin transforms and asymptotics: Finite differences and Rice's integrals*, Theoret. Comput. Sci., 144 (1995), pp. 101–124.
[26] P. Flajolet and J.-M. Steyaert, *A branching process arising in dynamic hashing, trie searching and polynomial factorization*, in Automata, Languages and Programming (Aarhus, 1982), Lecture Notes in Comput. Sci. 140, Springer-Verlag, Berlin, New York, 1982, pp. 239–251.
[27] E. Fredkin, *Trie memory*, Comm. ACM, 3 (1960), pp. 490–499.
[28] M. Fuchs, H.-K. Hwang, and R. Neininger, *Profiles of random trees: Limit theorems for random recursive trees and binary search trees*, Algorithmica, 46 (2006), pp. 367–407.
[29] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, UK, 1997.
[30] H.-K. Hwang, *Asymptotic expansions for the Stirling numbers of the first kind*, J. Combin. Theory Ser. A, 71 (1995), pp. 343–351.
[31] H.-K. Hwang, *Profiles of random trees: Plane-oriented recursive trees*, Random Structures Algorithms, 30 (2007), pp. 380–413.
[32] H.-K. Hwang, *Local Limit Theorems for the Profiles of Random Tries*, preprint, 2006.
[33] P. Jacquet and M. Régnier, *Trie partitioning process: Limiting distributions*, in CAAP '86 (Nice, 1986), Lecture Notes in Comput. Sci. 214, Springer-Verlag, Berlin, 1986, pp. 196–210.
[34] P. Jacquet and M. Régnier, *Normal limiting distribution of the size of tries*, in Performance '87 (Brussels, 1987), North-Holland, Amsterdam, 1988, pp. 209–223.
[35] P. Jacquet and W. Szpankowski, *Analysis of digital tries with Markovian dependency*, IEEE Trans. Inform. Theory, 37 (1991), pp. 1470–1475.
[36] P. Jacquet and W. Szpankowski, *Autocorrelation on words and its applications. Analysis of suffix trees by string-ruler approach*, J. Combin. Theory Ser. A, 66 (1994), pp. 237–269.

[37] P. Jacquet and W. Szpankowski, *Analytical de-Poissonization and its applications*, Theoret. Comput. Sci., 201 (1998), pp. 1–62.
[38] P. Jacquet, W. Szpankowski, and J. Tang, *Average profile of the Lempel–Ziv parsing scheme for a Markovian source*, Algorithmica, 31 (2001), pp. 318–360.
[39] S. Janson and W. Szpankowski, *Analysis of an asymmetric leader election algorithm*, Electron. J. Combin., 64 (1997), pp. 1–6.
[40] P. Kirschenhofer and H. Prodinger, *Some further results on digital search trees*, in Automata, Languages and Programming (Rennes, 1986), Lecture Notes in Comput. Sci. 226, Springer-Verlag, Berlin, 1986, pp. 177–185.
[41] P. Kirschenhofer and H. Prodinger, *b-tries: A paradigm for the use of number-theoretic methods in the analysis of algorithms*, in Contributions to General Algebra, Vol. 6, Hölder-Pichler-Tempsky, Vienna, 1988, pp. 141–154.
[42] P. Kirschenhofer and H. Prodinger, *Further results on digital search trees*, Theoret. Comput. Sci., 58 (1988), pp. 143–154.
[43] P. Kirschenhofer and H. Prodinger, *On some applications of formulae of Ramanujan in the analysis of algorithms*, Mathematika, 38 (1991), pp. 14–33.
[44] P. Kirschenhofer, H. Prodinger, and W. Szpankowski, *On the variance of the external path in a symmetric digital trie*, Discrete Appl. Math., 25 (1989), pp. 129–143.
[45] P. Kirschenhofer, H. Prodinger, and W. Szpankowski, *Analysis of a splitting process arising in probabilistic counting and other related algorithms*, Random Structures Algorithms, 9 (1996), pp. 379–401.
[46] C. Knessl, *A note on the asymptotic behavior of the depth of tries*, Algorithmica, 22 (1998), pp. 547–560.
[47] C. Knessl and W. Szpankowski, *A note on the asymptotic behavior of the heights in b-tries for b large*, Electron. J. Combin., 7 (2000), 16 pp.
[48] C. Knessl and W. Szpankowski, *Limit laws for the height in PATRICIA tries*, J. Algorithms, 44 (2002), pp. 63–97.
[49] C. Knessl and W. Szpankowski, *On the number of full levels in tries*, Random Structures Algorithms, 25 (2004), pp. 247–276.
[50] D. E. Knuth, *The Art of Computer Programming, Volume* III: *Sorting and Searching*, 2nd ed., Addison–Wesley, Reading, MA, 1998.
[51] K. Kukich, *Techniques for automatically correcting words in text*, ACM Comput. Surv., 24 (1992), pp. 377–439.
[52] G. Louchard, *Trie size in a dynamic list structure*, Random Structures Algorithms, 5 (1994), pp. 665–702.
[53] H. M. Mahmoud, *Evolution of Random Search Trees*, John Wiley & Sons, New York, 1992.
[54] M. E. Nebel, *On the Horton–Strahler number for combinatorial tries*, Theor. Inform. Appl., 34 (2000), pp. 279–296.
[55] M. E. Nebel, *The stack-size of combinatorial tries revisited*, Discrete Math. Theor. Comput. Sci., 5 (2002), pp. 1–16.
[56] M. E. Nebel, *The stack-size of tries: A combinatorial study*, Theoret. Comput. Sci., 270 (2002), pp. 441–461.
[57] R. Neininger and L. Rüschendorf, *A general limit theorem for recursive algorithms and combinatorial structures*, Ann. Appl. Probab., 14 (2004), pp. 378–418.
[58] M. Nguyen-The, *Distribution de valuations sur les arbres*, Ph.D. Thesis, LIX, École Polytechnique, Palaiseau, France, 2003.
[59] P. Nicodème, *Average profiles, from tries to suffix-trees*, in Proceedings of the 2005 International Conference on Analysis of Algorithms, C. Martínez, ed., Discrete Mathematics and Theoretical Computer Science, Nancy, France, 2005, pp. 257–266.
[60] S. Nilsson and M. Tikkanen, *An experimental study of compression methods for dynamic tries*, Algorithmica, 33 (2002), pp. 19–33.
[61] F. W. J. Olver, *Asymptotics and Special Functions*, Academic Press, New York, London, 1974.
[62] G. Park, *Profile of Tries*, Ph.D. thesis, Purdue University, West Lafayette, IN, 2006.
[63] G. Park and W. Szpankowski, *Towards a complete characterization of tries*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Vancouver, 2005, pp. 33–42.
[64] B. Pittel, *Asymptotic growth of a class of random trees*, Ann. Probab., 18 (1985), pp. 414–427.
[65] B. Pittel, *Paths in a random digital tree: Limiting distributions*, Adv. in Appl. Probab., 18 (1986), pp. 139–155.
[66] H. Prodinger, *How to select a loser*, Discrete Math., 120 (1993), pp. 149–159.
[67] S. T. Rachev and L. Rüschendorf, *Probability metrics and recursive algorithms*, Adv. in Appl. Probab., 27 (1995), pp. 770–799.

[68] M. Régnier and P. Jacquet, *New results on the size of tries*, IEEE Trans. Inform. Theory, 35 (1989), pp. 203–205.
[69] Y. Reznik, *Some results on tries with adaptive branching*, Theoret. Comput. Sci., 289 (2002), pp. 1009–1026.
[70] W. Schachinger, *On the variance of a class of inductive valuations of data structures for digital search*, Theoret. Comput. Sci., 144 (1995), pp. 251–275.
[71] W. Schachinger, *Asymptotic normality of recursive algorithms via martingale difference arrays*, Discrete Math. Theor. Comput. Sci., 4 (2001), pp. 363–397.
[72] W. Schachinger, *Concentration of size and path length of tries*, Combin. Probab. Comput., 13 (2004), pp. 763–793.
[73] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison–Wesley, Reading, MA, 1996.
[74] V. Srinivasan and G. Varghese, *Fast address lookups using controlled prefix expansions*, ACM Trans. Comput. Syst., 17 (1999), pp. 1–40.
[75] W. Szpankowski, *Average complexity of additive properties for multiway tries: A unified approach*, in TAPSOFT '87, Vol. 1 (Pisa, 1987), Lecture Notes in Comput. Sci. 249, Springer-Verlag, Berlin, 1987, pp. 13–25.
[76] W. Szpankowski, *Some results on V-ary asymmetric tries*, J. Algorithms, 9 (1988), pp. 224–244.
[77] W. Szpankowski, *On the height of digital trees and related problems*, Algorithmica, 6 (1991), pp. 256–277.
[78] W. Szpankowski, *A generalized suffix tree and its (un)expected asymptotic behaviors*, SIAM J. Comput., 22 (1993), pp. 1176–1198.
[79] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, Wiley-Interscience, New York, 2001.
[80] B. W. Watson, *Taxonomies and Toolkits of Regular Language Algorithms*, Ph.D. thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1995.
[81] M. D. Ward and W. Szpankowski, *Analysis of a randomized selection algorithm motivated by the LZ'77 scheme*, in Proceedings of the First Workshop on Analytic Algorithmics and Combinatorics (ANALCO 04), New Orleans, SIAM, Philadelphia, 2004, pp. 153–160.
[82] M. D. Ward and W. Szpankowski, *Analysis of the multiplicity matching parameter in suffix trees*, in Proceedings of the 2005 International Conference on Analysis of Algorithms, C. Martínez, ed., Discrete Mathematics and Theoretical Computer Science, Nancy, France, 2005, pp. 307–321.
[83] R. Wong, *Asymptotic Approximations of Integrals*, SIAM, Philadelphia, 2001 (corrected reprint of the 1989 original).

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1881–1911

ON FIXED-POINTS OF MULTIVALUED FUNCTIONS ON COMPLETE LATTICES AND THEIR APPLICATION TO GENERALIZED LOGIC PROGRAMS*

UMBERTO STRACCIA†, MANUEL OJEDA-ACIEGO‡, AND CARLOS V. DAMÁSIO§

Abstract. Unlike monotone single-valued functions, multivalued mappings may have zero, one, or (possibly infinitely) many minimal fixed-points. The contribution of this work is twofold. First, we overview and investigate the existence and computation of minimal fixed-points of multivalued mappings, whose domain is a complete lattice and whose range is its power set. Second, we show how these results are applied to a general form of logic programs, where the truth space is a complete lattice. We show that a multivalued operator can be defined whose fixed-points are in one-to-one correspondence with the models of the logic program.

Key words. fixed-points, multivalued functions, complete lattices, logic programming

AMS subject classifications. 47H10, 06B23, 68N17, 68Q55

DOI. 10.1137/070695976

1. Introduction. It is well known that fixed-point theorems are useful in many completely disparate and unrelated scientific branches and, thus, in computer science. Among the main fixed-point results is the Tarski theorem [47] (often called the Knaster–Tarski theorem) stating that the set of fixed-points of a monotone *single-valued* function $f\colon L \to L$, over a complete lattice $\langle L, \leq \rangle$, is a complete lattice and therefore has a least fixed-point.

The topic of this work is the overview and investigation of the fixed-points of *multivalued* functions $f\colon L \to 2^L$ (multivalued functions are also called *correspondences*, or *set-valued functions*, in the literature). Such functions naturally arise, e.g., in the specification of the semantics of nondeterministic programming languages [7, 8, 11, 18, 31, 36, 37, 44], in game theory [6, 33, 45, 53], and in disjunctive logic programming [22, 27, 32, 42, 52]; those of the latter case motivated our work. Informally, (i) in the first case, the meaning of a nondeterministic[1] program P may be seen as a function $p : S \to 2^S$, where S is the set of states a program may assume. The image of p is a finite nonempty set, as at a given step of a program execution, due to a nondeterministic statement, more than one successive state is possible. The semantics of a program is then related to the fixed-points of such functions ($s \in p(s)$); (ii) in the second case, a game is represented as a function $g : S \to 2^S$, where S is the strategy space of the involved players, and fixed-points ($s \in g(s)$) are related to the so-called Nash equilibria of the game. The image of g is a nonempty (usually finite) set, as at each step of the game, more than one incomparable strategic choice is possible; and

*Received by the editors July 2, 2007; accepted for publication (in revised form) September 10, 2008; published electronically January 9, 2009.
http://www.siam.org/journals/sicomp/38-5/69597.html
†ISTI - CNR, 56124 Pisa, Italy (straccia@isti.cnr.it).
‡Departamento de Matemática Aplicada, Universidad de Málaga, 29071 Málaga, Spain (aciego@uma.es). This author was partially supported by the Spanish Ministry of Science project TIN07-15455-C03-01 and Junta de Andalucía project P06-FQM-02049.
§CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal (cd@di.fct.unl.pt).

[1] An example of a nondeterministic statement is "π_1 or π_2" with informal semantics "execute either program π_1 or program π_2."

(iii) in the third case, models of disjunctive logic programs are related to fixed-points ($I \in T_{\mathcal{P}}(I)$) of a function $T_{\mathcal{P}} \colon \hat{L} \to 2^{\hat{L}}$, where $\hat{L}$ is the set of interpretations of a disjunctive logic program. Here, $T_{\mathcal{P}}$ is a so-called immediate consequence operator, which at each "step" provides a better approximation of the models of a disjunctive logic program. The image of $T_{\mathcal{P}}$ is a *possibly empty, nonfinite* set, as at each step of the model approximation computation, either no approximations, or a potentially infinite number of incomparable better approximations, are possible.

We point out that, in all three cases, fixed-point computations may be seen roughly as a tree, where a node is an element of the domain and the children of it are the alternative (nondeterministic) choices provided by the image of the multivalued function.

Generally, multivalued functions present the following fundamental challenge to the ordinary fixed-point approach: unlike monotone single-valued functions, it is possible that zero, one, or (infinitely) many minimal fixed-points exist.

The contribution of this work is twofold:

- We provide conditions for the existence of fixed-points and minimal fixed-points and show how to recursively obtain them in a slightly more general setting than considered so far (such as when the image of a multivalued function may be empty; see below). A summary of our main findings in described in Table 3.1. To the best of our knowledge, we have compared the results obtained with respect to all related work using similar order-theoretic approaches; when a reformulation or easier proof of a known result is presented, then appropriate credit is given.
- The results are then applied to a general form of logic programs, encompassing the disjunctive and many-valued extensions. The rules in such logic programs have the form $g(B_1, \ldots, B_k) \leftarrow f(A_1, \ldots, A_n)$, where f, g are arbitrary computable functions over a complete lattice (which acts as the truth space) and B_i and A_j are atoms. The form of the rules is sufficiently expressive to generalize all approaches that we are aware of in (monotone) many-valued logic programming. The main difference in this application to, e.g., semantics of nondeterministic programming languages and game theory, is that the image of $T_{\mathcal{P}}(I)$ may be empty or of infinite size, while in the former two cases both $p(s)$ and $g(s)$ are nonempty and finite. We show that a multivalued operator $T_{\mathcal{P}}(I)$ can be defined whose fixed-points are in one-to-one correspondence with the models of the logic program. The obtained relationship is novel and addresses some fundamental theoretical problems that have been neglected so far in the logic programming literature. We conclude by showing that our results extend current well-known results for classical disjunctive logic programs, where rules are of the form $B_1 \vee \cdots \vee B_k \leftarrow A_1 \wedge \cdots \wedge A_n$.

2. Preliminaries. We recall some basic definitions and notations.

With $\mathcal{L} = \langle L, \leq \rangle$, where $\leq$ is a partial order ($x \leq y$ may be read as "x approximates y") over the nonempty set L, we denote a *complete lattice*, with *join* (*meet*) operator $\vee$ ($\wedge$), least (greatest) element $\bot$ ($\top$).

Given $S \subseteq L$, by $\min S$ ($\max S$) we denote the set of *minimal* (*maximal*) elements in S and by $\bigwedge S$ ($\bigvee S$) the greatest lower bound (least upper bound) of S.[2] A nonempty subset S of L is a *sublattice* of L if for any x, y of S, both $x \vee y$ and $x \wedge y$ belong to S. A nonempty subset S of L is $\wedge$-*closed* ($\vee$-*closed*) if for any subset U

[2]We recall that $\bigwedge S = \bigwedge_{s \in S} s$ and $\bigvee S = \bigvee_{s \in S} s$.

of S, $\bigwedge_{x\in U} x$ ($\bigvee_{x\in U} x$) belongs to S. Note that S is $\wedge$-closed ($\vee$-closed) iff S is a complete meet semilattice (complete join semilattice). Furthermore, we say that S is *closed* if S is both $\wedge$-closed and $\vee$-closed, i.e., S is a complete sublattice of L. Given two elements $a, b \in L$ with $a \leq b$, we denote by $[a, b]$ the *interval* $\{x \in L \mid a \leq x \leq b\}$. Clearly, $\mathcal{L} = \langle [a, b], \leq \rangle$ is a complete lattice as well. Finally, by $\bar{\mathcal{L}} = \langle L, \geq \rangle$ we denote the *dual* lattice of $\mathcal{L} = \langle L, \leq \rangle$, where $x \geq y$ iff $y \leq x$. Of course, $\bar{\mathcal{L}}$ is a complete lattice as well, where $\geq$ is the reversed order of $\leq$ and $\top$ ($\bot$) is the least (greatest) element of $\bar{\mathcal{L}}$.

Two sets X and Y are *equipollent* iff there is a bijection from X to an Y. The *cardinality* $|X|$ of a set X is the least ordinal α such that there is a bijection between X and α.

We use the notation $(x_\alpha)_{\alpha\in I}$ to denote a (possibly transfinite) nonempty *sequence* of elements $x_\alpha \in L$, where I is an ordinal. We say that the sequence is *increasing* (*decreasing*) iff $x_\alpha \leq x_{\alpha+1}$ ($x_{\alpha+1} \leq x_\alpha$) for all $\alpha \in I$.

If there is an ordinal $\beta \in I$ such that $x_\beta = x_\alpha$ for all $\beta \leq \alpha \in I$, we say that $(x_\alpha)_{\alpha\in I}$ is *eventually stationary or constant*. A property we will frequently rely on is the following well-known fact.

PROPOSITION 2.1. *An increasing (decreasing) sequence $(x_\alpha)_{\alpha\in I}$ of elements $x_\alpha \in L$ with $|I| > |L|$ has the property that there is an ordinal $\beta \in I$ such that $|\beta| \leq |L|$ and $x_\beta = x_\alpha$ for all $\beta \leq \alpha \in I$ ($|S|$ is the* cardinal *of a set S).*

For ease of presentation and by abuse of terminology, under the condition of Proposition 2.1, we will say that the sequence $(x_\alpha)_{\alpha\in I}$ *converges* to x_β.

A function $f\colon L \to L$ is *monotone* iff for all $x, y \in L$, $x \leq y$ implies $f(x) \leq f(y)$. f is *inflationary* iff for all $x \in L$, $x \leq f(x)$. A *fixed-point* of f is an element $x \in L$ such that $f(x) = x$. By $Fix(f)$ we denote the set of fixed-points of f. f is $\bigvee$*-preserving* ($\bigwedge$-*preserving*) iff for all increasing (decreasing) sequences $(x_\alpha)_{\alpha\in I}$, $f(\bigvee_\alpha x_\alpha) = \bigvee_\alpha f(x_\alpha)$ ($f(\bigwedge_\alpha x_\alpha) = \bigwedge_\alpha f(x_\alpha)$). f is *limit-preserving* iff it is both $\bigvee$- and $\bigwedge$-preserving. It is easy to prove that $\bigvee$- or $\bigwedge$-preserving functions are monotone. However, a limit-preserving (in particular a monotone) function need not be inflationary.

Example 1. Consider $f : \{0, 1\} \to \{0, 1\}$ with $f(x) = 0$ for all $x \in \{0, 1\}$; then f is limit preserving and, thus, monotone, but $1 \not\leq f(1)$ and, thus, f is not inflationary. The Tarski theorem [47] establishes that a monotone function $f\colon L \to L$ has a fixed-point, the set of fixed-points of f is a complete lattice, and, thus, f has a least and a greatest fixed-point. The least (greatest) fixed-point can be obtained by transfinite iteration of f over $\bot$ ($\top$). Furthermore, let $\Phi(f) = \{x \in L\colon f(x) \leq x\}$, $\Psi(f) = \{x \in L\colon x \leq f(x)\}$, and, thus, $\top \in \Phi(f)$, while $\bot \in \Psi(f)$. Then the least fixed-point is $\bigwedge \Phi(f)$, while the greatest fixed-point is $\bigvee \Psi(f)$. If f is inflationary, then f has a fixed-point (e.g., obtained by transfinite iteration of f over $\bot$, also $\top \leq f(\top) = \top$), and $x \in \Phi(f)$ iff x is a fixed-point of f. However, inflationary functions may not have a least fixed-point.

Example 2. Consider $L = [0, 1]$ and function f with $f(0) = 1$ and for $x > 0$, $f(x) = x$. Then f is not monotone and is inflationary, all $x > 0$ are fixed-points, $\Phi(f) = \{x : x > 0\}$, $\bigwedge \Phi(f) = 0 \notin \Phi(f)$, and 0 is not a fixed-point of f.

3. Multivalued functions. Given $\mathcal{L} = \langle L, \leq \rangle$, a *multivalued function* is a function $f\colon L \to 2^L$ (if for all $x \in L$, $|f(x)| = 1$, then f is single valued). Note that we do not require $f(x) \neq \varnothing$ for all $x \in L$. We say that $x \in L$ is a *fixed-point* of f iff $x \in f(x)$. For instance, see the following example.

Example 3. Let $L = \{0, 1, 2\}$. Consider $f\colon L \to 2^L$ defined as $f(0) = \{0, 1, 2\}$, $f(1) = \{0, 1\}$, and $f(2) = \{0\}$. Then 0 and 1 are fixed-points, whereas 2 is not a fixed-point.

Furthermore, we say that f is *nonempty* (resp., $\wedge$-*closed*, $\vee$-*closed*) iff for all $x \in L$ we have that $f(x) \neq \varnothing$ ($f(x)$ is, resp., $\wedge$-*closed*, $\vee$-*closed*).

In order to define the notion of a (multivalued) monotone function, as $f(x)$ is now a set of elements, we need to extend the partial order $\leq$ to sets of elements. There are mainly three well-known *preorders* (reflexive, transitive, but not antisymmetric), namely the *Smyth ordering*, the *Hoare ordering*, and the *Egli-Milner ordering*, which have been proposed in the context of nondeterministic programming languages (see, e.g., [1, 25]):[3]

$$X \preceq_S Y \quad \text{iff} \quad \forall y \in Y\ \exists x \in X \text{ s.t. (such that) } x \leq y \quad \text{(Smyth ordering)}, \tag{3.1}$$

$$X \preceq_H Y \quad \text{iff} \quad \forall x \in X\ \exists y \in Y \text{ s.t. } x \leq y \quad \text{(Hoare ordering)}, \tag{3.2}$$

$$X \preceq_{EM} Y \quad \text{iff} \quad X \preceq_S Y \text{ and } X \preceq_H Y \quad \text{(Egli-Milner ordering)}\ . \tag{3.3}$$

These orderings may be read as follows: (i) $X \preceq_S Y$ iff all $y \in Y$ are approximated by some $x \in X$; (ii) $X \preceq_H Y$ iff all $x \in X$ approximate some $y \in Y$; and (iii) $X \preceq_{EM} Y$ iff all $y \in Y$ are approximated by some $x \in X$ and all $x \in X$ approximate some $y \in Y$.

Clearly the Hoare ordering is equivalent to the Smyth ordering in the dual underlying lattice. Indeed it is straightforward to show the following.

PROPOSITION 3.1. *Let X, Y be two subsets of L. Then $X \preceq_S Y$ in $\mathcal{L}$ iff $Y \preceq_H X$ in $\bar{\mathcal{L}}$.*

As a consequence, many properties we state with respect to the Smyth ordering in $\mathcal{L}$ have their dual with respect to the Hoare ordering in $\bar{\mathcal{L}}$.

f is *Smyth-monotone*, or simply *S-monotone*, iff for all $x, y \in L$, if $x \leq y$, then $f(x) \preceq_S f(y)$ holds. The notions of *Hoare-monotone*, or simply *H-monotone*, and *Egli-Milner-monotone*, or simply *EM-monotone*, are defined similarly. By using Proposition 3.1, it is straightforward to prove the following.

PROPOSITION 3.2. *Let $f\colon L \to 2^L$ be a multivalued function. Then f is S-monotone in $\mathcal{L}$ iff f is H-monotone in $\bar{\mathcal{L}}$.*

We say that f is *inflationary* iff for all x, $\{x\} \preceq_S f(x)$; i.e., all elements in $f(x)$ are greater than or equal to x. Dually, we say that f is *deflationary* iff for all $x \in L$, $f(x) \preceq_H \{x\}$; i.e., all elements in $f(x)$ are smaller than or equal to x. Of course, a deflationary function is an inflationary function in the dual lattice $\bar{\mathcal{L}}$.

PROPOSITION 3.3. *Let $f\colon L \to 2^L$ be a multivalued function. Then f is deflationary in $\mathcal{L}$ iff f is inflationary in $\bar{\mathcal{L}}$.*

We next generalize the notion of a limit-preserving function to the multivalued case. A multivalued function $f\colon L \to 2^L$ is $\bigvee$-*preserving* iff for all increasing sequences $(x_\alpha)_{\alpha \in I}$,

$$f\left(\bigvee_\alpha x_\alpha\right) = \left\{ y \mid \text{ there is } (y_\alpha)_{\alpha \in I} \text{ s.t. } y_\alpha \in f(x_\alpha) \text{ and } y = \bigvee_\alpha y_\alpha \right\}\ . \tag{3.4}$$

Dually, we say that $f\colon L \to 2^L$ is $\bigwedge$-*preserving* iff for all decreasing sequences $(x_\alpha)_{\alpha \in I}$,

$$f\left(\bigwedge_\alpha x_\alpha\right) = \left\{ y \mid \text{ there is } (y_\alpha)_{\alpha \in I} \text{ s.t. } y_\alpha \in f(x_\alpha) \text{ and } y = \bigwedge_\alpha y_\alpha \right\}\ . \tag{3.5}$$

f is *limit-preserving* iff it is both $\bigvee$- and $\bigwedge$-preserving. For ease of presentation, sometimes we use the notation $\bigvee_\alpha f(x_\alpha)$ (resp., $\bigwedge_\alpha f(x_\alpha)$) to denote the right-hand

[3] [36] describes another order, called the *Plotkin order*, which extends the Egli-Milner ordering. However, we will not address it here.

side of (3.4) (resp., (3.5)). Note that if for all $x \in L$, $|f(x)| = 1$, then the definition reduces to the usual one for single-valued functions.

Of course, we have the following.

PROPOSITION 3.4. *Let $f\colon L \to 2^L$ be a multivalued function. Then f is $\bigwedge$-preserving in $\mathcal{L}$ iff f is $\bigvee$-preserving in $\bar{\mathcal{L}}$.*

We can prove the following.

PROPOSITION 3.5. *Consider a multivalued function $f\colon L \to 2^L$.*

1. *If f is $\bigvee$-preserving, then f is S-monotone.*
2. *If f is $\bigwedge$-preserving, then f is H-monotone.*
3. *If f is limit-preserving, then f is EM-monotone.*

Proof.

Case 1. Let $x_1 \leq x_2$ and f $\bigvee$-preserving. Then for the increasing sequence $x_1 \leq x_2$, $f(x_2) = f(x_1 \vee x_2) = \{y\colon$ there are $y_i \in f(x_i)$ s.t. $y = y_1 \vee y_2\} = X$. If $f(x_2) = \varnothing$, then trivially $f(x_1) \preceq_S f(x_2) = \varnothing$. If $f(x_1) = \varnothing$, then by definition $X = \varnothing$ and, thus, $f(x_2) = \varnothing$. Therefore, $\varnothing = f(x_1) \preceq_S f(x_2) = \varnothing$. Otherwise assume $f(x_1)$ and $f(x_2)$ nonempty. Therefore, as f is $\bigvee$-preserving, for $y \in f(x_2) = X$ there are $y_i \in f(x_i)$ $(i = 1, 2)$ such that $y = y_1 \vee y_2$. In particular, $y_1 \leq y$. Therefore, $f(x_1) \preceq_S f(x_2)$ and, thus, f is S-monotone.

Case 2. The proof is dual to Case 1 (see the appendix, Proposition A.1).

Case 3. This case is straightforward by Cases 1 and 2. □

Note that a $\bigwedge$-preserving function need not be S-monotone and, similarly, a $\bigvee$-preserving function need not be H-monotone and, thus, an EM-monotone function need not be limit-preserving.

Example 4. Consider $L = \{0, 1\}$ with $0 \leq 1$. Then the multivalued function $f : L \to 2^L$, $f(0) = \varnothing, f(1) = \{1\}$ is $\bigwedge$-preserving, but not S-monotone, as $0 \leq 1$ and $f(0) = \varnothing \not\preceq_S f(1) = \{1\}$. Similarly, the multivalued function $g : L \to 2^L$, $g(0) = \{0\}, g(1) = \varnothing$ is $\bigvee$-preserving, but not H-monotone, as $0 \leq 1$ and $g(0) = \{0\} \not\preceq_H g(1) = \varnothing$.

But, we can easily show the following.

PROPOSITION 3.6. *Consider a multivalued function $f\colon L \to 2^L$ and $x_1 \leq x_2$ with $f(x_1) \neq \varnothing \neq f(x_2)$.*

1. *If f is $\bigwedge$-preserving, then $f(x_1) \preceq_S f(x_2)$.*
2. *If f is $\bigvee$-preserving, then $f(x_1) \preceq_H f(x_2)$.*

Proof.

Case 1. For the decreasing sequence $x_2 \geq x_1$, as f is $\bigwedge$-preserving, $f(x_1) = f(x_2 \wedge x_1) = \{y\colon$ there are $y_i \in f(x_i)$ s.t. $y = y_2 \wedge y_1\} = X$. Now, for $y \in f(x_2)$ choose a $y' \in f(x_1) \neq \varnothing$ and consider $y'' = y \wedge y'$. Therefore, $y'' \in X = f(x_1)$, $y'' \leq y$ and, thus, $f(x_1) \preceq_S f(x_2)$.

Case 2. This is similar to Case 1 (see the appendix, Proposition A.2). □

Example 1 can be adapted to multivalued functions and prove that a limit-preserving (in particular an S-monotone) function need not be inflationary.

Example 5. *Consider $f : \{0, 1\} \to 2^{\{0,1\}}$ such that for all $x \in \{0, 1\}$, $f(x) = \{0\}$; then f is limit-preserving and, thus, S-monotone, but $\{1\} \not\preceq_S f(1) = \{0\}$.*

We next want to investigate the existence of (minimal) fixed-points of multivalued functions. Similarly to the single-valued case, for $f\colon L \to 2^L$, let us define

$$\Phi(f) = \{x \in L\colon f(x) \preceq_S \{x\}\},$$
$$\Psi(f) = \{x \in L\colon \{x\} \preceq_H f(x)\}\ .$$

Note that, unlike the single-valued case, not necessarily $\top \in \Phi(f)$ (i.e., if $f(\top) = \varnothing$). Similarly, $\bot \in \Psi(f)$ iff $f(\bot) \neq \varnothing$. Also, if $f(x) = \varnothing$, then $x \notin \Phi(f)$; i.e., if $x \in \Phi(f)$, then $f(x) \neq \varnothing$. Finally, note that if $f(\top) \neq \varnothing$, then $\top \in \Phi(f)$ (we will use these straightforward facts often in the paper). Furthermore, note that $\Phi(f)$ is related to the $\preceq_S$ order, while $\Psi(f)$ is related to $\preceq_H$. One might wonder why we did not consider, for instance, $\Phi_H(f) = \{x \mid f(x) \preceq_H \{x\}\}$. As we will see later, $\bigwedge \Phi(f)$ relates to the least fixed-point of f (if it exists), while $\bigvee \Psi(f)$ relates to the greatest fixed-point of f. Example 6 shows that $\bigwedge \Phi_H(f)$ is not related to the least fixed-point of f.

Example 6. Consider $L = \{0, 1\}$ and the multivalued function $f\colon L \to 2^L$, $f(0) = \{0, 1\}, f(1) = \{1\}$. Then f is EM-monotone, $Fix(f) = \{0, 1\}$, but $\Phi_H(f) = \{x \mid f(x) \preceq_H \{x\}\} = \{1\}$ and $1 = \bigwedge \Phi_H(f)$ is not the least fixed-point of f.

We can show the following.

PROPOSITION 3.7. *Let $f\colon L \to 2^L$ be a multivalued function.*

1. *If f is inflationary, then $x \in \Phi(f)$ iff x is a fixed-point of f.*
2. *If f is deflationary, then $x \in \Psi(f)$ iff x is a fixed-point of f.*

Proof.

Case 1. Let $x \in \Phi(f)$. As f is inflationary, $\{x\} \preceq_S f(x) \preceq_S \{x\}$ and, thus, for $x \in \{x\}$ there is $y \in f(x)$ such that $x \leq y \leq x$, i.e., $x = y \in f(x)$. Vice versa, if $x \in f(x)$, then $f(x) \preceq_S \{x\}$ and, thus, $x \in \Phi(f)$.

Case 2. This is similar to Case 1 (see the appendix, Proposition A.3). ∎

Note that Proposition 3.7 does not hold if a function is, e.g., S-monotone, but not inflationary.

Example 7. In Example 3, f is S-monotone, not inflationary with $2 \in \Phi(f)$, but $2 \notin f(2)$.

The following examples show that a multivalued S-monotone function $f\colon L \to 2^L$ may have several minimal fixed-points or even no minimal fixed-point at all.

Example 8. Consider Belnap's truth space $\mathcal{FOUR}$ [3], $L = \{\bot, f, t, \top\}$ with f, t incomparable. Here, besides f for "false" and t for "true," $\bot$ stands for "unknown," whereas $\top$ stands for inconsistency. $\leq$ is the so-called knowledge order. Consider the multivalued function $g\colon L \to 2^L$ defined as $g(\bot) = \{f, t, \top\}, g(f) = \{f, \top\}, g(t) = \{t, \top\}$, and $g(\top) = \{\top\}$. Then g is EM-monotone, inflationary, and $\bigvee$-preserving. Furthermore, $f \in g(f), t \in g(t)$, and $\top \in g(\top)$, but $\bot \notin g(\bot)$, and thus f, t, and $\top$ are fixed-points of g, while $\bot$ is not. The minimal fixed-points are f and t. Note that $g(x)$ does not have a least element (e.g., $g(\bot)$) for all x. Additionally, note that $\Phi(g) = \{f, t, \top\}$, $\bigwedge \Phi(g) = \bot \notin \Phi(g)$, and $\min \Phi(g) = \{f, t\}$. Therefore, unlike the single-valued case, $\bigwedge \Phi(g)$ is not a fixed-point of g.

The four-element Belnap's truth space $\mathcal{FOUR}$ was introduced as a very suitable setting for computerized reasoning; it has a bilattice structure, since two orderings can be naturally defined and, as a result, it can be viewed as a class of truth values that can accommodate incomplete and inconsistent information, and in certain cases, default information.

Example 9. Let $L = [0, 1]$. Consider the multivalued function $f\colon L \to 2^L$ defined as $f(x) = \{y \mid y > 0, y \geq x)\}$. Then f is nonempty, $\bigvee$-preserving, and inflationary. Furthermore, for all $x > 0$, $x \in f(x)$, but $0 \notin f(0)$, and thus all $x > 0$ are fixed-points of f, while 0 is not. Therefore, f has no minimal fixed-point. Also, note that $\Phi(f) = \{x \mid x > 0\}$, $\bigwedge \Phi(f) = 0 \notin \Phi(f)$, and $\min \Phi(f) = \varnothing$. Similar to Example 8, $0 = \bigwedge \Phi(f)$ is not a fixed-point of f, but now $\min \Phi(f) = \varnothing$. Also note that $f(0)$ has no least element.

Similarly, let us consider now $g(x) = \{y \mid y < 1, y \leq x\}$. Then g is nonempty, $\bigwedge$-preserving, and deflationary. $\Psi(g) = \{x \mid x < 1\}$, $\bigvee \Psi(g) = 1 \notin \Psi(g)$, $\max \Psi(g) = \varnothing$, and $1 \notin g(1)$. Hence, g has no greatest fixed-point.

Likewise, $h(x) = \{y \mid 0 < y < 1\}$. Then h is nonempty and EM-monotone. $\Phi(h) = \{x \mid x > 0\}$, $\Psi(h) = \{x \mid x < 1\}$, and h has neither a least nor a greatest fixed-point.

Like the single-valued case, a multivalued inflationary function may not have a minimal fixed-point, even if $f(x)$ has a least element for all $x \in L$.

Example 10. Consider $f\colon [0,1] \to 2^{[0,1]}$, where $f(0) = \{1\}$ and for $x > 0$, $f(x) = \{x\}$. Then f is not S-monotone but is inflationary. Also, $f(x)$ has a least element for all $x \in L$. All $x > 0$ are fixed-points as $x \in f(x)$, $\Phi(f) = \{x \mid x > 0\}$ (in accordance with Proposition 3.7), and $\bigwedge \Phi(f) = 0$, but $0 \notin f(0)$. Note that $\min \Phi(f) = \varnothing$.

However, we will show later in Proposition 3.10 that a multivalued S-monotone function, such that $f(x)$ has a least element for all $x \in L$, has indeed a least fixed-point.

We next show that if $\Phi(f)$ has minimals, then an S-monotone or inflationary function f has minimal fixed-points.

PROPOSITION 3.8. *Let $f\colon L \to 2^L$ be a multivalued function.*

1. *If f is an S-monotone or inflationary multivalued function, and $\Phi(f)$ has minimals, then all $y \in \min \Phi(f)$ are minimal fixed-points of f. In particular, if $x = \bigwedge \Phi(f) \in \Phi(f)$, then x is the least fixed-point of f.*
2. *If f is an H-monotone or deflationary multivalued function, and $\Psi(f)$ has maximals, then all $y \in \max \Psi(f)$ are maximal fixed-points of f. In particular, if $x = \bigvee \Psi(f) \in \Psi(f)$, then x is the greatest fixed-point of f.*

Proof.

Case 1. To begin with, let us show that any $y \in \min \Phi(f)$ is a fixed-point of f. As $\Phi(f)$ has minimals, $\min \Phi(f) \neq \varnothing$. So, let $y \in \min \Phi(f)$. As $\varnothing \neq f(y) \preceq_S \{y\}$, thus, there is $y' \in f(y)$ such that $y' \leq y$. If f is S-monotone, then $f(y') \preceq_S f(y)$, and thus for $y' \in f(y)$ there is $y'' \in f(y')$ such that $y'' \leq y'$. Therefore, $f(y') \preceq_S \{y'\}$, and thus $y' \in \Phi(f)$. But $y \in \min \Phi(f)$, so it cannot be $y' < y$. Therefore, $y = y' \in f(y)$; i.e., y is a fixed-point of f. If f is inflationary, by Proposition 3.7, y is a fixed-point of f.

Now, let us show that any $y \in \min \Phi(f)$ is also a minimal fixed-point of f. So, consider $y \in \min \Phi(f)$ and, thus, y is a fixed-point of f. Now, consider another fixed-point $x \in f(x)$. Therefore, $f(x) \preceq_S \{x\}$, and thus $x \in \Phi(f)$. But $y \in \min \Phi(f)$, so it cannot be $x < y$, and thus y is a minimal fixed-point of f.

Finally, consider $x = \bigwedge \Phi(f)$. By hypothesis, $x \in \Phi(f)$ and x is a least element of $\Phi(f)$. Hence, we know that $x \in f(x)$. Let $y \in f(y)$. Hence $y \in \Phi(f)$, and thus $x \leq y$. As a consequence, x is the least fixed-point of f.

Case 2. This is similar to Case 1 (see the appendix, Proposition A.4). □

Note that $\Phi(f)$ in Examples 3 and 8 has minimals, while $\Phi(f)$ in Example 9 does not.

The following proposition establishes a condition on an S-monotone function f under which $\Phi(f)$ has minimals and, thus, minimal fixed-points.

PROPOSITION 3.9. *Let $f\colon L \to 2^L$ be a multivalued function.*

1. *If f is a $\bigwedge$-preserving multivalued function with $\Phi(f) \neq \varnothing$, then $\Phi(f)$ has minimals and, thus, minimal fixed-points.*

2. *If f is a $\bigvee$-preserving multivalued function with $\Psi(f) \neq \varnothing$, then $\Psi(f)$ has maximals and, thus, maximal fixed-points.*

Proof.

Case 1. By hypothesis $\Phi(f) \neq \varnothing$. Let $(x_\alpha)_{\alpha \in I}$ be a decreasing sequence of $x_\alpha \in \Phi(f)$ and let $\bar{x} = \bigwedge_\alpha x_\alpha$. As f is $\bigwedge$-preserving, by definition $f(\bar{x}) = \{y\colon$ there is $(y_\alpha)_{\alpha \in I}$ s.t. $y_\alpha \in f(x_\alpha)$ and $y = \bigwedge_\alpha y_\alpha\}$. Now, for any α, $x_{\alpha+1} \leq x_\alpha$, by Proposition 3.6 and, as $x_\alpha \in \Phi(f)$, $f(x_{\alpha+1}) \preceq_S f(x_\alpha) \preceq_S \{x_\alpha\}$. Therefore, for any x_α there is $y_\alpha \in f(x_\alpha)$ and $y_{\alpha+1} \in f(x_{\alpha+1})$ such that $y_{\alpha+1} \leq y_\alpha \leq x_\alpha$. Note that if α is a limit ordinal, then, as $x_\alpha \leq x_\beta$ for all $\beta < \alpha$, it follows that $f(x_\alpha) \preceq_S f(x_\beta) \preceq_S \{x_\beta\}$ and, thus, $y_\alpha \leq y_\beta \leq x_\beta$ for all $\beta < \alpha$. Therefore, there is a decreasing sequence $(y_\alpha)_{\alpha \in I}$ of elements $y_\alpha \in f(x_\alpha)$ such that $\bar{y} = \bigwedge_\alpha y_\alpha \leq \bigwedge_\alpha x_\alpha = \bar{x}$. By definition of $f(\bar{x})$, $\bar{y} \in f(\bar{x})$ and, thus, $f(\bar{x}) \preceq_S \{\bar{x}\}$. Therefore $\bar{x} \in \Phi(f)$ and, thus, every decreasing sequence has a lower bound in $\Phi(f)$. So, by Zorn's lemma, $\Phi(f)$ has minimals, which by Proposition 3.8 are also minimal fixed-points.

Case 2. This is the same as Case 1 (see the appendix, Proposition A.5). □

The converse of Proposition 3.9 above is not true.

Example 11. Consider $L = \{0, 0.5, 1\}$, where $f\colon L \to 2^L$ with $f(0) = \{0\}$, $f(0.5) = \{0.5\}$, and $f(1) = \{0, 1\}$. Then $\Phi(f) = L$ has minimals, but f is not S-monotone: $0.5 \leq 1$ but $f(0.5) \not\preceq_S f(1)$. Therefore, by Proposition 3.6, f cannot be $\bigwedge$-preserving.

One might wonder whether an S-monotone $f\colon L \to 2^L$ such that for all $x \in L$, $f(x)$ has minimals implies that $\Phi(f)$ has minimals. This is not true, as the following example shows.

Example 12. Consider $Y = \{y_\alpha : \alpha \in \omega\}$, Y antichain, $X = \{x_\alpha : \alpha \in \omega\}$, $x_{\alpha+1} \leq x_\alpha$, $\bar{x} = \bigwedge_\alpha x_\alpha$, $y_\alpha \leq x_\alpha$, each pair $\bar{x}, y_\alpha$ incomparable, $L = \{\bar{x}\} \cup X \cup Y \cup \{\bot, \top\}$, and $f\colon L \to 2^L$ with $f(\bot) = Y$, $f(\bar{x}) = Y$, $f(x_\alpha) = \{x_\alpha\}$, $f(y_\alpha) = \{x_\alpha\}$, and $f(\top) = \{\top\}$. Then f is S-monotone, for all $x \in L$, $f(x)$ has minimals, $\Phi(f) = X \cup \{\top\}$, and $(x_\alpha)_{\alpha \in \omega}$ is a decreasing sequence of elements in $\Phi(f)$. As neither $\bar{x}$ nor $\bot$ is in $\Phi(f)$, $\Phi(f)$ does not have minimals.

However, we can prove the following.

PROPOSITION 3.10. *Let $f\colon L \to 2^L$ be a multivalued function.*

1. *If f is S-monotone and for all $x \in L$, $f(x)$ has a least element, then f has a least fixed-point.*
2. *If f is H-monotone and for all $x \in L$, $f(x)$ has a greatest element, then f has a greatest fixed-point.*

Proof.

Case 1. As for all $x \in L$, $f(x)$ has a least element, by definition $\bigwedge f(x) \in f(x) \neq \varnothing$. Therefore, $\Phi(f) \neq \varnothing$ as $\varnothing \neq f(\top) \preceq_S \{\top\}$. Consider $a = \bigwedge_{c \in \Phi(f)} c$. If $a \in \Phi(f)$, then by Proposition 3.8, a is the least fixed-point of f. So, let us show that $a \in \Phi(f)$. For $c \in \Phi(f)$ there is an $x_c \in f(c)$ such that $x_c \leq c$. As $a \leq c$ and f is S-monotone, $f(a) \preceq_S f(c)$ and, thus, for $x_c \in f(c)$ there is $y_c \in f(a)$ such that $y_c \leq x_c \leq c$. Since $f(a)$ has a least element, there is $y \in f(a)$ such that $y \leq \bigwedge_{c \in \Phi(f)} y_c \leq \bigwedge_{c \in \Phi(f)} x_c \leq \bigwedge_{c \in \Phi(f)} c = a$. Hence, $f(a) \preceq_S \{a\}$, i.e., $a \in \Phi(f)$.

Case 2. This is the same as Case 1 (see the appendix, Proposition A.6). □

Note that if, e.g., $f(x)$ has a least element for all $x \in L$, then this does not imply necessarily that f is $\bigwedge$-preserving or $\bigvee$-preserving.

Example 13. Consider Belnap's truth space $\mathcal{FOUR}$, $L = \{f, t, \bot, \top\}$. Let $h(\top) = \{f\}$, $h(t) = \{\bot, f\}$, $h(f) = \{\bot, t\}$, $h(\bot) = \{\bot\}$. Then for all $x \in L$, $h(x)$ has a least element. Consider the decreasing sequence $(\top, f)$. Then $h(\top \wedge f) = h(f) = \{\bot, t\}$,

while $h(\top) \wedge h(f) = \{\bot\}$ and, thus, h is not $\bigwedge$-preserving. Consider the increasing sequence $(f, \top)$. Then $h(f \vee \top) = h(\top) = \{f\}$, while $h(f) \vee h(\top) = \{f, \top\}$ and, thus, h is not $\bigvee$-preserving.

The following example shows that, e.g., an H-monotone function such that for all $x \in L$, $f(x)$ has a least element, does not imply that f has a least fixed-point.

Example 14. Consider the lattice $\mathcal{FOUR}$ as in Example 13. Let $g(\bot) = \{t\}, g(f) = \{f, t, \bot\}, g(t) = \{f, t, \bot\}, g(\top) = \{\top\}$. g is H-monotone, but not S-monotone. Furthermore, for all $x \in L$, $g(x)$ has a least element. As $Fix(g) = \{f, t, \top\}$, g has no least fixed-point.

The following example shows that an H-monotone or S-monotone nonempty function may not have a fixed-point at all.

Example 15. Consider $L = [0, 1]$ and a multivalued function f, with $f(x) = \{(x + 1)/2\}$ for $x < 1$ and $f(1) = \{1 - 1/n \mid n = 1, 2, \ldots\}$. Then f is H-monotone without any fixed-point.

Similarly, let $g(x) = \{x/2\}$ for $x > 0$ and $g(0) = \{1/n \mid n = 1, 2, \ldots\}$. Then g is S-monotone without any fixed-point.

Next, we describe properties of the structure of the set of fixed-points. The following example shows that the meet of two fixed-points of a monotone multivalued function may not be a fixed-point and, thus, the set of fixed-points may not be a sublattice.

Example 16. Consider $L = \{f, t, \bot, \top, c\}$, where $\bot \leq c$, $c \leq f \leq \top$, and $c \leq t \leq \top$. Let $g(\bot) = \{\bot\}, g(c) = \{\bot\}, g(t) = \{t\}, g(f) = \{f\}, g(\top) = \{\top\}$. Then g is EM-monotone, limit-preserving, deflationary, but not inflationary, and for all $x \in L$, $g(x)$ is a closed sublattice of L. However, $Fix(g) = \{\bot, \top, f, t\}$ is not a sublattice of L, e.g., $f, t \in Fix(f)$, but $c = f \wedge t \notin Fix(f)$ ($Fix(f)$ is not even a meet semilattice).

However, we can show the following.

PROPOSITION 3.11. *Let $f\colon L \to 2^L$ be an S-monotone, nonempty, and $\wedge$-closed multivalued function. Then*

1. *$\Phi(f)$ is $\wedge$-closed; and*
2. *f has a least fixed-point.*

Proof. Note that $\Phi(f) \neq \varnothing$ as $\varnothing \neq f(\top) \preceq_S \{\top\}$.

1. Consider a subset S of $\Phi(f)$ and $a = \bigwedge S$. Let us show that $a \in \Phi(f)$. We know that for each $c \in S$, $f(c) \preceq_S \{c\}$ holds; i.e., there is $x_c \in f(c)$ such that $x_c \leq c$. But, f is S-monotone and, thus, from $a \leq c$, $f(a) \preceq_S f(c) \preceq_S \{c\}$ follows. That is, there is $y_c \in f(a)$ such that $y_c \leq x_c \leq c$. Let $y = \bigwedge_{c \in S} y_c$. As f is $\wedge$-closed, $y \in f(a)$ follows. Therefore, $y = \bigwedge_{c \in S} y_c \leq \bigwedge_{c \in S} c = a$, $f(a) \preceq_S \{a\}$, and, thus, $a \in \Phi(f)$. Therefore, $\Phi(f)$ is $\wedge$-closed.

2. From point 1, $\Phi(f)$ has a least element a and, thus, by Proposition 3.8, f has a as a least fixed-point. □

Dually, we have the following.

PROPOSITION 3.12. *Let $f\colon L \to 2^L$ be an H-monotone, nonempty, and $\vee$-closed multivalued function. Then*

1. *$\Psi(f)$ is $\vee$-closed; and*
2. *f has a greatest fixed-point.*

Proof. This is the dual of proof of Proposition 3.11 (see the appendix, Proposition A.7). □

Clearly, from Propositions 3.11 and 3.12 we immediately have the following.

PROPOSITION 3.13. *Let $f\colon L \to 2^L$ be an EM-monotone multivalued function such that for any $x \in L$, $f(x)$ is a nonempty closed sublattice of L. Then f has a*

least fixed-point and a greatest fixed-point.

Also, the next proposition follows immediately from Proposition 3.7.

PROPOSITION 3.14. *Let $f\colon L \to 2^L$ be a nonempty multivalued function. Then*

1. *if f is S-monotone, inflationary, and $\wedge$-closed, then $Fix(f)$ is nonempty and $\wedge$-closed and, thus, has a least element; and*
2. *if f is H-monotone, deflationary, and $\vee$-closed, then $Fix(f)$ is nonempty and $\vee$-closed and, thus, has a greatest element.*

Note that if f is both inflationary and deflationary, then for all $x \in L$ such that $f(x) \neq \varnothing$, we can easily show that $f(x) = \{x\}$; i.e., f is a single-valued, constant, limit-preserving function, and each such x is a fixed-point, and, thus, is not interesting.

We have seen in Proposition 3.14 that under rather strong conditions, we have a rather strong structure on the set of fixed-points (e.g., the conjunction of two fixed-points is a fixed-point). On the other hand, Example 16 shows that, e.g., if we omit the inflationary condition, then $Fix(f)$ is not $\wedge$-closed (e.g., the conjunction of two fixed-points need not be a fixed-point) and, thus, $Fix(f)$ cannot be a closed sublattice of L.

The following proposition, due to [53], establishes that the set of fixed-points is a complete lattice, though not a closed sublattice.

PROPOSITION 3.15 (Zhou [53]). *Let $f\colon L \to 2^L$ be a multivalued function. If f is EM-monotone and for any $x \in L$, $f(x)$ is a nonempty closed sublattice of L, then $Fix(f)$ is a nonempty complete lattice.*

We next look at limit-preserving functions and their impact on the set of fixed-points. We first notice the following.

PROPOSITION 3.16. *Let $f\colon L \to 2^L$ be a multivalued function. Then*

1. *if f is $\bigwedge$-preserving, then f is $\wedge$-closed;*
2. *if f is $\bigvee$-preserving, then f is $\vee$-closed; and*
3. *if f is limit-preserving, then for any $x \in L$, $f(x)$ is a closed sublattice of L.*

Proof.

1. Consider $x \in L$. If $f(x)$ is empty, then it is also $\wedge$-closed. Otherwise, consider any subset of $f(x)$ in the form of a sequence $(y_\alpha)_{\alpha\in I}$ of elements $y_\alpha \in f(x)$. We show that $f(x)$ is $\wedge$-closed by showing that $y = \bigwedge_{\alpha\in I} y_\alpha \in f(x)$. So, consider the decreasing sequence $(x_\alpha)_{\alpha\in I}$, where $x = x_\alpha$, for all $\alpha \in I$. By construction, $x = \bigwedge_{\alpha\in I} x_\alpha$. As f is $\bigwedge$-preserving, we have that

$$\begin{array}{rcl} f(x) & = & f(\bigwedge_\alpha x_\alpha) \\ & = & \{z \mid \text{ there is } (z_\alpha)_{\alpha\in I} \text{ s.t. } z_\alpha \in f(x_\alpha) \text{ and } z = \bigwedge_\alpha z_\alpha\} \\ & = & \{z \mid \text{ there is } (z_\alpha)_{\alpha\in I} \text{ s.t. } z_\alpha \in f(x) \text{ and } z = \bigwedge_\alpha z_\alpha\} \ . \end{array}$$

Therefore, as for $(y_\alpha)_{\alpha\in I}$ we have $y_\alpha \in f(x)$, it follows that $y = \bigwedge_{\alpha\in I} y_\alpha \in f(\bigwedge_\alpha x_\alpha) = f(x)$, which concludes the proof.

The other points can be shown similarly. □

Note that the converse in Proposition 3.16 does not hold. For instance, in Example 14, the function g is such that for all $x \in L$, $g(x)$ is a closed sublattice, but g is not $\bigwedge$-preserving (as g is not S-monotone).

We already know from Proposition 3.9 that if f is $\bigwedge$-preserving and $\Phi(f) \neq \varnothing$ (e.g., $f(\top) \neq \varnothing$), then f has minimal fixed-points and, similarly, from Proposition 3.9 we know that if f is $\bigvee$-preserving and $\Psi(f) \neq \varnothing$ (e.g., $f(\bot) \neq \varnothing$), then f has maximal fixed-points. By further relying on Propositions 3.14 and 3.16, we have the following.

PROPOSITION 3.17. *Let $f\colon L \to 2^L$ be a nonempty multivalued function. Then*

1. *if f is $\bigwedge$-preserving and inflationary, then $Fix(f)$ is nonempty, $\wedge$-closed, and thus, has a least element;*
2. *if f is $\bigvee$-preserving and deflationary, then $Fix(f)$ is nonempty, $\vee$-closed, and thus, has a greatest element; and*
3. *if f is limit-preserving, then $Fix(f)$ is a nonempty complete lattice.*

Note that the condition for nonemptiness in the above proposition is mandatory as, e.g., a $\bigwedge$-preserving function f may not necessarily imply that f is nonempty, as the example below shows. This example also shows that Proposition 3.17 neither subsumes nor contrasts with Proposition 3.8.

Example 17. Consider the lattice $\mathcal{FOUR}$. Let g be a multivalued function on L such that $g(\bot) = \varnothing$, $g(\top) = \{\top\}$, $g(f) = \{f\}$, and $g(t) = \{t\}$. It can be easily verified that g is $\bigwedge$-preserving and deflationary, though $Fix(g) = \{f, t, \top\}$, and, thus, no least fixed-point exists. g has two minimal fixed-points instead.

As already pointed out, we are more interested in cases in which $f(x)$ may be empty for some $x \in L$. The literature we are aware of does not report results in such cases [6, 22, 33, 45, 53]. The following result (compare to Proposition 3.15) reveals the structure of the set of fixed-points for limit-preserving functions under weaker conditions than those in Proposition 3.17. It says that the set of fixed-points of a limit-preserving function, if not empty, is a complete multilattice. A *complete multilattice* [4, 29, 30] is a partially ordered set $\mathcal{M} = \langle M, \leq \rangle$, such that for every subset $X \subseteq M$, the set of upper (resp., lower) bounds of X has minimal (resp., maximal) elements, which are called *multisuprema* (resp., *multi-infima*). The sets of multisuprema and multi-infima of a set X are denoted multsup(X) and multinf(X).

PROPOSITION 3.18. *Let $f\colon L \to 2^L$ be a multivalued function. If f is limit-preserving and $Fix(f)$ is nonempty, then $Fix(f)$ is a complete multilattice.*

Proof. The proof is inspired by the proof of Proposition 3.15.

Let us show that $\langle Fix(f), \leq \rangle$ is a complete multilattice. By assumption, $Fix(f)$ is nonempty; by Proposition 3.5, f is EM-monotone; and by Proposition 3.16, for any $x \in L$, $f(x)$ is a closed sublattice of L. Let $S \subseteq Fix(f)$. Let us show that the set multsup(S) is nonempty in $\langle Fix(f), \leq \rangle$. So, consider $a = \bigvee S = \bigvee_{c \in S} c$ and the complete lattice $\mathcal{B} = \langle [a, \top], \leq \rangle$. Let g be the multivalued function from $[a, \top]$ to $2^{[a,\top]}$ defined by $g(s) = f(s) \cap [a, \top]$ for all $s \in [a, \top]$. Since both f and h, which assign to each $s \in [a, \top]$ the constant interval $[a, \top]$, are $\bigwedge$-preserving on S, it is not difficult to check that $g = f \cap h$ is $\bigwedge$-preserving on $[a, \top]$.

Now, let's show that $\Phi(g) \neq \varnothing$. For $c \in S$, as $c \leq a$ and f is H-monotone, $f(c) \preceq_H f(a)$ follows. Hence, for $c \in f(c)$ there is $x_c \in f(a)$ such that $c \leq x_c$. Consider $b = \bigvee_{c \in S} x_c$. Therefore, $a = \bigvee_{c \in S} c \leq \bigvee_{c \in S} x_c = b$. We show now that $b \in f(a)$. Consider the sequence $(a, a, \ldots, a)$ of length $|S|$. As f is limit-preserving and all $x_c \in f(a)$, we have that $b = \bigvee_{c \in S} x_c \in f(a \vee a \vee \cdots \vee a) = f(a)$, i.e., $b \in f(a)$. Now, consider $s \in [a, \top]$. As $a \leq s$ and f is H-monotone, $f(a) \preceq_H f(s)$ follows; i.e., for $b \in f(a)$ there is an $s_b \in f(s)$ such that $a \leq b \leq s_b$. It follows that $g(s) = f(s) \cap [a, \top] \neq \varnothing$ for all $s \in [a, \top]$. In particular, $g(\top) \neq \varnothing$ and, thus, $g(\top) \preceq_S \{\top\}$, i.e., $\top \in \Phi(g) \neq \varnothing$.

As a consequence, by Proposition 3.9, g has minimal fixed-points S'. Obviously, as $Fix(g) = Fix(f) \cap [a, \top]$, any $a' \in S'$ is also a fixed-point of f, with $a \leq a'$. In fact, a' is a minimal fixed-point of f, which is an upper bound of all elements of S; in other words, $a' \in$ multsup(S) and $a' \in Fix(f)$, which concludes the proof.

Similarly, it can be shown that multinf(S) is nonempty in $\langle Fix(f), \leq \rangle$, and, thus, we can conclude that $\langle Fix(f), \leq \rangle$ is a complete multilattice. □

TABLE 3.1
Main results about $Fix(f)$.

Prop.	$\bigwedge$-pr.	$\bigvee$-pr.	S-mo.	H-mo.	$f(x)$	infl.	defl.	$\Phi(f)$	$\Psi(f)$	$Fix(f)$
3.7						•		$\neq \varnothing$		$\neq \varnothing$
3.7							•		$\neq \varnothing$	$\neq \varnothing$
3.8			•					min		min
3.8			•					$\bigwedge$		$\bigwedge$
3.8				•					max	max
3.8				•					$\bigvee$	$\bigvee$
3.8						•		min		min
3.8						•		$\bigwedge$		$\bigwedge$
3.8							•		max	max
3.8							•		$\bigvee$	$\bigvee$
3.9	•							$\neq \varnothing$		min
3.9		•							$\neq \varnothing$	max
3.10			•		$\bigwedge$					$\bigwedge\!\!\bigvee$
3.10				•		$\bigvee$				$\bigvee\!\!\bigwedge$
3.11			•		$\neq \varnothing$, $\wedge$-cl.					$\bigwedge\!\!\bigvee$
3.12				•	$\neq \varnothing$, $\vee$-cl.					$\bigvee\!\!\bigwedge$
3.14			•		$\neq \varnothing$, $\wedge$-cl.	•				$\neq \varnothing$, $\wedge$-cl.
3.14				•	$\neq \varnothing$, $\vee$-cl.		•			$\neq \varnothing$, $\vee$-cl.
3.15			•	•	$\neq \varnothing$, sublatt.					$\neq \varnothing$, compl. latt.
3.17	•				$\neq \varnothing$	•				$\neq \varnothing$, $\wedge$-cl.
3.17		•			$\neq \varnothing$		•			$\neq \varnothing$, $\vee$-cl.
3.17	•	•			$\neq \varnothing$					$\neq \varnothing$, compl. latt.
3.18	•	•								compl. multilatt.

TABLE 3.2
Impact of multivalued functions in the examples on $Fix(f)$.

Ex.	$\bigwedge$-pr.	$\bigvee$-pr.	S-mo.	H-mo.	$f(x)$	infl.	defl.	$\Phi(f)$	$\Psi(f)$	$Fix(f)$
10					$\bigwedge, \bigvee$	•		$\neq \varnothing$, $\nexists \bigwedge$	$\bigvee$	$\bigvee$, $\nexists$ min
8		•	•	•	$\bigvee$	•		$\exists$ min, $\nexists \bigwedge$	$\bigvee$	$\bigvee$, $\exists$ min, $\nexists \bigwedge$
9		•			$\bigvee$	•		$\bigvee$, $\nexists$ min	$\bigvee$	$\bigvee$, $\nexists$ min
9	•				$\bigwedge$		•	$\bigwedge$	$\bigwedge$, $\nexists$ max	$\bigwedge$, $\nexists$ max
9			•	•	$\neq \varnothing$, $\nexists$ min $\nexists$ max			$\neq \varnothing$, $\nexists$ min $\nexists$ max	$\neq \varnothing$, $\nexists$ min $\nexists$ max	$\neq \varnothing$, $\nexists$ min $\nexists$ max
14			.	•	$\bigwedge$			$\exists$ min $\nexists \bigwedge$	compl. latt.	$\nexists$ min $\bigvee$
15				•	$\bigwedge\!\!\bigvee$			$\bigvee$	$\bigwedge$	$= \varnothing$
15			•		$\bigwedge\!\!\bigvee$			$\bigvee$	$\bigwedge$	$= \varnothing$
16	•	•			closed sublatt.		•	compl. latt.	$\bigwedge$	$\exists \bigwedge\!\!\bigvee$, $\exists \bigvee\!\!\bigwedge$, $\neg\wedge$-cl.
17	•						•	$\exists$ min $\bigvee$	$\exists$ min $\bigvee$	$\exists$ min, $\bigvee$

Note that by Proposition 3.9, in Proposition 3.18 above, $\Phi(f) \neq \varnothing$ guarantees that $Fix(f)$ is nonempty.

For convenience, Table 3.1 reports a summary of the main results about $Fix(f)$ reported in this section. In the table, min (max) means that the set contains minimals (maximals), while $\bigwedge$ ($\bigvee$) means that the set contains a least (greatest) element.

For completeness, Table 3.2 summarizes the impact of the multivalued functions in the examples on the set of fixed-points.

3.1. Orbits. We next describe how to obtain minimal fixed-points (if they exist) of multivalued functions $f\colon L \to 2^L$. An *orbit*[4] of f is a (possibly transfinite) sequence $(x_\alpha)_{\alpha\in I}$ of elements $x_\alpha \in L$, with $|I| > |L|$ and

$$\begin{array}{lcl} x_0 & = & \bot, \\ x_{\alpha+1} & \in & f(x_\alpha), \\ x_\lambda & = & \bigvee_{\alpha<\lambda} x_\alpha \text{ for limit ordinals } \lambda\,. \end{array}$$

[4]The definition is a generalization of the usual iteration of f over $\bot$ for single-valued functions.

Some comments are in order:

- Due to the nondeterministic choice of $x_{\alpha+1}$, f may have many possible orbits.
- For the sake of this paper we consider the starting point of the orbit $x_0 = \bot$. However, this can be made more flexible by considering any $x_0 = a \in L$ as a starting point. We consider $x_0 = \bot$, as we are interested in how to obtain minimal fixed-points. Of course, a special and interesting alternative case is $x_0 = \top$ (in that case, we postulate that for limit ordinal λ, $x_\lambda = \bigwedge_{\alpha<\lambda} x_\alpha$), which relates to the computation of maximal fixed-points. We call such sequences $\top$-*orbits*.
- A sequence $x_0, x_1, \ldots, x_\alpha$, where $x_{\beta+1} \in f(x_\beta)$ for $\beta < \alpha$ and $f(x_\alpha) = \varnothing$, is *not* an orbit.
- For convenience, we require that the length $|I|$ of an orbit be strictly greater than $|L|$, so that, if the orbit is increasing (decreasing), we may apply Proposition 2.1, which guarantees then that the orbit eventually becomes stationary.
- If an orbit $(x_\alpha)_{\alpha\in I}$ becomes stationary, i.e., there is $\beta \in I$ such that $|\beta| \leq |L|$ and $x_\alpha = x_\beta$ for all $\beta \leq \alpha \in I$, then by construction $x_\beta = x_{\beta+1} \in f(x_\beta)$ and, thus, x_β is a fixed-point of f.
- As any increasing (decreasing) orbit converges to a fixed-point, it is clear that if we can guarantee that such an orbit exists, then also the existence of a fixed-point is shown.
- Of course, from a practical point of view, whenever we try to build an orbit, we may stop as soon as we have $x_\beta = x_{\beta+1}$.

Example 18. Consider the lattice $\mathcal{FOUR}$. Let g be a multivalued function such that $g(\bot) = \{f, t\}$, $g(f) = \{f, t\}$, $g(t) = \{f, t\}$, $g(\top) = \{\top\}$. It can easily be verified that g is S-monotone and $Fix(g) = \{f, t, \top\}$. Then, for instance, we may have the following orbits:

$$\begin{array}{lcl} o_1 & = & (\bot, f, f, f, f), \\ o_2 & = & (\bot, t, t, t, t, t), \\ o_3 & = & (\bot, f, t, t, t), \\ o_4 & = & (\bot, t, f, f, t), \\ o_5 & = & (\bot, f, t, f, t, f, t)\ . \end{array}$$

As already pointed out, unlike the single-valued case, Examples 9 and 18 show that, e.g., S-monotonicity does not guarantee the existence of a minimal fixed-point. Also, S-monotonicity does not guarantee that an orbit $(x_\alpha)_{\alpha\in I}$ eventually becomes stationary (consider the orbit $(0, 2, 0, 2, \ldots)$ in Example 3 or orbit o_5 in Example 18). Note also that in Example 18 no orbit converges to the fixed-point $\top$.

Our main contribution in this context is the following.

PROPOSITION 3.19. *For a multivalued function f,*

1. *if f is inflationary, then each orbit is increasing;*
2. *each increasing orbit converges to a fixed-point of f (if no fixed-point exists, then there is no orbit); and*
3. *if f is S-monotone and inflationary, then for any minimal fixed-point of f there is an orbit converging to it.*

Proof. Let $(x_\alpha)_{\alpha\in I}$ be an orbit of f. Recall that for ordinal α we have $x_{\alpha+1} \in f(x_\alpha) \neq \varnothing$. As f is inflationary, $\{x_\alpha\} \preceq_S f(x_\alpha)$. But, by the definition of $\preceq_S$, for $x_{\alpha+1} \in f(x_\alpha)$, $x_\alpha \leq x_{\alpha+1}$. For a limit ordinal λ, $x_\lambda = \bigvee_{\alpha<\lambda} x_\alpha$, $\{x_\lambda\} \preceq_S f(x_\lambda) \neq \varnothing$, and, thus, there is $x_{\lambda+1} \in f(x_\lambda)$ such that $x_\lambda \leq x_{\lambda+1}$.

For the second point, as $(x_\alpha)_{\alpha\in I}$ is an increasing sequence and $|I| > |L|$, by Proposition 2.1 there is an ordinal α such that $x_\alpha = x_{\alpha+1} \in f(x_\alpha)$. That is, x_α is a fixed-point of f.

Finally, for the third point, assume $\bar{x} \in f(\bar{x})$ is a minimal fixed-point of f. Now, let us show by (transfinite) induction on α that there is an increasing orbit $(x_\alpha)_{\alpha\in I}$ of f such that $x_\alpha \leq \bar{x}$ for all α.

The case where $\boldsymbol{\alpha = 0}$. $x_0 = \bot \leq \bar{x}$.

$\boldsymbol{\alpha}$ successor ordinal. By induction, $x_\alpha \leq \bar{x}$. As f is S-monotone and inflationary, $\{x_\alpha\} \preceq_S f(x_\alpha) \preceq_S f(\bar{x})$. But, $\bar{x} \in f(\bar{x})$, so we can choose $x_{\alpha+1} \in f(x_\alpha)$ such that $x_\alpha \leq x_{\alpha+1} \leq \bar{x}$.

$\boldsymbol{\alpha}$ limit ordinal. By induction, $x_\beta \leq \bar{x}$ holds for all $\beta < \alpha$, which implies that $x_\alpha = \bigvee_{\beta<\alpha} x_\beta \leq \bar{x}$.

The sequence $(x_\alpha)_{\alpha\in I}$ is increasing and, thus, by Proposition 2.1 there is an ordinal α such that $x_\alpha = x_{\alpha+1} \in f(x_\alpha)$. So, x_α is a fixed-point of f with $x_\alpha \leq \bar{x}$. As $\bar{x}$ is a minimal, $x_\alpha = \bar{x}$. ∎

Example 19. Consider the lattice $\mathcal{FOUR}$. Let g be a multivalued function such that $g(\bot) = \{f, t\}$, $g(f) = \{f\}$, $g(t) = \{t\}$, $g(\top) = \{\top\}$. It can easily be verified that g is S-monotone and inflationary and that $Fix(g) = \{f, t, \top\}$. Then, we may have the following orbits:

$$\begin{array}{rcl} o_1 & = & (\bot, f, f, f, f), \\ o_2 & = & (\bot, t, t, t, t, t)\ . \end{array}$$

Orbit o_1 converges to the minimal fixed-point f, while o_2 converges to the minimal fixed-point t.

Of course, the dual of Proposition 3.19 holds as well.

PROPOSITION 3.20. *For a multivalued function f,*

1. *if f is deflationary, then each $\top$-orbit is decreasing;*
2. *each decreasing $\top$-orbit converges to a fixed-point of f (if no fixed-point exists, then there is no orbit); and*
3. *if f is H-monotone and deflationary, then for any maximal fixed-point of f there is a $\top$-orbit converging to it.*

Proof. This is dual to Proposition 3.19 (see the appendix, Proposition A.8). ∎

By a straightforward adaptation of the proof of point 3 in Proposition 3.19, we can show the following.

PROPOSITION 3.21. *Let f be an H-monotone, nonempty multivalued function such that for any increasing sequence $(y_\alpha)_{\alpha\in I}$ there is $y \in f(\bigvee_{\alpha\in I} y_\alpha)$ such that $y_\alpha \leq y$ for all $\alpha \in I$. Then, there is an increasing orbit and, thus, a fixed-point of f.*

Proof. Let us show by (transfinite) induction on α that there is an increasing orbit $(x_\alpha)_{\alpha\in I}$ of f and that by Proposition 3.19, point 2, it converges to a fixed-point of f.

The case where $\boldsymbol{\alpha = 0}$. $x_0 = \bot$.

$\boldsymbol{\alpha}$ successor ordinal. By induction, $x_{\alpha-1} \leq x_\alpha$ and $x_\alpha \in f(x_{\alpha-1})$. As f is H-monotone, we have $f(x_{\alpha-1}) \preceq_H f(x_\alpha)$. So, for $x_\alpha \in f(x_{\alpha-1})$, there is $x_{\alpha+1} \in f(x_\alpha)$ s.t. $x_\alpha \leq x_{\alpha+1}$.

$\boldsymbol{\alpha}$ limit ordinal. Consider $(x_\beta)_{\beta\in\alpha}$. By hypothesis, there is $x_{\alpha+1} \in f(\bigvee_{\beta\in\alpha} x_\beta)$ with $x_\beta \leq x_{\alpha+1}$ for all $\beta < \alpha$, and, thus, $x_\alpha = \bigvee_{\beta<\alpha} x_\beta \leq x_{\alpha+1}$. ∎

Note that the condition on the limit is essential, as Example 15 shows: (0,0.5,0.75,. . .) is the increasing sequence that can be built, which converges to 1. But, there is no $x \in f(1)$ such that $1 \leq x$. The dual of Proposition 3.21 is as follows.

PROPOSITION 3.22 (Khamsi and Misane [22]). *Let f be an S-monotone, nonempty multivalued function such that for any decreasing sequence $(y_\alpha)_{\alpha\in I}$ there is $y \in f(\bigwedge_{\alpha\in I} y_\alpha)$ such that $y \leq y_\alpha$ for all $\alpha \in I$. Then there is a decreasing $\top$-orbit and, thus, a fixed-point of f.*

We recall that Proposition 3.22 is the main result described in [22] (see also [16]).

A closer look at the induction step in the previous proof of point 3 of Proposition 3.19 reveals a useful practical case. Indeed, rather than choosing an arbitrary $x_{\alpha+1} \in f(x_\alpha)$ s.t. $x_{\alpha+1} \leq \bar{x}$ with $x_\alpha \leq x_{\alpha+1}$, if $\min f(x_\alpha)$ is nonempty, we may choose an appropriate $x_{\alpha+1} \in \min f(x_\alpha)$.

In the following, let $(x_\alpha)_{\alpha\in I}$ be an orbit ($\top$-orbit) of f. We say that $(x_\alpha)_{\alpha\in I}$ is an *orbit ($\top$-orbit) of minimals (maximals)* of f iff $x_{\alpha+1} \in \min f(x_\alpha)$ if $\min f(x_\alpha) \neq \varnothing$ ($x_{\alpha+1} \in \max f(x_\alpha)$ if $\max f(x_\alpha) \neq \varnothing$). Hence, we have the following.

PROPOSITION 3.23. *Consider a multivalued function $f\colon L \to 2^L$.*

1. *If f is inflationary and S-monotone, then for any minimal fixed-point of f there is an orbit $(x_\alpha)_{\alpha\in I}$ of minimals converging to it.*
2. *If f is deflationary and H-monotone, then for any maximal fixed-point of f there is a $\top$-orbit$(x_\alpha)_{\alpha\in I}$ of maximals converging to it.*

Similarly, we have the following.

PROPOSITION 3.24. *Consider a multivalued function $f\colon L \to 2^L$.*

1. *If $f\colon L \to 2^L$ is S-monotone and for all $x \in L$, $f(x)$ has a least element, then there is an orbit $(x_\alpha)_{\alpha\in I}$ of least elements, i.e., $x_{\alpha+1} = \bigwedge f(x_\alpha)$, converging to the least fixed-point of f.*
2. *If $f\colon L \to 2^L$ is H-monotone and for all $x \in L$, $f(x)$ has a greatest element, then there is a $\top$-orbit $(x_\alpha)_{\alpha\in I}$ of greatest elements, i.e., $x_{\alpha+1} = \bigvee f(x_\alpha)$, converging to the greatest fixed-point of f.*

Proof.

1. From Proposition 3.10, we know that f has a least fixed-point $\bar{x}$. Now, we proceed similarly as for Proposition 3.19, point 3. Let us show by (transfinite) induction on α that there is an increasing orbit $(x_\alpha)_{\alpha\in I}$ of f s.t. $x_{\alpha+1} = \bigwedge f(x_\alpha)$ (if α ordinal), and $x_\alpha \leq \bar{x}$ for all α.

The case where $\boldsymbol{\alpha = 0}$. $x_0 = \bot \leq \bar{x}$.

$\boldsymbol{\alpha}$ successor ordinal. By induction, $x_{\alpha-1} \leq x_\alpha \leq \bar{x}$ and $x_\alpha = \bigwedge f(x_{\alpha-1})$. As f is S-monotone, $f(x_{\alpha-1}) \preceq_S f(x_\alpha) \preceq_S f(\bar{x})$. But, $\bar{x} \in f(\bar{x})$, and, thus, there is $y_1 \in f(x_\alpha)$ such that $y_1 \leq \bar{x}$. Consider $x_{\alpha+1} = \bigwedge f(x_\alpha)$. As $x_{\alpha+1} \in f(x_\alpha)$, $x_{\alpha+1} \leq y_1 \leq \bar{x}$ follows. But then, for $x_{\alpha+1} \in f(x_\alpha)$ there is $y_2 \in f(x_{\alpha-1})$ such that $y_2 \leq x_{\alpha+1}$. Consider $x_\alpha = \bigwedge f(x_{\alpha-1})$. By induction, $x_\alpha \in f(x_{\alpha-1})$ and, thus, $x_\alpha \leq y_2 \leq x_{\alpha+1} \leq y_1 \leq \bar{x}$.

$\boldsymbol{\alpha}$ limit ordinal. By induction, $x_\beta \leq x_{\beta+1} \leq \bar{x}$ holds for all $\beta < \alpha$, which implies that $x_\alpha = \bigvee_{\beta<\alpha} x_\beta = \bigvee_{\beta<\alpha} x_{\beta+1} \leq \bar{x}$. As f is S-monotone, $f(x_\beta) \preceq_S f(x_\alpha) \preceq_S f(\bar{x})$ for $\beta < \alpha$. But, $\bar{x} \in f(\bar{x})$, and, thus, there is $y_1 \in f(x_\alpha)$ such that $y_1 \leq \bar{x}$. Consider $x_{\alpha+1} = \bigwedge f(x_\alpha)$. As $x_{\alpha+1} \in f(x_\alpha)$, $x_{\alpha+1} \leq y_1 \leq \bar{x}$ follows. Similarly, as $f(x_\beta) \preceq_S f(x_\alpha)$, for $x_{\alpha+1} \in f(x_\alpha)$ and $x_{\beta+1} = \bigwedge f(x_\beta)$, we have by induction $x_{\beta+1} \in f(x_\beta)$ and, thus, $x_{\beta+1} \leq x_{\alpha+1}$. Therefore, $x_\beta \leq x_{\beta+1} \leq x_{\alpha+1} \leq \bar{x}$ and, thus, $x_\alpha = \bigvee_{\beta<\alpha} x_\beta = \bigvee_{\beta<\alpha} x_{\beta+1} \leq x_{\alpha+1} \leq \bar{x}$.

The sequence $(x_\alpha)_{\alpha\in I}$ is increasing and, thus, by Proposition 2.1 there is an ordinal α such that $x_\alpha = x_{\alpha+1} \in f(x_\alpha)$. So, x_α is a fixed-point of f with $x_\alpha \leq \bar{x}$. As $\bar{x}$ is the least fixed-point, $x_\alpha = \bar{x}$.

Point 2 can be shown similarly. □

Interestingly, f being S-monotone and inflationary does not guarantee that $\Phi(f)$ has minimals, and, thus, a minimal fixed-point may not exist (Example 9). However, we have the following.

PROPOSITION 3.25. *Let f be an inflationary, $\bigwedge$-preserving multivalued function such that $\Phi(f) \neq \varnothing$.*

1. *Then f has minimal fixed-points and there are orbits converging to them.*
2. *If f is also $\bigvee$-preserving, then ω steps are sufficient to reach a minimal fixed-point.*

Proof. The first item follows immediately from Propositions 3.7, 3.9, and 3.19. For the second item, consider an orbit $(x_\alpha)_{\alpha\in I}$ converging to a minimal fixed-point $\bar{x}$ of f. Let us show that x_ω is a fixed-point of f. As f is inflationary, the orbit is increasing. Then $x_\omega = \bigvee_{\alpha<\omega} x_\alpha$. As f is $\bigvee$-preserving we have that $f(x_\omega) = f(\bigvee_{\alpha<\omega} x_\alpha) = \{y\colon$ there is $(y_\alpha)_{\alpha<\omega}$ s.t. $y_\alpha \in f(x_\alpha)$ and $y = \bigvee_{\alpha<\omega} y_\alpha\}$. For $0 \leq \alpha < \omega$, let $y_\alpha = x_{\alpha+1}$. Therefore, $y_\alpha \in f(x_\alpha)$ and, thus, $x_\omega = y = \bigvee_{\alpha<\omega} y_\alpha \in f(x_\omega)$. That is, x_ω is a fixed-point of f and $x_\omega \leq \bar{x}$ and, thus, $x_\omega = \bar{x}$. □

Clearly, the dual of Proposition 3.25 holds as well.

PROPOSITION 3.26. *If a multivalued function f is deflationary and $\bigvee$-preserving, and $\Psi(f) \neq \varnothing$, then f has maximal fixed-points and there are $\top$-orbits converging to them. If f is also $\bigwedge$-preserving, then ω steps are sufficient to reach a maximal fixed-point.*

We conclude this part by showing a strict relationship between S-monotone and inflationary operators. For a multivalued function $f\colon L \to 2^L$, let us define

$$g(x) = x \oplus f(x) = \{x \vee y\colon y \in f(x)\}\ . \tag{3.6}$$

Note that if $f(x) = \varnothing$, then $g(x) = \varnothing$.

PROPOSITION 3.27. *For $f\colon L \to 2^L$, $g(x) = x \oplus f(x)$ is inflationary. Furthermore, if f is S-monotone, then*

1. *g is S-monotone;*
2. *$x \in f(x)$ implies $x \in g(x)$;*
3. *$x \in g(x)$ implies $f(x) \preceq_S \{x\}$;*
4. *if x is a minimal fixed point of g, then x is a minimal fixed point of f; and*
5. *if x is a minimal fixed point of f and f is also inflationary, then x is a minimal fixed point of g.*

Proof. Consider f and g. If $f(x) = \varnothing$, then $\{x\} \preceq_S g(x) = \varnothing$. Otherwise, for $y \in g(x)$, $x \leq y$. Therefore, $\{x\} \preceq_S g(x)$ and, thus, g is inflationary. Now, suppose f is S-monotone.

1. This is easy to prove, as g is a combination of S-monotone functions.

2. If $x \in f(x)$, then by definition of g, $x = x \vee x \in g(x)$.

3. If $x \in g(x)$, then for some $y \in f(x)$, $x = x \vee y$. Therefore, $y \leq x$ and, thus, $f(x) \preceq_S \{x\}$.

4. Assume x is a minimal fixed-point of g, i.e., $x \in g(x) = x \oplus f(x)$. Therefore, there is $y \in f(x)$ such that $y \leq x$. As f is S-monotone, $f(y) \preceq_S f(x)$. That is, there is $z \in f(y)$ such that $z \leq y$ and, thus, $y = y \vee z$. Therefore, $y \in g(y)$. As x is minimal and $y \leq x$, $y = x$ follows, and, thus, $x \in f(x)$. To prove that x is a minimal fixed-point of f, assume there is $y \leq x$ such that $y \in f(y)$. By point 2, $y \in g(y)$, and, thus, as x is a minimal fixed-point of g, $y = x$ follows.

5. Assume x is a minimal fixed-point of f. By point 2 $x \in g(x)$. To prove that x is a minimal fixed-point of g, assume there is $y \leq x$ such that $y \in g(y)$. Then by

point 3 $f(y) \preceq_S \{y\}$ and, thus, $y \in \Phi(f)$. By Proposition 3.7, $y \in f(y)$, and, thus, as x is a minimal fixed-point of f, $y = x$ follows. □

We note that the inflationary condition in point 5 in Proposition 3.27 is necessary.

Example 20. Consider $L = \{0\} \cup \{1/n : n = 1, 2, \dots\}$ and the multivalued mapping $f : L \to 2^L$ defined as follows:

$$f(0) = \{1/n : n = 1, 2, \dots\},$$
$$f(1/n) = \{1\} \cup \{1/(n+k) : k = 1, 2, \dots\}.$$

f is S-monotone, but not inflationary ($\{1/n\} \not\preceq_S f(1/n)$), and 1 is its only fixed-point. However, the function $g(x) = x \oplus f(x)$ has the following definition:

$$g(0) = \{1/n : n = 1, 2, \dots\},$$
$$g(1/n) = \{1, 1/n\} ,$$

which has infinitely many fixed points and none is minimal.

Of course, Proposition 3.27 has its dual as well. Let

$$h(x) = x \otimes f(x) = \{x \wedge y \colon y \in f(x)\} . \tag{3.7}$$

PROPOSITION 3.28. *For $f \colon L \to 2^L$, $h(x) = x \otimes f(x)$ is deflationary. Furthermore, if f is H-monotone, then*

1. *h is H-monotone;*
2. *$x \in f(x)$ implies $x \in h(x)$;*
3. *$x \in h(x)$ implies $\{x\} \preceq_H f(x)$;*
4. *if x is a maximal fixed point of h, then x is a maximal fixed point of f; and*
5. *if x is a maximal fixed point of f and f is also deflationary, then x is a maximal fixed point of h.*

Proof. This is dual to Proposition 3.27 (see the appendix, Proposition A.9). □

We report here some other related results known in the literature. For instance, [45] (which relies on [33]) gives a condition for the existence of a least fixed-point.

PROPOSITION 3.29 (Stouti [45]). *Let $f \colon L \to 2^L$ be a multivalued function, where $\mathcal{L} = \langle L, \preceq \rangle$ is a* complete partial order *(CPO) with $\perp$, i.e., any nonempty chain in L has a supremum in L, and $\perp \in L$. Assume that for any $x \in L$, $f(x)$ is nonempty, and that if for any $x, y \in L$ with $x < y$, then for every $a \in f(x)$ and $b \in f(y)$, we have that $a \leq b$.*[5]

1. *Then f has a least fixed-point.*
2. *If there is $a \in L$ such that for all $b \in f(a)$ we have $a \leq b$, then f has a least fixed-point in the subset $\{a \in L \mid a \leq x\}$.*

For completeness, we recall that [33] states the following.

PROPOSITION 3.30 (Orey [33]). *Let $f \colon L \to 2^L$ be a multivalued function, where $\mathcal{L} = \langle L, \preceq \rangle$ is a CPO with $\perp$, i.e., any nonempty chain in L has a supremum in L, and $\perp \in L$. Assume that for any $x \in L$, $f(x)$ is nonempty, and that if for any $x, y \in L$ with $x < y$, then for every $a \in f(x)$ and $b \in f(y)$, we have that $a \leq b$. If there is $a \in L$ such that $\{a\} \preceq_S f(a)$, then f has a fixed-point.*

The above proposition relies on the fact that under its condition we have that $\{a\} \preceq_S f(a) \preceq_S f^2(a) \preceq_S \cdots$, which allows us to build an increasing and, thus, eventually stationary, orbit.

[5] Hence, this is a strictly stronger monotonicity condition than the EM-monotonicity.

We conclude this section by extending $\leq$ to L^n pointwise: for $(x_1, \ldots, x_n) \in L^n$ and $(y_1, \ldots, y_n) \in L^n$, we say that $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n)$ iff for all i, $x_i \leq y_i$. For $\mathbf{x}, \mathbf{y} \in L^n$, $\mathbf{x} \wedge \mathbf{y}$ and $\mathbf{x} \vee \mathbf{y}$ are defined pointwise, i.e., $\mathbf{x} \wedge \mathbf{y} = (x_1 \wedge y_1, \ldots, x_n \wedge y_n)$ and $\mathbf{x} \vee \mathbf{y} = (x_1 \vee y_1, \ldots, x_n \vee y_n)$. Since $\mathcal{L} = \langle L, \leq \rangle$ is a complete lattice, so is $\mathcal{L}^n = \langle L^n, \leq \rangle$. All definitions and properties of single-valued functions and multivalued functions over the domain L of $\mathcal{L}$ can be extended to $\mathcal{L}^n$ as well.

4. Generalized logic programs. We now apply the results developed so far to a general form of logic programs. Consider a complete lattice $\mathcal{L} = \langle L, \leq \rangle$, which will act as our truth-value set. Formulae will have a degree of truth in L. Let $\mathcal{F}$ be a family of computable n-ary functions $f\colon L^n \to L$, called (logical) *connectors*.[6] Connectors will be used to build logical formulae from logical atoms. For instance, the join (disjunction function) $\vee$ and the meet (conjunction function) $\wedge$ are connectors. $f(x, y) = \max(0, x + y - 1)$ is also a connector over $[0, 1]^2$. Connectors need not necessarily be monotone functions. Let $\mathcal{V}$ be a set of variable symbols and $\mathcal{A}$ be a set of atomic formulae $P(t_1, \ldots, t_m)$, where P is an m-ary predicate symbol and all t_i are terms. A *term* is defined inductively, as usual, as being either a variable, a constant, or the application of a logical function symbol to terms [26].

A *formula* is either an atom A or an expression of the form $f(A_1, \ldots, A_n)$, where f is an n-ary connector and each A_i is an atom. For ease of presentation, the connectors $\wedge$ and $\vee$ are used in fix notation. The intuition behind a formula $f(A_1, \ldots, A_n)$ is that the truth degree of the formula is given by evaluating the truth degree of each A_i and then applying f to these degrees to obtain the final degree. Of course, the function f may well be the composition of functions, $f_1 \circ \cdots \circ f_n$. For instance, over $[0, 1]$, $\min(A(x, y), B(y, z)) \cdot \max(\neg R(z), 0.7) + G(x)$ is a formula. In this case, the truth of the formula is determined from the truth of the atoms $A(x, y), B(y, z), R(z)$, and $G(x)$ by applying the specified arithmetic functions. Truth degrees in L may appear in formulae (like 0.7 above).

A *logic program* $\mathcal{P}$ is a set of rules $\psi \leftarrow \varphi$, where ψ and φ are formulae (respectively, called the head and the body); i.e., rules are of the form

$$g(B_1, \ldots, B_k) \leftarrow f(A_1, \ldots, A_n) \ ,$$

where f, g are connectors and B_i and A_j are atoms. Free variables in a rule are understood to be *universally quantified*. For instance, over $[0, 1]$,

$$\max(A(x), B(x)) \leftarrow 0.7 \cdot \max(0, A(x, y) + B(y, z) - 1)$$

is a rule. The intuition is that the truth of either $A(x)$ or $B(x)$ is at least the truth degree of the body. We point out that the form of the rules is sufficiently expressive to encompass all approaches we are aware of to monotone many-valued logic programming.[7] So far, in many-valued logic programming, rules are either of the "deterministic" form $B \leftarrow f(A_1, \ldots, A_n)$ or of the form $B_1 \vee \ldots \vee B_k \leftarrow A_1 \wedge \ldots \wedge A_n$ (see, e.g., [46]).

In the following, by $\mathcal{P}^*$ we denote the ground instantiation of $\mathcal{P}$. If there is no constant in $\mathcal{P}$, then we consider some constant, say c, to form ground terms. Note

[6] By computable we mean that the result of f is computable in a finite amount of time.

[7] Also note that any classical first order clause $A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \neg B_n$ (with $k + n > 0$) is a rule of the form $A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_n$. If $k = 0$, we use $\bot$ in the left-hand side, while if $n = 0$, we use $\top$ in the right-hand side.

that $|\mathcal{P}^*|$ may be not finite, but it is countable. If we restrict a term to be either a variable or a constant, then $|\mathcal{P}^*|$ is finite.

We next consider the usual notion of interpretation and generalize the notion of satisfiability (see, e.g., [46]) to our setting. An *interpretation* is a mapping I from ground atoms to members of L. For a ground atom A, $I(A)$ indicates the degree of truth to which A is true under I. An interpretation I is extended from atoms to nonatomic formulae in the usual way as follows:

1. for $b \in L$, $I(b) = b$; and
2. $I(f(A_1, \ldots, A_n)) = f(I(A_1), \ldots, I(A_n))$.

An interpretation I *satisfies* (is a *model* of) a ground rule $\psi \leftarrow \varphi \in \mathcal{P}^*$, denoted $I \models \psi \leftarrow \varphi$ iff $I(\varphi) \leq I(\psi)$. Essentially, we postulate that the consequent ψ of the ground rule (implication) is at least as true as the antecedent φ. We further say that I *satisfies* (is a *model* of) a logic program $\mathcal{P}$, denoted $I \models \mathcal{P}$, iff I satisfies all ground rules in $\mathcal{P}^*$. Given an interpretation I, by $\mathcal{P}[I]$ we denote the set of ground rules of $\mathcal{P}^*$ in which the body has been evaluated by means of I, i.e.,

$$\mathcal{P}[I] = \{\psi \leftarrow I(\varphi) \colon \psi \leftarrow \varphi \in \mathcal{P}^*\} \ .$$

It is easily verified that $I \models \mathcal{P}$ iff $I \models \mathcal{P}[I]$.

Given two interpretations I, J, we define $I \leq J$ pointwise; i.e., $I \leq J$ iff for all ground atoms $I(A) \leq J(A)$. It is easily verified that the set of interpretations, denoted $\hat{L}$, forms a complete lattice as well, i.e., $\langle \hat{L}, \leq \rangle$ is a complete lattice, with least element $I_\bot$ (mapping all atoms to $\bot$) and greatest element $I_\top$ (mapping all atoms to $\top$). If L is countable, then so is $\hat{L}$. If L is finite and a term is either a variable or a constant, then $\hat{L}$ is finite as well.

It is worth noting that $I \leq J$ does not necessarily imply that $I(\psi) \leq J(\psi)$ for a formula ψ. However, as one may expect, if the functions involved in ψ are monotone, then from $I \leq J$, $I(\psi) \leq J(\psi)$ follows.

PROPOSITION 4.1. *Let I, J be two interpretations such that $I \leq J$. If ψ is a formula involving monotone functions $f \in \mathcal{F}$, then $I(\psi) \leq J(\psi)$.*

Proof. The proof is on the structure of ψ. Assume ψ is an atomic formula A. Then by definition of $I \leq J$, $I(A) \leq J(A)$. If $\psi = f(A_1, \ldots, A_n)$, then using induction on A_i and the fact that f is monotone we have that

$$\begin{aligned} I(f(A_1, \ldots, A_n)) &= f(I(A_1), \ldots, I(A_n)) \\ &\leq f(J(A_1), \ldots, J(A_n)) \\ &= J(f(A_1, \ldots, A_n)) \ , \end{aligned}$$

which concludes the proof. □

Note that the connectors $\wedge, \vee$ are monotone. More generally, let us define the *evaluation* function

$$e(I, \psi) = I(\psi) \ .$$

Then the above proposition establishes that the function $e(I, \psi)$ is monotone in I if all the connectors in ψ are monotone; i.e., if $I \leq J$, then $e(I, \psi) \leq e(J, \psi)$. Similarly, we can show that if all the connectors in ψ are $\bigvee$-preserving ($\bigwedge$-preserving), then $e(I, \psi)$ is $\bigvee$-preserving ($\bigwedge$-preserving) in I.

PROPOSITION 4.2. *If all the connectors in ψ are $\bigvee$-preserving ($\bigwedge$-preserving), then $e(I, \psi)$ is $\bigvee$-preserving ($\bigwedge$-preserving) in I.*

Proof. Let us prove the $\bigwedge$-preserving case. The other case is similar. Consider a decreasing sequence of interpretations $(I_\alpha)_{\alpha\in I}$. We have to show that $e(\bigwedge_\alpha I_\alpha, \psi) = \bigwedge_\alpha e(I_\alpha, \psi)$. That is, $(\bigwedge_\alpha I_\alpha)(\psi) = \bigwedge_\alpha I_\alpha(\psi)$. Let $\bar{I}$ be the interpretation $\bar{I} = \bigwedge_\alpha I_\alpha$. The proof is on the structure of ψ. Assume ψ is an atomic formula A. Then by definition, $e(\bar{I}, A) = \bar{I}(A) = (\bigwedge_\alpha I_\alpha)(A) = \bigwedge_\alpha I_\alpha(A) = \bigwedge_\alpha e(I_\alpha, A)$. If $\psi = f(A_1, \ldots, A_n)$, then using induction on A_i and the fact that f is $\bigwedge$-preserving we have that

$$\begin{aligned}
e(\bar{I}, f(A_1, \ldots, A_n)) &= \bar{I}(f(A_1, \ldots, A_n)) \\
&= f(\bar{I}(A_1), \ldots, \bar{I}(A_n)) \\
&= f(e(\bar{I}, A_1), \ldots, e(\bar{I}, A_n)) \\
&= f\left(\bigwedge_\alpha e(I_\alpha, A_1), \ldots, \bigwedge_\alpha e(I_\alpha, A_n)\right) \\
&= \bigwedge_\alpha f(e(I_\alpha, A_1), \ldots, e(I_\alpha, A_n)) \\
&= \bigwedge_\alpha f(I_\alpha(A_1), \ldots, I_\alpha(A_n)) \\
&= \bigwedge_\alpha I_\alpha(f(A_1, \ldots, A_n)) \\
&= \bigwedge_\alpha e(I_\alpha, f(A_1, \ldots, A_n)) \ ,
\end{aligned}$$

which concludes the proof. □

Useful to note is the following.

PROPOSITION 4.3. *$\vee$ ($\wedge$) is $\bigvee$-preserving ($\bigwedge$-preserving).*

Proof. Let us show that $\vee$ is $\bigvee$-preserving. Indeed, for all increasing sequences $(\langle x_\alpha, y_\alpha\rangle)_{\alpha\in I}$, we have that

$$\begin{aligned}
\vee\left(\bigvee_\alpha \langle x_\alpha, y_\alpha\rangle\right) &= \vee\left(\left\langle \bigvee_\alpha x_\alpha, \bigvee_\alpha y_\alpha \right\rangle\right) \\
&= \left(\bigvee_\alpha x_\alpha\right) \vee \left(\bigvee_\alpha y_\alpha\right) = \bigvee_\alpha (x_\alpha \vee y_\alpha) \\
&= \bigvee_\alpha \vee(x_\alpha, y_\alpha) \ .
\end{aligned}$$

In a similar way, $\wedge$ is $\bigwedge$-preserving. □

In general, $\vee$ ($\wedge$) is not $\bigwedge$- ($\bigvee$-) preserving.

Example 21 (see [5]). Let us show that the meet function is not $\bigvee$-preserving in general. Consider the complete lattice obtained from the set of closed subsets of the unit disk, with the meet defined as the set-intersection and the join defined as the topological closure of set-union (closure is needed here because the arbitrary union of closed sets need not be closed). This definition provides a complete distributive lattice structure. Now, for all $n \in \mathbb{N}$, define $x_{n,1} = a =$ the unit circle, i.e., the points $\langle x, y\rangle$ satisfying $x^2 + y^2 = 1$, and define $x_{n,2} =$ the disk of radius $1 - 1/n$, that is, the points $\langle x, y\rangle$ satisfying $x^2 + y^2 \leq 1 - 1/n$. The sequence $(\langle x_{n,1}, x_{n,2}\rangle)_{n\in\mathbb{N}}$ is an increasing sequence. $\bigvee_n x_{n,2}$ turns out to be the whole unit disk; therefore $(\bigvee_n x_{n,1}) \wedge (\bigvee_n x_{n,2}) = a \wedge (\bigvee_n x_{n,2})$ is the unit circle. On the other hand, $x_{n,1} \wedge x_{n,2} = a \wedge x_{n,2}$ is the empty set (which is a closed subset), and hence $\bigvee_n (x_{n,1} \wedge x_{n,2}) = \bigvee_n (a \wedge x_{n,2})$

is the empty set. As a consequence, $(\bigvee_n x_{n,1}) \wedge (\bigvee_n x_{n,2}) \neq \bigvee_n (x_{n,1} \wedge x_{n,2})$ and, thus, the meet function $\wedge$ is not $\bigvee$-preserving.

However, it can easily be shown that $\vee$ ($\wedge$) is $\bigwedge$- ($\bigvee$-) preserving if $\mathcal{L} = \langle L, \leq \rangle$ is *finite*, i.e., $|L| \in \mathbb{N}$. From a practical point of view this is a limitation we can live with, especially taking into account that computers have finite resources. In particular, this includes also the case of the rational numbers in $[0, 1]$ under a given fixed decimal precision p (e.g., $p = 2$) and the Boolean lattice over $\{0, 1\}$.

PROPOSITION 4.4. *If $\mathcal{L} = \langle L, \leq \rangle$ is* finite*, then $\vee$ and $\wedge$ are limit-preserving.*

Note that Proposition 4.4 can be extended to any *finite* n-ary meet (join) function. Furthermore, Proposition 4.4 holds also for any *infinite* n-ary meet (join) function, as for a finite lattice, an infinite meet (join) is equivalent to a finite meet (join). Indeed, only finitely many values can appear in the infinite meet (join). Another useful and special case is when $\mathcal{L} = \langle [0, 1], \leq \rangle$, as it is used in fuzzy logic programming (see, e.g., [48]).

PROPOSITION 4.5. *$\vee$ and $\wedge$ are limit-preserving on $[0, 1] \times [0, 1]$.*

4.1. Fixed-point characterization of logic programs. The aim of this section is to extend the usual fixed-point characterization of classical logic programs [26] to the case of generalized logic programs. So, let $\mathcal{P}$ be a logic program. Consider $\mathcal{L} = \langle L, \leq \rangle$ and the related complete lattice of interpretations $\langle \hat{L}, \leq \rangle$. We next define a multivalued function over $\hat{L}$ whose set of fixed-points coincides with the set of models of $\mathcal{P}$.

The *multivalued immediate consequence* operator mapping interpretations into sets of interpretations, $T_{\mathcal{P}} \colon \hat{L} \to 2^{\hat{L}}$, is defined as

$$T_{\mathcal{P}}(I) = \{J \colon J \models \mathcal{P}[I], I \leq J\} \ .$$

Note that either $T_{\mathcal{P}}(I_\top) = \varnothing$ or $T_{\mathcal{P}}(I_\top) = \{I_\top\}$. Also note that, unlike in the single-valued case, we do not necessarily have $T_{\mathcal{P}}(I) \neq \varnothing$.

Example 22. For any interpretation I and for $\mathcal{P} = \{A \vee B \leftarrow \top, \ \ \bot \leftarrow A, \ \ \bot \leftarrow B \ \}$, $T_{\mathcal{P}}(I) = \varnothing$ holds.

However, note that for the specific case of rules of the form below (where A_i, B_j is neither $\top$ nor $\bot$ and $k \geq 1$)

$$A_1 \vee \cdots \vee A_k \leftarrow f(B_1, \ldots, B_n) \ ,$$

it is easily verified that for any I, $I_\top \in T_{\mathcal{P}}(I) \neq \varnothing$, and in particular $T_{\mathcal{P}}(I_\top) = \{I_\top\}$. Also, note that $T_{\mathcal{P}}(I)$ may not be countable.

Example 23. Consider $L = [0, 1]$ and $\mathcal{P}$ with rule $A \leftarrow 0$. Then for any interpretation $I \neq I_\top$, $T_{\mathcal{P}}(I) = \{J \mid I \leq J \text{ and } J(A) \geq 0.3\}$ holds. Hence, $T_{\mathcal{P}}(I)$ is not countable.

The $T_{\mathcal{P}}$ function has the desired property in which models of logic programs are fixed-points and vice versa.

PROPOSITION 4.6. *$I \models \mathcal{P}$ iff $I \in T_{\mathcal{P}}(I)$.*

Proof. $I \models \mathcal{P}$ iff $I \models \mathcal{P}[I]$ iff $I \in T_{\mathcal{P}}(I)$. □

Example 24. Over $\mathcal{L} = \langle \{0, 1\}, \leq \rangle$, consider $\mathcal{P} = \{A \leftarrow 1 - B\}$ and $I(A) = 0, I(B) = 1$. Then

$$\begin{aligned} T_{\mathcal{P}}(I) &= \{J \mid J \models \mathcal{P}[I], I \leq J\} \\ &= \{J \mid J \models A \leftarrow 0, I \leq J\} \\ &= \{J \mid I \leq J\} \\ &= \{I, I'\}, \end{aligned}$$

where $I'(A) = I'(B) = 1$. Note that $I \in T_{\mathcal{P}}(I)$ and I is a model of $\mathcal{P}$. Note also that the truth combination function $f(x) = 1-x$ in rule $A \leftarrow 1-B$ is not monotone. Hence determining models of a logic program is equivalent to investigating the fixed-points of the multivalued function $T_{\mathcal{P}}$.

In the following, we will determine which properties of section 3 about multivalued functions apply to $T_{\mathcal{P}}$ and which are specific of $T_{\mathcal{P}}$ only. To start with, as definition $J \in T_{\mathcal{P}}(I)$ implies $I \leq J$ we immediately have the following.

PROPOSITION 4.7. *$T_{\mathcal{P}}$ is inflationary.*

Furthermore, we also can show the following.

PROPOSITION 4.8. *If all connector functions in the body φ of rules $\psi \leftarrow \varphi \in \mathcal{P}$ are $\bigvee$-preserving, then $T_{\mathcal{P}}$ is $\bigvee$-preserving and, thus, S-monotone.*

Proof. Let $(I_\alpha)_{\alpha\in I}$ be an increasing sequence of interpretations. Let $\bar{I} = \bigvee_\alpha I_\alpha$. We have to show that $T_{\mathcal{P}}(\bar{I}) = \{J\colon$ there is $(J_\alpha)_{\alpha\in I}$ s.t. $J_\alpha \in T_{\mathcal{P}}(I_\alpha)$ and $J = \bigvee_\alpha J_\alpha\}$ $(= \bigvee_\alpha T_{\mathcal{P}}(I_\alpha))$. So, let $J \in T_{\mathcal{P}}(\bar{I})$. Then $J \models \mathcal{P}[\bar{I}]$ and $\bar{I} \leq J$ and, thus, $I_\alpha \leq J$. Then, using Proposition 4.2, for all ground rules $\psi \leftarrow \varphi \in \mathcal{P}^*$, $I_\alpha(\varphi) \leq \bigvee_\alpha I_\alpha(\varphi) = \bar{I}(\varphi) \leq J(\psi)$. Therefore, $J \models \mathcal{P}[I_\alpha]$ and, thus, $J \in T_{\mathcal{P}}(I_\alpha)$. Hence, $J \in \bigvee_\alpha T_{\mathcal{P}}(I_\alpha)$. Vice versa, let $J \in \bigvee_\alpha T_{\mathcal{P}}(I_\alpha)$. Thus $J = \bigvee_\alpha J_\alpha$ with $J_\alpha \in T_{\mathcal{P}}(I_\alpha)$. It follows that $I_\alpha \leq J_\alpha \leq J$ and $J_\alpha \models P[I_\alpha]$. Then, using Proposition 4.2, for all ground rules $\psi \leftarrow \varphi \in \mathcal{P}^*$, $\bar{I}(\varphi) = \bigvee_\alpha I_\alpha(\varphi) \leq \bigvee_\alpha J_\alpha(\psi) = J(\psi)$ and, thus, $J \models \mathcal{P}[\bar{I}]$. As $\bar{I} = \bigvee_\alpha I_\alpha \leq \bigvee_\alpha J_\alpha = J$, $J \in T_{\mathcal{P}}(\bar{I})$ follows. S-monotonicity follows from Proposition 3.5. ∎

The analogue of Proposition 4.8 does not hold for $\bigwedge$-preserving connector functions.

Example 25. Consider $L = [0,1]$, $a \geq 1$, the function $f(x) = 1/(a+1-x)$, and the logic program $\mathcal{P} = \{\frac{1}{a+1} \leftarrow f(A)\}$. Consider a decreasing sequence of interpretations $I_n(A) = 1/n$, $n \in \mathbb{N}$. Then $\bar{I}(A) = \bigwedge_\alpha I_\alpha(A) = I_\perp(A) = 0$. The function f is monotone—more precisely, $\bigwedge$-preserving—, with maximum value $\frac{1}{a}$ and minimum value $\frac{1}{a+1}$. Furthermore, $f(I_1(A)) = \frac{1}{a}$, while $f(\bar{I}(A)) = \frac{1}{a+1}$ and $f(I_n(A)) = \frac{1}{a+1-1/n} > \frac{1}{a+1}$. Therefore, $T_{\mathcal{P}}(\bar{I}) = \{J\colon J$ interpretation$\}$. On the other hand, $T_{\mathcal{P}}(I_n) = \varnothing$ and, thus,[8] $\bigwedge_n T_{\mathcal{P}}(I_n) = \varnothing$. Therefore, $T_{\mathcal{P}}(\bigwedge_n I_n) \not\subseteq \bigwedge_n T_{\mathcal{P}}(I_n)$; i.e., $T_{\mathcal{P}}$ is not $\bigwedge$-preserving.

Let us define

$$G_{\mathcal{P}}(I) = \{J \vee I\colon J \models \mathcal{P}[I]\}\ . \tag{4.1}$$

Then it is easily verified that $T_{\mathcal{P}}(I) \subseteq G_{\mathcal{P}}(I)$ (from $I \leq J$, $J \vee I = J$). On the other hand, for $J \in G_{\mathcal{P}}(I)$, $J = J' \vee I$, $J' \models \mathcal{P}[I]$, $J' \leq J$, and $I \leq J$. If all connector functions in the head of rules in $\mathcal{P}$ are monotone, then for all ground rules $\psi \leftarrow \varphi \in \mathcal{P}^*$ (using Proposition 4.1), $I(\varphi) \leq J'(\psi) \leq J(\psi)$. Therefore, $J \in T_{\mathcal{P}}(I)$, i.e., $G_{\mathcal{P}}(I) \subseteq T_{\mathcal{P}}(I)$. Therefore we have what follows.

PROPOSITION 4.9. *For any interpretation I, $T_{\mathcal{P}}(I) \subseteq G_{\mathcal{P}}(I)$. If all connector functions in the head of rules in $\mathcal{P}$ are monotone, then $T_{\mathcal{P}}(I) = G_{\mathcal{P}}(I)$.*

Monotonicity is a necessary condition for guaranteeing equivalence among $T_{\mathcal{P}}$ and $G_{\mathcal{P}}$.

Example 26. Over $\mathcal{L} = \langle\{0,1\}, \leq\rangle$, consider the logic program $\mathcal{P} = \{\neg A \leftarrow A\}$. The negation function $\neg x = 1 - x$ is obviously not monotone. Consider $I(A) = 1$ and $J'(A) = 0$. Then, $J' \models \mathcal{P}[I]$ and, thus, $J = I \vee J' = I_\top \in G_{\mathcal{P}}(I)$, but $J \notin T_{\mathcal{P}}(I)$.

[8] Recall that $\bigwedge_n T_{\mathcal{P}}(I_n)$ is shorthand for the right-hand side of (3.5).

A closer analysis shows that we can write $G_{\mathcal{P}}$ similarly to (3.6). Indeed, let $F_{\mathcal{P}}$ be the multivalued function

$$F_{\mathcal{P}}(I) = \{J\colon J \models \mathcal{P}[I]\} \ .$$

Then, it can easily verified that

$$G_{\mathcal{P}}(I) = I \oplus F_{\mathcal{P}}(I) \ .$$

We can show the following.

PROPOSITION 4.10. *If all connector functions in the body φ of rules $\psi \leftarrow \varphi \in \mathcal{P}$ are monotone, then $F_{\mathcal{P}}$ is a multivalued S-monotone operator.*

Proof. Consider interpretations I, J s.t. $I \leq J$. Let us show that $F_{\mathcal{P}}(I) \preceq_S F_{\mathcal{P}}(J)$. If $F_{\mathcal{P}}(J) = \varnothing$, then obviously $F_{\mathcal{P}}(I) \preceq_S F_{\mathcal{P}}(J)$. Otherwise, assume $F_{\mathcal{P}}(J) \neq \varnothing$. Let $J' \in F_{\mathcal{P}}(J)$ and, thus, by definition $J' \models \mathcal{P}[J]$; i.e., for all ground rules $\psi \leftarrow \varphi \in \mathcal{P}^*$, $J(\varphi) \leq J'(\psi)$. But, $I \leq J$ and, using Proposition 4.1, $I(\varphi) \leq J(\varphi) \leq J'(\psi)$. Therefore, $J' \models \mathcal{P}[I]$ and, thus, $J' \in F_{\mathcal{P}}(I)$, which concludes the proof. □

Note that the proof of the proposition above shows in fact that if $I \leq J$, then $F_{\mathcal{P}}(J) \subseteq F_{\mathcal{P}}(I)$ and, thus, $F_{\mathcal{P}}(I) \preceq_S F_{\mathcal{P}}(J)$.

Now, taking into account Propositions 3.27, 4.7, and 4.9, the following analogue of Proposition 3.27 can be obtained.

PROPOSITION 4.11. *$G_{\mathcal{P}}$ is inflationary. Furthermore, if all connector functions in $\mathcal{P}$ are monotone, then* (i) *$T_{\mathcal{P}} = G_{\mathcal{P}}$;* (ii) *$T_{\mathcal{P}}$ is S-monotone;* (iii) *$I \in F_{\mathcal{P}}(I)$ implies $I \in T_{\mathcal{P}}(I)$;* (iv) *$I \in T_{\mathcal{P}}(I)$ implies $F_{\mathcal{P}}(I) \leq \{I\}$; and* (v) *for any interpretation I, I is a minimal fixed-point of $F_{\mathcal{P}}$ iff I is a minimal fixed-point of $T_{\mathcal{P}}$.*

By relying on Propositions 4.6, 3.7, and 3.19, we have the following.

PROPOSITION 4.12. *Let $\mathcal{P}$ be a logic program. Then*

1. *$\Phi(T_{\mathcal{P}}) \neq \varnothing$ iff $\mathcal{P}$ has a model;*
2. *each orbit of $T_{\mathcal{P}}$ is increasing and converges to a model of $\mathcal{P}$;*
3. *if I is a minimal model of $\mathcal{P}$ and all connector functions in $\mathcal{P}$ are monotone, then there is an orbit converging to I.*

Unlike the general case, for $T_{\mathcal{P}}$ we can be even more precise and reach any model.

PROPOSITION 4.13. *If I is a model of $\mathcal{P}$ and all connector functions in $\mathcal{P}$ are monotone, then there is an orbit converging to I.*

Proof. We show that if $I \models \mathcal{P}$, then there is an orbit converging to I. By Proposition 4.6, $I \in T_{\mathcal{P}}(I)$. The proof is similar for point 3 in Proposition 3.19. We know that each orbit of $T_{\mathcal{P}}$ converges to a model of $\mathcal{P}$. As in Proposition 3.19, we can show by induction on α that there is an orbit $(I_\alpha)_{\alpha \in I}$ of elements $I_{\alpha+1} \in T_{\mathcal{P}}(I_\alpha)$ with $I_0 = I_\perp$, such that $I_\alpha \leq I$ for all α. Therefore, the orbit converges to a model $I_{\bar{\alpha}}$ of $\mathcal{P}$, where $I_{\bar{\alpha}} = I_{\bar{\alpha}+1}$, $I_{\bar{\alpha}} \leq I$. By Proposition 4.6, $I_{\bar{\alpha}} \in T_{\mathcal{P}}(I_{\bar{\alpha}})$. Now, let us show that $I \in T_{\mathcal{P}}(I_{\bar{\alpha}})$. Indeed, from $I_{\bar{\alpha}} \models \mathcal{P}$ and $I \models \mathcal{P}$, for all $\psi \leftarrow \varphi \in \mathcal{P}^*$, from $I_{\bar{\alpha}} \leq I$, using Proposition 4.1, we have $I_{\bar{\alpha}}(\varphi) \leq I(\varphi) \leq I(\psi)$. Therefore, $I \in T_{\mathcal{P}}(I_{\bar{\alpha}})$ and, thus, the sequence $I_0 = \perp, \ldots, I_{\bar{\alpha}}, I, I, \ldots$ is an orbit converging to I. □

Example 27. Consider the logic program over the Boolean lattice on $\{0,1\}$, $\mathcal{P} = \{(a \vee b \leftarrow 1), (c \leftarrow a), (a \wedge c \wedge d \leftarrow b)\}$. The unique minimal model is $\bar{I}(a,b,c,d) = \langle 1,0,1,0\rangle$. The following are two orbits p_1, p_2 of $T_{\mathcal{P}}$:

$$p_1 = \langle 0,0,0,0\rangle \rightarrow \langle 1,0,0,0\rangle \rightarrow \langle 1,0,1,0\rangle \rightarrow \langle 1,0,1,0\rangle \ ,$$
$$p_2 = \langle 0,0,0,0\rangle \rightarrow \langle 0,1,0,0\rangle \rightarrow \langle 1,1,1,1\rangle \rightarrow \langle 1,1,1,1\rangle \ .$$

Both $\langle 1,0,1,0\rangle$ and $\langle 1,1,1,1\rangle$ are fixed-points, i.e., models, and p_1 reaches the minimal one.

Note that the previous two propositions allow us also to decide, if the lattice is *finite*, whether or not a logic program has a model. Indeed, it suffices to try to build an orbit, starting with $I_\bot$, and systematically use all alternatives (which are finite) at each step. If no orbit can be built, no model exists.

As for the general case (see Example 9), $T_\mathcal{P}$ may not have minimal fixed-points.

Example 28. Consider the logic program $\mathcal{P} = \{f(A) \leftarrow 1\}$, where $f(x) = 1$ if $x > 0$ and $f(0) = 0$. Then $I \models \mathcal{P}$ iff $I(A) > 0$, and no minimal model exists.

The following example shows that if a connector function is not $\bigwedge$-preserving, then there is a decreasing sequence of models not converging to a model.

Example 29. Consider $L = [0, 1]$ and the connector function f such that $f(0) = 0$ and for $x > 0$, $f(x) = 1$. Now, consider the logic program $\mathcal{P} = \{A \vee f(B) \leftarrow 1\}$. Then the decreasing sequence $(I_n)_{n\in\mathbb{N}}$ of interpretations I_n, where $I_n(A) = 0$ and $I_n(B) = 1/n$ is a decreasing sequence of models of $\mathcal{P}$ converging to the interpretation $I(A) = 0, I(B) = 0$, which, however, is not a model of $\mathcal{P}$. (Note: f is not $\bigwedge$-preserving.) Also note that $\mathcal{P}$ has a minimal model with $I(A) = 1$ and $I(B) = 0$, despite the fact that the connector function f is not $\bigwedge$-preserving.

We next want to establish a proposition like Proposition 3.9, guaranteeing the existence of minimal fixed points.

PROPOSITION 4.14. *If all connector functions in $\mathcal{P}$ are $\bigwedge$-preserving and $\mathcal{P}$ has models, then $\Phi(T_\mathcal{P})$ has minimals.*

Proof. As $\mathcal{P}$ has models, models are fixed-points of $T_\mathcal{P}$ (Proposition 4.6), and $T_\mathcal{P}$ is inflationary, by Proposition 3.7, $\Phi(T_\mathcal{P}) \neq \varnothing$. So, let $(I_\alpha)_{\alpha\in I}$ be a decreasing sequence of interpretations in $\Phi(T_\mathcal{P})$, and let $I = \bigwedge_\alpha I_\alpha$. Again, by Zorn's lemma it suffices to show that $I \in \Phi(T_\mathcal{P})$.

By Propositions 4.11 and 3.7, $I_\alpha \in T_\mathcal{P}(I_\alpha)$; i.e., I_α are fixed-points. Now, let us show that $I \in T_\mathcal{P}(I)$. From $I_\alpha \in T_\mathcal{P}(I_\alpha)$ and $\psi \leftarrow \varphi \in \mathcal{P}^*$, $I_\alpha(\varphi) \leq I_\alpha(\psi)$ holds. Therefore, by Proposition 4.2, $I(\varphi) = (\bigwedge_\alpha I_\alpha)(\varphi) = \bigwedge_\alpha I_\alpha(\varphi) \leq \bigwedge_\alpha I_\alpha(\psi) = (\bigwedge_\alpha I_\alpha)(\psi) = I(\psi)$. As a consequence, $I \models \mathcal{P}[I]$ and, thus, $I \in T_\mathcal{P}(I)$. Therefore, $I \in \Phi(T_\mathcal{P})$, which concludes the proof. □

We note that, by Proposition 3.19, if $T_\mathcal{P}(I_\top) \neq \varnothing$, then, as $T_\mathcal{P}$ is inflationary, $\mathcal{P}$ has a model. Then, by Propositions 3.8 and 4.6, the next proposition directly follows.

PROPOSITION 4.15. *If all connector functions in $\mathcal{P}$ are $\bigwedge$-preserving and $\mathcal{P}$ has models, then $T_\mathcal{P}$ has minimal fixed-points and, thus, $\mathcal{P}$ has minimal models.*

The analogue of Proposition 3.25 is as follows.

PROPOSITION 4.16. *If $\mathcal{P}$ has models and all connector functions in $\mathcal{P}$ are $\bigwedge$-preserving, then $\mathcal{P}$ has minimal models and there are orbits converging to them. If all connector functions in $\mathcal{P}$ are also $\bigvee$-preserving, then ω steps are sufficient to reach a minimal model.*

4.2. The case of classical logic programs. We conclude this part by applying our results to classical logic programs [26, 27, 32]. As already pointed out, any classical first order clause $A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \neg B_n$ (with $k + n > 0$) is a rule of the form

$$A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_n \ . \tag{4.2}$$

If $k = 0$, we use $\bot$ in the left-hand side, while if $n = 0$, we use $\top$ in the right-hand side. The truth space is $L = \{0, 1\}$. Note that usually in disjunctive logic programs $k \geq 1$ is assumed and A_i, B_j is neither $\top$ nor $\bot$. This slight difference has an impact on the set of models of a disjunctive logic program, as we show next.

Example 30. Consider the truth space $L = \{0, 1\}$ and consider $\mathcal{P}$ with rules

$$\bot \leftarrow A,$$
$$A \leftarrow \top\ .$$

The former rule states that A should be false, while the latter states that A should be true. Of course, $T_{\mathcal{P}}(I) = \emptyset$, for any interpretation I and, thus, $T_{\mathcal{P}}$ has no fixed-point; thus, $\mathcal{P}$ has no model.

On the other hand, if we assume that $k \geq 1$ and that A_i, B_j is neither $\top$ nor $\bot$, as usual for disjunctive logic programs, as L is finite, by Proposition 4.4, $\vee$ and $\wedge$ are limit-preserving. Furthermore, it is easily verified that for any I, $I_\top \in T_{\mathcal{P}}(I) \neq \varnothing$, in particular $T_{\mathcal{P}}(I_\top) = \{I_\top\}$, $T_{\mathcal{P}}$ is $\bigvee$-preserving (thus, S-monotone), and, as $T_{\mathcal{P}}$ inflationary, $\mathcal{P}$ has a model. By Propositions 4.16 and 3.23 we immediately have the following well-known fact [27, 32].

PROPOSITION 4.17. *Any classical disjunctive logic program $\mathcal{P}$ has minimal models and there are orbits (of length ω) of minimals converging to them.*

Finally, let us further restrict logic programs to the case where the head contains one atom only (i.e., $k = 1$). That is, rules are of the usual deterministic form

$$A \leftarrow B_1 \wedge \cdots \wedge B_n\ . \tag{4.3}$$

Then, for any I, $T_{\mathcal{P}}(I)$ has a least element.

PROPOSITION 4.18. *For any classical deterministic logic program $\mathcal{P}$ and interpretation I, $T_{\mathcal{P}}(I)$ has a least element.*

Proof. Consider $\bar{J} = \bigwedge T_{\mathcal{P}}(I)$. Let us show that $\bar{J} \in T_{\mathcal{P}}(I)$. As for all $J \in T_{\mathcal{P}}(I)$ we have $I \leq J$, it follows that $I \leq \bigwedge_{J \in T_{\mathcal{P}}(I)} J = \bar{J}$. Now, consider $A \leftarrow I(\varphi)$ with $A \leftarrow \varphi \in \mathcal{P}^*$. Then by Proposition 4.2, as for all $J \in T_{\mathcal{P}}(I)$, $I(\varphi) \leq J(A)$ holds,

$$I(\varphi) \ \leq \ \bigwedge_{J \in T_{\mathcal{P}}(I)} J(A) \ = \ \bigwedge_{J \in T_{\mathcal{P}}(I)} e(J, A) \ = \ e\left(\bigwedge_{J \in T_{\mathcal{P}}(I)}, A\right) \ = \ e(\bar{J}, A) \ = \ \bar{J}(A),$$

and, thus, $\bar{J} \models \mathcal{P}[I]$. As a consequence, $\bar{J} \in T_{\mathcal{P}}(I)$. □

Now, using Propositions 3.10, 3.24, and 4.17 we immediately have the following well-known fact [26].

PROPOSITION 4.19. *Any classical deterministic logic program $\mathcal{P}$ has a least model and there is an orbit (of length ω) of least elements converging to it.*

If terms are restricted to be either variables or constants, then for disjunctive logic programs the set of minimal models is finite (as there are finitely many interpretations). For both Propositions 4.17 and 4.19 the length of the orbits is finite.

5. Conclusions and related work. We have provided conditions for the existence of fixed-points, and minimal and maximal fixed-points of multivalued functions over complete lattices, and have shown how to obtain them. Our main contribution establishes that an inflationary, S-monotone, multivalued function with $\Phi(f) \neq \varnothing$ has minimal fixed-points, where each orbit converges to a fixed-point and for each minimal fixed-point an orbit converging to it exists. We have also shown that (see Table 3.1) the set of fixed-points of a limit-preserving multivalued function is a complete multilattice. We also reported the results of related work we are aware of.

We then applied our results to a general form of logic programs, where the truth space is a complete lattice. We have shown that a multivalued operator can be

defined whose fixed-points are in one-to-one correspondence with the models of the logic program.

Related work. To the best of our knowledge, the fixed-point theory over complete lattices is mainly single-value oriented. Nonetheless, [6, 14, 15, 16, 20, 21, 22, 33, 45, 53] establish a version of the Knaster–Tarski theorem, though requiring the condition that $f(x)$ be always nonempty and some other conditions. References [20, 21, 22, 14, 15, 16, 17, 38] also investigate the case where metric spaces or Banach spaces are considered in place of complete lattices, and then use the well-known contraction principle (see also [24, 41]) or continuity to guarantee the existence of a fixed-point (if $f(x)$ is always nonempty, of course). They also then apply some of their results to disjunctive logic programs (with nonmonotone negation). Close in spirit, using mainly Banach spaces, topological spaces, and metric spaces in place of complete lattices, are works of the mathematical community such as [2, 10, 19, 23, 49, 40, 34, 35, 43, 50, 51]. We point out that these works do not cover our results. As our initial objective was to study generalized many-valued logic programs, our analysis tried to parallel the usual analyses made for single-valued functions over complete lattices.

The research area of semantics for nondeterministic programming languages (see, e.g., [8, 36, 37, 44]) instead does not address multivalued functions directly, but rather "lifts" a multivalued function $f : D \to 2^D$ to a function $g : \mathcal{P}^*(D) \to \mathcal{P}^*(D)$, where $\mathcal{P}^*(D)$ is a rather complicated and appropriately ordered subset of the powerset of D (so-called *power domains* [1, 36, 44]), and then applies usual fixed-point theory. Here, D is a so-called *domain*, i.e., a complete partial ordered set with some additional constraints [1]. As in all order cases, $f(x)$ is assumed to be nonempty and finite. This constraint is related to the application of nondeterministic programming languages (as indeed, at each step of a program execution, there is at least one next state and there are at most finitely many possible nondeterministic alternatives).

Concerning the application of multivalued functions to logic programming, to the best of our knowledge, no work considers such general rules. Related to our approach are [14, 15, 16, 20, 21, 22] in which classical disjunctive logic programs have been considered with nonmonotone negation. We did not consider nonmonotonic negation so far, as an appropriate semantics (for generalized nonmonotone many-valued logic programs) has still to be developed. We also point to works such as [13, 39, 52] in which disjunctive logic programs are studied from a domain-theoretic (i.e., Smyth powerdomain) point of view. One feature of these works is that, by using an appropriate domain, as in the case of nondeterminsitic programming languages, the concept of a multivalued function is avoided by representing "disjunctive states"[9] (again, the image of a multivalued function is assumed to be nonempty and finite). On the other hand, we follow a direct approach, which requires less formal and abstract theory and is likely amenable to a less formal audience as well.

We envisage several directions for future research. The fixed-point theory of multivalued functions is interesting per se (there are many options worth investigating, such as using some other sets in place of complete lattices, CPOs, domains, Banach spaces, metric spaces, topological spaces, or some specific sets such as $[0, 1]$, etc., which have mainly been considered by mathematicians—see also [12]). On the other hand, related to general logic programs, besides considering special cases for connectors in the head and body, it would be interesting to generalize the stable model semantics

[9] This is similar to [42] in which an immediate consequence operator has been defined over sets of interpretations.

for classical disjunctive logic programs [9] to our case. More generally, we would like to bring the theory of fixed-points of multivalued functions to the attention of the knowledge representation and reasoning community, where multivalued functions may be applied to several problems and logic-based languages for knowledge representation.

Appendix A. Some other proofs.

PROPOSITION A.1. *Consider a multivalued function $f\colon L \to 2^L$. If f is $\bigwedge$-preserving, then f is H-monotone.*

Proof. Consider $x_1 \leq x_2$. Then for the decreasing sequence $x_2 \geq x_1$, $f(x_1) = f(x_2 \wedge x_1) = \{y\colon$ there are $y_i \in f(x_i)$ s.t. $y = y_2 \wedge y_1\} = X$. If $f(x_1) = \varnothing$, then trivially $\varnothing = f(x_1) \preceq_H f(x_2)$. If $f(x_2) = \varnothing$, then by definition $X = \varnothing$ and, thus, $f(x_1) = \varnothing$. Therefore, $\varnothing = f(x_1) \preceq_H f(x_2) = \varnothing$. Otherwise assume $f(x_1)$ and $f(x_2)$ are nonempty. Therefore, as f is $\bigwedge$-preserving, for $y \in f(x_1) = X$ there are $y_i \in f(x_i)$ $(i = 1, 2)$ such that $y = y_2 \wedge y_1$. In particular, $y \leq y_2$. Therefore, $f(x_1) \preceq_H f(x_2)$ and, thus, f is H-monotone. □

PROPOSITION A.2. *Consider a multivalued function $f\colon L \to 2^L$ and $x_1 \leq x_2$ with $f(x_1) \neq \varnothing \neq f(x_2)$. If f is $\bigvee$-preserving, then $f(x_1) \preceq_H f(x_2)$.*

Proof. For the increasing sequence $x_1 \leq x_2$, as f is $\bigvee$-preserving, $f(x_2) = f(x_1 \vee x_2) = \{y\colon$ there are $y_i \in f(x_i)$ s.t. $y = y_2 \vee y_1\} = X$. Now, for $y \in f(x_1)$ choose a $y' \in f(x_2) \neq \varnothing$ and consider $y'' = y \vee y'$. Therefore, $y'' \in X = f(x_2)$, $y \leq y''$, and, thus, $f(x_1) \preceq_H f(x_2)$. □

PROPOSITION A.3. *Let $f\colon L \to 2^L$ be a multivalued function. If f is deflationary, then $x \in \Psi(f)$ iff x is a fixed-point of f.*

Proof. Let $x \in \Psi(f)$. As f is deflationary, $\{x\} \preceq_H f(x) \preceq_H \{x\}$ and, thus, for $x \in \{x\}$ there is $y \in f(x)$ such that $x \leq y \leq x$, i.e., $x = y \in f(x)$. Vice versa, if $x \in f(x)$, then $\{x\} \preceq_H f(x)$ and, thus, $x \in \Psi(f)$. □

PROPOSITION A.4. *Let $f\colon L \to 2^L$ be a multivalued function. If f is an H-monotone or deflationary multivalued function, and $\Psi(f)$ has maximals, then all $y \in \max \Psi(f)$ are maximal fixed-points of f. In particular, if $x = \bigvee \Psi(f) \in \Psi(f)$, then x is the greatest fixed-point of f.*

Proof. As $\Psi(f)$ has maximals, $\max \Psi(f) \neq \varnothing$. So, let $y \in \max \Psi(f)$. Therefore, $\{y\} \preceq_H f(y) \neq \varnothing$ and, thus, there is $y' \in f(y)$ such that $y \leq y'$. If f is H-monotone, then $f(y) \preceq_H f(y')$ and, thus, for $y' \in f(y)$ there is $y'' \in f(y')$ such that $y' \leq y''$. Therefore, $\{y'\} \preceq_H f(y')$ and, thus, $y' \in \Psi(f)$. But $y \in \max \Psi(f)$, so it cannot be $y < y'$. Therefore, $y = y' \in f(y)$; i.e., y is a fixed-point of f. If f is deflationary, by Proposition 3.7, y is a fixed-point of f. Now, assume $x \in f(x)$. Therefore, $\{x\} \preceq_H f(x)$ and, thus, $x \in \Psi(f)$. But $y \in \max \Psi(f)$, so it cannot be $y < x$, and, thus, y is a maximal fixed-point of f. Finally, consider $x = \bigvee \Psi(f)$. By hypothesis, $x \in \Psi(f)$ and x is the greatest element of $\Psi(f)$. Hence, we know that $x \in f(x)$. Let $y \in f(y)$. Hence $y \in \Psi(f)$, and, thus, $y \leq x$. As a consequence, x is the greatest fixed-point of f. □

PROPOSITION A.5. *Let $f\colon L \to 2^L$ be a multivalued function. If f is a $\bigvee$-preserving multivalued function with $\Psi(f) \neq \varnothing$, then $\Psi(f)$ has maximals and, thus, maximal fixed-points.*

Proof. By hypothesis, $\Psi(f) \neq \varnothing$. Let $(x_\alpha)_{\alpha \in I}$ be an increasing sequence of $x_\alpha \in \Psi(f)$, and let $\bar{x} = \bigvee_\alpha x_\alpha$. As f is $\bigvee$-preserving, by definition, $f(\bar{x}) = \{y\colon$ there is $(y_\alpha)_{\alpha \in I}$ s.t. $y_\alpha \in f(x_\alpha)$ and $y = \bigvee_\alpha y_\alpha\}$.

Now, for any α, $x_\alpha \leq x_{\alpha+1}$, by Proposition 3.6 and, as $x_\alpha \in \Psi(f)$, $\{x_\alpha\} \preceq_H f(x_\alpha) \preceq_H f(x_{\alpha+1})$. Therefore, for any x_α there are $y_\alpha \in f(x_\alpha)$ and $y_{\alpha+1} \in f(x_{\alpha+1})$ such that $x_\alpha \leq y_\alpha \leq y_{\alpha+1}$.

Note that if α is a limit ordinal, then, as $x_\beta \leq x_\alpha$ for all $\beta < \alpha$, it follows that $\{x_\beta\} \preceq_H f(x_\beta) \preceq_H f(x_\alpha)$ and, thus, $x_\beta \leq y_\beta \leq y_\alpha$ for all $\beta < \alpha$. Therefore, there is an increasing sequence $(y_\alpha)_{\alpha \in I}$ of elements $y_\alpha \in f(x_\alpha)$ such that $\bar{x} = \bigvee_\alpha x_\alpha \leq \bigvee_\alpha y_\alpha = \bar{y}$. By definition of $f(\bar{x})$, $\bar{y} \in f(\bar{x})$, and, thus, $\{\bar{x}\} \preceq_H f(\bar{x})$. Therefore $\bar{x} \in \Psi(f)$, and, thus, every increasing sequence has an upper bound in $\Psi(f)$. So, by Zorn's lemma, $\Psi(f)$ has maximals, which by Proposition 3.8 are also maximal fixed-points. □

PROPOSITION A.6. *Let $f\colon L \to 2^L$ be a multivalued function. If f is an H-monotone, multivalued function, and for all $x \in L$, $f(x)$ has the greatest element, then f has the greatest fixed-point.*

Proof. As for all $x \in L$, $f(x)$ has the greatest element, by definition, $\bigvee f(x) \in f(x) \neq \varnothing$. Therefore, $\Psi(f) \neq \varnothing$ as $\{\bot\} \preceq_H f(\bot)$. Consider $a = \bigvee_{c \in \Psi(f)} c$. If $a \in \Psi(f)$, then by Proposition 3.8, a is the greatest fixed-point of f. So, let us show that $a \in \Psi(f)$. For $c \in \Psi(f)$ there is an $x_c \in f(c)$ such that $c \leq x_c$. As $c \leq a$ and f is H-monotone, $f(c) \preceq_H f(a)$, and, thus, for $x_c \in f(c)$ there is $y_c \in f(a)$ such that $c \leq x_c \leq y_c$. Since $f(a)$ has the greatest element, there is $y \in f(a)$ such that $a = \bigwedge_{c \in \Psi(f)} c \leq \bigwedge_{c \in \Psi(f)} x_c \leq \bigwedge_{c \in \Psi(f)} y_c \leq y$. Hence, $\{a\} \preceq_H f(a)$, i.e., $a \in \Psi(f)$. □

PROPOSITION A.7. *Let $f\colon L \to 2^L$ be an H-monotone, nonempty, and $\vee$-closed multivalued function. Then*

1. *$\Psi(f)$ is $\vee$-closed;*
2. *f has a greatest fixed-point.*

Proof. Note that $\Psi(f) \neq \varnothing$ as $\{\bot\} \preceq_H f(\top) \neq \varnothing$.

1. Consider a subset S of $\Psi(f)$ and $a = \bigvee S$. Let us show that $a \in \Psi(f)$. We know that for each $c \in S$, $\{c\} \preceq_H f(c)$ holds; i.e., there is $x_c \in f(c)$ such that $c \leq x_c$. But, f is H-monotone, and, thus, from $c \leq a$, $\{c\} \preceq_H f(c) \preceq_H f(a)$ follows. That is, there is $y_c \in f(a)$ such that $c \leq x_c \leq y_c$. Let $y = \bigvee_{c \in S} y_c$. As f is $\vee$-closed, $y \in f(a)$ follows. Therefore, $a = \bigvee_{c \in S} c \leq \bigvee_{c \in S} y_c = y$, $\{a\} \preceq_H f(a)$, and, thus, $a \in \Psi(f)$. Therefore, $\Psi(f)$ is $\vee$-closed.

2. From point 1, $\Psi(f)$ has the greatest element a, and, thus, by Proposition 3.8, f has a as the greatest fixed-point. □

PROPOSITION A.8. *For a multivalued function f,*

1. *if f is deflationary, then each $\top$-orbit is decreasing;*
2. *each decreasing $\top$-orbit converges to a fixed-point of f (if no fixed-point exists, then there is no orbit);*
3. *if f is H-monotone and deflationary, then for any maximal fixed-point of f there is a $\top$-orbit converging to it.*

Proof. Let $(x_\alpha)_{\alpha \in I}$ be an orbit of f. Recall that for ordinal α, $x_{\alpha+1} \in f(x_\alpha) \neq \varnothing$. As f is deflationary, $f(x_\alpha) \preceq_H \{x_\alpha\}$. But, by definition of $\preceq_H$, for $x_{\alpha+1} \in f(x_\alpha)$, $x_{\alpha+1} \leq x_\alpha$. For a limit ordinal λ, $x_\lambda = \bigvee_{\alpha<\lambda} x_\alpha$, $\varnothing \neq f(x_\lambda) \preceq_H \{x_\lambda\}$, and, thus, there is $x_{\lambda+1} \in f(x_\lambda)$ such that $x_{\lambda+1} \leq x_\lambda$.

For the second point, as $(x_\alpha)_{\alpha \in I}$ is a decreasing sequence and $|I| > |L|$, by Proposition 2.1 there is an ordinal α such that $x_\alpha = x_{\alpha+1} \in f(x_\alpha)$. That is, x_α is a fixed-point of f.

Finally, for the third point, assume $\bar{x} \in f(\bar{x})$ is a maximal fixed-point of f. Now, let us show by (transfinite) induction on α that there is a decreasing orbit $(x_\alpha)_{\alpha \in I}$ of f s.t. $\bar{x} \leq x_\alpha$ for all α.

The case when $\alpha = 0$. $\bar{x} \leq \top = x_0$.

α successor ordinal. By induction, $\bar{x} \leq x_\alpha$. As f is H-monotone and deflationary,

$f(\bar{x}) \preceq_H f(x_\alpha) \preceq_H \{x_\alpha\}$. But, $\bar{x} \in f(\bar{x})$, so we can choose $x_{\alpha+1} \in f(x_\alpha)$ s.t. $\bar{x} \leq x_{\alpha+1} \leq x_\alpha$.

α limit ordinal. By induction, $\bar{x} \leq x_\beta$ holds for all $\beta < \alpha$, which implies that $\bar{x} \leq \bigvee_{\beta<\alpha} x_\beta = x_\alpha$.

The sequence $(x_\alpha)_{\alpha\in I}$ is decreasing, and, thus, by Proposition 2.1 there is an ordinal α such that $x_\alpha = x_{\alpha+1} \in f(x_\alpha)$. So, x_α is a fixed-point of f with $\bar{x} \leq x_\alpha$. As $\bar{x}$ is maximal, $x_\alpha = \bar{x}$. □

PROPOSITION A.9. *For $f\colon L \to 2^L$, $h(x) = x \otimes f(x)$ is deflationary. Furthermore, if f is H-monotone, then*

1. *h is H-monotone;*
2. *$x \in f(x)$ implies $x \in h(x)$;*
3. *$x \in h(x)$ implies $\{x\} \preceq_H f(x)$;*
4. *if x is a maximal fixed point of h, then x is a maximal fixed point of f;*
5. *if x is a maximal fixed point of f, and f is also deflationary, then x is a maximal fixed point of h.*

Proof. Consider f and h. If $f(x) = \varnothing$, then $\varnothing = h(x) \preceq_H \{x\}$. Otherwise, for $y \in h(x)$, $y \leq x$. Therefore, $h(x) \preceq_H \{x\}$, and, thus, h is deflationary. Now, suppose f is H-monotone.

1. This is easy. h is a combination of H-monotone functions.

2. If $x \in f(x)$, then by definition of h, $x = x \wedge x \in h(x)$.

3. If $x \in h(x)$, then for some $y \in f(x)$, $x = x \wedge y$. Therefore, $x \leq y$ and, thus, $\{x\} \preceq_H f(x)$.

4. Assume x is a maximal fixed-point of h, i.e., $x \in h(x) = x \otimes f(x)$. Therefore, there is $y \in f(x)$ such that $x \leq y$. As f is H-monotone, $f(x) \preceq_H f(y)$. That is, there is $z \in f(y)$ such that $y \leq z$ and, thus, $y = y \wedge z$. Therefore, $y \in h(y)$. As x is maximal and $x \leq y$, $y = x$ follows and, thus, $x \in f(x)$. To prove that x is a maximal fixed-point of f, assume there is $x \leq y$ such that $y \in f(y)$. By point 2, $y \in h(y)$, and, thus, as x is a maximal fixed-point of h, $y = x$ follows.

5. Assume x is a maximal fixed-point of f. By point 2 $x \in h(x)$. To prove that x is a maximal fixed-point of h, assume there is $x \leq y$ such that $y \in h(y)$. Then by point 3 $\{y\} \preceq_H f(y)$ and, thus, $y \in \Psi(f)$. By Proposition 3.7, $y \in f(y)$, and, thus, as x is a maximal fixed-point of f, $y = x$ follows. □

Disclaimer. The authors of this work apologize both to the authors and to the readers for all the relevant works and results which are not cited here that we are unaware of.

REFERENCES

[1] S. ABRAMSKY AND A. JUNG, *Domain theory*, in Handbook of Logic in Computer Science, Vol. 3, S. Abramsky, D. Gabbay, and T. S. E. Maibaum, eds., Oxford University Press, Oxford, UK, 1994, pp. 1–168.

[2] C. AVRAMESCU, *A fixed-point theorem for multivalued mappings*, Electron. J. Qual. Theory Differ. Equ., 10 (2004), pp. 1–10.

[3] N. D. BELNAP, *A useful four-valued logic*, in Modern Uses of Multiple-Valued Logic, G. Epstein and J. Michael Dunn, eds., D. Reidel, Dordrecht, The Netherlands, 1977, pp. 5–37.

[4] M. BENADO, *Les ensembles partiellement ordonnés et le théorème de raffinement de Schreier,* II. *Théorie des multistructures*, Czech. Math. J., 5 (1955), pp. 308–344.

[5] G. BIRKHOFF, *Lattice Theory*, American Mathematical Society, Providence, RI, 1973.

[6] F. ECHENIQUE, *A short and constructive proof of Tarski's fixed-point theorem*, Internat. J. Game Theory, 33 (2005), pp. 215–218.

[7] H. EGLI, *A Mathematical Model for Nondeterministic Computations*, Technological University, Zürich, Zürich, Switzerland, 1975.

[8] N. Francez, C. A. R. Hoare, D. J. Lehmann, and W. P. de Roever, *Semantics of nondeterminism, concurrency, and communication*, J. Comput. System Sci., 19 (1979), pp. 290–308.
[9] M. Gelfond and V. Lifschitz, *Classical negation in logic programs and disjunctive databases*, New Gener. Computing, 9 (1991), pp. 365–386.
[10] L. Górniewicz, *Topological Fixed Point Theory of Multivalued Mappings*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.
[11] C. A. Gunter and D. S. Scott, *Semantic domains*, in Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier, Amsterdam, 1990, pp. 633–674.
[12] J. Hillas, *Fixed Points Theorems*, http://yoda.eco.auckland.ac.nz/~jhillas/616.781/fpb_01.pdf (2001).
[13] P. Hitzler, *Default reasoning over domains and concept hierarchies*, in Proceedings of the 27th German Conference on Artificial Intelligence (KI-04), Lecture Notes in Artificial Intelligence 3238, Springer, Berlin, 2004, pp. 351–365.
[14] P. Hitzler and A. K. Seda, *The fixed-point theorems of priess-crampe and ribenboim in logic programming*, in Proceedings of the 1999 Valuation Theory Conference, Fields Institute Communications Series 23, American Mathematical Society, Providence, RI, 1999, pp. 219–235.
[15] P. Hitzler and A. K. Seda, *Multivalued mappings, fixed-point theorems and disjunctive databases*, in Proceedings of the 3rd Irish Workshop on Formal Methods, Galway, Eire, 1999.
[16] P. Hitzler and A. K. Seda, *Some issues concerning fixed points in computational logic: Quasi-metrics, multivalued mappings, and the Knaster-Tarski theorem*, Topology Proc., 24 (1999), pp. 223–250.
[17] P. Hitzler and A. K. Seda, *Generalized metrics and uniquely determined logic programs*, Theoret. Comput. Sci., 305 (2003), pp. 187–219.
[18] C. A. R. Hoare, *Communicating sequential processes*, Comm. ACM, 21 (1978), pp. 666–677.
[19] S. Kakutani, *A generalization of Brouwer's fixed-point theorem*, Duke Math. J., 8 (1941), pp. 457–459.
[20] M. A. Khamsi, V. Kreinovich, and D. Misane, *A new method of proving the existence of answer sets for disjunctive logic programs: A metric fixed point theorem for multivalued maps*, in Proceedings of the Workshop on Logic Programming with Incomplete Information, Vol. 32, Vancouver, BC, Canada, 1993, pp. 58–73.
[21] M. A. Khamsi and D. Misane, *Disjunctive signed logic programs*, Fund. Inform., 32 (1996), pp. 349–357.
[22] M. A. Khamsi and D. Misane, *Fixed point theorems in logic programming*, Ann. Math. Artificial Intell., 21 (1997), pp. 231–243.
[23] I.-S. Kim, *Fixed points of condensing multivalued maps in topological vector spaces*, Fixed Point Theory Appl., 2 (2004), pp. 107–112.
[24] A. Kirk and B. Sims, *Handbook of Metric Fixed Point Theory*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.
[25] P. Knijnenburg, *Algebraic Domains, Chain Completion and the Plotkin Powerdomain Construction*, Technical report RUU-CS-93-03, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 1993.
[26] J. W. Lloyd, *Foundations of Logic Programming*, Springer, Heidelberg, 1987.
[27] J. Lobo, J. Minker, and A. Rajasekar, *Foundations of Disjunctive Logic Programs*, MIT Press Series in Logic Programming, Ellis Horwood, New York, 1992.
[28] Y. Loyer and U. Straccia, *Epistemic foundation of stable model semantics*, J. Theory Practice Logic Program., 6 (2006), pp. 355–393.
[29] J. Martínez, G. Gutiérrez, I. P. de Guzmán, and P. Cordero, *Generalizations of lattices via non-deterministic operators*, Discrete Math., 295 (2005), pp. 107–141.
[30] J. Medina, M. Ojeda-Aciego, and J. Ruiz-Calviño, *Fuzzy logic programming via multilattices*, Fuzzy Sets and Systems, 158 (2007), pp. 674–688.
[31] R. Milner, *An approach to the semantics of parallel programs*, in Proceedings of the Convegno di Informatica Theorica, Istituto di Elaborazione dell'Informazione, Pisa, 1973.
[32] J. Minker and A. Rajasekar, *A fixpoint semantics for disjunctive logic programs*, J. Logic Program., 9 (1990), pp. 45–74.
[33] V. Orey, *Fixed points for correspondences with values in a partially ordered set and extended supermodular games*, J. Math. Econom., 25 (1996), pp. 345–354.
[34] S. Park, *New versions of the Fan–Browder fixed point theorem and existence of economic equilibria*, Fixed Point Theory Appl., 2 (2004), pp. 149–158.

[35] A. Petrusel, *Fixed points for multifunctions on generalized metric spaces with applications to a multivalued Cauchy problem*, Comment. Math. Univ. Carolinae, 38 (1997), pp. 657–663.
[36] G. D. Plotkin, *A powerdomain construction*, SIAM J. Comput., 5 (1976), pp. 452–487.
[37] G. D. Plotkin, *Pisa Notes* (*On Domain Theory*), Department of Computer Science, University of Edinburgh, Edinburgh, 1983. Available online at http://homepages.inf.ed.ac.uk/gdp/publications/Domains.ps
[38] S. Priess-Crampe and P. Ribenboim, *Ultrametric spaces and logic programming*, J. Logic Program., 42 (2000), pp. 59–70.
[39] W. C. Rounds and G.-Q. Zhang, *Clausal logic and logic programming in algebraic domains*, Inform. Comput., 171 (2001), pp. 183–200.
[40] A. K. Seda, *Topology and the semantics of logic programs*, Fund. Inform., 24 (1995), pp. 359–386.
[41] A. K. Seda, *Quasi-metrics and the semantics of logic programs*, Fund. Inform., 29 (1997), pp. 97–117.
[42] D. Seipel, J. Minker, and C. Ruiz, *Model generation and state generation for disjunctive logic programs*, J. Logic Program., 32 (1997), pp. 49–69.
[43] N. Shahzad and A. Lone, *Fixed points of multimaps which are not necessarily nonexpansive. Theorem and existence of economic equilibria*, Fixed Point Theory Appl., 2 (2005), pp. 169–176.
[44] M. B. Smyth, *Power domains*, J. Comput. System Sci., 16 (1978), pp. 23–36.
[45] A. Stouti, *A generalized Amman's fixed point theorem and its application to Nash equilibrium*, Acta Math. Acad. Paedagog. Nyházi. (N.S.), 21 (2005), pp. 107–112.
[46] U. Straccia, *Managing uncertainty and vagueness in description logics, logic programs and description logic programs*, in Reasoning Web, 4th International Summer School, Tutorial Lectures, Lecture Notes in Comput. Sci. 5224, Springer-Verlag, Berlin, New York, 2008, pp. 54–103.
[47] A. Tarski, *A lattice-theoretical fixpoint theorem and its applications*, Pacific J. Math., 5 (1955), pp. 285–309.
[48] P. Vojtáš, *Fuzzy logic programming*, Fuzzy Sets and Systems, 124 (2001), pp. 361–370.
[49] L. E. Ward, Jr., *A fixed-point theorem for multi-valued functions*, Pacific J. Math., 8 (1958), pp. 921–927.
[50] X. Wu, *A new fixed point theorem and its applications*, Proc. Amer. Math. Soc., 125 (1997), pp. 1779–1788.
[51] Z. Wu, *A note on fixed point theorems for semi-continuous correspondences on* [0,1], Proc. Amer. Math Soc., 126 (1998), pp. 3061–3064.
[52] G.-Q. Zhang and W. C. Rounds, *Semantics of logic programs and representation of Smyth powerdomain*, in Domains and Processes: Proceedings of the 1st International Symposium on Domain Theory, K. Keimel et al., eds., Kluwer, Dordrecht, 1999, pp. 151–181.
[53] L. Zhou, *The set of Nash equilibria of a supermodular game is a complete lattice*, Games Econom. Behav., 7 (1994), pp. 295–300.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1912–1951

IMPOSSIBILITY RESULTS AND LOWER BOUNDS FOR CONSENSUS UNDER LINK FAILURES*

ULRICH SCHMID†, BETTINA WEISS†, AND IDIT KEIDAR‡

Abstract. We provide a suite of impossibility results and lower bounds for the required number of processes and rounds for synchronous consensus under transient link failures. Our results show that consensus can be solved even in the presence of $O(n^2)$ moving omission and/or arbitrary link failures per round, provided that both the number of affected outgoing and incoming links of every process is bounded. Providing a step further toward the weakest conditions under which consensus is solvable, our findings are applicable to a variety of dynamic phenomena such as transient communication failures and end-to-end delay variations. We also prove that our model surpasses alternative link failure modeling approaches in terms of assumption coverage.

Key words. fault-tolerant distributed algorithms, consensus, transient link failures, impossibility results, lower bounds, assumption coverage analysis

AMS subject classifications. 68Q25, 68Q85, 68W15, 68W40

DOI. 10.1137/S009753970443999X

1. Introduction.

Most research on fault-tolerant distributed algorithms conducted in the past rests on process failure models. Every failure occurring in a system is attributed to either the sending or the receiving process here, irrespectively of whether the actual error occurs at this process or rather on the intermediate communication path. Moreover, a process that commits a single failure is often "statically" considered faulty during the whole execution, even if its failure is transient.

Although such process failure models adequately capture many important scenarios, including crash failures, where a faulty process just stops operating, and Byzantine failures, where a faulty process can do anything, they are not particularly suitable for modeling more dynamic phenomena. In particular, given the steadily increasing dominance of communication over computation in modern distributed systems, in conjunction with the high reliability of modern processors and robust operating system designs, transient communication failures such as lost or unrecognized packets (synchronization errors), CRC errors (data corruption), and receiver overruns (packet buffer overflow) are increasingly dominating real-world failures. Another dynamic phenomenon that is encountered frequently in practice is unpredictable variations of the end-to-end delays in multihop networks such as the Internet, which are caused, for example, by temporary network congestion and intermediate router failures. Since excessive end-to-end delays appear as omissions in classic (semi)synchronous systems and other time(out)-based approaches, for example, [3, 4, 5, 53, 7, 51, 43, 39], such timing variations can also be considered as transient link failures.

The distinguishing properties of such failures are (a) that they affect the path (termed the *link* in what follows) connecting two processes, rather than the endpoints

*Received by the editors January 19, 2004; accepted for publication (in revised form) September 9, 2008; published electronically January 9, 2009.
http://www.siam.org/journals/sicomp/38-5/43999.html

†Embedded Computing Systems Group (E182/2), Technische Universität Wien, Treitlstraße 1-3, A-1040 Vienna, Austria (s@ecs.tuwien.ac.at, bw@ecs.tuwien.ac.at). The research of these authors was supported by the Austrian START programme Y41-MAT and the FWF project P18264 (SPAWN).

‡Department of Electrical Engineering, Technion Haifa, Technion City, Haifa, Israel (idish@ee.technion.ac.il).

(the processes), and (b) that they are *mobile* [58], as different links may fail at different times. Hence, the ability to communicate (in a timely manner) with other processes in the system cannot be statically attributed to a process here: If the link failure rate is sufficiently high, there will never be a nonempty set of processes that appear nonfaulty to each other, i.e., never send (timing) faulty messages to each other. As a consequence, classic process failure models are inappropriate for such applications.

This paper focuses on impossibility results and lower bounds for synchronous deterministic consensus in the presence of such mobile link failures. In the consensus problem (also called the "Byzantine agreement" problem [47]), processes have to agree on a common output value, despite failures, based on some input values distributed among the processes. Consensus is widely recognized as one of the most fundamental problems in fault-tolerant distributed computing. Synchronous consensus algorithms execute in a sequence of lock-step rounds $k = 1, 2, \ldots$, which consist of sending, receiving, and processing round k messages exchanged among all the processes.

Unfortunately, there is a discouraging impossibility result for deterministic synchronous consensus in the presence of general link failures (see Theorem 1 in section 3), which goes back to Gray's 1978 paper [33] on atomic commitment in distributed databases. This result was strengthened by Santoro and Widmayer [58], who introduced the *mobile omissive process failure model*: In each round, some process can be send-omissive (or omissive for short) in the sense that it can experience any number of transmission failures on its outgoing links. Even a single mobile omissive process failure was shown to render consensus unsolvable [58].

Despite those negative results, however, there are synchronous consensus algorithms for restricted link failure patterns. For example, it has been known for a long time that consensus can be solved when sufficient connectivity is always maintained [20, 37, 46].

More recently, Schmid, Weiss, and Rushby introduced a hybrid failure model for synchronous systems [66] which—in addition to classic process failures—admits up to $O(n^2)$ moving link failures per round. The link failure patterns must be such, however, that no more than f_ℓ^s outgoing links and no more than f_ℓ^r incoming links are affected at any process per round. An analysis of the assumption coverage [62] in the presence of independent, identically distributed probabilistic link failures confirmed that this model is suitable even for substantial link failure rates. Most existing consensus algorithms [48, 32, 47, 10, 68] were shown to work essentially unchanged under this hybrid failure model [65, 71, 14, 64, 11], provided that the number of processes n in the system is increased by $C_r f_\ell^r + C_s f_\ell^s$ for some small integers C_r, C_s that depend on the particular algorithm.

Naturally, the different values of C_r and C_s obtained for different consensus algorithms raised the question of lower bounds, both on the number of failures that can be tolerated and on the number of rounds required to solve consensus. In the present paper, we provide the results of our comprehensive theoretical study of this subject:

1. In section 2, we provide a precise definition of our system model, which involves both moving omission and moving arbitrary link failures (but no process failures).
2. In section 3, we use a refinement of the bivalency proof techniques introduced in [58] for proving a versatile generalization of Gray's result, which reveals the importance of unimpaired *bidirectional* communication for solving consensus.
3. Using this general result, as well as a new instance of an easy impossibility proof [28], we provide a complete suite of impossibility results and lower

bounds for the required number of processes (section 4) and rounds (section 5).

4. In section 6, we show that our lower bounds are tight and characterize the threshold that, when exceeded, turns a correct process exhibiting omission (resp., arbitrary link failures according to our model) into a classic omission (resp., Byzantine faulty process).
5. In section 7, we survey alternative approaches for modeling link failures and analyze their assumption coverage in a simple probabilistic setting. It turns out that our model is the only one with a coverage that approaches 1 (rather than 0) for large n.

Some conclusions and directions of future work in section 8 eventually complete our paper.

2. System model. We consider a system of n distributed *processes*, each identified by a unique id $p \in \Pi = \{1, \dots, n\}$. The processes are fully connected by a point-to-point network made up of unidirectional *links*. Every pair of processes p and $q \neq p$ is hence connected by a pair of links (p, q), from sender process p to receiver process q, and (q, p), from sender process q to receiver process p, which are considered independent of each other. To simplify our presentation, we also assume that there is a link (p, p) from every process $p \in \Pi$ to itself. Our system hence contains n^2 unidirectional links. Links may reorder messages, that is, are not assumed to be FIFO.

2.1. Computing model. For our computing model, we employ the standard lock-step round model as used in [58]. Every process p is modeled as a deterministic state machine—acting on some local state $state_p \in State_p$—that can send and receive messages from some (possibly infinite) alphabet $\mathcal{M}$. The initial state of process p is drawn from a set of initial states $Init_p \subseteq State_p$. All processes execute, in perfect synchrony, a sequence of atomic computing steps forming a sequence of lock-step rounds $k \in K = \{1, 2, \dots\}$: In round k, process p performs the following steps:

1. Initializes its *received messages vector* Rm_p to $\forall q \in \Pi : rm_p[q] = \emptyset$, where $\emptyset$ represents "no message," and sends at most one message $msg_p^k = Msg_p(state_p, k)$ to every process $q \in \Pi$ (including itself); $Msg_p : State_p \times K \to \mathcal{M} \cup \{\emptyset\}$ denotes the *message sending function* of the algorithm executed by p.
2. Waits for some time while receiving messages into Rm_p. This time must be sufficiently long to allow delivery of (most of) the round k messages. We assume that no messages arrive after this waiting period is over; practically, if late messages arrive, they are discarded.
3. Performs a state transition from $state_p$ to $state'_p = Trans_p(state_p, rm_p, k)$, where $Trans_p : State_p \times Rm_p \times K \to State_p$ denotes the *state transition function* of the algorithm executed by p.

Note that the round number k can be viewed as global time in this model and is typically part of $state_p$.

The distributed algorithm executed by the processes is hence specified by the pairs of message sending function and message transition function $\{(Msg_p, Trans_p) | \, p \in \Pi\}$.

The *configuration* C^k of the system at the end of round k is the vector of states $(state_1^k, \dots, state_n^k)$ obtained at the end of round k (after the state transition); the initial configuration is $C^0 = (init_1, \dots, init_n)$ with $\forall p \in \Pi : init_p \in Init_p$. The system-wide $n \times n$ *received messages matrix* $\mathcal{R}^k$ for round k is the column vector

$(rm_1^k, \ldots, rm_n^k)^T$ of all processes' received messages vectors in round k, i.e.,

$$\mathcal{R}^k = \begin{pmatrix} rm_1^k[1] & rm_1^k[2] & \ldots & rm_1^k[n] \\ rm_2^k[1] & rm_2^k[2] & \ldots & rm_2^k[n] \\ \vdots & \vdots & \vdots & \vdots \\ rm_n^k[1] & rm_n^k[2] & \ldots & rm_n^k[n] \end{pmatrix}, \tag{2.1}$$

with entry $rm_p^k[q]$ denoting the message that process p received from process q via its incoming link in round k, or $\emptyset$ if no such message was received.

A *run* (also termed *execution*) of the distributed algorithm is an infinite sequence $C^0, \mathcal{R}^1, C^1, \mathcal{R}^2, \ldots$ of configurations alternating with received messages matrices, starting from some initial configuration $C^0 \in (Init_1, \ldots, Init_n)$.

2.2. Failure model. We assume that all processes are correct,[1] but links may fail transiently.

Consider the system-wide $n \times n$ *sent messages matrix* $\mathcal{S}^k$ for round k, which consists of n identical rows containing the message sent by every process in round k, i.e.,

$$\mathcal{S}^k = \begin{pmatrix} msg_1^k & msg_2^k & \ldots & msg_n^k \\ msg_1^k & msg_2^k & \ldots & msg_n^k \\ \vdots & \vdots & \vdots & \vdots \\ msg_1^k & msg_2^k & \ldots & msg_n^k \end{pmatrix}, \tag{2.2}$$

with msg_p^k denoting the message process p sends to all processes via its outgoing links in round k, or $\emptyset$ if no message is sent.

Clearly, in case of no link failures in round k, $\mathcal{R}^k = \mathcal{S}^k$ since every message sent by p via its outgoing link to q is received faithfully by q via its incoming link from p. A link failure hitting the directed link (p, q) results in $rm_q^k[p] \neq msg_p^k$, however, so $\mathcal{R}^k \neq \mathcal{S}^k$ in this case. As in [58], we distinguish the following types of link failures of (p, q) in a single round k:

- *Correct link:* $rm_q^k[p] = msg_p^k$.
- *Lossy link:* $\emptyset = rm_q^k[p] \neq msg_p^k$.
- *Erroneous link (corruption):* $\emptyset \neq rm_q^k[p] \neq msg_p^k \neq \emptyset$.
- *Erroneous link (spurious):* $\emptyset \neq rm_q^k[p] \neq msg_p^k = \emptyset$.

For some round k, a lossy link is called *omission faulty*, an erroneous link (corrupted or spurious) is termed *arbitrary faulty*. A link producing either type of failure is termed *faulty*.

Our link failure model, originally introduced in [66, 65], is such that, system-wide, up to $c \cdot n^2$ links for some $c < 1$ may be faulty in any round. We cannot allow *any* set of $c \cdot n^2$ links to be hit by link failures, however, but require some restriction on the allowed link failure patterns: Let $\mathcal{F}^k$ be the $n \times n$ *failure pattern matrix* with entries

$$f_q^k[p] = \begin{cases} ok & \text{if } rm_q^k[p] = msg_p^k, \\ om & \text{if } \emptyset = rm_q^k[p] \neq msg_p^k, \\ arb(e) & \text{otherwise, where } e \text{ encodes the type of the actual error,} \end{cases}$$

which is just the difference of $\mathcal{R}^k$ and $\mathcal{S}^k$ interpreted as ok, om, or $arb(e)$ on a per entry basis, depending on the corresponding link behavior. The *feasible* pattern of

[1] Except for section 5, where we also allow process crashes.

system-wide link failures must be such that for every process p and every round k, the following hold:

- (A^r) p's row $(f_p[1], \ldots, f_p[n])$ in $\mathcal{F}^k$ contains at most f_ℓ^r entries $\neq ok$, with at most $f_\ell^{r,a} \leq f_\ell^r$ of those equal to $arb(.)$. Since row p corresponds to p acting as a receiver process, we say that p may perceive at most f_ℓ^r *receive link failures* (on its incoming links), with up to $f_\ell^{r,a}$ arbitrary ones among those.
- (A^s) p's column $(f_1[p], \ldots, f_n[p])^T$ in $\mathcal{F}^k$ contains at most f_ℓ^s entries $\neq ok$, with at most $f_\ell^{s,a} \leq f_\ell^s$ of those equal to $arb(.)$. Since column p corresponds to p acting as a sender process, we say that p may experience at most f_ℓ^s *send link failures* (on its outgoing links), with up to $f_\ell^{s,a}$ arbitrary ones among those.

Note that *every* process in the system may experience up to f_ℓ^s send link failures ($f_\ell^{s,a}$ of them arbitrary), and up to f_ℓ^r receive link failures ($f_\ell^{r,a}$ of them arbitrary) in every round. In addition, the particular links actually hit by a link failure may be different in different rounds. Of course, they may also remain the same, which makes our link failure model, for example, applicable to not fully connected networks as well; cf. [67].

In the above modeling, the primary failure instance is the link failure pattern in the matrix $\mathcal{F}^k$. It determines how many link failures could be experienced by every process, both as a sender (send link failures f_ℓ^s, $f_\ell^{s,a}$) and as a receiver (receive link failures f_ℓ^r, $f_\ell^{r,a}$). Clearly, assumptions (A^r) and (A^s) imply that at most nf_ℓ^s outgoing links and at most nf_ℓ^r incoming links may be hit by a link failure. Since every outgoing link is of course some receiver's incoming link, it follows that the maximum allowed number of link failures occurs when $f_\ell^s = f_\ell^r = f_\ell$.

In our subsequent analysis, however, our assumptions on send link failures, as captured by (A^s) and f_ℓ^s, $f_\ell^{s,a}$, will be independent of the assumptions made on receive link failures, as captured by (A^r) and f_ℓ^r, $f_\ell^{r,a}$. Doing this allows us to extend the range of applicability of our model. In particular, by restricting the assumption "*every* process may commit up to f_ℓ^s send link failures" to "at most f_ℓ^r processes in some fixed subset of the processes may commit up to f_ℓ^s send link failures," we can also model *restricted process failures*: For example, a restricted omission faulty process is perceived as omission faulty only by at most f_ℓ^s receiver processes per round (rather than by all receivers, as allowed in the case of a standard omission faulty process); see section 6 for details.

Note that adding classic process failures to the picture would provide a "fallback" in cases where the link failure restrictions (A^s) and/or (A^r) are violated: As long as the numbers of link failures experienced by a process p do not exceed the thresholds f_ℓ^r, $f_\ell^{r,a}$, f_ℓ^s, and $f_\ell^{s,a}$, process p can be considered correct. Otherwise, p can just be considered faulty, in which case the link failure restrictions of course do not apply. The resulting hybrid perception-based failure model has been applied successfully in the analysis of several different algorithms [66, 65, 71, 14, 64, 72, 11]. In order to focus on the intrinsic costs of link failures, we will not add standard process failures to the model of this paper, however.

2.3. The consensus problem. Binary consensus is the problem of computing a common binary output value from binary input values distributed among all processes. We assume that every process p has a read-only *input value* $x_p \in \{0, 1\}$ in $state_p$, which is supplied via the initial state $init_p \in Init_p$ to p's local instance of a synchronous deterministic distributed consensus algorithm. In addition, p has a write-once *output value* $y_p \in \{0, 1\}$ in $state_p$, initially undefined. Process p irreversibly computes ("decides upon") y_p according to the following requirements:

(C0) *Termination*: Every process decides within a finite number of rounds (that may be different for different processes and may depend on the particular execution).

(C1) *Agreement*: Every two processes p and q compute the same output value $y_p = y_q$.

(C2) *Weak validity* [29]: If all processes start with the same input value, then every process p computes
- $y_p = 1$ if $\forall q : x_q = 1$ and no link failure has occurred in the entire execution,
- $y_p = 0$ if $\forall q : x_q = 0$.

Note that practical consensus algorithms usually guarantee the following stronger validity property:

(C2') *Validity*: If all processes start with the same input value $\forall q : x_q = v$, then every process p computes $y_p = v$.

In particular, $y_p = 1$ in the case of $\forall q : x_q = 1$ even when link failures have occurred. We will employ the weaker form of validity in our proofs most of the time, since impossibility of consensus under (C2) obviously implies impossibility of consensus under (C2') as well.

3. Basic results. In this section, we will provide a generic analysis of consensus in our setting, which essentially follows the approach taken in [58]:[2] Using bivalence arguments, we will show that consensus is impossible if every process p can *withhold* its information from a nonempty subset $Q = Q(p)$ of processes for an arbitrary number of rounds. Withholding is a weaker property than the "adjacency-preserving" property used in [58], however, and so our generic results are slightly stronger than those of [58] and hence need a different proof. In particular, our findings reveal the importance of unimpaired *bidirectional* communication between processes for solving consensus.

In order to motivate the need for restricting failure patterns and to set the stage for our more advanced proofs, we start with Gray's well-known result [33], in the formalization of [50, Thm. 5.1]. It is devoted to the coordinated attack problem, which is just consensus with weak validity (C2) as stated in section 2.3. This result assumes, however, that there are no constraints on the link failure pattern matrices (except that only omission failures are allowed).

THEOREM 1 (Gray's impossibility [50, Thm. 5.1]). *There is no deterministic algorithm that solves the coordinated attack problem in a synchronous* 2*-process system with arbitrary lossy links.*

Proof. Suppose that the failure-free execution $\overline{E}$ of a 2-process system with omission faulty links terminates at the end of round r when starting with initial values $[1, 1]$. By validity, the common decision value must be 1 in $\overline{E}$. Since decisions are irreversible, we can safely drop all the messages some algorithm might send in rounds $> r$ without changing the decision value. The resulting "truncated" execution E shown in Figure 3.1 is obviously feasible.

By means of induction on the number of messages k sent in E, we show that the decision value 1 does not change even when we drop all messages in E: For $k = 0$, the claim holds trivially. For the induction step, assume that $k > 0$ messages are sent in E where both processes decide by some round r and no messages are sent after

[2] Note that we could also have employed the *layering framework* [52] by Moses and Rajsbaum, which provides generic consensus impossibility and lower bound results in a model-independent way. Although similar in its general structure, it is based on quite different lower level "tools," for example, potence instead of valence, which are—contrary to [42]—not required in our relatively simple setting.

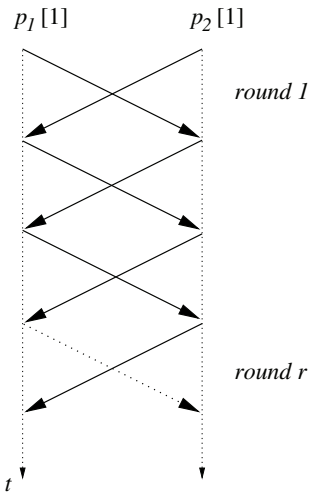


FIG. 3.1. *Execution of a 2-process synchronous consensus algorithm with unrestricted link omission failures, starting with initial values* $[1,1]$*, truncated after round* r *by which both processes decide.*

round r. Let m be the last message from, w.l.o.g., $p_1 \to p_2$ in E (cf. the dotted line in Figure 3.1). If m is dropped, the resulting execution $\overline{E}'$ is indistinguishable from E for p_1, so p_1 and, by agreement, also p_2 must eventually decide upon 1. Clearly, in $\overline{E}'$, process p_2 could decide after round r and even send out additional messages in some round $r' > r$—we can guarantee only that the decision value is the same. However, p_1 has already decided by round r and hence cannot make use of such "late" messages. Consequently, we can safely drop all those late messages in $\overline{E}'$, if any, leading to an execution E' where only $k-1$ messages are sent, both processes decide upon 1 by some round r', and no messages are sent after round r'. We can hence apply the induction hypothesis to E', which completes the induction step.

Since the processes are fully isolated from each other in the resulting execution where all messages have been dropped, changing the initial values to $[1,0]$ and then to $[0,0]$ cannot affect the decision value either, but now the outcome of the final execution would violate validity. □

As our first "real" result, we will now show that solving consensus is even impossible when a link—viewed as a pair of unidirectional links—loses or, in case of arbitrary link failures, corrupts messages only in one direction, i.e., when either process (but not necessarily both) can withhold information for an arbitrary number of rounds from the other. Eventually bidirectional communication is hence mandatory for any deterministic consensus algorithm. Solutions exist, however, if the direction of the message loss is fixed; see the remarks following Theorem 2 below.

Unfortunately, this stronger result cannot be shown by generalizing the proof of Theorem 1: We are not allowed to simply drop all messages in later rounds to "hide" the effect of dropping/restoring round r-messages here, since this would amount to a link failure in both directions and hence an infeasible execution. It was shown in [58], however, that bivalency arguments [29] can successfully be applied in this setting.

We start with some notation: Recall that we defined an execution as an infinite sequence $C^0, \mathcal{R}^1, C^1, \mathcal{R}^2, \ldots$ of configurations alternating with received message matrices, starting from some initial configuration $C^0 \in (Init_1, \ldots, Init_n)$. For a con-

figuration $C^k = (c_1^k, \ldots, c_n^k)$, let $state_p(C^k) = c_p^k$ denote the local state of process p in C^k. Since we are dealing with deterministic algorithms, executions are uniquely determined by the initial configuration, i.e., the initial values x_p for all processes p, and the sequence of link failure pattern matrices $\mathcal{F}^k$ in rounds $k \geq 1$. Consequently, we can unambiguously specify executions as infinite sequences $C^0, \mathcal{F}^1, C^1, \mathcal{F}^2, \ldots$ of configurations alternating with failure pattern matrices. Moreover, a finite sequence of failure pattern matrices $\mathcal{F}_x = \mathcal{F}^k, \ldots, \mathcal{F}^{k+x}$ starting from configuration C^{k-1} uniquely determines the finite *execution segment* $C^{k-1}, \mathcal{F}^k, C^k, \mathcal{F}^{k+1}, C^{k+1}, \ldots, \mathcal{F}^{k+x}, C^{k+x}$, and we write $C^{k+x} = \mathcal{F}_x(C^{k-1})$. An execution is called *feasible* if all $\mathcal{F}^k$ are feasible, i.e., adhere to the link failure pattern constraints (A^r) and (A^s) in section 2.2. Finally, a configuration C' is called *reachable* from configuration C if there is a feasible finite sequence of failure pattern matrices $\mathcal{F}_x$ such that $C' = \mathcal{F}_x(C)$. A configuration C is reachable if it is reachable from some initial configuration.

A configuration is called *v-decided* (*decided* for short) if all processes have decided on a common decision value $v \in \{0, 1\}$. A configuration C is *v-valent* (*univalent* if v is not known or irrelevant) if all decided configurations reachable from C are v-decided; in particular, it is impossible to reach a 1-decided configuration from a 0-valent C. On the other hand, C is *bivalent* if both 0-decided and 1-decided configurations can be reached from C.

For example, in the case of $n = 2$, given any configuration $C^{k-1} = (c_1^{k-1}, c_2^{k-1})$, there are only four possible *successor configurations* C_{00}^k, C_{01}^k, C_{10}^k, and C_{11}^k in the case of message omissions: Configuration C^{k-1} is followed by the successor configuration C_{xy}^k, depending on whether the message $p_2 \to p_1$ (x) and/or $p_1 \to p_2$ (y) is lost $(x, y = 0)$ or correct $(x, y = 1)$ in round $k \geq 1$.[3] Note that C_{00}^k is feasible only in the setting of Theorem 1, where unrestricted losses are allowed, since both messages are lost there. In the context of Theorem 2, however, C_{00}^k cannot be reached from C^{k-1} since losing both messages is not feasible.

The situation is more complicated in case of arbitrary link failures, however, where C^{k-1} can have more than four successor configurations: After all, different errors e, experienced, for example, by the message from $p_2 \to p_1$ due to an arbitrary link failure with $f_1^k[2] = arb(e)$ in the failure pattern matrix $\mathcal{F}^k$, might result in different states of p_1. Fortunately, we can keep the convenient assumption of just four successors if we replace a single successor state C_{xy}^k by the set of possible successor states. Using this extended interpretation, C_{01}^k actually consists of all configurations reachable from C^{k-1}, where only the message $p_2 \to p_1$ is lost or erroneous, for example. Univalence of a set of configurations C_{xy}^k means that all individual configurations $C^k \in C_{xy}^k$ are univalent, whereas bivalence means that at least one individual configuration $C^k \in C_{xy}^k$ is bivalent. Bivalence proofs are easily generalized to this extended interpretation.

Finally, our notation can be easily generalized to $n > 2$: (Sets of) successor configurations are indexed by strings of $n(n-1)$ 0's or 1's, corresponding to every link in a system of n processes, with 0 denoting a lost or faulty message, and 1 denoting a nonfaulty one.

Two successor configurations C_v^k and C_w^k are called *neighbors* if the received message matrices $\mathcal{R}_v^k$ and $\mathcal{R}_w^k$ that lead to C_v^k and C_w^k, respectively, differ in at most one entry: W.l.o.g., this entry contains the correct message in $\mathcal{R}_v^k$ but a lost/erroneous one in $\mathcal{R}_w^k$. Consequently, all configurations in C_{00}^k and C_{01}^k are neighbors in the

[3] Message "self-transmission," from $p_1 \to p_1$ and $p_2 \to p_2$, is always assumed to be failure-free here. Since we are dealing with impossibility results and lower bounds, this can safely be assumed.

above system, but the ones in C^k_{01} and C^k_{10} are not. The *successor graph* $\mathcal{G}_C$ of some configuration C consists of all successor configurations of C, where all neighbors are connected by an edge. We can make the following well-known observation.

LEMMA 1. *The successor graph $\mathcal{G}_C$ of any configuration C of a consensus algorithm under our system model is connected.*

Proof. Let $k \geq 1$ be the round at the end of which the transition from C to one of its successor configurations takes place. Obviously, the failure-free successor configuration $C_{1..1}$ where no round k messages has been lost or corrupted must be in $\mathcal{G}_C$. Let C_X be any other successor configuration caused by a feasible link failure pattern, with $\overline{\mathcal{M}}_X$ denoting the corresponding set of lost or faulty messages. Since the link failure pattern $\overline{\mathcal{M}}'_X$, obtained from $\overline{\mathcal{M}}_X$ by removing (= repairing) exactly one of the lost or faulty messages, is of course also feasible, the resulting successor configuration C'_X is a neighbor of C_X and obviously $C'_X \in \mathcal{G}_C$. Since $|\overline{\mathcal{M}}'_X| = |\overline{\mathcal{M}}_X| - 1$, this argument can be repeated until the failure-free successor configuration $C'_X = C_{1..1}$ is reached. Hence, there is a path from any C_X to $C_{1..1}$ in the successor graph $\mathcal{G}_C$. □

The result of Lemma 1 will be used primarily in conjunction with Lemma 2.

LEMMA 2. *Suppose that all successor configurations of some configuration C with successor graph $\mathcal{G}_C$ are univalent. If there are two arbitrary successor configurations C' and C'' among those that are* 0*-valent and* 1*-valent, respectively, then there are also two* neighboring *successor configurations $\overline{C}'$ and $\overline{C}''$ that are* 0*-valent and* 1*-valent.*

Proof. Since C' and C'' are connected in $\mathcal{G}_C$ and have different valences, there is a path of configurations connecting C' and C''. This implies that there must be neighbors $\overline{C}'$ and $\overline{C}''$ on this path where the valence changes. □

With these two lemmas, it is fairly easy to show that eventually bidirectional communication is mandatory for solving consensus in a 2-process system: Theorem 2 considers link omission failures only (that may change arbitrarily from round to round, however) and strengthens Gray's theorem, Theorem 1. Note that it could also be derived from the impossibility of consensus under a single moving process omission failure [58] in a system of $n = 2$ processes.

THEOREM 2 (unidirectional 2-process impossibility). *There is no deterministic algorithm that solves consensus in a synchronous system with two nonfaulty processes connected by a lossy link, even if communication is reliable in one direction in every round.*

Proof. Assume that there are algorithms $\mathcal{A}_{p_1}$ and $\mathcal{A}_{p_2}$ running on processes p_1 and p_2 that jointly solve consensus in a 2-process system with unidirectional communication. We will show inductively that every bivalent configuration has at least one bivalent successor. This implies that it is impossible to always reach a final decision within any finite number of rounds.

For the base case $k = 0$ of our inductive construction, we have to show that there is a bivalent initial configuration. Consider the configuration $C^0(01)$, where p_1 starts with initial value 0 and p_2 starts with initial value 1. If $C^0(01)$ is bivalent, we are done. If $C^0(01)$ is 0-valent, the execution where all messages from $p_1 \to p_2$ are lost in all rounds must also lead to a 0-decided configuration. However, this execution is indistinguishable for p_2 from the equivalent execution that starts from $C^0(11)$ (where p_1 has initial value 1 instead of 0), which implies that the common decision value must also be 0 here. Since $C^0(11)$ must lead to a 1-decided configuration in case of no link failures by validity, we have shown that $C^0(11)$ is bivalent in this case. An analogous argument can be used to show that $C^0(00)$ would be bivalent when $C^0(01)$

is 1-valent.

For the induction step $k \geq 1$, we assume that we have already reached a bivalent configuration C^{k-1} at the end of round $k-1$ and show that at least one of the feasible successor configurations C^k_{01}, C^k_{10}, and C^k_{11} reached at the end of round k is bivalent. Assuming the contrary, all of those must be univalent. The bivalence of C^{k-1} implies that at least one of C^k_{01}, C^k_{10}, and C^k_{11} must be 0-valent and one must be 1-valent (i.e., C^{k-1} must be a *fork* [69]). By Lemma 2 in conjunction with Lemma 1, either the neighbors C^k_{11} and C^k_{01}, or else C^k_{11} and C^k_{10} must be v-valent and $(1-v)$-valent, respectively. W.l.o.g. assume that C^k_{11} is v-valent and C^k_{01} is $(1-v)$-valent for some $v \in \{0,1\}$. Since the only difference between those two configurations is that the message from $p_2 \to p_1$ arrives in the former but not in the latter, we consider the execution C^{k*}_{11}, where all messages from $p_1 \to p_2$ are lost in all rounds $> k$. As p_1 cannot tell p_2 whether it has received a round k message, the execution C^{k*}_{11} starting from C^k_{11} is indistinguishable for p_2 from the same execution starting from C^k_{01}. Since p_2 must eventually decide upon the same value, C^k_{11} and C^k_{01} cannot have different valences. □

Remarks.

1. If message losses can occur only in one direction, and if that direction is known to the algorithm, then there is a trivial 1-round algorithm that solves consensus in a 2-process system: The process that can communicate with its peer sends its own value and decides upon it; the other process decides upon the value received from its peer.
2. If message losses can occur only in one and the same but unknown direction, there is a simple 2-round algorithm that solves consensus in a 2-process system: Every p_i initially sets $v_i := x_i$. In the first round, p_1 sends v_1 to p_2. If p_2 receives v from p_1, it sets $v_2 := v$. In the second round, p_2 sends its value v_2 to p_1. If p_1 receives v from p_2, it sets $v_1 := v$. At the end of the second round, process p_i, $i = 1, 2$, decides upon v_i. It is easy to verify that the decision values satisfy validity and agreement.

Our next goal will be to show that consensus cannot be solved in any system of $n \geq 2$ processes if, for every process, bidirectional communication with every peer cannot eventually be guaranteed. More specifically, we will show that this is the case when every process p can *withhold* its information from some nonempty subset $\mathcal{Q} = \mathcal{Q}(p)$ of processes (but not necessarily vice versa) from any round $k+1$ on, namely, when there is a sequence of failure pattern matrices for rounds $k+1, k+2, \ldots$ such that every $q \in \mathcal{Q}$ has the same view of the resulting execution after round $k+x$, independently of the state of p in the starting configuration C^k.

DEFINITION 1 (withholding). *A process p, in some reachable configuration C^k, can* withhold its information from round $k+1$ on*, if there is an infinite sequence of failure pattern matrices $\mathcal{F} = \mathcal{F}^{k+1}, \mathcal{F}^{k+2}, \ldots$ and a nonempty set $\mathcal{Q}$ of processes with $p \notin \mathcal{Q}$ (where both $\mathcal{F}$ and $\mathcal{Q}$ may depend on p and C^k) such that, for any reachable configuration $\overline{C}^k$ with $\forall q \in \mathcal{Q} : state_q(C^k) = state_q(\overline{C}^k)$, there exists an infinite sequence of failure pattern matrices $\overline{\mathcal{F}} = \overline{\mathcal{F}}^{k+1}, \overline{\mathcal{F}}^{k+2}, \ldots$ such that $\forall q \in \mathcal{Q} : state_q\big(\mathcal{F}_x(C^k)\big) = state_q\big(\overline{\mathcal{F}}_x(\overline{C}^k)\big)$ for any finite prefix $\mathcal{F}_x = \mathcal{F}^{k+1}, \ldots, \mathcal{F}^{k+x}$ of $\mathcal{F}$ and $\overline{\mathcal{F}}_x = \overline{\mathcal{F}}^{k+1}, \ldots, \overline{\mathcal{F}}^{k+x}$ of $\overline{\mathcal{F}}$.*

We say that p can withhold its information *if it can withhold its information from round $k+1$ on, for every $k \geq 0$, starting from any reachable configuration C^k.*

Definition 1 implies that if p can withhold its information, then, since $|\mathcal{Q}| \geq 1$,

there is at least one process $q = q(p)$ which never gets any useful information from p at all during $\mathcal{F}$—neither directly nor indirectly via other processes.

It is important to note that withholding is a weaker property than *adjacency-preserving*, on which the generic results of [58] are based: The latter requires that *all* processes $\neq p$ have the same state in $\mathcal{F}_x(C^k)$ and $\mathcal{F}_x(\overline{C}^k)$, i.e., correspond to p withholding its state from *every* $q \neq p$. Consequently, adjacency-preserving implies withholding, but not vice versa. In fact, the proofs of Theorems 3 and 6 will show that withholding—but not adjacency-preservation—is possible under our failure model for certain parameter values. Basically, this is due to some "unidirectional" partitioning of the n processes in the system, which is possible when (A^s) and (A^r) hold, provided that n is small enough.

As a consequence, we cannot use the generic consensus impossibility results from [58]. The following Lemma 3 reveals, however, that the ability of every process p to withhold its information from some nonempty subset of the processes arbitrarily long already prohibits a solution to the consensus problem.

LEMMA 3 (n-process impossibility). *Consider a synchronous n-process system with omission and/or arbitrary link failures. There is no deterministic algorithm that solves consensus in such a system if any process p can withhold its information in any configuration.*

Proof. The proof is a generalization of the proof of Theorem 2, although more involved: We assume here that there are n algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$ running on the n processes $p_1, \ldots, p_n$ in the system that jointly solve consensus. We will show inductively that there is an infinite execution involving bivalent configurations only, which makes it impossible to always reach a decision within a finite number of rounds.

For the base case $k = 0$ of our inductive construction, we have to show that there is a bivalent initial configuration C^0. As in [29], we consider the initial configuration $C^0(111..1)$, where all processes start with the initial value 1. If this configuration is bivalent, we are done. Otherwise, $C^0(111..1)$ can only be 1-valent, since validity requires a decision value 1 in the failure-free case. Now consider $C^0(011..1)$, where process p_1 starts with 0 and all others with 1. If this configuration is bivalent, we are done. If not, we assume first that it is 0-valent and choose the execution starting from $C^0(011..1)$ where p_1 withholds its value from some process $q(p_1) = p_x \neq p_1$; such an execution must exist according to Definition 1 since p can withhold its information. This execution is indistinguishable for p_x from the analogous execution starting from $C^0(111..1)$, however, so p_x's (and hence the common) decision must be 1 here. This contradicts the stipulated 0-valence of $C^0(011..1)$, however, which could hence only be 1-valent.

The whole argument can now be repeated for p_2 in place of p_1, etc., until either a bivalent initial configuration has been found or the 1-valent initial configuration $C^0(0..001)$ has been reached; in $C^0(0..001)$, the processes $p_1, \ldots, p_{n-1}$ start with 0 and p_n starts with 1. We again consider the execution where p_n withholds its information from some process $q(p_n) = p_y \neq p_n$. For p_y, this execution is indistinguishable from the same one starting from $C^0(0..000)$, which must lead to a decision value of 0 by weak validity (C2). This contradicts the stipulated 1-valence of $C^0(0..001)$, however.

For the induction step $k \geq 1$, we assume that we have already reached a bivalent configuration C^{k-1} at the end of round $k-1$. We must show that at least one of the feasible successor configurations C^k that can be reached at the end of round k is bivalent. If this is true in the first place, we are done. If not, all successor configurations C^k must be univalent. However, the bivalence of C^{k-1} implies that at least

one of those must be 0-valent and one must be 1-valent (i.e., C^{k-1} must be a *fork* [69]). By Lemma 2, there must also be 0-valent and 1-valent successor configurations C_0^k and C_1^k, respectively, that are neighbors. Assume that they differ only in the state of process r that has some specific round k message correct in C_0^k but incorrect in C_1^k (or vice versa). Now consider the two executions starting with C_0^k and C_1^k, where r withholds its round k information from some process $q(r) = p_z \neq r$ in any future round $> k$. They are indistinguishable for p_z, which means that p_z and, by agreement, all other processes must compute the same decision in both executions. This contradicts the stipulated different valences of C_0^k and C_1^k, however. □

Lemma 3 has a number of important consequences. First, it reveals an interesting asymmetry in the "severeness" of receive link failures (A^r) versus send link failures (A^s). This can be seen by considering two instances of a 3-process system, where two processes A, B cannot communicate bidirectionally due to receive and/or send link failures: In the system shown in Figure 3.2 (called of type R), processes A and B may not receive the messages from both peers due to excessive receive link failures ($f_\ell^r = 2$ and $f_\ell^s = 1$). In the system shown in Figure 3.3 (of type S), processes A and B may fail to send to both peers due to excessive send link failures ($f_\ell^r = 1$ and $f_\ell^s = 2$).

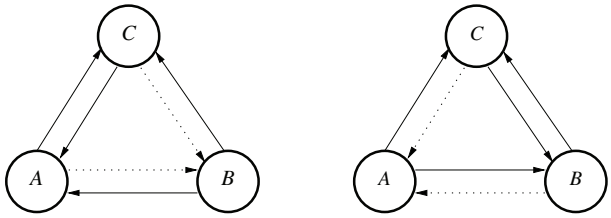


FIG. 3.2. *3-process system (type R), where processes A and B cannot communicate in one direction due to excessive receive link failures ($f_\ell^r = 2$ and $f_\ell^s = 1$). The left scenario shows the case $A \not\rightarrow B$ and the right one $B \not\rightarrow A$, which may alternate arbitrarily.*

It follows from Lemma 3 that no algorithm can solve consensus in a system of type R, even if process C is fixed and known to the algorithm. For, since A may fail to receive any information from any other process in the system, choosing $q(p) = A$ secures withholding by every $p \neq A$. Similarly, since B may also fail to receive the information from any peer, it provides the required $q(p)$ for withholding by process $p = A$. Hence, consensus is impossible in a 3-process system with $f_\ell^r = 2$ and $f_\ell^s = 1$; note that C is not fixed here, which makes consensus even harder to solve.

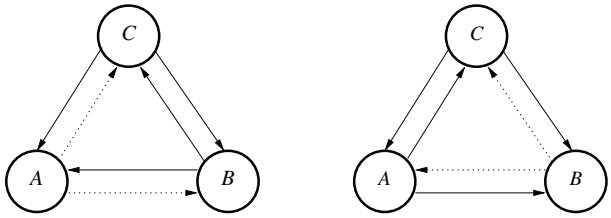


FIG. 3.3. *3-process system (type S), where processes A and B cannot communicate in one direction due to excessive send link failures ($f_\ell^r = 1$ and $f_\ell^s = 2$). The left scenario shows the case $A \not\rightarrow B$ and the right one $B \not\rightarrow A$, which may alternate arbitrarily.*

On the other hand, for systems of type S where C is fixed and known to the algorithm, there is a trivial solution that lets all processes decide upon the value of

process C. If process C is fixed but not known, consensus can be solved by means of the algorithm described in [34]. No solution exists in a system of type S only if C is not fixed—as is the case in a 3-process system with $f_\ell^r = 1$ and $f_\ell^s = 2$.

As a final remark, we note that the above observations are in accordance with the results of [43], where it was shown that bounded-time consensus is impossible in the $\diamond$MFM ("majority from majority") link timing model.

4. Number of processes. Using the results of section 3, we will first establish a lower bound for purely omissive link failures ($f_\ell^{r,a} = f_\ell^{s,a} = 0$).

For the special case $f_\ell^s = f_\ell^r = f_\ell > 0$, such a lower bound can be inferred from Theorem 2 (even from Theorem 1) by a straightforward simulation-type proof: Assume that there is a deterministic algorithm $\mathcal{C}$ that solves consensus for $n = 2f_\ell$. Using $\mathcal{C}$, it is possible to construct a solution for consensus in a 2-process system with lossy links, which is impossible, however.

The detailed proof is as follows: Partition the n processes into two subsets P_A and P_B of size f_ℓ each. Two *superprocesses* A and B are used to simulate the execution of the processes in P_A and P_B, respectively. All the links between the simulated processes in the two superprocesses are routed over a single *superlink*. For a superprocess's decision value, any simulated process's decision value can be taken. In order to ensure that $\mathcal{C}$ achieves consensus among all (simulated) processes, we must show that our link failure assumptions are not violated for any simulated process in any superprocess in case of a superlink failure: Any simulated process must not get more than $f_\ell^r = f_\ell$ receive link failures and must not produce more than $f_\ell^s = f_\ell$ send link failures. This is trivially satisfied since $f_\ell^s = f_\ell^r = f_\ell$, however. Hence, our solution would achieve consensus among the two superprocesses, which violates Theorem 2 (even Theorem 1, since bidirectional partitioning could happen here).

For the general case of arbitrary f_ℓ^s and f_ℓ^r, the lower bound for omission link failures can immediately be derived from Lemma 3.

THEOREM 3 (lower bound processes 1). *Any deterministic algorithm that solves consensus under our system model with $f_\ell^s, f_\ell^r > 0$ needs $n > f_\ell^r + f_\ell^s$.*

Proof. We first show that, for any process p, we can arbitrarily choose a set $\mathcal{P}$ of f_ℓ^r processes including p, where no process in $\mathcal{P}$ sends any messages to a process in $\mathcal{Q} = \Pi \backslash \mathcal{P}$ in case of $n = f_\ell^s + f_\ell^r$ in some feasible execution: Since there are $f_\ell^s \geq 1$ processes in $\mathcal{Q}$, every process in $\mathcal{P}$ may commit send link failures that omit all processes in $\mathcal{Q}$. Any process in $\mathcal{Q}$ thus experiences exactly f_ℓ^r receive link failures, which is also feasible with respect to our failure model. Hence, there is no information flow from processes in $\mathcal{P}$ to processes in $\mathcal{Q}$ at all, such that every process p can trivially withhold its information. According to Lemma 3, solving consensus is impossible here. ∎

Remarks.

1. According to Corollary 1 in section 6, the lower bound $n > f_\ell^r + f_\ell^s$ provided by Theorem 3 is tight; it is, for example, matched by the authenticated algorithm ZA [65].
2. The result of Theorem 3 implies that, in order to be able to solve consensus, link failures (A^s) and (A^r) may affect at most a minority of processes only. In the setting of Gray's theorem, Theorem 1, however, there is no point in considering this case at all: There is no nonempty minority of processes for $n = 2$. Focusing on overly simple cases hence sometimes hides ways of escaping impossibility results.

In order to find a lower bound for arbitrary link failures, we will again start with

the special case $f_\ell^s = f_\ell^r = f_\ell^{s,a} = f_\ell^{r,a} = f_\ell > 0$. Our derivation will be based on Theorem 4, which shows that no algorithm can solve consensus (with validity (C2')) in a 4-process system in the presence of a single arbitrary link failure per process ($f_\ell^{r,a} = f_\ell^{s,a} = 1$).

THEOREM 4 (4-process impossibility). *There is no deterministic algorithm that solves consensus with validity* (C2') *under our system model for a single arbitrary link failure in a* 4*-process system.*

Proof. We employ a new instance of the "easy impossibility proof techniques" of [28] to show that any deterministic algorithm violates agreement if every process may see an inconsistent value from one of its neighbors. Suppose that our four processes execute a distributed algorithm consisting of specific programs A, B, C, D, which solve consensus under our system model with $f_\ell^{r,a} = f_\ell^{s,a} = f_\ell^s = f_\ell^r = 1$. In order to derive a contradiction, we arrange eight processes in a cube as shown in Figure 4.1. For example, the lower leftmost process labeled $A[0]$ executes algorithm A starting with initial value 0 (the $\boxed{0}$ on its left displays this process's decision value, as explained below). Note carefully that all processes and all links are assumed to be nonfaulty here.

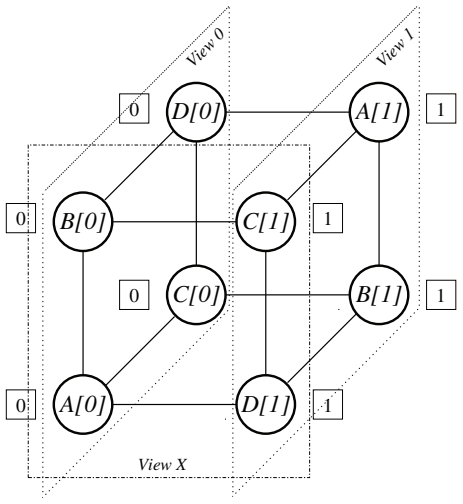


FIG. 4.1. *Topology used for proving the violation of agreement in a* 4*-process system. Eight nonfaulty processes with perfect links are arranged in a cube in a neighborhood-preserving way. The assignment of initial values ensures that all processes in view* 0 *(resp., view* 1*) decide* 0 *(resp.,* 1*), but this violates agreement in view* X*.*

Of course, dealing with a solution for a 4-process system, we cannot expect to achieve consensus in the 8-process system of Figure 4.1. However, due to the special assignment of algorithms to processes, each process observes a neighborhood as in a 4-process system. More specifically, the four processes at any side of the cube (we call it a *view*) can be interpreted as an instance of a legitimate 4-process system. In fact, as can be checked easily, our assignment ensures that any process in a view is connected to exactly one process outside this view. Since we assumed that every process may see an arbitrary faulty input from at most one neighbor, the input from the process outside the view may be arbitrary—it just appears as a process the links of which deliver arbitrary faulty messages.

Now consider the processes in view 0, which all have initial value 0. By the validity

property for consensus, all processes must decide 0 here (the initial value 1 of the processes outside view 0 does not matter, as the links to them are considered arbitrary faulty with respect to view 0). Similarly, in view 1, all processes must decide 1 since they have initial value 1. But now the processes in view X face a problem: Since e.g., the lower leftmost process $A[0]$ observes exactly the same messages in view X as in view 0 by construction, it must decide 0 as observed above. Similarly, as the lower rightmost process $D[1]$ observes exactly the same messages in view X as in view 1, it must decide 1—but this would violate agreement in the 4-process system corresponding to view X. We hence established the required contradiction, thereby completing the proof of Theorem 4. □

Using Theorem 4, a similar simulation-type argument as in the pure omission failure case can be used to show the lower bound $n > 4f_\ell$ for $f_\ell^s = f_\ell^{s,a} = f_\ell^r = f_\ell^{r,a} = f_\ell > 0$ arbitrary link failures. Corollary 1 in section 6 reveals that this lower bound is also tight; it is, for example, matched by the nonauthenticated algorithm OMH [66, 65].

THEOREM 5 (lower bound processes 2). *Any deterministic algorithm that solves consensus with validity* (C2') *under our system model with* $f_\ell^r = f_\ell^s = f_\ell^{s,a} = f_\ell^{r,a} = f_\ell > 0$ *needs* $n > 4f_\ell$.

Proof. Assume that there is a deterministic algorithm $\mathcal{C}$ that solves consensus for $n = 4f_\ell$ in our model. We use $\mathcal{C}$ to construct a solution for a 4-process system of Theorem 4, which provides the required contradiction.

We partition the set of all processes P into four subsets P_A, P_B, P_C, P_D of the same cardinality f_ℓ, and we let each *superprocess* A, B, C, D simulate all the instances of the algorithm in the respective subset. For the superprocess's decision value, any simulated process's decision value can be taken. In order to ensure that $\mathcal{C}$ achieves consensus among all (simulated) processes, we must show that our link failure assumptions are not violated for any process in any superprocess if at most one superlink per process may experience an arbitrary link failure. Since any superlink hosts the links to and from exactly f_ℓ processes, this is trivially fulfilled, however: In case of a superlink failure, every sender process commits at most f_ℓ send link failures (affecting the f_ℓ processes in the receiving superprocess), and every receiver process experiences at most f_ℓ receive link failures (from the f_ℓ processes in the sending superprocess).

Therefore, we have constructed an implementation of a consensus algorithm for a 4-process system which can withstand a single arbitrary link failure. Since this is impossible by Theorem 4, the proof of Theorem 5 is completed. □

Unfortunately, we did not find an easy way to generalize the above simulation-type argument for an arbitrary number $f_\ell^s, f_\ell^{s,a}, f_\ell^r, f_\ell^{r,a} \geq 0$ of link failures (and weak validity (C2)). In order to derive a lower bound for n for this general case, we must hence resort to our key lemma, Lemma 3, again. What needs to be shown here, however, is that every process p can withhold its information: Lemma 5 below will prove that as many as $f_\ell^r + f_\ell^{r,a}$ processes can withhold their information from as many as $f_\ell^s + f_\ell^{s,a}$ processes in case of $n = f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$, provided that

$$\frac{f_\ell^{r,a}}{f_\ell^r} = \frac{f_\ell^{s,a}}{f_\ell^s}. \tag{4.1}$$

Hence, by Lemma 3, $n > f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ is a lower bound for solving consensus if (4.1) holds. If (4.1) does not hold, there are cases where consensus can be solved also for $n \leq f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$. A lower bound in this case is $n > \overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$,

however, where $\overline{f}_\ell^{\,r} \le f_\ell^r$, $\overline{f}_\ell^{\,ra} \le f_\ell^{r,a}$, $\overline{f}_\ell^{\,s} \le f_\ell^s$, $\overline{f}_\ell^{\,sa} \le f_\ell^{s,a}$ are such that (4.1) holds and n is maximal (see the comments prior to Theorem 6 for details).

Diving into the details of our lower bound proof, we start with a simple "balls and boxes" technical lemma. It shows that it is possible to drop white, orange, and purple balls into a matrix such that each row and each column contain some specific numbers of balls of each color. This result will be used subsequently to assert the existence of a certain mapping of send and receive link failures, by interpreting a white, orange, and purple ball as a correct, omission faulty, and arbitrary faulty transmission between a particular sender process (column index) and receiver process (row index).

LEMMA 4 (balls and boxes). *Consider a matrix with $s + s^a$ rows and $r + r^a$ columns, where $s \ge s^a > 0$, $r \ge r^a > 0$, and*

$$r/r^a = s/s^a. \tag{4.2}$$

Then it is possible to drop white, orange, and purple balls into the matrix (one ball per entry) such that any single row contains exactly r^a white, $r - r^a$ orange, and r^a purple balls, whereas any single column contains exactly s^a white, $s - s^a$ orange, and s^a purple balls.

Proof. First, we note that summing up the number of balls of the same color by rows and columns, respectively, in any such assignment yields the same result: For example, we need $w_r = (s + s^a)r^a$ white balls when summing over rows and $w_c = s^a(r+r^a)$ white balls when summing over columns. Since (4.2) implies $sr^a = s^a r$, it follows that $w_r = w_c$. We will now construct such an assignment explicitly.

Consider the first row in our matrix, and let

$$\pi_0, \pi_1, \ldots, \pi_{r+r^a-1}$$

with $\pi_i \in \{\mathit{white}, \mathit{orange}, \mathit{purple}\}$ be its assignment of balls to places according to the following rule: For any integer $x \ge 0$,

$$\pi_x = \begin{cases} \mathit{orange} \vee \mathit{purple} & \text{if } x = c(i) \text{ for some integer } i \ge 0, \\ \mathit{purple} & \text{iff } x = c\big(a(j)\big) \text{ for some integer } j \ge 0, \\ \mathit{white} & \text{otherwise,} \end{cases}$$

where

$$\begin{aligned} c(i) &= \left\lfloor \frac{r + r^a}{r} \cdot i \right\rfloor, \\ a(j) &= \left\lfloor \frac{r}{r^a} \cdot j \right\rfloor, \\ p(j) = c\big(a(j)\big) &= \left\lfloor \frac{r + r^a}{r} \cdot \left\lfloor \frac{r}{r^a} \cdot j \right\rfloor \right\rfloor. \end{aligned}$$

This assignment distributes colored (orange or purple), as well as purple balls alone, as regularly as possible over the $r + r^a$ available places in the first row. The following periodicity properties are immediately apparent from the above definitions: For $0 \le i \le r - 1$, $0 \le j \le r^a - 1$, and any integer $y \ge 0$,

$$0 \le c(i) \le r + r^a - 1 \quad \text{and} \quad c(i + r) = c(i) + r + r^a, \tag{4.3}$$

$$0 \le p(j) \le r + r^a - 1 \quad \text{and} \quad p(j + r) = p(j) + r + r^a, \tag{4.4}$$

$$\pi_{y+r+r^a} = \pi_y. \tag{4.5}$$

From the properties of $c(i)$ and $p(j)$, it follows immediately that $\pi_0, \pi_1, \ldots, \pi_{r+r^a-1}$ contains exactly r colored balls and r^a white ones. Clearly, every cyclic permutation (rotation) $\pi_y, \pi_{y+1}, \ldots, \pi_{y+r+r^a-1}$ of the original $\pi_0, \pi_1, \ldots, \pi_{r+r^a-1}$ has this property as well. Note that index addition in this cyclic permutation should actually be modulo $r + r^a$; (4.5) reveals that this is automatically taken care of, however. Below, we will assign $\pi_y, \pi_{y+1}, \ldots, \pi_{y+r+r^a-1}$ to row y of our matrix to prove our lemma.

By using the equivalences $(r + r^a)/r = (s + s^a)/s$ and $r/r^a = s/s^a$, which follow immediately from (4.2), in the definitions of $c(i)$ and $p(i)$, we obtain similar periodicity properties for $0 \le i \le s-1$, $0 \le j \le s^a - 1$, and any integer $x \ge 0$:

$$0 \le c(i) \le s + s^a - 1 \quad \text{and} \quad c(i+s) = c(i) + s + s^a, \tag{4.6}$$

$$0 \le p(j) \le s + s^a - 1 \quad \text{and} \quad p(j+s) = p(j) + s + s^a, \tag{4.7}$$

$$\pi_{x+s+s^a} = \pi_x. \tag{4.8}$$

As before, this implies that $\pi_0, \pi_1, \ldots, \pi_{s+s^a-1}$ contains exactly s colored balls and s^a white ones. Even more, the periodicity properties (4.6) and (4.7) imply that every cyclic permutation (rotation) $\pi_x, \pi_{x+1}, \ldots, \pi_{x+s+s^a-1}$ of $\pi_0, \pi_1, \ldots, \pi_{s+s^a-1}$ has this property as well; again, (4.8) takes care of index addition modulo $s + s^a$.

Hence, we just have to assign $\pi_y, \pi_{y+1}, \ldots, \pi_{y+r+r^a-1}$ to row y of our matrix, meaning that the entry in column 0 of row y contains the same ball as the entry in column y of row 0, for example. Our findings on the number of balls in cyclic permutations of $\pi_0, \ldots, \pi_{r+r^a-1}$ shows that this assignment respects our lemma's requirement on rows. Similarly, inspection of the resulting matrix shows that column x contains the pattern $\pi_x, \pi_{x+1}, \ldots, \pi_{x+s+s^a-1}$, which respects our requirement on the number of balls in columns as well. For example, for $r = 4, r^a = 2, s = 2, s^a = 1$, we obtain the following assignment:

$$\begin{pmatrix} p & o & w & p & o & w \\ o & w & p & o & w & p \\ w & p & o & w & p & o \end{pmatrix}. \qquad \square$$

Now we are ready to prove our major Lemma 5, which shows that, in case of $n = f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ processes satisfying (4.1), any two executions that lead to two sufficiently "similar" configurations, in the sense that $|\mathcal{Q}| = f_\ell^s + f_\ell^{s,a}$ processes have identical states in both, can be extended by one round in a way that again yields two "similar" configurations for the processes in $\mathcal{Q}$. This implies that all the remaining $f_\ell^r + f_\ell^{r,a}$ processes can withhold their information in the resulting execution. Hence, Lemma 3 can be applied again, which will finally establish our general lower bound result.

LEMMA 5 (similarity). *Consider two configurations $C = (c_1, \ldots, c_n)$ and $C' = (c'_1, \ldots, c'_n)$ generated by executions E and E' in a system of $n = f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ processes satisfying $f_\ell^r / f_\ell^{r,a} = f_\ell^s / f_\ell^{s,a}$, where the states $c_1 = c'_1, \ldots, c_{f_\ell^s + f_\ell^{s,a}} = c'_{f_\ell^s + f_\ell^{s,a}}$ of $f_\ell^s + f_\ell^{s,a}$ processes are the same. Then, E and E' can be feasibly extended by one round, such that the same $f_\ell^s + f_\ell^{s,a}$ processes again have the same states $d_1 = d'_1, \ldots, d_{f_\ell^s + f_\ell^{s,a}} = d'_{f_\ell^s + f_\ell^{s,a}}$ in the resulting successor configurations $D = (d_1, \ldots, d_n)$ and $D' = (d'_1, \ldots, d'_n)$.*

Proof. Let $\mathcal{Q} = \{q_0, \ldots, q_{f_\ell^s + f_\ell^{s,a} - 1}\}$ be the set of processes with equal states in C and C', and let $\mathcal{P} = \{p_0, \ldots, p_{f_\ell^r + f_\ell^{r,a} - 1}\}$ be the set of the remaining processes with possibly different states in C and C'. We claim that there is a feasible link failure pattern $\mathcal{F}$ extending E by one round, yielding the execution $E \cup (\mathcal{F}, D)$, where $\cup$

denotes the concatenation operation. Therefore, every process in $\mathcal{Q}$ gets exactly $f_\ell^{r,a}$ arbitrary link failures from some processes in $\mathcal{P}$, delivering the message that would have been sent in the failure-free extension of E' (i.e., in the absence of link failures). In addition, every process in $\mathcal{Q}$ also experiences exactly $f_\ell^r - f_\ell^{r,a}$ omission link failures from some processes in $\mathcal{P}$, whereas the messages from the remaining $f_\ell^{r,a}$ processes in $\mathcal{P}$ are received correctly. All messages from processes in $\mathcal{Q}$ to processes in $\mathcal{Q}$, as well as all messages to processes in $\mathcal{P}$, are failure-free.

Not surprisingly, the required link failure pattern has already been established in Lemma 4: We just have to map $r = f_\ell^r$, $r^a = f_\ell^{r,a}$, $s = f_\ell^s$, and $s^a = f_\ell^{s,a}$ and interpret white, orange, and purple balls as correct, omission faulty, and arbitrary faulty transmissions from the $f_\ell^r + f_\ell^{r,a}$ processes in $\mathcal{P}$ (columns) to the $f_\ell^s + f_\ell^{s,a}$ processes in $\mathcal{Q}$ (rows). The results of Lemma 4 reveal that the corresponding link failure pattern respects both the maximum numbers of send and receive link failures.

Knowing that such an $\mathcal{F}$ indeed exists, our lemma follows from extending E with $\mathcal{F}$ and E' with the pattern matrix $\mathcal{F}'$, which is exactly $\mathcal{F}$ except that a process that committed an arbitrary send link failure in $\mathcal{F}$ transmits correctly in $\mathcal{F}'$, whereas a process that transmitted correctly in $\mathcal{F}$ commits an arbitrary send link failure in $\mathcal{F}'$, which (erroneously) delivers the message that would have been transmitted in the failure-free extension of E: After this round, every process in $\mathcal{Q}$ has the same view of the execution both in $E \cup (\mathcal{F}, D)$ and $E' \cup (\mathcal{F}', D')$ and hence reaches the same configuration in D and D' as asserted. □

Now it is not difficult to prove our general lower bound result as given by Theorem 6. It reveals that $n > f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ is required for solving consensus if (4.1) holds. According to Corollary 1 in section 6, this bound is tight and is, for example, matched by the exponential algorithm OMH [66, 65]. If (4.1) does not hold, consensus can be solved with fewer processes, namely, with $n > \overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$ ones. The quantities $\overline{f}_\ell^r$, $\overline{f}_\ell^{ra}$, $\overline{f}_\ell^s$, and $\overline{f}_\ell^{sa}$ are nonnegative integers solving (4.1), subject to the constraints $\overline{f}_\ell^r \le f_\ell^r$, $\overline{f}_\ell^{ra} \le f_\ell^{r,a}$, $\overline{f}_\ell^s \le f_\ell^s$, and $\overline{f}_\ell^{sa} \le f_\ell^{s,a}$, such that $\overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$ is maximal. Note that this is in accordance with the observation that, if (4.1) does not hold, OMH solves consensus also for $n = f_\ell^r + f_\ell^{r,a} + f_\ell^s + f_\ell^{s,a}$ (if it is allowed to execute some additional rounds). Although we do not know whether the lower bound given by Theorem 6 is also tight in this case, we nevertheless conjecture that this is the case.

THEOREM 6 (lower bound processes 3). *Any deterministic algorithm that solves consensus under our system model needs* $n > \overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$, *where* $\overline{f}_\ell^r \le f_\ell^r$, $\overline{f}_\ell^{ra} \le f_\ell^{r,a}$, $\overline{f}_\ell^s \le f_\ell^s$, $\overline{f}_\ell^{sa} \le f_\ell^{s,a}$ *are such that* $\overline{f}_\ell^r / \overline{f}_\ell^{ra} = \overline{f}_\ell^s / \overline{f}_\ell^{sa}$ *holds and the sum* $\overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$ *is maximal.*

Proof. Due to Theorem 3, it remains to provide an impossibility proof for $f_\ell^{s,a}, f_\ell^{r,a} > 0$. According to Lemma 3, we just have to show that every process p can withhold its information under the conditions of our theorem: More specifically, for every $k \ge 0$, we need a failure pattern for rounds $k+1, k+2, \ldots$ such that a nonempty set of processes $\mathcal{Q} = \mathcal{Q}(p)$ has the same view of the resulting execution after round $r \ge k+1$, independent of the information p has gathered by round r.

This follows easily from inductively applying Lemma 5, however: If we assume that p is just one of the $f_\ell^r + f_\ell^{r,a}$ processes in $\mathcal{P}$ that may have a different state in two k-round executions E (resp., E') leading to configurations C (resp., C'), we are guaranteed that the remaining $|\mathcal{Q}| = f_\ell^s + f_\ell^{s,a}$ processes that had identical states in E and E' have identical states in some 1-round extension $E \cup (\mathcal{F}, D)$ and $E' \cup (\mathcal{F}', D')$ of E and E' again. Hence, no such process ever gets information from p. Since this

can go on for an arbitrary number of rounds, Definition 1 reveals that every p can indeed withhold its information as required. □

5. Number of rounds. In this section, we will show that being able to handle link failures comes at the price of additional running time. More specifically, compared to the case without link failures, solving consensus in case of $f_\ell^s, f_\ell^r > 0$ requires one additional round. Our proofs are again based on bivalency arguments and reuse some of the results developed in the previous sections.

THEOREM 7 (lower bound rounds 1). *Any deterministic algorithm that solves consensus under our system model for $f_\ell^s, f_\ell^r > 0$ needs at least two rounds.*

Proof. Assume that there is a 1-round algorithm that solves consensus in the presence of link failures. Obviously, since any process p may suffer from a send link failure to any receiver process q, any process p can withhold its information from at least one process $q(p)$ here: The 1-round assumption does not allow other processes to learn about p's information in some later round. Therefore, Lemma 3 reveals that solving consensus is impossible here. Note that Lemma 3 is applicable here, since x-round consensus is an instance of consensus where no messages are sent in rounds $> x$. So if no consensus algorithms exists, as guaranteed by Lemma 3, x-round consensus is impossible also. □

The above result can be extended to the case where both process and link failures can occur. Using our ideas in the simple bivalency proof of the well-known $f + 1$ lower bound for the number of rounds required for solving consensus in the presence of f process crashes developed in [6], it is possible to show that $f + 2$ is a tight lower bound (matched, for example, by algorithm OMH of [66, 65] and also confirmed by Corollary 1 for $f = 0$).

THEOREM 8 (lower bound rounds 2). *Any deterministic algorithm that solves consensus under our system model with $n \geq f+f_\ell^s+f_\ell^r$, where f denotes the maximum number of process crash failures and $f_\ell^s, f_\ell^r > 0$, needs at least $f + 2$ rounds in the worst case.*

Proof. In [6], a simple forward induction based on a bivalency argument involving message losses due to process crashes is used to show that any consensus algorithm has executions that lead to at least one bivalent configuration at the end of round $f - 1$. The executions considered in this proof are such that at most one process may crash in every round; clearly, no link failures are assumed to occur here. The impossibility of consensus within f rounds follows by contradiction: It is shown in [6, Lemma 1] that the existence of such a solution would imply that all configurations reached after $f - 1$ rounds must be univalent.

In order to prove Theorem 8, we have only to provide an analogue to [6, Lemma 1]: The existence of a consensus algorithm that decides after $f+1$ rounds in our failure model would imply that all configurations reached after $f - 1$ rounds are univalent. The proof is by contradiction: Assuming that not all configurations reached after $f-1$ rounds are univalent, there must be a bivalent configuration C^{f-1} after round $f - 1$. Since at most one process may have crashed during each of the first $f - 1$ rounds, there is still one process p that may crash in round f or $f + 1$. Note that it is the crash of this process p and/or the occurrence of link failures in round f or $f + 1$ that "allows" C^{f-1} to be bivalent.

Let v be the algorithm's decision in the execution E, where no failure (i.e., neither a crash of p nor any link failure) occurs in the two rounds following C^{f-1}. Due to the bivalence of C^{f-1}, there must be a different execution $\overline{E}$ also starting from C^{f-1}, where the decision is $1 - v$. Assume first that p crashes in round f in $\overline{E}$. Then, there

must be two executions, E^q leading to the decision value v, and $\overline{E}^q$ leading to $1-v$, which differ only in that the (crashing) p sends its round f message to q in E^q but not in $\overline{E}^q$: Starting from E where p sends all its messages, we remove the messages p succeeds in sending one by one until the decision value changes; this happens at the latest when we arrive at the execution $\overline{E}^q$.

By construction, q is the only process that can distinguish between E^q and $\overline{E}^q$ after round f. If we allow q to produce a send link failure to some other correct process r (this process must exist since $n \geq f+2$) in the final round $f+1$, then r has the same view at the end of E^q and $\overline{E}^q$. Hence, the resulting decisions cannot be different, providing the required contradiction of some C^{f-1} being bivalent in this case.

We still have to deal with the case where p does not crash in round f in $\overline{E}$. Then, we claim that there is some bivalent configuration C^f reachable from C^{f-1} in round f. For the sake of contradiction, assume that all configurations reachable from C^{f-1} in a single round (where no process crashes) are univalent. Since C^{f-1} is bivalent, the configuration C^f reached by the link-failure-free single-round extension of C^{f-1} must be v-valent, whereas some configuration $\overline{C}^f$ reached by another single-round extension with link failures must be $(1-v)$-valent. Due to Lemma 2, there are two neighboring configurations C_q^f and $\overline{C}_q^f$ that are also v-valent and $(1-v)$-valent, respectively. Those configurations differ only in a single link failure from some sender $s \to q$ in round f, perceived by q in $\overline{C}_q^f$ but not in C_q^f.

Again, q is the only process that can distinguish between the resulting executions E^q and $\overline{E}^q$ after round f. If we allow q to produce a send link failure to some other correct process r (this process must exist since $n \geq f+2$) in the final round $f+1$, then r has the same view at the end of E^q and $\overline{E}^q$. Therefore, the resulting decisions cannot be different, contradicting the stipulated univalence of all C^f. Hence, there is indeed some reachable bivalent configuration C^f at the end of round f.

Since we still have a processor p to crash in round $f+1$ in this case, however, the original proof [6, Lemma 1] applies: In order to decide at the end of round $f+1$, all configurations at the end of round f must be univalent. Since C^f is bivalent, we have established the required contradiction also in this case. □

As a concluding remark, we note that the additional round required for solving consensus in the presence of link failures is not a new result. For $f=0$, it has been shown in [45] that two rounds are needed for solving consensus. Interestingly, the general case follows also from the general result of [25] on indulgent consensus algorithms. More specifically, link omission failures can be interpreted as false suspicions of a local failure detector module. Our algorithms tolerate such link failures and hence must be indulgent with respect to their "failure detectors" (= message reception). Note that our constraints (A^s) and (A^r) also ensure termination of such algorithms.

6. Other results. In this section, we will elaborate on some consequences of our model and the results obtained so far. We start with some considerations related to connectivity in the underlying communication graph in our model, which can be used for confirming the tightness of our lower bounds $n > f_\ell^s + f_\ell^r$ and $n > f_\ell^s + f_\ell^{s,a} + f_\ell^r + f_\ell^{r,a}$.

Consider the single-round communication graph G for some round k. It consists of n vertices corresponding to the processes in Π and contains a directed edge (p,q) iff there is no link failure on the link connecting $p \to q$ in round k. Recall from elementary graph theory that a graph G is *at least c-connected* if it remains connected when at

most $c-1$ vertices and their adjacent edges are removed. Two paths connecting processes p and q are called *process-disjoint* iff they do not have common processes except p and q.

THEOREM 9 (connectivity). *Every single-round communication graph G of a system of $n > f_\ell^s + f_\ell^r$ processes complying to our system model is at least c-connected with $c = n - f_\ell^s - f_\ell^r > 0$. In fact, every pair of processes p, q is connected by c process-disjoint paths consisting of at most two nonfaulty links.*

Proof. Since at least c-connectivity follows trivially if every pair of processes is connected by c process-disjoint paths, it suffices to show the latter: From (A^s), we know that p is connected to at least $n - f_\ell^s$ processes (possibly including itself) via nonfaulty links. From (A^r), it follows that q is connected to a set of at least $n - f_\ell^s - f_\ell^r = c$ of these processes via nonfaulty links. Let $\mathcal{I}$ with $|\mathcal{I}| \geq c$ be this set of processes. If $p \notin \mathcal{I}$ and $q \notin \mathcal{I}$ (i.e., if p and q are not adjacent), then p and q are connected by c paths consisting of two nonfaulty links routed over the processes in $\mathcal{I}$. Otherwise, there are only $c-1$ paths of length 2 and a direct path from p to q, which are of course also process-disjoint. ∎

Theorem 9 implies that one can build a 2-round simulation of reliable communication in our model by using the well-known echo broadcasting scheme [15] ("crusader's agreement" [23]): p sends (msg, p) to all in the first round, and every q rebroadcasts (msg, p) in the second round. One can easily show that this broadcasting scheme allows every q to deliver (msg, p) correctly if $c = n - f_\ell^s - f_\ell^r > 0$. Interestingly, as proved in [11, 12], this simulation even works in case of arbitrary link failures if n is sufficiently large.

COROLLARY 1 (reliable link simulation [11, Thm.2]). *There is a 2-round simulation, which implements reliable broadcasting in our link failure model if $n - f_\ell^s - f_\ell^{s,a} - f_\ell^r - f_\ell^{r,a} > 0$.*

Any synchronous consensus algorithm resilient to f classic process failures can hence be used in conjunction with this simulation for solving consensus in the hybrid failure model of [66, 65] (and therefore, trivially, in the model of section 2). Note, however, that this simulation doubles the number of rounds and is hence suboptimal: Theorem 8 revealed a lower bound of $f+2$ rounds for solving consensus in our model and the algorithms provided in [65] confirm that this bound is tight.

Certain consequences of the results of the previous sections also shed some light on classic process failures. After all, the effect of an omission (resp., arbitrary) faulty process on its peers is principally the same as the effect of omission (resp., arbitrary) send link failures (A^s) committed by a nonfaulty process: Some receiving processes inconsistently get no (resp., erroneous) messages instead of the correct ones. So, we question why f omission faulty processes require at least $f+1$ rounds of execution (Theorem 8), whereas any number of processes committing send link failures can be handled in just 2 rounds according to Corollary 1; cf. the exponential algorithm OMH [65, Thm. 5.4], for example.

From our link failure model, it is apparent that arbitrary send link failures (A^s) can also be viewed as the consequence of a *restricted process failure* with inconsistency limited to f_ℓ^s. Since $f_\ell^s < n$, however, the inconsistency caused by send link failures is strictly less than that of an arbitrary faulty process, since the latter is not restricted in the number of recipients that may get an erroneous message. If at most $f_\ell^r = f_\ell^{r,a}$ processes suffer from restricted process failures with inconsistency limited to $f_\ell^s = f_\ell^{s,a}$, both (A^s) and (A^r) are satisfied, which implies that this alternative interpretation leads to feasible link failure patterns in our model. Note carefully that (A^s) and (A^r)

admit more general failure patterns than just restricted process failures, however: A receive failure may hit any incoming link in the former but is restricted to one of the links from the f_ℓ^r restricted faulty processes in the latter.

Anyway, using the alternative interpretation of arbitrary link failures as restricted arbitrary process failures, the result of Theorem 6 allows us to characterize what makes a process failure a truly arbitrary (Byzantine) one: Choosing $f_\ell^r = f_\ell^{r,a} > 0$ arbitrary and fixing $n > 2f_\ell^{r,a}$, an optimal consensus algorithm such as OMH solves consensus with $n \geq 2f_\ell^{s,a} + 2f_\ell^{r,a} + 1$ in only two rounds. It hence tolerates $f_\ell^{r,a}$ restricted arbitrarily process failures with inconsistency limited to

$$f_\ell^{s,a} \leq \lfloor (n-1)/2 \rfloor - f_\ell^{r,a}. \tag{6.1}$$

For example, $n = 9$ processes are required for tolerating two restricted arbitrary failures with inconsistency $f_\ell^{s,a} = 2$ in just two (instead of three) rounds. Due to (6.1), at least $\lceil (n-1)/2 \rceil + f_\ell^{r,a}$ processes, i.e., a majority[4] of the nonfaulty processes, get the correct message, even in the broadcast of a restricted arbitrary faulty process. Provided that n is chosen appropriately, *any* number $f_\ell^{r,a}$ of process failures with inconsistency limited to $f_\ell^{s,a}$ can hence be tolerated in just two rounds here, i.e., in a number of rounds that does not depend on the number of failures $f_\ell^{r,a}$. If, on the other hand, more than $f_\ell^{s,a}$, i.e., more than a minority of the nonfaulty processes, can get a faulty message in the broadcast of a process, then the sender must be considered arbitrary faulty and thus increases the number of rounds required for solving consensus.

A similar observation can be made for omission failures. Choosing $f_\ell^{s,a} = f_\ell^{r,a} = 0$, and $f_\ell^r > 0$ arbitrary, and fixing $n > f_\ell^r$, an optimal consensus algorithm such as ZA [65] solves consensus for $n \geq f_\ell^r + f_\ell^s + 1$ in only two rounds. It hence tolerates f_ℓ^r restricted omission process failures with inconsistency

$$f_\ell^s \leq n - 1 - f_\ell^r. \tag{6.2}$$

It follows from (6.2) that at least $f_\ell^r + 1$ processes, i.e., at least one nonfaulty process, must get the correct message, even in the broadcast of a restricted omission faulty process. If this is warranted, any number f_ℓ^r of such restricted process failures can be tolerated within two rounds.

It hence follows that a process that disseminates inconsistent information cannot do much harm—in the sense of requiring additional rounds for solving consensus—if at most a certain number of recipients can get inconsistent information:

- A process must be considered arbitrary faulty if it can supply erroneous information to a majority of the nonfaulty processes.
- A process must be considered omission faulty if it can fail to provide information to any and all nonfaulty processes.

For suboptimal algorithms, the thresholds (numbers of affected receivers) are of course smaller.

Note, finally, that our observations do not contradict the lower bound $f+1$ for the number of rounds required for consensus in the presence of f faulty processes (recall Theorem 8 or [8, Sec. 5.1.4]), since this result relies heavily on the fact that a faulty process can disseminate inconsistent information to as many processes as desired.

[4]For a suboptimal algorithm, even more than a majority of the nonfaulty processes must get the correct message here.

7. Relation to other models. In this section, we will relate our model to alternative link failure modeling approaches. Particular emphasis will be put on the issue of assumption coverage, which will be analyzed and compared in detail in a simple probabilistic setting.

7.1. Overview of related approaches. It has long been known that consensus is impossible with arbitrarily lossy links [33]. Therefore, every useful failure model must restrict link failures in some way. Some previous work has considered links that eventually become reliable "for sufficiently long," at least among a majority of the processes, (e.g., [46, 24]), or "stubborn links" [36] that eventually deliver every message provided the message is sent sufficiently many times. These models in essence provide safety despite link failures but require communication to eventually become reliable in order to ensure liveness. A similar approach is employed in the crash-recovery model [2], which also deals with transient failures, albeit at the level of whole processes and on a much larger time-scale. There are only a few failure models for synchronous systems in the literature that deal with transient link failures that continue to occur indefinitely.

One straightforward way to deal with link failures is to map link failures to sender process failures [32] or, preferably, to general send/receive-omission failures [54]. Unfortunately, this approach suffers from poor model coverage (see section 7.2) and is also quite restrictive in the sense that only specific processes—the faulty ones—may experience link failures.

Another class of models [55, 61, 67] considers a small number of link failures explicitly: Those papers assume that at most $O(n)$ links may be faulty system-wide during the entire execution. Obviously, such models can be applied only when link failures are rare. Hadzilacos [37] presents a theoretical study of connectivity requirements for solving consensus in arbitrary networks with stopping and omission failures.

The most severe problem of the models surveyed so far is their inability to deal with the "moving" nature of transient link failures: In a real network, there is a positive probability for message loss (or delay) on every link. In the aforementioned models, once a single message is lost, either the link or the process is deemed faulty. Since failures are considered persistent during an execution in the above models, the "exhaustion" of nonfaulty processes and links progresses rapidly with every round. This makes any attempt to solve consensus in models such as those presented in [32, 54, 37] void in case of significant link failure rates (see section 7.2 for a detailed analysis). A more adequate approach to capturing message loss is to allow for transient failures that hit different processes or links in different communication rounds.

Santoro and Widmayer were the first to introduce this assumption: In [58, 60], they showed that consensus cannot be solved in the presence of $n-1$ (resp., $\lfloor n/2 \rfloor$) omission (resp., Byzantine) link failures per round, particularly if those link failures hit the same sender process. As a consequence, consensus cannot be solved in the presence of just a single *mobile* omission or Byzantine faulty process, i.e., a single process—which may be different in different rounds—that suffers from omission or even Byzantine failures. This result has been proven in [58] by means of a similar approach as employed in section 3 and reproven in the layering framework by Moses and Rajsbaum [52].

On the other hand, if the number of moving link failures is further restricted to less than $n-1$ (resp., $\lfloor n/2 \rfloor$) per round in case of omission (resp., Byzantine) link failures, consensus can be solved in a constant number of rounds [59, 49]. Other distributed computing problems [21] and special system architectures [22] have also

been studied under this model.

The failure model introduced in [56] can be seen as a first step in the direction of increasing the link failure resilience from $O(n)$ to $O(n^2)$: For a system with $n \geq 20f+1$ processes, at most f of which may be Byzantine faulty per round, a consensus algorithm was given that tolerates a small number $l = n/20$ of link failures at every node. A different (but related) model has been proposed in [31], which considers at most f Byzantine process failures per round that may move from one process to another with a certain maximum speed.

Cristian et al. [20] provide a suite of synchronous atomic broadcast protocols with much better link failure resilience (which is comparable to results we presented in an earlier paper [65]). Although atomic broadcast is usually investigated in a more communications-oriented context, it can be used to solve consensus as well; see, e.g., [38] for an overview. The three algorithms of Cristian et al. [20] tolerate an arbitrary number of processes with omission, timing, or Byzantine failures (if authentication is available) and work on general communication graphs subject to link failures. Instead of making the number of link failures explicit, however, it is just assumed that any two processes in the system are always connected via a path of nonfaulty links.

Unlike in the deterministic setting, link failures are easily tolerated by *randomized* consensus algorithms such as the one of [70]. Such algorithms circumvent Gray's impossibility result (cf. Theorem 1) by adding nondeterminism (coin tossing) to the computations. Still, due to the inherent nonzero probability of failure/nontermination within a fixed number of rounds, randomized algorithms are unsuitable for some applications. Moreover, there is a lower bound $1/(R+1)$ for the probability of disagreement after R rounds with arbitrary loss [50, Thm. 5.5]. It is interesting to note, however, that our link failure modeling approach also circumvents this lower bound: For a well-known randomized algorithm, Schmid and Fetzer established a probability of disagreement of only $(1/2)^R$ [64].

Though our paper primarily deals with message omissions in the synchronous timing model, our model can also be used to reason about *timeliness* of some links in otherwise asynchronous round-based systems (where late messages are discarded). In this context, our threshold f_ℓ^r (resp., f_ℓ^s) can be seen as a restriction on the number of late (or untimely) messages a process receives (resp., sends) in a round. Restrictions of this type have received much attention recently: A suite of papers [3, 4, 5, 53, 7, 51, 43, 39, 40, 44] provided weaker and weaker models that are still sufficiently strong for implementing failure detectors and/or solving consensus in the presence of at most f process crash failures and dynamic timing variations.

In particular, Keidar and Shraer introduced a model called *all from majority* ($\diamond$AFM) in their round-by-round GIRAF framework [43], which is closely related to the moving link failure model introduced in section 2.2: It allows $O(n^2)$ links per round to be nontimely, provided that every process has at least $m+1$ timely outgoing links and $n-m$ timely incoming links at any time for some suitable m. The set of timely links may be moving. $\diamond$AFM was shown in [44] to be the only model (out of those defined thus far in this context) that scales with n, in the sense that consensus can be guaranteed to terminate in a constant expected number of rounds in the independent identically distributed probabilistic link failure model, even for $n \to \infty$.

Our model is also related to the *heard-of model* (HO Model) developed by Charron-Bost and Schiper [18, 19]. The HO Model is a round-based distributed computing model, which unifies synchrony and (benign) failures of both processes and links. It has recently been extended to Byzantine failures [13] as well. Our link failure restric-

tions (A^s) and (A^r) can be elegantly expressed as simple communication predicates in the HO model, namely, $\forall p \in \Pi, k \geq 1 : |HO(p,k)| \geq n - f_\ell^r \wedge |TT(p,k)| \geq n - f_\ell^s$ in case of omission link failures, where $HO(p,k)$ is the set of processes p hears from in round k and $TT(p,k)$ is the set of processes p talks to in round k. Moreover, the reliable link simulation that led to Corollary 1 can be seen as a 2-round translation that simulates a global kernel (of size n, i.e., a failure-free system) in our model. It is also interesting to compare the upper bound results of [13] with our lower bounds: Theorem 5 reveals that we can allow at most $n/4$ arbitrary link failures per round. The algorithm $\mathcal{A}_{T,E}$ in [13] admits up to $n/4$ arbitrary receive link failures per round, without posing a restriction to send link failures, however. There is no contradiction here, due to the fact that the analysis of $\mathcal{A}_{T,E}$ separates safety and liveness: The algorithm actually needs some rounds with much less than $n/4$ link failures for guaranteed termination. By contrast, our lower bounds guarantee both safety and liveness simultaneously.

Finally, we already noted that, in the context of round-by-round fault detectors [30], false suspicions of a local failure detector [16] can also be interpreted as transient link failures. Our results, such as the lower bound of $f+2$ rounds for solving consensus, are hence also applicable to stable periods [26] and stable runs [41, 25] of indulgent [35] consensus algorithms.

An alternative way to cope with transient link failures involves reliable link simulation protocols based on retransmissions [1, 9, 2]. Asynchronous algorithms can then be used atop such protocols for solving consensus. However, since retransmission protocols can obviously mask omissions only (but not timing failures and/or erroneous messages), they are no panacea. Moreover, using time redundancy for tolerating link failures necessarily increases the end-to-end delay in case of a failure, which eventually affects the consensus algorithms' termination time. And, last but not least, since it is impossible to solve consensus in asynchronous systems with even a single crash failure [29], one has to add some synchrony to the system anyway [3, 4, 5, 53, 7, 51, 40, 43, 39]. This makes our synchronous lower bound results applicable again, at least to asynchronous algorithms designed for round-by-round-based frameworks such as [64, 18, 19, 43]. A detailed survey of link failures in partially synchronous and asynchronous systems can be found in [63].

Note that using reliable link protocols in conjunction with *synchronous* consensus algorithms is not particularly useful, since the duration of the rounds must be fixed a priori. As a consequence, only a certain number of retransmissions can be accomodated in a round, which is not sufficient for simulating reliable links in the presence of high link failure rates. In sharp contrast, our approach toward handling link failures uses additional processes (i.e., some larger value of n) instead of retransmissions and therefore does not suffer from this problem: The duration of a round just needs to encompass a single end-to-end delay.

7.2. Model coverage. In this section, we will prove that our link failure model also surpasses alternative approaches in terms of assumption coverage.

If link failures are just mapped to sender process failures, as in [32], even a single link failure per process and round ($f_\ell = 1$ in our terminology) would end up with $f = n$ faulty processes in some runs. Figure 7.1 shows an example for $n = 4$ and $f_\ell = 1$.

In the more elaborate send/receive-omission failure model of [54], an omission can be attributed to either the sender or the receiver process. Still, in the example of Figure 7.1, only at most two processes can be considered correct. Ending up with

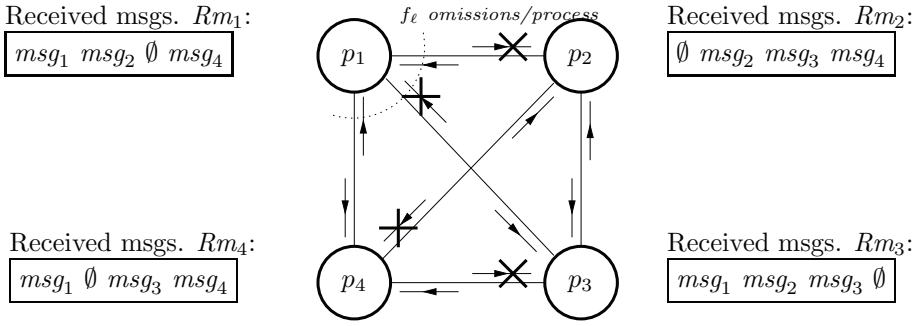


FIG. 7.1. *Example of a 4-process system with $f_\ell = 1$ send and receive omission failures per process in each round, where all processes must be considered faulty in traditional process failure models.*

less than a majority of correct processes renders uniform consensus unsolvable [17], however. Hence, by attributing link failures to processes, we may miss the opportunity to solve consensus in scenarios which can be handled in our model.

We now turn to examine whether the additional scenarios captured by our model are significant. After all, one could argue that failure patterns as depicted above almost never occur in practice, such that more refined models have only marginal added value. We counter this argument by quantifying the *coverage*[5] of various models, using a simple probabilistic "benchmarking scenario."

DEFINITION 2. *Consider a synchronous system of n processes, where each of the $n(n-1)$ unidirectional links fails with some independent probability $0 \leq p < 1$, independently in every round. The* coverage *$Cov(M)$ of a model M is the probability that the model assumptions hold true during the execution of an m-round algorithm for some arbitrary $m \geq 1$.*

Combining ideas from [62] and [44], we analyze the coverage of the following failure models:

- f general omission-faulty processes (GO_f) [54],
- at least one process with f nonmoving timely links (TL_f) [5],
- at most $n-2$ moving link failures per round (ML_{n-2}) [58],
- f_ℓ moving link failures per process and round (MLO_{f_ℓ}), the model of section 2.2.

It will turn out that the link failure model introduced in section 2.2 surpasses all other modeling approaches above in terms of coverage. In particular, it is the only model that scales with n, in the sense that, for example, $\mathrm{Cov}(MLO_{n/2}) \to 1$ for $n \to \infty$. By contrast, the coverage of all the other models even goes to 0 for $n \to \infty$ in comparable settings.

We should mention, though, that our coverage analysis does not aim at replacing a direct analysis of a distributed algorithm in our probabilistic "benchmarking scenario." A particular algorithm $\mathcal{A}$ may perform much better than our coverage analysis predicts, since $\mathcal{A}$ may also work well in executions where the deterministic model M is violated. Hence, $\mathrm{Cov}(M)$ is just a lower bound on the achievable performance of $\mathcal{A}$ but is of course a meaningful measure for assessing the quality of a deterministic

[5] We note that the term coverage suffers from overloading in the literature; throughout this paper, "coverage" must be read as "model coverage in synchronous systems with independent link failure probability p."

model independently of a particular protocol.

Analysis of f general omission-faulty processes (GO_f). Under this failure model, there must be a set K of at least $k = n - f$ processes that never commit a send nor a receive omission. This requirement is mapped to our probabilistic link failure setting as follows: All the links among the processes in K must be correct in all rounds, and the links connecting processes in $\Pi\backslash K$ with processes in $\Pi\backslash K$ may be either correct or faulty. The links to/from processes in K from/to processes in $\Pi\backslash K$ may also be either correct or faulty, since we can attribute a link failure to the adjacent process in $\Pi\backslash K$.

So let K be a subset of $k = n - f \geq 1$ distinct processes where all $k(k-1)$ links among processes in K never experience any omission failure during m rounds. Since there are $\binom{n}{k}$ different sets K of k processes out of n processes, the probability $P_n(k)$ that there is at least one such set in a run satisfies

$$P_n(k) \leq \binom{n}{k}(1-p)^{k(k-1)m}. \tag{7.1}$$

Note that $P_n(k)$ is not equal to $\binom{n}{k}(1-p)^{k(k-1)m}$, since there may be multiple sets K in a given run (recall that the links outside K can also be correct).

Hence, the model coverage of the standard general omission failure model with at most f faulty processes GO_f satisfies

$$\mathrm{Cov}(GO_f) = P_n(n-f) \leq \binom{n}{n-f}(1-p)^{(n-f)(n-f-1)m}. \tag{7.2}$$

In case of $f = \lambda n$ for any $0 < \lambda < 1$, it is not difficult to prove that $\mathrm{Cov}(GO_{\lambda n}) \to 0$ for $n \to \infty$. We will use asymptotic analysis for this purpose,[6] with Stirling's formula

$$n! \sim \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \qquad \text{for } n \to \infty \tag{7.3}$$

as our major ingredient.

LEMMA 6 (asymptotic expansion $\binom{n}{n\lambda}$). *For any $0 < \lambda < 1$ and $n \to \infty$,*

$$\binom{n}{\lambda n} \sim \frac{1}{\sqrt{2\pi n\lambda(1-\lambda)}} \cdot \left((1-\lambda)^{-(1-\lambda)} \cdot \lambda^{-\lambda}\right)^n. \tag{7.4}$$

Proof. Using Stirling's formula in $\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$, we find

$$\begin{aligned}
\binom{n}{\lambda n} &\sim \frac{\left(\frac{n}{e}\right)^n}{\left(\frac{n(1-\lambda)}{e}\right)^{n(1-\lambda)}\left(\frac{\lambda n}{e}\right)^{\lambda n}} \cdot \frac{\sqrt{2\pi n}}{\sqrt{2\pi n(1-\lambda)} \cdot \sqrt{2\pi\lambda n}} \\
&\sim \left(\frac{n}{n(1-\lambda)}\right)^n \cdot \left(\frac{n(1-\lambda)}{\lambda n}\right)^{\lambda n} \cdot \frac{1}{\sqrt{1-\lambda} \cdot \sqrt{2\pi\lambda n}} \\
&\sim (1-\lambda)^{-n} \cdot \left(\frac{1-\lambda}{\lambda}\right)^{\lambda n} \cdot \frac{1}{\sqrt{2\pi n\lambda(1-\lambda)}},
\end{aligned}$$

from which (7.4) follows immediately. ∎

[6] We use the notation $f(n) \sim g(n) \Leftrightarrow \lim_{n\to\infty} f(n)/g(n) = 1$ and $f(n) \propto g(n) \Leftrightarrow 0 \leq \lim_{n\to\infty} f(n)/g(n) \leq 1$.

Using the result of Lemma 6 in (7.2), the coverage of $GO_{n\lambda}$ evaluates to

$$\text{Cov}(GO_{n\lambda}) \propto \frac{\left((1-\lambda)^{-(1-\lambda)} \cdot \lambda^{-\lambda}\right)^n}{\sqrt{2\pi n\lambda(1-\lambda)}} \cdot (1-p)^{\left(n(1-\lambda)\right)\left(n(1-\lambda)-1\right)m}. \tag{7.5}$$

Since the exponent of $1-p<1$ is quadratic in n, it is obvious that $\text{Cov}(GO_{n\lambda}) \to 0$ for $n \to \infty$, for any $p>0$, $0<\lambda<1$, and $m \geq 1$. In particular, for $f = n/2$, we obtain

$$\text{Cov}(GO_{n/2}) \propto \frac{2^n}{\sqrt{\pi n/2}}(1-p)^{(n/2)(n/2-1)m}. \tag{7.6}$$

$\text{Cov}(GO_{n/2})$ hence very quickly goes to 0 for $n \to \infty$ for any $0<p<1$ and any $m \geq 1$. Figure 7.2 gives some numerical values which reveal that the coverage of the general omission process failure model in our benchmarking scenario is indeed very poor. For example, for $p=0.01$ and $m=2$ (first plot in the left figure), the coverage is only about 10^{-25} in a system of $n=40$ processes (log-scale)!

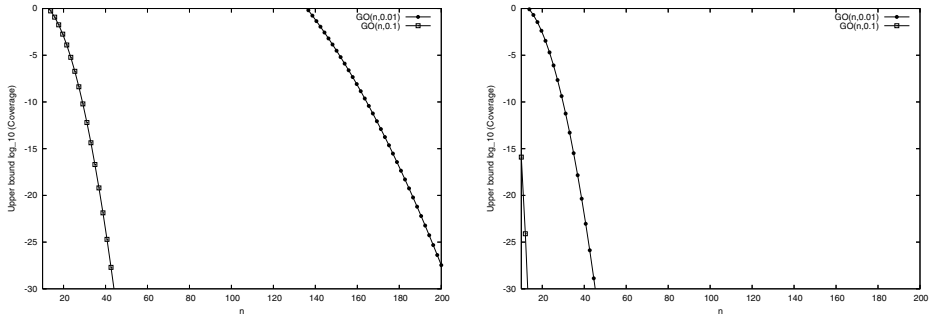


FIG. 7.2. *Coverage general process omission failure model: Upper bound on* $\log_{10}(Cov(GO_{n/2}))$ *over* n *with* $p=0.01$ *and* $p=0.1$ *for* $m=2$ *(left) and* $m=20$ *(right).*

Analysis of at least one process with f nonmoving timely links (TL_f). We now turn our attention to the "nonmoving" link timing models [3, 4, 5] used for solving Ω and consensus in an (almost) asynchronous system of n processes with up to f process crash failures and eventually reliable links. The weakest model among those (denoted TL_f here) assumes that (eventually) there is at least one process p with at least f timely outgoing links in each of its broadcasts. Note that those f links are fixed throughout the (suffix of the) execution.

We note that the model TL_f is actually too weak for solving consensus within *bounded* time: As shown in [43, 44], considerably more timely links are required to solve consensus within a bounded number m of (timely) rounds. We incorporate the analysis of TL_f here, however, since it provides sort of an "upper bound" with respect to coverage: Any model that, in addition to TL_f, requires additional timely links must have an even lower coverage in our benchmarking scenario. Bear in mind, however, that the number of rounds m should be considered large here.

With p representing the probability that a link is nontimely in a round here, the probability $P_{n-1}(f) = Cov(TL_f)$ that some process has at least f timely links during m rounds is at most $n\binom{n-1}{f}(1-p)^{fm}$: We have n processes, and there are $\binom{n-1}{f}$ different subsets of f processes among the $n-1$ neighbors of a process; $(1-p)^{fm}$

gives the probability that the links to a fixed set of f neighbors are correct during all m rounds. Hence,

$$\mathrm{Cov}(TL_f) \le n \cdot \binom{n-1}{f}(1-p)^{fm}. \tag{7.7}$$

As in the analysis of GO_f, this is only a (quite conservative) upper bound for $P_{n-1}(f)$, however, since the involved events are not independent.

For $f = \lambda(n-1) = \lambda n'$, $0 < \lambda < 1$, where we employed the abbreviation

$$n' = n - 1$$

used throughout this section, we obtain

$$\begin{aligned}\mathrm{Cov}(TL_{\lambda n'}) &\le (n'+1) \cdot \binom{n'}{\lambda n'}(1-p)^{m\lambda n'} \\ &\propto \sqrt{\frac{n'}{2\pi\lambda(1-\lambda)}} \cdot \left((1-\lambda)^{-(1-\lambda)} \cdot \lambda^{-\lambda}\right)^{n'} \cdot (1-p)^{m\lambda n'}. \end{aligned} \tag{7.8}$$

Choosing $\lambda = 1/2$, i.e., $f = (n-1)/2 = n'/2$, to facilitate comparison with our other results, we obtain

$$\mathrm{Cov}(TL_{n'/2}) \propto \sqrt{\frac{2n'}{\pi}}\left(4(1-p)^m\right)^{n'/2}. \tag{7.9}$$

The above bound goes to 0 for $n' \to \infty$ if $4(1-p)^m < 1$, i.e., when the number of rounds satisfies

$$m > -\frac{\log 4}{\log(1-p)} > \frac{\log 4}{p}, \tag{7.10}$$

according to the series expansion $\log(1-x) = -\sum_{k\ge 1} x^k/k$, valid for $|x| < 1$. For smaller values of m, (7.9) increases exponentially. Since $\mathrm{Cov}(TL_{n'/2}) = P_{n'}(n'/2)$ is a probability and hence ≤ 1, however, the question arises whether the range of m where $\mathrm{Cov}(TL_{n'/2}) \to 0$ could be extended by a refined analysis.

Some advanced results on the distribution of the maximum degree of nodes in a geometric random graph [57] can be used for this purpose: The sought probability $P_{n-1}(f)$ is just the probability that the maximum degree Δ of the nodes in a random graph with n nodes (where an edge exists, independently of the other edges, with some fixed probability $0 < q < 1$) satisfies $\Delta \ge f$. More specifically, we have to consider the random graph with n nodes, corresponding to our processes, where an edge (x, y) exists if there is no link failure on the link $x \to y$ during m rounds. Clearly, the probability of the latter event is $q = (1-p)^m$. Note that [57] actually deals with undirected graphs. Considering the undirected random graph RG corresponding to an execution, instead of its directed counterpart $\overline{RG}$, provides an upper bound: Since every directed edge in $\overline{RG}$ is also present in RG, it follows that $\mathbf{P}\{\Delta_{RG} \ge f\} \ge \mathbf{P}\{\Delta_{\overline{RG}} \ge f\}$.

THEOREM 10 (maximum degree in geometric random graphs [57]). *Given a geometric random graph with n nodes and edge probability q, the maximum degree Δ is strongly concentrated, in the sense that almost always*

$$\left|\Delta - qn - \sqrt{2q(1-q)n\log n} + \log\log n\sqrt{\frac{q(1-q)n}{8\log n}}\right| \le \log\log\sqrt{\frac{n}{\log n}}. \tag{7.11}$$

Moreover, the tail satisfies $\mathbf{P}\{\Delta < qn + b\sqrt{nq(1-q)}\} = \big(c(b) + o(1)\big)^n$ *for* $n \to \infty$, *where* $c(b) < 1$ *is the root of a certain equation;* $c(0) = 0.6102$*; and* $c(b)$ *is independent of* q.

According to our exposition above, we must set $q := (1-p)^m$, $n := n'+1$, and $f = n'/2$. Now, if $n'/2 > nq = (n'+1)(1-p)^m$ with $n'/2-(n'-1)(1-p)^m \geq C \cdot n'$ for some constant $C > 0$, then the area $\Delta \geq n'/2$ is to the right of the area of concentration $\big[nq - O(\sqrt{n \log n}), nq + O(\sqrt{n \log n})\big]$ of Δ given in (7.11) if n is sufficiently large. As a consequence, $\mathrm{Cov}(TL_{n'/2}) = \mathbf{P}\{\Delta_{\overline{RG}} \geq n'/2\} \leq \mathbf{P}\{\Delta_{RG} \geq n'/2\}$ actually goes to 0 for $n \to \infty$ if $(1-p)^m < 1/2$, i.e., when $m > \frac{\log 2}{p} > \frac{\log 2}{p}$; cf. (7.10). For smaller values of m, $\mathrm{Cov}(TL_{n'/2}) \to 1$ for $n \to \infty$ (with a fast transition phase in between, as usual in random graphs).

Figure 7.3 provides some numerical values in a system with $p = 0.1$ for $m = 20$ rounds. It reveals that the coverage of $n'/2$ fixed timely links in our benchmarking scenario is poor if m is above its critical value (7.10) with respect to p (which is the case for $m = 20$ and $p = 0.1$).

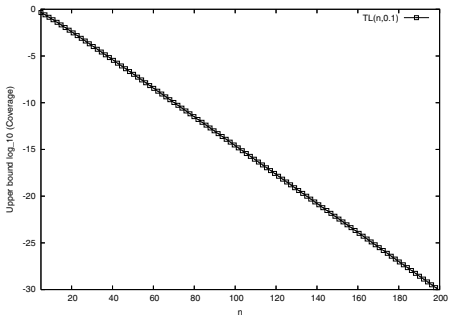


FIG. 7.3. *Coverage of nonmoving timely links: Upper bound on* $\log_{10}\big(Cov(TL_{n'/2})\big)$ *over* n *with* $p = 0.1$ *for* $m = 20$. *The number of rounds* $m = 20$ *is above its critical value* (7.10) *here.*

Analysis of $n-2$ moving link failures per round (ML_{n-2}). Classic moving link failure models, particularly [58], admit only $O(n)$ link failures per round. Let $ML_{\lambda n}$ be the model that admits at most λn link failures per round for some real constant $\lambda > 0$. In [58], it was shown that consensus possibility demands at most $\lambda n = n - 2$ link failures per round; hence $\lambda = (n-2)/n$ here.

We start our derivations with a simple bound on the tail of the binomial distribution taken from Feller's book [27],[7] which will also be required in the analysis of the coverage of our model MLO_{f_ℓ}. Consider the binomial distribution $B(\overline{n}, \overline{p})$, where $\overline{p}$ is the "success" probability, and let

$$p_{\overline{n}}(f_\ell) = \sum_{l=0}^{f_\ell} \binom{\overline{n}}{l} \overline{p}^l (1-\overline{p})^{\overline{n}-l} \tag{7.12}$$

be the probability of at most f_ℓ "successes," and let $q_{\overline{n}}(f_\ell) = 1 - p_{\overline{n}}(f_\ell)$ be the

[7]This method gives a better bound than Chernoff's, i.e., $q_n(f_\ell) \leq \min_{z \geq 1} B(z; n, p)/z^{f_\ell}$, where $B(z; n, p) = (pz + 1 - p)^n$ is the generating function of the binomial distribution.

probability of more than f_ℓ "successes." Clearly,

$$q_{\overline{n}}(f_\ell) = \sum_{l=f_\ell+1}^{\overline{n}} \binom{\overline{n}}{l} \overline{p}^l (1-\overline{p})^{\overline{n}-l}. \tag{7.13}$$

Lemma 7 gives an upper bound for this quantity.

LEMMA 7 (upper bound for binomial tail [27]). *For any $0 \le \overline{p} \le 1$, $\overline{n} \ge 1$, and $f_\ell + 1 > \overline{np}$, we have*

$$q_{\overline{n}}(f_\ell) \le \frac{(1-\overline{p})(f_\ell+1)}{f_\ell+1-\overline{np}} \cdot \binom{\overline{n}}{f_\ell+1} \overline{p}^{f_\ell+1} (1-\overline{p})^{\overline{n}-f_\ell-1}. \tag{7.14}$$

Proof. Following the argument [27, p. 151, eq. (3.4)], let $b(k;n,p) = \binom{n}{k} p^k (1-p)^{n-k}$, and note that $q_{\overline{n}}(f_\ell) = \sum_{k=0}^{\infty} b(k+f_\ell+1;\overline{n},\overline{p})$. Using the straightforward identity $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$, one obtains for any $k \ge 1$

$$\begin{aligned}
b(k+f_\ell+1;n,p) &= \frac{(n-k-f_\ell-1+1)p}{(k+f_\ell+1)(1-p)} \cdot b(k-1+f_\ell+1;n,p) \\
&= \Big(1 - \frac{k+f_\ell+1-(n+1)p}{(k+f_\ell+1)(1-p)}\Big) \cdot b(k-1+f_\ell+1;n,p) \\
&= \prod_{j=1}^{k} \Big(1 - \frac{j+f_\ell+1-(n+1)p}{(j+f_\ell+1)(1-p)}\Big) \cdot b(f_\ell+1;n,p) \\
&= \prod_{j=1}^{k} \Big(1 - \frac{1-\frac{(n+1)p}{j+f_\ell+1}}{1-p}\Big) \cdot b(f_\ell+1;n,p).
\end{aligned}$$

Since it is easily checked that, for any $j \ge 1$,

$$1 - \frac{1-\frac{(n+1)p}{j+f_\ell+1}}{1-p} \le 1 - \frac{1-\frac{np}{f_\ell+1}}{1-p},$$

it follows that

$$b(k+f_\ell+1;n,p) \le \Big(1 - \frac{1-\frac{np}{f_\ell+1}}{1-p}\Big)^k \cdot b(f_\ell+1;n,p),$$

which holds even for $k \ge 0$. Consequently,

$$\begin{aligned}
q_{\overline{n}}(f_\ell) &= \sum_{k=0}^{\infty} b(k+f_\ell+1;\overline{n},\overline{p}) \le b(f_\ell+1;\overline{n},\overline{p}) \sum_{k=0}^{\infty} \Big(1 - \frac{1-\frac{\overline{np}}{f_\ell+1}}{1-\overline{p}}\Big)^k \\
&\le \frac{1-\overline{p}}{1-\frac{\overline{np}}{f_\ell+1}} \cdot \binom{\overline{n}}{f_\ell+1} \overline{p}^{f_\ell+1} (1-\overline{p})^{\overline{n}-f_\ell-1},
\end{aligned}$$

as asserted in (7.14). ∎

In case of $ML_{\lambda n}$, we set $\overline{n} := n(n-1)$, $\overline{p} := 1-p$, and $f_\ell := n(n-1) - \lambda n - 1 = n(n-1-\lambda) - 1$ in Lemma 7, such that $q_{n(n-1)}\big(n(n-1-\lambda)-1\big)$ is the probability that at least $n(n-1-\lambda)$ nonfaulty links (and hence at most λn faulty links) occur per round. It evaluates to

$$\begin{aligned}
q_{n(n-1)}\big(n(n-1-\lambda)-1\big) \le \frac{pn(n-1-\lambda)}{n(n-1-\lambda) - n(n-1)(1-p)} \\
\cdot \binom{n(n-1)}{n(n-1-\lambda)} (1-p)^{n(n-1-\lambda)} p^{\lambda n}.
\end{aligned}$$

By independence, the probability that at most λn link failures occur during m rounds is $\big[q_{n(n-1)}\big(n(n-1-\lambda)-1\big)\big]^m$, and thus

$$\text{Cov}(ML_{\lambda n}) \le \left[\frac{p\cdot(n-1-\lambda)}{p\cdot(n-1)-\lambda}\cdot\binom{n(n-1)}{n(n-1-\lambda)}(1-p)^{n(n-1-\lambda)}p^{\lambda n}\right]^m \tag{7.15}$$

by Lemma 7. In order to determine the asymptotic value of $\text{Cov}(ML_{\lambda n})$, we will need

$$\begin{aligned}\Big(1-\frac{x}{n}\Big)^n &\sim e^{-x} \qquad \text{for any fixed } x \text{ and } n\to\infty,\\ \Big(1-\frac{x}{n-1}\Big)^{n(n-1)} &= e^{n(n-1)\log\left(1-\frac{x}{n-1}\right)}\\ &= e^{n(n-1)\cdot\left(-\frac{x}{n-1}-\frac{x^2}{2(n-1)^2}+O(x^3/n^3)\right)}\\ &= e^{-nx-\frac{nx^2}{2(n-1)}+O(x^3/n)}\\ &\sim e^{-nx-\frac{x^2}{2}} \qquad \text{for } |x|<1,\end{aligned}$$

where we used the series expansion $\log(1-x) = -\sum_{k\ge1}x^k/k$. Applying Stirling's formula (7.3) again yields

$$\begin{aligned}\binom{n(n-1)}{n(n-1-\lambda)} &\sim \frac{\left(\frac{n(n-1)}{e}\right)^{n(n-1)}}{\left(\frac{n(n-1-\lambda)}{e}\right)^{n(n-1-\lambda)}\left(\frac{\lambda n}{e}\right)^{\lambda n}}\cdot\frac{\sqrt{2\pi n(n-1)}}{\sqrt{2\pi n(n-1-\lambda)}\cdot\sqrt{2\pi\lambda n}}\\ &\sim \Big(\frac{n-1}{n-1-\lambda}\Big)^{n(n-1)}\cdot\Big(\frac{n-1-\lambda}{\lambda}\Big)^{\lambda n}\cdot\frac{1}{\sqrt{1-\frac{\lambda}{n-1}}\cdot\sqrt{2\pi\lambda n}}\\ &\sim \frac{1}{\left(1-\frac{\lambda}{n-1}\right)^{n(n-1)}}\cdot\Big(\frac{n}{\lambda}\Big)^{\lambda n}\cdot\Big(1-\frac{\lambda\cdot(1+\lambda)}{\lambda n}\Big)^{\lambda n}\cdot\frac{1}{\sqrt{2\pi\lambda n}}\\ &\sim e^{\lambda n+\lambda^2/2}\cdot\Big(\frac{n}{\lambda}\Big)^{\lambda n}\cdot e^{-\lambda(1+\lambda)}\cdot\frac{1}{\sqrt{2\pi\lambda n}}\\ &\sim \frac{e^{-\lambda(1+\lambda/2)}}{\sqrt{2\pi\lambda n}}\cdot\Big(\frac{en}{\lambda}\Big)^{\lambda n} \qquad (7.16)\\ &\sim e^{\lambda n\log n+\lambda n(1-\log\lambda)-\frac12\cdot\log n-\lambda(1+\lambda/2)-\frac12\cdot\log(2\pi\lambda)}. \qquad (7.17)\end{aligned}$$

The dominant term in the exponent in (7.17) is clearly $\lambda n\log n>0$. On the other hand,

$$(1-p)^{n(n-1-\lambda)}p^{\lambda n} = e^{n^2\log(1-p)-(1+\lambda)n\log(1-p)+(\lambda n)\log p},$$

which rapidly goes to 0 for $n\to\infty$ since the dominant term in the exponent is $\log(1-p)n^2<0$. Multiplying this with (7.16) according to (7.15) hence yields

$$\text{Cov}(ML_{\lambda n}) \propto \left[\frac{e^{-\lambda(1+\lambda/2)}}{\sqrt{2\pi\lambda n}}\cdot\Big(\frac{enp}{\lambda}\Big)^{\lambda n}(1-p)^{n(n-1-\lambda)}\right]^m, \tag{7.18}$$

which quickly goes to 0 for $n\to\infty$, for any λ and m.

For the special case $\lambda = (n-2)/n$, which implies at most $n-2$ link failures per round, we obtain for $n \to \infty$

$$\begin{aligned}
\lambda &\sim 1,\\
\frac{e^{-\lambda(1+\lambda/2)}}{\sqrt{2\pi\lambda n}} &\sim \frac{e^{-3/2}}{\sqrt{2\pi n}},\\
\Big(\frac{enp}{\lambda}\Big)^{\lambda n} = \Big(\frac{enp}{\frac{n-2}{n}}\Big)^{n-2} &= \frac{(1-2/n)^2\cdot(enp)^{n-2}}{(1-2/n)^n} \sim \frac{(enp)^{n-2}}{e^{-2}},\\
(1-p)^{n(n-1-\lambda)} = (1-p)^{n(n-1)-n+2} &= \Big(\frac{1}{(1-p)^2}\Big)^{n-2}(1-p)^{n^2-2}.
\end{aligned}$$

Using this in expression (7.18) hence yields

$$\begin{aligned}
\mathrm{Cov}(ML_{n-2}) &\sim \Big(\frac{e^{-3/2}}{e^{-2}\sqrt{2\pi n}}\Big)^m \Big(\frac{enp}{(1-p)^2}\Big)^{(n-2)m}(1-p)^{(n^2-2)m}\\
&\sim \Big(\sqrt{\frac{e}{2\pi n}}\Big)^m \Big(\frac{enp}{(1-p)^2}\Big)^{(n-2)m}(1-p)^{(n^2-2)m},
\end{aligned}$$

which very quickly goes to 0 for $n \to \infty$. Figure 7.4 gives some numerical values which reveal that the coverage of this model in our benchmarking scenario is very poor even for relatively small system sizes n.

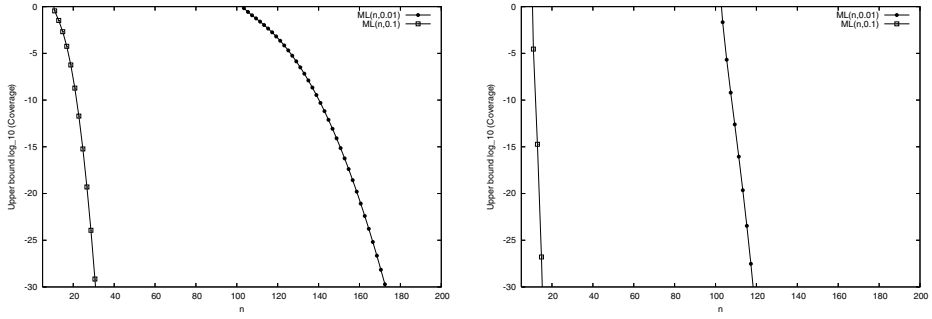


FIG. 7.4. *Coverage of $O(n)$ moving link failures per round: Upper bound on $\log_{10}\big(Cov(ML_{n-2})\big)$ over n with $p = 0.01$ and $p = 0.1$ for $m = 2$ (left) and $m = 20$ (right).*

7.2.1. Analysis of f_ℓ moving link failures per process and round (MLO_{f_ℓ}). Finally, we will show that the moving link omission failure model MLO_{f_ℓ} with $f_\ell^r = f_\ell^s = f_\ell$ introduced in section 2.2 does not suffer from poor coverage: On the contrary, in accordance with [44], we will show that $\mathrm{Cov}(MLO_{n\lambda}) \to 1$ for $n \to \infty$, for any $p < 1/2$ and $\lambda > p$.

Recalling (7.12), we have to set $\overline{n} := n-1$ and choose $\overline{p}$ to be our link failure probability p in this equation in order to obtain the probability $p_{n-1}(f_\ell)$ that at most f_ℓ outgoing links are faulty in the broadcast of a single process to its $n-1$ receivers in a round. Similarly, the probability $q_{n-1}(f_\ell) = 1 - p_{n-1}(f_\ell)$ that more than f_ℓ links are faulty in this event is given by (7.13), and Lemma 7 provides an upper bound for $q_{n-1}(f_\ell)$ for $f_\ell + 1 > (n-1)p$. Since we will eventually choose $f_\ell = (n-1)/2 - 1$ and $p < 1/2$, the latter condition is indeed satisfied. Note that there is a reasonably small upper bound for $q_{n-1}(f_\ell)$ also in case of small values $f_\ell + 1 \le (n-1)p$; see [62].

The probability that none of the n processes in the system experiences more than f_ℓ link failures on its outgoing links in a single round is $P_s = p_{n-1}(f_\ell)^n$, since the failures on the outgoing links of different processes are independent.

Obviously, (7.12) for $p_{n-1}(f_\ell)$ also provides the probability that a single receiver process experiences at most f_ℓ link failures on its incoming links. As before, the probability that none of the n processes in the system experiences more than f_ℓ link failures on its incoming links in a round is $P_r = p_{n-1}(f_\ell)^n$.

The probability P_{sr} that none of the n processes in the system experiences more than f_ℓ link failures on its outgoing links *and* no more than f_ℓ link failures on its incoming links is not just the product of P_s and P_r, however, since they are not independent. However, $P_{sr} = P_{r|s}P_s$, where $P_{r|s}$ denotes the conditional probability that no process perceives more than f_ℓ link failures on its incoming links, conditioned on the fact that no process experiences more than f_ℓ link failures on its outgoing links. Since trivially $P_{r|s} \geq P_r$, we obtain $P_{sr} \geq P_s P_r = p_{n-1}(f_\ell)^{2n}$ and hence

$$\mathrm{Cov}(MLO_{f_\ell}) = P_{sr}^m \geq (1 - q_{n-1}(f_\ell))^{2nm}.$$

By the Bernoulli inequality $(1+\alpha)^n \geq 1 + n\alpha$ for any $\alpha > -1$, we obtain

$$(1 - q_{n-1}(f_\ell))^{2nm} \geq 1 - 2nmq_{n-1}(f_\ell), \tag{7.19}$$

which is valid for $q_{n-1}(f_\ell) < 1$; since the latter is a probability < 1, this condition is of course satisfied.

Consequently, using Lemma 7, $1 - \mathrm{Cov}(MLO_{f_\ell})$ can be upper bounded by

$$1 - \mathrm{Cov}(MLO_{f_\ell}) \leq \frac{2nm(f_\ell+1)(1-p)}{f_\ell + 1 - (n-1)p}\binom{n-1}{f_\ell+1}p^{f_\ell+1}(1-p)^{n-1-f_\ell-1}. \tag{7.20}$$

A very similar analysis as for $ML_{\lambda n}$ proves that $\mathrm{Cov}(MLO_{f_\ell})$ quickly approaches 1 as $n \to \infty$ for any $f_\ell + 1 = \lambda(n-1)$ with $\lambda > p$ and $p < 1/2$: Recalling Lemma 6, we immediately obtain

$$\binom{n}{\lambda n}p^{\lambda n}(1-p)^{n(1-\lambda)} \sim \Big(\frac{1-p}{1-\lambda}\Big)^{n(1-\lambda)}\Big(\frac{p}{\lambda}\Big)^{\lambda n}\frac{1}{\sqrt{2\pi n\lambda(1-\lambda)}}.$$

Plugging $n' := n-1$ and $f_\ell + 1 = (n-1)\lambda = n'\lambda$ according into (7.20) into the above equation provides

$$1 - \mathrm{Cov}(MLO_{n'\lambda-1}) \propto \frac{m(1-p)}{\lambda - p}\cdot\sqrt{\frac{2n'\lambda}{\pi(1-\lambda)}}\left[\Big(\frac{1-p}{1-\lambda}\Big)^{1-\lambda}\Big(\frac{p}{\lambda}\Big)^{\lambda}\right]^{n'}.$$

Since $x^\lambda \leq x$ for any $0 \leq \lambda \leq 1$, and $p < \lambda$,

$$\Big(\frac{1-p}{1-\lambda}\Big)^{1-\lambda}\Big(\frac{p}{\lambda}\Big)^{\lambda} = \frac{1-p}{1-\lambda}\Big(\frac{p(1-\lambda)}{\lambda(1-p)}\Big)^{\lambda} \leq \frac{1-p}{1-\lambda}\cdot\frac{p(1-\lambda)}{\lambda(1-p)} = \frac{p}{\lambda} < 1,$$

so $\mathrm{Cov}(MLO_{(n-1)\lambda-1})$ indeed quickly approaches 1 as $n \to \infty$. In the special case $\lambda = 1/2 > p$, where $f_\ell + 1 = (n-1)/2 = n'/2$ with $n = n'+1 \sim n'$ for $n \to \infty$, we obtain

$$1 - \mathrm{Cov}(MLO_{n'/2-1}) \propto \frac{m(1-p)}{1/2 - p}\cdot\sqrt{\frac{2n'}{\pi}}\cdot\big(4p(1-p)\big)^{n'/2}, \tag{7.21}$$

which rapidly goes to 0 for $n \to \infty$, for any $p < 1/2$, and for any $m = O(n^k)$, with k arbitrary but fixed. Figure 7.5 confirms that our link failure model[8] indeed scales well with n and gives excellent coverage for any reasonable choice of parameters. Note carefully that those figures, in sharp contrast to all previous ones, show an upper bound on $1 - \mathrm{Cov}(MLO_{n'/2})$, i.e., the difference to ideal coverage.

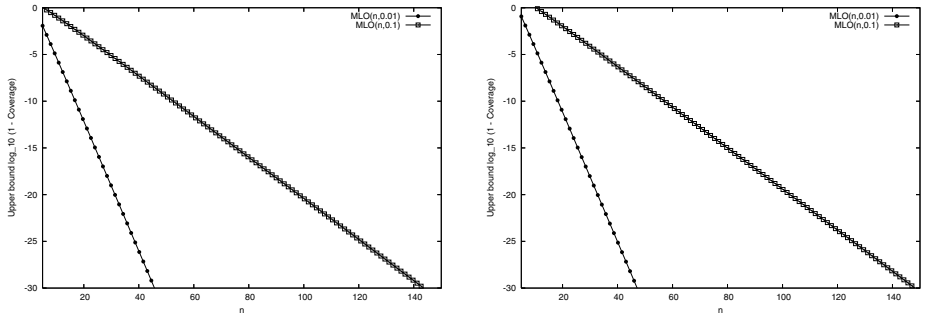


FIG. 7.5. *Coverage of f_ℓ moving link failures per round per process: Upper bound on* $\log_{10}\big(1 - Cov(MLO_{n'/2})\big)$ *over n with $p = 0.01$ and $p = 0.1$ for $m = 2$ (left) and $m = 20$ (right).*

Finally, Figure 7.6 compares our upper bounds on $\mathrm{Cov}(GO_{n/2})$, $\mathrm{Cov}(TL_{n'/2})$, and $\mathrm{Cov}(ML_{n-2})$ in a system of $n = 30$ processes, for $m = 2$ and $m = 20$ rounds, under varying link failure rates p; Figure 7.7 does the same for $n = 60$. The numerical results reveal that all those models provide poor coverage in our benchmarking scenario, particularly under substantial link failure rates. Note that this is also true for $TL_{n'/2}$, unless the number of rounds m is not below the critical value (7.10) with respect to p.

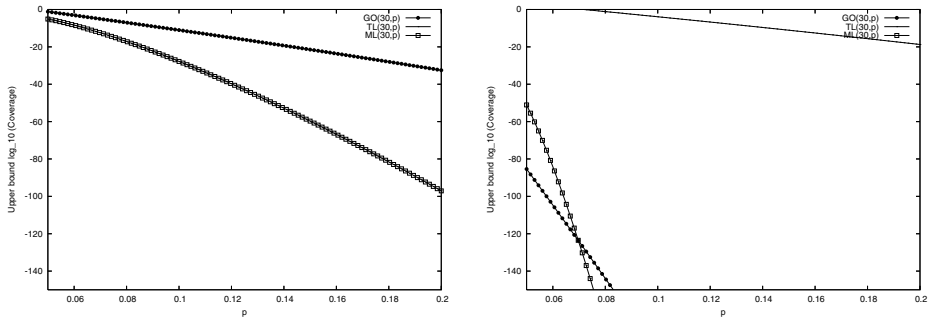


FIG. 7.6. *Upper bound on $Cov(GO_{n/2})$, $Cov(TL_{n'/2})$, and $Cov(ML_{n-1})$ over p for a system of $n = 30$ processes, for $m = 2$ (left) and $m = 20$ (right) rounds.*

By contrast, Figures 7.8 and 7.9 provide numerical results for our upper bound on $1 - \mathrm{Cov}(MLO_{n'/2})$, under the same parameter values for n and m as in Figures 7.6 and 7.7. They reveal a high coverage also under substantial link failure rates, as well as a remarkably low dependence on the number m of rounds. They finally justify our claim that MLO is the only model that performs well in our benchmarking scenario.

[8] As well as the moving timely link model for Ω and consensus of [39, 40], which can be analyzed in a similar way.

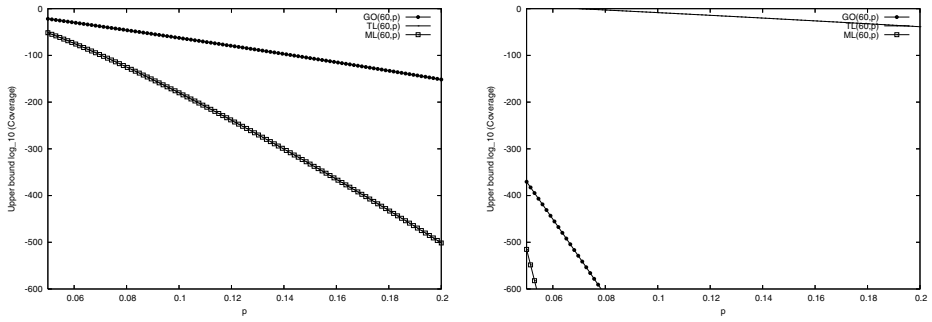


FIG. 7.7. *Upper bound on* $Cov(GO_{n/2})$, $Cov(TL_{n'/2})$, *and* $Cov(ML_{n-1})$ *over* p *for a system of* $n = 60$ *processes, for* $m = 2$ *(left) and* $m = 20$ *(right) rounds.*

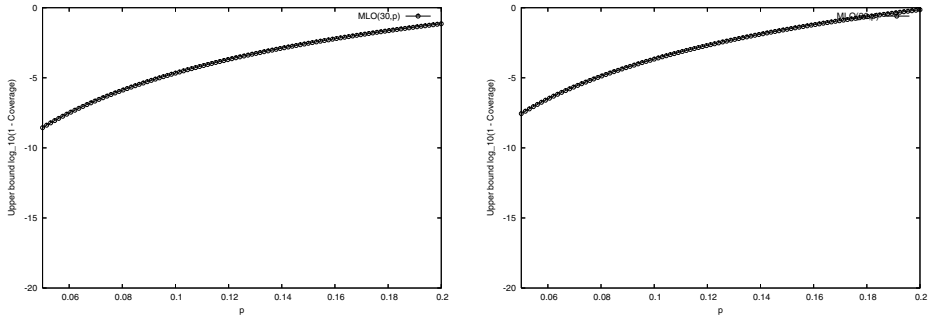


FIG. 7.8. *Upper bound on* $\log_{10}\big(1 - Cov(MLO_{n'/2})\big)$ *over* p *for a system of* $n = 30$ *processes, for* $m = 2$ *(left) and* $m = 20$ *(right) rounds.*

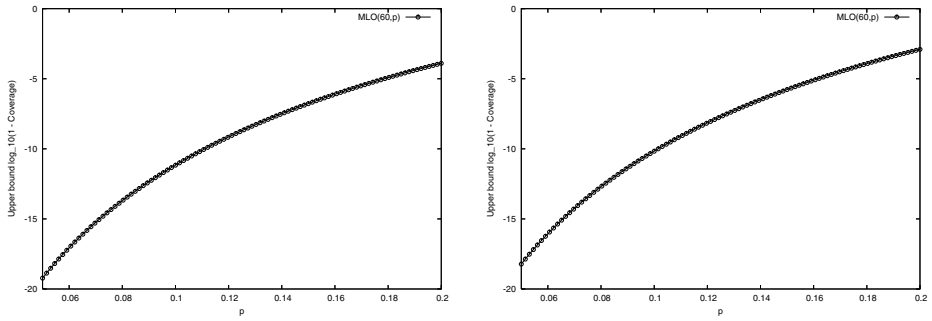


FIG. 7.9. *Upper bound on* $\log_{10}\big(1 - Cov(MLO_{n'/2})\big)$ *over* p *for a system of* $n = 60$ *processes, for* $m = 2$ *(left) and* $m = 20$ *(right) rounds.*

8. Conclusions. We provided a complete theoretical treatment of the impossibility of deterministic synchronous consensus under a novel link failure model, which grants every process a certain maximum number of send and receive link failures per round. Link failures may both be omissive and arbitrary and can hit messages to/from different processes in every round. Using novel instances of "easy impossibility" and bivalency proofs, we provided related lower bounds for the number of processes and rounds as well. Most of them are matched by existing consensus algorithms and hence

are tight. An analysis of the assumption coverage in a simple probabilistic setting revealed that our model is the only one with a coverage that approaches 1 (rather than 0) for large n.

Part of our current/future theoretical research in this area is devoted to consensus lower bounds under our fully fledged hybrid failure model, which captures both process and link failures simultaneously. We analyzed several algorithms under this model and found that the respective numbers of processes required just add up. This suggested that tolerating link failures and process failures is more or less orthogonal. In [11], however, it was shown that this is not true in general. Generalized lower bounds are hence required for reasoning about optimal algorithms here.

Acknowledgments. We are grateful to Michael Drmota for providing us with Theorem 10 from [57] and to the anonymous reviewers for their thorough and very helpful comments.

REFERENCES

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang, and L. Zuck, *Reliable communication over unreliable channels*, J. ACM, 41 (1994), pp. 1267–1297.

[2] M. K. Aguilera, W. Chen, and S. Toueg, *Failure detection and consensus in the crash-recovery model*, in Distributed Computing, Vol. 13, Springer-Verlag, London, 2000, pp. 99–125.

[3] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, *Stable leader election*, in Proceedings of the 15th International Conference on Distributed Computing, Springer-Verlag, New York, 2001, pp. 108–122.

[4] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, *On implementing Omega with weak reliability and synchrony assumptions*, in Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC'03), ACM Press, New York, 2003, pp. 306–314.

[5] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, *Communication-efficient leader election and consensus with limited link synchrony*, in Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'04) (St. John's, Newfoundland, Canada), ACM, New York, 2004, pp. 328–337.

[6] M. K. Aguilera and S. Toueg, *A simple bivalency proof that t-resilient consensus requires t+1 rounds*, Inform. Process. Lett., 71 (1999), pp. 155–158.

[7] E. Anceaume, A. Fernández, A. Mostéfaoui, G. Neiger, and M. Raynal, *A necessary and sufficient condition for transforming limited accuracy failure detectors*, J. Comput. System Sci., 68 (2004), pp. 123–133.

[8] H. Attiya and J. Welch, *Distributed Computing*, McGraw-Hill, New York, 1998.

[9] A. Basu, B. Charron-Bost, and S. Toueg, *Crash failures vs. crash + link failures*, in Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (Philadelphia, PA), ACM, New York, 1996, p. 246.

[10] P. Berman, J. A. Garay, and K. J. Perry, *Asymptotically optimal distributed consensus*, available online at http://www.bell-labs.com/user/garay/#distributed-pub, 1992.

[11] M. Biely, *An optimal Byzantine agreement algorithm with arbitrary node and link failures*, in Proceedings of the 15th Annual IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'03), Marina Del Rey, CA, 2003, pp. 146–151.

[12] M. Biely, *Towards an Optimal Algorithm for Hybrid Byzantine Agreement*, Tech. report 183/1-130, Department of Automation, Technische Universität Wien, Vienna, Austria, 2003.

[13] M. Biely, B. Charron-Bost, A. Gaillard, M. Hutle, A. Schiper, and J. Widder, *Tolerating corrupted communication*, in Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC'07), Portland, OR, 2007, pp. 244–253.

[14] M. Biely and U. Schmid, *Message-Efficient Consensus in Presence of Hybrid Node and Link Faults*, Tech. report 183/1-116, Department of Automation, Technische Universität Wien, Vienna, Austria, 2001.

[15] G. Bracha and S. Toueg, *Resilient consensus protocols*, in Proceedings of the 2nd Symposium on the Principles of Distributed Computing (PODC'83), Montreal, Canada, 1983, pp. 12–26.

[16] T. D. Chandra and S. Toueg, *Unreliable failure detectors for reliable distributed systems*, J. ACM, 43 (1996), pp. 225–267.
[17] B. Charron-Bost and A. Schiper, *Uniform consensus is harder than consensus*, J. Algorithms, 51 (2004), pp. 15–37.
[18] B. Charron-Bost and A. Schiper, *The heard-of model: Unifying all benign failures*, Tech. report LSR-REPORT-2006-004, EPFL, 2006.
[19] B. Charron-Bost and A. Schiper, *Improving fast Paxos: Being optimistic with no overhead*, in Proceedings of the 12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2006), IEEE Computer Society, Washington, DC, 2006, pp. 287–295.
[20] F. Cristian, H. Aghili, R. Strong, and D. Dolev, *Atomic broadcast: From simple message diffusion to Byzantine agreement*, in Proceedings of the 15th International Conference on Fault-Tolerant Computing (FTCS-15), Ann Arbor, MI, 1985, pp. 200–206.
[21] S. Dobrev, *Computing input multiplicity in anonymous synchronous networks with dynamic faults*, in Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'00), Springer-Verlag, London, 2000, pp. 137–148.
[22] S. Dobrev and I. Vrto, *Optimal broadcasting in hypercubes with dynamic faults*, Inform. Process. Lett., 71 (1999), pp. 81–85.
[23] D. Dolev, *The Byzantine generals strike again*, J. Algorithms, 3 (1982), pp. 14–30.
[24] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi, *Failure detectors in omission failure environments*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, p. 286.
[25] P. Dutta and R. Guerraoui, *The inherent price of indulgence*, in Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (Monterey, CA), ACM, New York, 2002, pp. 88–97.
[26] P. Dutta, R. Guerraoui, and I. Keidar, *The overhead of consensus failure recovery*, in Distributed Computing, Vol. 19, Springer-Verlag, London, pp. 373–386.
[27] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 3rd ed., John Wiley & Sons, New York, 1968.
[28] M. Fischer, N. Lynch, and M. Merritt, *Easy impossibility proofs for the distributed consensus problem*, in Distributed Computing, Vol. 1, Springer-Verlag, London, pp. 26–39.
[29] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.
[30] E. Gafni, *Round-by-round fault detectors (extended abstract): Unifying synchrony and asynchrony*, in Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing (Puerto Vallarta, Mexico), ACM, New York, 1998, pp. 143–152.
[31] J. A. Garay, *Reaching (and maintaining) agreement in the presence of mobile faults*, in Proceedings of the 8th International Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 857, Springer-Verlag, New York, 1994, pp. 253–264.
[32] L. Gong, P. Lincoln, and J. Rushby, *Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults*, in Proceedings of Dependable Computing for Critical Applications 5, Champaign, IL, 1995, pp. 139–157.
[33] J. N. Gray, *Notes on data base operating systems*, in Operating Systems: An Advanced Course, G. Seegmüller, R. Bayer, and R. M. Graham, eds., Lecture Notes in Comput. Sci. 60, Springer-Verlag, New York, 1978, p. 465.
[34] G. Gridling, *An Algorithm for Three-Process Consensus under Restricted Link Failures*, Tech. report 183/1-123, Department of Automation, Technische Universität Wien, Vienna, Austria, 2002.
[35] R. Guerraoui, *Indulgent algorithms (preliminary version)*, in Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (Portland, OR), ACM, New York, 2000, pp. 289–297.
[36] R. Guerraoui, R. Oliveira, and A. Schiper, *Stubborn Communication Channels*, Tech. report 98-278, Dèpartment d'Informatique, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1998.
[37] V. Hadzilacos, *Connectivity requirements for Byzantine agreement under restricted types of failures*, in Distributed Computing, Vol. 2, Springer-Verlag, London, 1987, pp. 95–103.
[38] V. Hadzilacos and S. Toueg, *Fault-tolerant broadcasts and related problems*, in Distributed Systems, 2nd ed., S. Mullender, ed., ACM/Addison–Wesley, New York, 1993, pp. 97–145.
[39] M. Hutle, D. Malkhi, U. Schmid, and L. Zhou, *Brief announcement: Chasing the weakest system model for implementing* Ω *and consensus*, in Proceedings of the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2006), Lecture Notes in Comput. Sci. 4280, Springer-Verlag, New York, 2006, pp. 576–577.

[40] M. Hutle, D. Malkhi, U. Schmid, and L. Zhou, *Chasing the weakest system model for implementing* Ω *and consensus*, IEEE Trans. Dependable and Secure Computing, to appear.
[41] I. Keidar and S. Rajsbaum, *On the cost of fault-tolerant consensus when there are no faults (preliminary version)*, SIGACTN: SIGACT News, 32 (2001), pp. 45–63.
[42] I. Keidar and S. Rajsbaum, *A simple proof of the uniform consensus synchronous lower bound*, Inform. Process. Lett., 85 (2003), pp. 47–52.
[43] I. Keidar and A. Shraer, *Timeliness, failure detectors, and consensus performance*, in Proceedings of the 25th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'06), ACM, New York, 2006, pp. 169–178.
[44] I. Keidar and A. Shraer, *How to choose a timing model*, in Proceedings of the 37th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), IEEE, Washington, DC, 2007, pp. 389–398.
[45] L. Lamport, *Lower Bounds on Consensus*, unpublished manuscript; available online from http://lamport.org, 2000.
[46] L. Lamport, *The part-time parliament*, ACM Trans. Comput. Systems, 16 (1998), pp. 133–169.
[47] L. Lamport, R. Shostak, and M. Pease, *The Byzantine generals problem*, ACM Trans. Programming Languages and Systems, 4 (1982), pp. 382–401.
[48] P. Lincoln and J. Rushby, *A formally verified algorithm for interactive consistency under a hybrid fault model*, in Proceedings of the 23rd Fault Tolerant Computing Symposium, Toulouse, France, 1993, pp. 402–411.
[49] Z. Lipták and A. Nickelsen, *Broadcasting in complete networks with dynamic edge faults*, in Proceedings of the 4th International Conference on Principles of Distributed Systems (OPODIS'00), Paris, France, 2000, pp. 123–142.
[50] N. Lynch, *Distributed Algorithms*, Morgan Kaufman, San Francisco, 1996.
[51] D. Malkhi, F. Oprea, and L. Zhou, Ω *meets paxos: Leader election and stability without eventual timely links*, in Proceedings of the 19th Symposium on Distributed Computing (DISC'05), Lecture Notes of Comput. Sci. 3724, Springer-Verlag, New York, 2005, pp. 199–213.
[52] Y. Moses and S. Rajsbaum, *A layered analysis of consensus*, SIAM J. Comput., 31 (2002), pp. 989–1021.
[53] A. Mostéfaoui and M. Raynal, *Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach*, in Distributed Computing: Proceedings of the 13th International Symposium (DISC'99), P. Jayanti, ed., Lecture Notes in Comput. Sci. 1693, Springer-Verlag, New York, 1999, pp. 49–63.
[54] K. J. Perry and S. Toueg, *Distributed agreement in the presence of processor and communication faults*, IEEE Trans. Software Engineering, SE-12 (1986), pp. 477–482.
[55] S. S. Pinter and I. Shinahr, *Distributed agreement in the presence of communication and process failures*, in Proceedings of the 14th IEEE Convention of Electrical & Electronics Engineers in Israel, IEEE, Washington, DC, 1985.
[56] R. Reischuk, *A new solution for the Byzantine generals problem*, Inform. and Control, 64 (1985), pp. 23–42.
[57] O. Riordan and A. Selby, *The maximum degree of a random graph*, Combin. Probab. Comput., 9 (2000), pp. 549–572.
[58] N. Santoro and P. Widmayer, *Time is not a healer*, in Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89), Lecture Notes in Comput. Sci. 349, Springer-Verlag, New York, pp. 304–313.
[59] N. Santoro and P. Widmayer, *Distributed function evaluation in the presence of transmission faults*, in SIGAL International Symposium on Algorithms, Tokyo, Japan, 1990, pp. 358–367.
[60] N. Santoro and P. Widmayer, *Agreement in synchronous networks with ubiquitous faults*, Theoret. Comput. Sci., 384 (2007), pp. 232–249.
[61] H. M. Sayeed, M. Abu-Amara, and H. Abu-Amara, *Optimal asynchronous agreement and leader election algorithm for complete networks with Byzantine faulty links*, in Distributed Computing, Vol. 9, Springer-Verlag, London, 1995, pp. 147–156.
[62] U. Schmid, *Failure Model Coverage under Transient Link Failures*, Research report 2/2004, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, 2004, submitted.
[63] U. Schmid and C. Fetzer, *Randomized Asynchronous Consensus with Imperfect Communications*, Technical report 183/1-120, Department of Automation, Technische Universität Wien, Vienna, Australia, 2002.

[64] U. Schmid and C. Fetzer, *Randomized asynchronous consensus with imperfect communications*, in Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS'03), Florence, Italy, 2003, pp. 361–370.

[65] U. Schmid and B. Weiss, *Synchronous Byzantine agreement under hybrid process and link failures*, Technical report 183/1-124, Department of Automation, Technische Universität Wien, Vienna, Australia, 2002.

[66] U. Schmid, B. Weiss, and J. Rushby, *Formally verified Byzantine agreement in presence of link faults*, in Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, 2002, pp. 608–616.

[67] H.-S. Siu, Y.-H. Chin, and W.-P. Yang, *Byzantine agreement in the presence of mixed faults on processors and links*, IEEE Trans. Parallel and Distributed Systems, 9 (1998), pp. 335–345.

[68] T. K. Srikanth and S. Toueg, *Simulating authenticated broadcasts to derive simple fault-tolerant algorithms*, in Distributed Computing, Vol. 2, Springer-Verlag, London, 1987, pp. 80–94.

[69] G. Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, Cambridge, UK, 1994.

[70] G. Varghese and N. A. Lynch, *A tradeoff between safety and liveness for randomized coordinated attack protocols*, in Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, 1992, pp. 241–250.

[71] B. Weiss and U. Schmid, *Consensus with written messages under link faults*, in Proceedings of the 20th Symposium on Reliable Distributed Systems (SRDS'01), New Orleans, LA, 2001, pp. 194–197.

[72] J. Widder and U. Schmid, *Booting clock synchronization in partially synchronous systems with hybrid process and link failures*, in Distributed Computing, Vol. 20, Springer-Verlag, London, 2007, pp. 115–140.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1952–1969

LOCALLY DECODABLE CODES FROM NICE SUBSETS OF FINITE FIELDS AND PRIME FACTORS OF MERSENNE NUMBERS*

KIRAN S. KEDLAYA† AND SERGEY YEKHANIN‡

Abstract. A k-query locally decodable code (LDC) encodes an n-bit message x as an N-bit codeword $C(x)$, such that one can probabilistically recover any bit x_i of the message by querying only k bits of the codeword $C(x)$, even after some constant fraction of codeword bits has been corrupted. The major goal of LDC related research is to establish the optimal trade-off between length and query complexity of such codes. Recently vast improvements in upper bounds for the length of LDCs were achieved via constructions that rely on existence of certain special ("nice") subsets of finite fields. In this work we extend the constructions of LDCs from "nice" subsets. We argue that further progress on upper bounds for LDCs via these methods is tied to progress on an old number theory question regarding the size of the largest prime factors of Mersenne numbers. Specifically, we show that every Mersenne number $m = 2^t - 1$ that has a prime factor $p > m^\gamma$ yields a family of $k(\gamma)$-query LDCs of length $\exp(n^{1/t})$. Conversely, if for some fixed k and all $\epsilon > 0$ one can use the "nice" subsets technique to obtain a family of k-query LDCs of length $\exp(n^\epsilon)$, then infinitely many Mersenne numbers have prime factors larger than currently known.

Key words. locally decodable codes, Mersenne primes

AMS subject classifications. 11L05, 11B99, 94B60

DOI. 10.1137/070696519

1. Introduction.

Classical error-correcting codes allow one to encode an n-bit string x into an N-bit codeword $C(x)$, in such a way that x can still be recovered even if $C(x)$ gets corrupted in a number of coordinates. It is well known [21] that codewords $C(x)$ of length $N = O(n)$ already suffice to correct errors in up to δN locations of $C(x)$ for any constant $\delta < 1/4$. The disadvantage of classical error-correction is that one needs to consider all or most of the (corrupted) codeword to recover anything about x. Now suppose that one is interested only in recovering one or a few bits of x. In such a case more efficient schemes are possible. Such schemes are known as locally decodable codes (LDCs). LDCs allow reconstruction of an arbitrary bit x_i from looking only at k randomly chosen coordinates of $C(x)$, where k can be as small as 2. LDCs have numerous applications in complexity theory [17, 33], cryptography [6, 13], and the theory of fault tolerant computation [28]. The following is a slightly informal definition of LDCs.

A (k, δ, ϵ)-LDC encodes n-bit strings to N-bit codewords $C(x)$, such that for every $i \in [n]$, the bit x_i can be recovered with probability $1 - \epsilon$ by a randomized decoding procedure that makes only k queries, even if the codeword $C(x)$ is corrupted in up to δN locations.

One should think of $\delta > 0$ and $\epsilon < 1/2$ as constants. The main parameters of interest in LDCs are the length N and the query complexity k. Ideally we would like to have both of them as small as possible. The concept of LDCs was explicitly discussed in various papers in the early 1990s [2, 32, 25]. Katz and Trevisan [17]

*Received by the editors July 10, 2007; accepted for publication (in revised form) July 22, 2008; published electronically January 14, 2009. A preliminary version of this paper appeared in *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC'08)* [18].

http://www.siam.org/journals/sicomp/38-5/69651.html

†Department of Mathematics, MIT, Cambridge, MA 02139 (kedlaya@mit.edu). This author's research was supported by NSF CAREER grant DMS-0545904 and by the Sloan Research Fellowship.

‡Microsoft Research, Mountain View, CA 94043 (yekhanin@microsoft.com).

were the first to provide a formal definition of LDCs. Further work on LDCs includes [3, 10, 24, 4, 19, 35, 38, 37, 16, 27, 36, 15].

Below is a brief summary of what was known regarding the length of optimal LDCs prior to a recent paper of Yekhanin [38]. The length of optimal 2-query LDCs was settled by Kerenidis and de Wolf [19] and is $\exp(n)$.[1] The best upper bound for the length of 3-query LDCs is $\exp(n^{1/2})$ due to Beimel, Ishai, and Kushilevitz [3], and the best lower bound is $\tilde{\Omega}(n^2)$ [37]. For general (constant) k the best upper bound is $\exp(n^{O(\log\log k/(k\log k))})$ due to Beimel et al. [4], and the best lower bound is $\tilde{\Omega}(n^{1+1/(\lceil k/2\rceil-1)})$ [37].

The recent work of Yekhanin [38] improved the upper bounds to the extent that it changed the common perception of what may be achievable [14, 13]. Yekhanin [38] introduced a novel technique to construct codes from so-called nice subsets of finite fields and showed that every Mersenne prime $p = 2^t - 1$ yields a family of 3-query LDCs of length $\exp(n^{1/t})$. Based on the largest known Mersenne prime [7], this translates to a length of less than $\exp(n^{10^{-7}})$. Combined with the recursive construction from [4], this result yields vast improvements for all values of $k > 2$. It has often been conjectured that the number of Mersenne primes is infinite. If indeed this conjecture holds, Yekhanin [38] gets 3-query LDCs of length $N = \exp(n^{O(\frac{1}{\log\log n})})$ *for infinitely many* n. Finally, assuming that the conjecture of Lenstra, Pomerance, and Wagstaff (see [34, p. 388], [26]) regarding the density of Mersenne primes holds, Yekhanin [38] gets 3-query LDCs of length $N = \exp(n^{O(\frac{1}{\log^{1-\epsilon}\log n})})$ *for all* n, for every $\epsilon > 0$.

1.1. Our results. We address two natural questions left open by [38]:

1. Are Mersenne primes necessary for the constructions of [38]?
2. Has the technique of [38] been pushed to its limits, or can one construct better codes through a more clever choice of nice subsets of finite fields?

We extend the work of [38] and answer both of these questions. In what follows let $P(m)$ denote the largest prime factor of m. We show that one does not necessarily need to use Mersenne primes. It suffices to have Mersenne numbers with polynomially large prime factors. Specifically, every Mersenne number $m = 2^t - 1$ such that $P(m) \geq m^\gamma$ yields a family of $k(\gamma)$-query LDCs of length $\exp(n^{1/t})$. A partial converse also holds. Namely, if for some fixed $k \geq 3$ and all $\epsilon > 0$ one can use the technique of [38] to (unconditionally) obtain a family of k-query LDCs of length $\exp(n^\epsilon)$, then for infinitely many t we have

$$P(2^t - 1) \geq (t/2)^{1+1/(k-2)}. \tag{1}$$

The bound (1) may seem quite weak in light of the widely accepted conjecture that the number of Mersenne primes is infinite. However, (for any $k \geq 3$) this bound is substantially stronger than what is currently known unconditionally. Lower bounds for $P(2^t - 1)$ have received a considerable amount of attention in the number theory literature [29, 30, 11, 31, 23, 22, 12]. The strongest result to date is due to Stewart [31]. It says that, for all integers t ignoring a set of asymptotic density zero and for all functions $\epsilon(t) > 0$ where $\epsilon(t)$ tends to zero monotonically and arbitrarily slowly,

$$P(2^t - 1) > \epsilon(t)t\,(\log t)^2 \,/\, \log\log t. \tag{2}$$

There are no better bounds known to hold for infinitely many values of t, unless one is willing to accept some number theoretic conjectures [23, 22]. We hope that our

[1] Throughout the paper we use the standard notation $\exp(x) = e^{\Theta(x)}$.

work will further stimulate the interest in proving lower bounds for $P(2^t - 1)$ in the number theory community.

In summary, we show that one may be able to improve the unconditional bounds of [38] (say, by discovering a new Mersenne number with a very large prime factor) using the same technique. However, any attempts to reach the $\exp(n^\epsilon)$ length for some fixed query complexity and all $\epsilon > 0$ require either progress on an old number theory problem or a substantially different approach.

In this paper we deal only with binary codes for the sake of clarity of presentation. We remark, however, that our results as well as the results of [38] can be easily generalized to larger alphabets. Such a generalization is discussed in detail in [39].

1.2. Outline. In section 3 we introduce the key concepts of [38], namely, those of combinatorial and algebraic niceness of subsets of finite fields. We also briefly review the construction of LDCs from nice subsets. In section 4 we show how Mersenne numbers with large prime factors yield nice subsets of prime fields. In section 5 we prove a partial converse. Namely, we show that every finite field $\mathbb{F}_q$ containing a sufficiently nice subset is an extension of a prime field $\mathbb{F}_p$, where p is a large prime factor of a large Mersenne number. Our main results are summarized in sections 4.3 and 5.4.

2. Notation. We use the following standard mathematical notation:

- $[s] = \{1, \ldots, s\}$.
- $\mathbb{Z}_n$ denotes integers modulo n.
- $\mathbb{F}_q$ is the finite field of q elements.
- $d_H(x, y)$ denotes the Hamming distance between binary vectors x and y.
- (u, v) stands for the dot product of vectors u and v.
- For a linear space $L \subseteq \mathbb{F}_2^m$, $L^\perp$ denotes the *dual* space. That is, $L^\perp = \{u \in \mathbb{F}_2^m \mid \text{for all } v \in L,\ (u, v) = 0\}$.
- For an odd prime p, $\mathrm{ord}_p(2)$ denotes the smallest integer t such that $p \mid 2^t - 1$.

3. Nice subsets of finite fields and locally decodable codes. In this section we introduce the key technical concepts of [38], namely that of combinatorial and algebraic niceness of subsets of finite fields. We briefly review the construction of LDCs from nice subsets. Our review is concise although self-contained. We refer the reader interested in a more detailed and intuitive treatment of the construction to the original paper [38]. We start by formally defining LDCs.

DEFINITION 1. *A binary code $C : \{0,1\}^n \to \{0,1\}^N$ is said to be (k, δ, ϵ)-locally decodable if there exists a randomized decoding algorithm $\mathcal{A}$ such that the following hold:*

1. *For all $x \in \{0,1\}^n$, $i \in [n]$, and $y \in \{0,1\}^N$ such that $d_H(C(x), y) \le \delta N$, we have $Pr[\mathcal{A}(y, i) = x_i] \ge 1 - \epsilon$, where the probability is taken over the random coin tosses of the algorithm $\mathcal{A}$.*
2. *$\mathcal{A}$ reads at most k coordinates of y.*

We now introduce the concepts of combinatorial and algebraic niceness of subsets of finite fields. Our definitions are syntactically slightly different from the original definitions in [38]. We prefer these formulations since they are more appropriate for the purposes of the current paper. In what follows let $\mathbb{F}_q^*$ denote the multiplicative group of $\mathbb{F}_q$.

DEFINITION 2. *A set $S \subseteq \mathbb{F}_q^*$ is called t-combinatorially nice if for some constant $c > 0$ and every positive integer m there exist two $n = \lfloor cm^t \rfloor$-sized collections of vectors $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_n\}$ in $\mathbb{F}_q^m$, such that*

1. *for all* $i \in [n]$, $(u_i, v_i) = 0$;
2. *for all* $i, j \in [n]$ *such that* $i \neq j$, $(u_j, v_i) \in S$.

DEFINITION 3. *A set* $S \subseteq \mathbb{F}_q^*$ *is called* k*-algebraically nice if* k *is odd and there exist an odd* $k' \leq k$ *and two sets* $S_0, S_1 \subseteq \mathbb{F}_q$ *such that*

1. S_0 *is not empty;*
2. $|S_1| = k'$;
3. *for all* $\alpha \in \mathbb{F}_q$ *and* $\beta \in S$, *we have* $|S_0 \cap (\alpha + \beta S_1)| \equiv 0 \bmod (2)$.

The following lemma shows that for an algebraically nice set S, the set S_0 can always be chosen to be large. It is a straightforward generalization of [38, Lemma 15].

LEMMA 4. *Let* $S \subseteq \mathbb{F}_q^*$ *be a* k*-algebraically nice set. Let* $S_0, S_1 \subseteq \mathbb{F}_q$ *be sets realizing the definition of* k*-algebraic niceness of* S. *One can always redefine the set* S_0 *to satisfy* $|S_0| \geq \lceil q/2 \rceil$.

Proof. Let L be the linear subspace of $\mathbb{F}_2^q$ spanned by the incidence vectors of the sets $\alpha + \beta S_1$ for $\alpha \in \mathbb{F}_q$ and $\beta \in S$. Observe that L is invariant under the actions of a 1-transitive permutation group (permuting the coordinates in accordance with addition in $\mathbb{F}_q$). This implies that the space $L^\perp$ is also invariant under the actions of the same group. Note that $L^\perp$ has positive dimension since it contains the incidence vector of the set S_0. The last two observations imply that $L^\perp$ has *full support*; i.e., for every $i \in [q]$ there exists a vector $v \in L^\perp$ such that $v_i \neq 0$. Note that the expected weight of a random vector in a linear subspace of $\mathbb{F}_2^q$ of full support is $q/2$. Let $v \in L^\perp$ be an arbitrary vector of weight $\lceil q/2 \rceil$. Redefining the set S_0 to be the set of nonzero coordinates of v, we conclude the proof. □

We now proceed to the core proposition of [38] that shows how sets exhibiting both combinatorial and algebraic niceness yield LDCs.

PROPOSITION 5. *Suppose* $S \subseteq \mathbb{F}_q^*$ *is* t*-combinatorially nice and* k*-algebraically nice; then for every message length* n *there exists a code of length* $\exp(n^{1/t})$ *that is* $(k, \delta, 2k\delta)$*-locally decodable for all* $\delta > 0$.

Proof. Our proof proceeds in three steps. We specify encoding and local decoding procedures for our codes and then argue the lower bound for the probability of correct decoding. We use the notation from Definitions 2 and 3.

Encoding. We assume that our message has length $n = \lfloor cm^t \rfloor$ for some integer value of m. (Otherwise we pad the message with zeros. It is easy to see that such padding does not affect the asymptotic length of the code.) Our code will be $\mathbb{F}_2$-linear. Therefore it suffices to specify the encoding of unit vectors $e_1, \ldots, e_n$, where e_j has length n and a unique nonzero coordinate j. We define the encoding of e_j to be a q^m long vector, whose coordinates are labelled by elements of $\mathbb{F}_q^m$. For all $w \in \mathbb{F}_q^m$ we set

$$\mathrm{Enc}(e_j)_w = \begin{cases} 1 & \text{if } (u_j, w) \in S_0, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

It is straightforward to verify that we defined a code encoding n bits to $\exp(n^{1/t})$ bits.

Local decoding. Given a (possibly corrupted) codeword y and an index $i \in [n]$, the decoding algorithm $\mathcal{A}$ picks $w \in \mathbb{F}_q^m$ such that $(u_i, w) \in S_0$ uniformly at random, reads $k' \leq k$ coordinates of y, and outputs the modulo 2 sum

$$\sum_{\lambda \in S_1} y_{w + \lambda v_i}. \tag{4}$$

Probability of correct decoding. First we argue that decoding is always correct if $\mathcal{A}$ picks $w \in \mathbb{F}_q^m$ such that all bits of y in locations $\{w + \lambda v_i\}_{\lambda \in S_1}$ are not corrupted.

We need to show that for all $i \in [n]$, $x \in \{0,1\}^n$, and $w \in \mathbb{F}_q^m$, such that $(u_i, w) \in S_0$,

$$\sum_{\lambda \in S_1} \left(\sum_{j=1}^{n} x_j \operatorname{Enc}(e_j) \right)_{w+\lambda v_i} = x_i. \tag{5}$$

Note that

$$\begin{aligned} \sum_{\lambda \in S_1} \left(\sum_{j=1}^{n} x_j \operatorname{Enc}(e_j) \right)_{w+\lambda v_i} &= \sum_{j=1}^{n} x_j \sum_{\lambda \in S_1} \operatorname{Enc}(e_j)_{w+\lambda v_i} \\ &= \sum_{j=1}^{n} x_j \sum_{\lambda \in S_1} I\left[(u_j, w+\lambda v_i) \in S_0\right], \end{aligned} \tag{6}$$

where $I[\gamma \in S_0] = 1$ if $\gamma \in S_0$ and zero otherwise. Now note that

$$\begin{aligned} \sum_{\lambda \in S_1} I\left[(u_j, w+\lambda v_i) \in S_0\right] &= \sum_{\lambda \in S_1} I\left[(u_j, w) + \lambda(u_j, v_i) \in S_0\right] \\ &= \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{7}$$

The last identity in (7) for $i = j$ follows from $(u_i, v_i) = 0$, $(u_i, w) \in S_0$, and $k' = |S_1|$ is odd. The last identity for $i \neq j$ follows from $(u_j, v_i) \in S$ and the algebraic niceness of S. Combining identities (6) and (7), we get (5).

Now assume that up to δ fraction of bits of y are corrupted. Let T_i denote the set of coordinates whose labels belong to $\{w \in \mathbb{F}_q^m \mid (u_i, w) \in S_0\}$. Recall that by Lemma 4, $|T_i| \geq q^m/2$. Thus at most 2δ fraction of coordinates in T_i contain corrupted bits. Let $Q_i = \left\{\{w+\lambda v_i\}_{\lambda \in S_1} \mid w : (u_i, w) \in S_0\right\}$ be the family of k'-tuples of coordinates that may be queried by $\mathcal{A}$. The fact that $(u_i, v_i) = 0$ implies that every element of T_i belongs to the same number of k'-tuples from Q_i. Combining the last two observations, we conclude that with probability at least $1 - 2k\delta$ $\mathcal{A}$ picks an uncorrupted k'-tuple and outputs the correct value of x_i. □

All LDCs constructed in this paper are obtained by applying Proposition 5 to certain nice sets. Thus all our codes have the same dependence of ϵ (the probability of the decoding error) on δ (the fraction of corrupted bits). In what follows we often ignore these parameters and consider only the length and query complexity of codes.

4. Mersenne numbers with large prime factors yield nice subsets of prime fields. In what follows let $\langle 2 \rangle \subseteq \mathbb{F}_p^*$ denote the multiplicative subgroup of $\mathbb{F}_p^*$ generated by 2. In [38, Lemma 13] it is shown that for every Mersenne prime $p = 2^t - 1$ the set $\langle 2 \rangle \subseteq \mathbb{F}_p^*$ is simultaneously 3-algebraically nice and $\operatorname{ord}_p(2)$-combinatorially nice. In this section we prove the same conclusion for a substantially broader class of primes.

LEMMA 6. *Suppose p is an odd prime; then $\langle 2 \rangle \subseteq \mathbb{F}_p^*$ is $\operatorname{ord}_p(2)$-combinatorially nice.*

Proof. Let $t = \operatorname{ord}_p(2)$. Clearly, t divides $p-1$. We need to specify a constant $c > 0$ such that for every positive integer m there exist two $n = \lfloor cm^t \rfloor$-sized collections of vectors $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_n\}$ in $\mathbb{F}_p^m$ satisfying

- for all $i \in [n]$, $(u_i, v_i) = 0$;
- for all $i, j \in [n]$ such that $i \neq j$, $(u_j, v_i) \in \langle 2 \rangle$.

We start with some notation. For $y \in \mathbb{F}_p^w$ and a positive integer l, let $y^{\otimes l} \in \mathbb{F}_p^{w^l}$ denote the lth tensor power of y. Coordinates of $y^{\otimes l}$ are labelled by all possible sequences in $[w]^l$ and $y^{\otimes l}_{i_1,\ldots,i_l} = \prod_{j=1}^{l} y_{i_j}$. The following identity relating tensor powers and dot products of vectors is standard. For every positive integer l and vectors u, v in $\mathbb{F}_p^w$,

$$\begin{aligned}(u^{\otimes l}, v^{\otimes l}) &= \sum_{(i_1,\ldots,i_l)\in[w]^l} \left(\prod_{j=1}^{l} u_{i_j} \prod_{j=1}^{l} v_{i_j}\right) \\ &= \sum_{(i_1,\ldots,i_l)\in[w]^l} \left(\prod_{j=1}^{l} u_{i_j} v_{i_j}\right) = \left(\sum_{i_1\in[w]} u_{i_1} v_{i_1}\right) \ldots \left(\sum_{i_l\in[w]} u_{i_l} v_{i_l}\right) = (u,v)^l.\end{aligned} \tag{8}$$

We now proceed to the construction of vectors $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_n\}$ in $\mathbb{F}_p^m$. First assume that m has the shape $m = w^{(p-1)/t}$ for some integer $w \geq p-1$. Let $n = \binom{w}{p-1}$. Observe that $n \geq cm^t$ for a suitably chosen constant $c > 0$. For $i \in [n]$ let vectors u'_i in $\mathbb{F}_p^w$ be the incidence vectors of all possible subsets of $[w]$ of cardinality $(p-1)$ and let vectors v'_i be their complements (i.e., for every $i \in [n]$, $v_i = 1^w - u_i$). Observe that

- for all $i \in [n]$, $(u'_i, v'_i) = 0$;
- for all $i, j \in [n]$ such that $i \neq j$, $(u'_j, v'_i) \neq 0$.

Let $l = (p-1)/t$. For $i \in [n]$ set $u_i = u_i'^{\otimes l}$ and $v_i = v_i'^{\otimes l}$. Formula (8) and cyclicity of $\mathbb{F}_p^*$ imply that vectors $\{u_1, \ldots, u_n\}$, $\{v_1, \ldots, v_n\}$ in $\mathbb{F}_p^m$ have the desired properties, i.e.,

- for all $i \in [n]$, $(u_i, v_i) = 0$;
- for all $i, j \in [n]$ such that $i \neq j$, $(u_j, v_i) \in \langle 2 \rangle$.

Now assume that m does not have the right shape, and let m_1 be the largest integer smaller than m that does have it. In order to get vectors of length m we use vectors of length m_1 coming from the construction above padded with zeros. It is not hard to verify that such a construction still gives us $n \geq cm^t$ large families of vectors for a suitably chosen constant c. □

We use the standard notation $\overline{\mathbb{F}}$ to denote the algebraic closure of the field $\mathbb{F}$. Also let

$$C_p = \{x \in \overline{\mathbb{F}}_2^* \mid x^p = 1\}$$

denote the multiplicative subgroup of pth roots of unity in $\overline{\mathbb{F}}_2$. The next lemma generalizes [38, Lemma 14].

LEMMA 7. *Let p be a prime and k be odd. Suppose there exist $\zeta_1, \ldots, \zeta_k \in C_p$ such that*

$$\zeta_1 + \cdots + \zeta_k = 0; \tag{9}$$

then $\langle 2 \rangle \subseteq \mathbb{F}_p^*$ is k-algebraically nice.

Proof. In what follows we define the set $S_1 \subseteq \mathbb{F}_p$ and prove the existence of a set S_0 such that together S_0 and S_1 yield k-algebraic niceness of $\langle 2 \rangle$. Identity (9) and the fact that we are working in characteristic 2 imply that there exist an odd integer $k' \leq k$ and k' *distinct* pth roots of unity $\zeta'_1, \ldots, \zeta'_k \in C_p$ such that

$$\zeta'_1 + \cdots + \zeta'_{k'} = 0. \tag{10}$$

Let $t = \mathrm{ord}_p(2)$. Observe that $C_p \subseteq \mathbb{F}_{2^t}$. Let g be a generator of C_p. Identity (10) yields $g^{\gamma_1} + \cdots + g^{\gamma_{k'}} = 0$ for some distinct values $\{\gamma_i\}_{i\in[k']}$ in $\mathbb{Z}_p$. Set $S_1 = \{\gamma_1, \ldots, \gamma_{k'}\}$.

Consider a natural one-to-one correspondence between subsets S' of $\mathbb{F}_p$ and polynomials $\phi_{S'}(x)$ in the ring $\mathbb{F}_2[x]/(x^p - 1)$: $\phi_{S'}(x) = \sum_{s\in S'} x^s$. It is easy to see that for all sets $S' \subseteq \mathbb{F}_p$ and all $\alpha, \beta \in \mathbb{F}_p$, such that $\beta \neq 0$,

$$\phi_{\alpha+\beta S'}(x) = x^\alpha \phi_{S'}(x^\beta).$$

Let α be a variable ranging over $\mathbb{F}_p$ and β be a variable ranging over $\langle 2\rangle$. We are going to argue the existence of a set S_0 that has even intersections with all sets of the form $\alpha+\beta S_1$, by showing that all polynomials $\phi_{\alpha+\beta S_1}$ belong to a certain linear space $L \subset \mathbb{F}_2[x]/(x^p - 1)$ of dimension less than p. In this case any nonempty set $T \subseteq \mathbb{F}_p$ such that $\phi_T \in L^\perp$ can be used as the set S_0. Let $\tau(x) = \gcd(x^p - 1, \phi_{S_1}(x))$. Note that $\tau(x) \neq 1$ since g is a common root of $x^p - 1$ and $\phi_{S_1}(x)$. Let L be the space of polynomials in $\mathbb{F}_2[x]/(x^p - 1)$ that are multiples of $\tau(x)$. Clearly, $\dim L = p - \deg\tau$. Fix some $\alpha \in \mathbb{F}_p$ and $\beta \in \langle 2\rangle$. Let us prove that $\phi_{\alpha+\beta S_1}(x)$ is in L:

$$\phi_{\alpha+\beta S_1}(x) = x^\alpha \phi_{S_1}(x^\beta) = x^\alpha (\phi_{S_1}(x))^\beta.$$

The last identity above follows from the fact that for any $f \in \mathbb{F}_2[x]$ and any integer i we have $f(x^{2^i}) = (f(x))^{2^i}$. □

In what follows we present sufficient conditions for the existence of k-tuples of pth roots of unity in $\overline{\mathbb{F}}_2$ that sum to zero. We treat the $k = 3$ case separately since in that case we can use a specialized argument to derive a more explicit conclusion.

4.1. A sufficient condition for the existence of three *p*th roots of unity summing to zero.

LEMMA 8. *Let p be an odd prime. Suppose $\mathrm{ord}_p(2) < (4/3)\log_2 p$; then there exist three pth roots of unity in $\overline{\mathbb{F}}_2$ that sum to zero.*

Proof. Let $t = \mathrm{ord}_p(2)$. Note that $C_p \subseteq \mathbb{F}_{2^t}$. Consider the homogeneous equation

$$x_1^{(2^t-1)/p} + x_2^{(2^t-1)/p} + x_3^{(2^t-1)/p} = 0. \tag{11}$$

We count the $\mathbb{F}_{2^t}$-rational solutions to (11) up to multiplication by nonzero scalars. Let N_{2^t} denote the total number of such solutions. Davenport and Hasse [9] proved that

$$\left|N_{2^t} - (2^t + 1)\right| \leq (d-1)(d-2)2^{t/2}, \tag{12}$$

where $d = (2^t - 1)/p$ is the degree of (11). The estimate (12) is also a special case of the Weil bound [20, p. 330].

A solution $a = (x_1, x_2, x_3)$ to (11) is called trivial if one of the coordinates of a is zero. Observe that setting one of x_i, $1 \leq i \leq 3$, to zero in (11) leaves us with an equation $(x_j/x_k)^{(2^t-1)/p} = 1$ in the other two variables. Cyclicity of $\mathbb{F}_{2^t}^*$ implies that such an equation has exactly $(2^t - 1)/p$ solutions in $\mathbb{F}_{2^t}$ up to scalar multiplication. Therefore there are exactly $3(2^t - 1)/p$ trivial solutions to (11). Note that every nontrivial solution yields a triple of elements of C_p that sum to zero.

Identity (12) implies that in the case

$$2^t + 1 > \left(\frac{2^t - 1}{p} - 1\right)\left(\frac{2^t - 1}{p} - 2\right)2^{t/2} + 3\frac{2^t - 1}{p} \tag{13}$$

there exists a nontrivial solution to (11). Note that (13) follows from

$$2^t + 1 > \left(\frac{2^t}{p}\right)\left(\frac{2^t}{p}\right)2^{t/2} - \frac{2^{3t/2+1}}{p} + \frac{3\cdot 2^t}{p}, \tag{14}$$

and (14) follows from

$$2^t > 2^{2t+t/2}/p^2 \quad \text{and} \quad 2^{t/2+1} > 3.$$

Now note that the first inequality above follows from $t < (4/3)\log_2 p$, and the second follows from $t > 1$. □

Note that the constant $4/3$ in Lemma 8 cannot be improved to 2: there are no three elements of $C_{13264529}$ that sum to zero, even though $\mathrm{ord}_2(13264529) = 47 < 2\cdot\log_2 13264529 \approx 47.3$. We briefly discuss our method for verifying this at the end of section 5.3.

4.2. A sufficient condition for the existence of *k* *p*th roots of unity summing to zero. Our argument in this section proceeds in three steps. First we briefly review the notion of (additive) Fourier coefficients of subsets of $\mathbb{F}_{2^t}$. Next, we invoke a folklore argument to show that subsets of $\mathbb{F}_{2^t}$ with appropriately small nontrivial Fourier coefficients contain k-tuples of elements that sum to zero. Finally, we use a recent result of Bourgain and Chang [5] (generalizing the classical estimate for Gauss sums) to argue that (under certain constraints on p) all nontrivial Fourier coefficients of C_p are small.

For $x \in \mathbb{F}_{2^t}$ let $\mathrm{Tr}(x) = x + x^2 + \cdots + x^{2^{t-1}}$ denote the trace of x. It is a standard fact that for all x, $\mathrm{Tr}(x) \in \mathbb{F}_2$. Characters of $\mathbb{F}_{2^t}$ are homomorphisms from the additive group of $\mathbb{F}_{2^t}$ into the multiplicative group $\{\pm 1\}$. There exist 2^t characters. We denote characters by χ_a, where a ranges in $\mathbb{F}_{2^t}$, and set $\chi_a(x) = (-1)^{\mathrm{Tr}(ax)}$ [20]. Let $C(x)$ denote the incidence function of a set $C \subseteq \mathbb{F}_{2^t}$. For arbitrary $a \in \mathbb{F}_2^t$ the Fourier coefficient $\hat{C}(\chi_a)$ is defined by $\hat{C}(\chi_a) = \sum \chi_a(x)C(x)$, where the sum is over all $x \in \mathbb{F}_{2^t}$. Fourier coefficient $\hat{C}(\chi_0) = |C|$ is called trivial, and other Fourier coefficients are called nontrivial. In what follows $\sum_\chi$ stands for summation over all 2^t characters of $\mathbb{F}_{2^t}$. We need the following two standard properties of characters and Fourier coefficients:

$$\sum_\chi \chi(x) = \begin{cases} 2^t & \text{if } x = 0, \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

$$\sum_\chi \hat{C}^2(\chi) = 2^t|C|. \tag{16}$$

The following lemma is folklore.

LEMMA 9. *Let $C \subseteq \mathbb{F}_{2^t}$ and let $k \geq 3$ be a positive integer. Let F be the largest absolute value of a nontrivial Fourier coefficient of C. Suppose*

$$\frac{F}{|C|} < \left(\frac{|C|}{2^t}\right)^{1/(k-2)}; \tag{17}$$

then there exist k elements of C that sum to zero.

Proof. Let $M(C) = \#\{(\zeta_1, \ldots, \zeta_k) \in C^k \mid \zeta_1 + \cdots + \zeta_k = 0\}$. Formula (15) yields

$$M(C) = \frac{1}{2^t}\sum_{x_1,\ldots,x_k \in \mathbb{F}_{2^t}} C(x_1)\ldots C(x_k)\sum_\chi \chi(x_1 + \cdots + x_k). \tag{18}$$

Note that $\chi(x_1+\cdots+x_k)=\chi(x_1)\dots\chi(x_k)$. Changing the order of summation in (18), we get

$$M(C)=\frac{1}{2^t}\sum_{\chi}\sum_{x_1,\dots,x_k\in\mathbb{F}_{2^t}} C(x_1)\dots C(x_k)\chi(x_1)\dots\chi(x_k)=\frac{1}{2^t}\sum_{\chi}\hat{C}^k(\chi). \tag{19}$$

Note that

$$\begin{aligned}\frac{1}{2^t}\sum_{\chi}\hat{C}^k(\chi)&=\frac{|C|^k}{2^t}+\frac{1}{2^t}\sum_{\chi\neq\chi_0}\hat{C}^k(\chi)\\ &\geq\frac{|C|^k}{2^t}-F^{k-2}\frac{1}{2^t}\sum_{\chi}\hat{C}^2(\chi)=\frac{|C|^k}{2^t}-F^{k-2}|C|,\end{aligned} \tag{20}$$

where the last identity follows from (16). Combining (19) and (20) we conclude that (17) implies $M(C)>0$. □

The following lemma is a special case of [5, Theorem 1].

LEMMA 10. *Assume that $n\mid 2^t-1$ and satisfies the condition*

$$\gcd\left(n,\frac{2^t-1}{2^{t'}-1}\right)<2^{t(1-\epsilon)-t'}\quad \textit{for all}\quad 1\leq t'<t,\ t'\mid t,$$

where $\epsilon>0$ is arbitrary and fixed. Then for all $a\in\mathbb{F}_{2^t}^*$

$$\left|\sum_{x\in\mathbb{F}_{2^t}}(-1)^{\mathrm{Tr}(ax^n)}\right|<c_1 2^{t(1-\delta)}, \tag{21}$$

where $\delta=\delta(\epsilon)>0$ and $c_1=c_1(\epsilon)$ are constants.

Below is the main result of this section. Recall that C_p denotes the set of pth roots of unity in $\overline{\mathbb{F}}_2$.

LEMMA 11. *For every $c>0$ there exists an odd integer $k=k(c)$ such that the following implication holds. If p is an odd prime and $\mathrm{ord}_p(2)<c\log_2 p$, then some k elements of C_p sum to zero.*

Proof. Note that if there exist k' elements of a set $C\subseteq\overline{\mathbb{F}}_2$ that sum to zero, where k' is odd, then there exist k elements of C that sum to zero for every odd $k\geq k'$. Also note that the sum of all pth roots of unity is zero. Therefore given c it suffices to prove the existence of an odd $k=k(c)$ that works for all *sufficiently large* p. Let $t=\mathrm{ord}_p(2)$. Observe that $p>2^{t/c}$. Assume that p is sufficiently large so that $t>2c$. Next we show that the precondition of Lemma 10 holds for $n=(2^t-1)/p$ and $\epsilon=1/(2c)$. Let $t'\mid t$ and $1\leq t'<t$. Clearly $\gcd(2^{t'}-1,p)=1$. Therefore

$$\gcd\left(\frac{2^t-1}{p},\frac{2^t-1}{2^{t'}-1}\right)=\frac{2^t-1}{p(2^{t'}-1)}<\frac{2^{t(1-1/c)})}{2^{t'}-1}, \tag{22}$$

where the inequality follows from $p>2^{t/c}$. Clearly, $t>2c$ yields $2^{t/(2c)}/2>1$. Multiplying the right-hand side of (22) by $2^{t/(2c)}/2$ and using $2(2^{t'}-1)\geq 2^{t'}$ we get

$$\gcd\left(\frac{2^t-1}{p},\frac{2^t-1}{2^{t'}-1}\right)<2^{t(1-1/(2c))-t'}. \tag{23}$$

Combining (23) with Lemma 10 we conclude that there exist $\delta > 0$ and c_1 such that for all $a \in \mathbb{F}_{2^t}^*$

$$\left| \sum_{x \in \mathbb{F}_{2^t}} (-1)^{\mathrm{Tr}\left(ax^{(2^t-1)/p}\right)} \right| < c_1 2^{t(1-\delta)}. \tag{24}$$

Observe that $x^{(2^t-1)/p}$ takes every value in C_p exactly $(2^t-1)/p$ times when x ranges over $\mathbb{F}_{2^t}^*$. Therefore the left-hand side of (24) is

$$\left| (-1)^{\mathrm{Tr}(0)} + \frac{2^t-1}{p} \sum_{x \in C_p} (-1)^{\mathrm{Tr}(ax)} \right| = \left| 1 + \frac{2^t-1}{p} \hat{C}_p(\chi_a) \right|. \tag{25}$$

Let $F = \max_{a \in \mathbb{F}_2^*} |\hat{C}_p(\chi_a)|$ denote the largest absolute value of a nontrivial Fourier coefficient of C_p. Combining (24) and (25) we get

$$(2^t-1)(F/p) < c_1 2^{t(1-\delta)} + 1. \tag{26}$$

Assuming that t is sufficiently large, we get

$$(2^t-1)(F/p) < c_2 2^{t(1-\delta)} \tag{27}$$

for a suitably chosen constant c_2. Inequality (27) yields $F/p < (2c_2)2^{-\delta t}$. Pick $k \geq 3$ to be the smallest odd integer such that $(1-1/c)/(k-2) < \delta$. We now have

$$\frac{F}{p} < 2^{-\frac{(1-1/c)t}{(k-2)}} \tag{28}$$

for all sufficiently large values of p. Combining $p > 2^{t/c}$ with (28) we get

$$\frac{F}{|C_p|} < \left(\frac{|C_p|}{2^t} \right)^{1/(k-2)},$$

and the application of Lemma 9 concludes the proof. □

4.3. Summary. In this section we summarize our positive results and show that one does not necessarily need to use Mersenne primes to construct LDCs via the methods of [38]. It suffices to have Mersenne numbers with polynomially large prime factors. Recall that $P(m)$ denotes the largest prime factor of an integer m. Our first theorem gets 3-query LDCs from Mersenne numbers m with prime factors larger than $m^{3/4}$.

THEOREM 12. *Suppose $P(2^t-1) > 2^{0.75t}$; then for every message length n there exists a 3-query LDC of length $\exp(n^{1/t})$.*

Proof. Let $P(2^t-1) = p$. Observe that $p \mid 2^t-1$ and $p > 2^{0.75t}$ yield $\mathrm{ord}_p(2) < (4/3)\log_2 p$. Combining Lemmas 8, 7, and 6 with Proposition 5 we obtain the statement of the theorem. □

As an example application of Theorem 12 one can observe that $P(2^{23}-1) = 178481 > 2^{(3/4)*23} \approx 155872$ yields a family of 3-query LDCs of length $\exp(n^{1/23})$. Theorem 12 immediately yields the following theorem.

THEOREM 13. *Suppose that for infinitely many t we have $P(2^t-1) > 2^{0.75t}$; then for every $\epsilon > 0$ there exists a family of 3-query LDCs of length $\exp(n^{\epsilon})$.*

The next theorem gets constant-query LDCs from Mersenne numbers m with prime factors larger than m^γ for every value of γ.

THEOREM 14. *For every $\gamma > 0$ there exists an odd integer $k = k(\gamma)$ such that the following implication holds. Suppose that $P(2^t - 1) > 2^{\gamma t}$; then for every message length n there exists a k-query LDC of length $\exp(n^{1/t})$.*

Proof. Let $P(2^t - 1) = p$. Observe that $p \mid 2^t - 1$ and $p > 2^{\gamma t}$ yield $\mathrm{ord}_p(2) < (1/\gamma)\log_2 p$. Combining Lemmas 11, 7, and 6 with Proposition 5 we obtain the statement of the theorem. ∎

As an immediate corollary we get the following theorem.

THEOREM 15. *Suppose for some $\gamma > 0$ and infinitely many t we have $P(2^t - 1) > 2^{\gamma t}$; then there is a fixed k such that for every $\epsilon > 0$ there exists a family of k-query LDCs of length $\exp(n^\epsilon)$.*

5. Nice subsets of finite fields yield Mersenne numbers with large prime factors. We start the section with the following definition.

DEFINITION 16. *We say that a sequence $\{S_i \subseteq \mathbb{F}_{q_i}^*\}_{i\geq 1}$ of subsets of finite fields is k-nice if every S_i is k-algebraically nice and $t(i)$-combinatorially nice, for some integer valued strictly increasing function t.*

The core Proposition 5 asserts that a subset $S \subseteq \mathbb{F}_q^*$ that is k-algebraically nice and t-combinatorially nice yields a family of k-query LDCs of length $\exp(n^{1/t})$. Clearly, to get k-query LDCs of length $\exp(n^\epsilon)$ for some fixed k and every $\epsilon > 0$ via this proposition, one needs to exhibit a k-nice sequence. In this section we show how the existence of a k-nice sequence implies that infinitely many Mersenne numbers have large prime factors.

Our argument proceeds in two steps. In section 5.1 we show that a k-nice sequence yields an infinite sequence of primes $\{p_i\}_{i\geq 1}$, where every C_{p_i} contains a k-tuple of elements summing to zero. In sections 5.2 and 5.3 we show that C_p contains a (nontrivial) short additive dependence only if p is a large factor of a Mersenne number.

5.1. A nice sequence yields infinitely many primes p with short dependencies between pth roots of unity. Our argument in this section proceeds in three steps. First we study algebraically nice subsets of finite fields. Second we study combinatorially nice subsets of finite fields. Third we show how an interplay between the structural properties of algebraically and combinatorially nice subsets translates nice sequences into infinite families of primes p with short nontrivial additive dependencies in C_p.

Algebraically nice subsets of $\mathbb{F}_q^*$. We start with some notation. Consider a finite field $\mathbb{F}_q = \mathbb{F}_{p^l}$, where p is prime. Fix a basis $e_1, \ldots, e_l$ of $\mathbb{F}_q$ over $\mathbb{F}_p$. In what follows we often write $(\alpha_1, \ldots, \alpha_l) \in \mathbb{F}_p^l$ to denote $\alpha = \sum_{i=1}^l \alpha_i e_i \in \mathbb{F}_q$. Let R denote the ring $\mathbb{F}_2[x_1, \ldots, x_l]/(x_1^p - 1, \ldots, x_l^p - 1)$. Consider a natural one-to-one correspondence between subsets S_1 of $\mathbb{F}_q$ and polynomials $\phi_{S_1}(x_1, \ldots, x_l) \in R$:

$$\phi_{S_1}(x_1, \ldots, x_l) = \sum_{(\alpha_1,\ldots,\alpha_l)\in S_1} x_1^{\alpha_1} \ldots x_l^{\alpha_l}.$$

It is easy to see that for all sets $S_1 \subseteq \mathbb{F}_q$ and all $\alpha, \beta \in \mathbb{F}_q$

$$\phi_{(\alpha_1,\ldots,\alpha_l)+\beta S_1}(x_1, \ldots, x_l) = x_1^{\alpha_1} \ldots x_l^{\alpha_l} \phi_{\beta S_1}(x_1, \ldots, x_l). \tag{29}$$

Let Γ be a family of subsets of $\mathbb{F}_q$. It is straightforward to verify that a set $S_0 \subseteq \mathbb{F}_q$ has even intersections with every element of Γ if and only if ϕ_{S_0} belongs to $L^\perp$, where L is

the linear subspace of R spanned by $\{\phi_{S_1}\}_{S_1\in\Gamma}$. Combining the last observation with formula (29) we conclude that a set $S \subseteq \mathbb{F}_q^*$ is k-algebraically nice if and only if there exists a set $S_1 \subseteq \mathbb{F}_q$ of odd size $k' \leq k$ such that the ideal generated by polynomials $\{\phi_{\beta S_1}\}_{\{\beta\in S\}}$ is a proper ideal of R. Note that polynomials $\{f_1, \ldots, f_h\} \in R$ generate a proper ideal if and only if polynomials $\{f_1, \ldots, f_h, x_1^p - 1, \ldots, x_l^p - 1\}$ generate a proper ideal in $\mathbb{F}_2[x_1, \ldots, x_l]$. Also note that a family of polynomials generates a proper ideal in $\mathbb{F}_2[x_1, \ldots, x_l]$ if and only if it generates a proper ideal in $\overline{\mathbb{F}}_2[x_1, \ldots, x_l]$. Now an application of Hilbert's Nullstellensatz [8, p. 168] implies that a set $S \subseteq \mathbb{F}_q^*$ is k-algebraically nice if and only if there is a set $S_1 \subseteq \mathbb{F}_q$ of odd size $k' \leq k$ such that the polynomials $\{\phi_{\beta S_1}\}_{\{\beta\in S\}}$ and $\{x_i^p - 1\}_{1\leq i\leq l}$ have a common root in $\overline{\mathbb{F}}_2$.

LEMMA 17. *Let $\mathbb{F}_q = \mathbb{F}_{p^l}$, where p is prime. Suppose $\mathbb{F}_q$ contains a nonempty k-algebraically nice subset; then there exist $\zeta_1, \ldots, \zeta_k \in C_p$ such that $\zeta_1 + \cdots + \zeta_k = 0$.*

Proof. Assume that $S \subseteq \mathbb{F}_q^*$ is nonempty and k-algebraically nice. The discussion above implies that there exists $S_1 \subseteq \mathbb{F}_q$ of odd size $k' \leq k$ such that all polynomials $\{\phi_{\beta S_1}\}_{\{\beta\in S\}}$ vanish at some $(\zeta_1, \ldots, \zeta_l) \in C_p^l$. Fix an arbitrary $\beta_0 \in S$, and note that C_p is closed under multiplication. Thus each of the k' monomials of $\phi_{\beta_0 S_1}$ is a root of unity when evaluated at $(\zeta_1, \ldots, \zeta_l)$, and

$$\phi_{\beta_0 S_1}(\zeta_1, \ldots, \zeta_l) = 0 \tag{30}$$

yields k' pth roots of unity that sum to zero. It is readily seen that (since we are working in characteristic 2) one can extend (30) by adding an appropriate number of pairs of identical roots to obtain k pth roots of unity that sum to zero for any odd $k \geq k'$. □

Note that Lemma 17 does not suffice to prove that a k-nice sequence $\{S_i \subseteq \mathbb{F}_{q_i}^*\}_{i\geq 1}$ yields infinitely many primes p with short (nontrivial) additive dependencies in C_p. We need to argue that the set $\{\operatorname{char} \mathbb{F}_{q_i}\}_{i\geq 1}$ cannot be finite. This is our goal for the remainder of the section.

To proceed, we need some more notation. Recall that $q = p^l$ and p is prime. For $x \in \mathbb{F}_q$ let $\operatorname{Tr}(x) = x + \cdots + x^{p^{l-1}} \in \mathbb{F}_p$ denote the (absolute) trace of x. For $\gamma \in \mathbb{F}_q$, $c \in \mathbb{F}_p^*$ we call the set $\pi_{\gamma,c} = \{x \in \mathbb{F}_q \mid \operatorname{Tr}(\gamma x) = c\}$ a *proper-affine hyperplane* of $\mathbb{F}_q$.

LEMMA 18. *Let $\mathbb{F}_q = \mathbb{F}_{p^l}$, where p is prime. Suppose $S \subseteq \mathbb{F}_q^*$ is k-algebraically nice; then there exist $h \leq p^k$ proper-affine hyperplanes $\{\pi_{\gamma_r,c_r}\}_{1\leq r\leq h}$ of $\mathbb{F}_q$ such that $S \subseteq \bigcup_{r=1}^h \pi_{\gamma_r,c_r}$.*

Proof. The discussion preceding Lemma 17 implies that there exists a set $S_1 = \{\sigma_1, \ldots, \sigma_{k'}\} \subseteq \mathbb{F}_q$ of odd size $k' \leq k$ such that all polynomials $\{\phi_{\beta S_1}\}_{\{\beta\in S\}}$ vanish at some $(\zeta_1, \ldots, \zeta_l) \in C_p^l$. Let ζ be a generator of C_p. For every $1 \leq i \leq l$, pick $\omega_i \in \mathbb{Z}_p$ such that $\zeta_i = \zeta^{\omega_i}$. For every $\beta \in S$, $\phi_{\beta S_1}(\zeta_1, \ldots, \zeta_l) = 0$ yields

$$\sum_{\mu=(\mu_1,\ldots,\mu_l)\in\beta S_1} \zeta^{\sum_{i=1}^l \mu_i\omega_i} = 0. \tag{31}$$

Observe that for fixed values $\{\omega_i\}_{1\leq i\leq l} \in \mathbb{Z}_p$ the map $D(\mu) = \sum_{i=1}^l \mu_i\omega_i$ is a linear map from $\mathbb{F}_q$ to $\mathbb{F}_p$. It is a standard fact that every such map can be expressed as $D(\mu) = \operatorname{Tr}(\delta\mu)$ for an appropriate choice of $\delta \in \mathbb{F}_q$ [20]. Therefore we can rewrite (31) as

$$\sum_{\mu\in\beta S_1} \zeta^{\operatorname{Tr}(\delta\mu)} = \sum_{\sigma\in S_1} \zeta^{\operatorname{Tr}(\delta\beta\sigma)} = 0. \tag{32}$$

Let $W = \left\{(w_1, \dots, w_{k'}) \in \mathbb{Z}_p^{k'} \;\middle|\; \zeta^{w_1} + \dots + \zeta^{w_{k'}} = 0\right\}$ denote the set of exponents of k'-dependencies between powers of ζ. Clearly, $|W| \le p^k$. Identity (32) implies that every $\beta \in S$ satisfies

$$\begin{cases} \operatorname{Tr}((\delta\sigma_1)\beta) &= w_1, \\ \vdots & \\ \operatorname{Tr}((\delta\sigma_{k'})\beta) &= w_{k'} \end{cases} \tag{33}$$

for an appropriate choice of $(w_1, \dots, w_{k'}) \in W$. Note that the all-zeros vector does not lie in W since k' is odd and the characteristic is even. Therefore at least one of the identities in (33) has a nonzero right-hand side and defines a proper-affine hyperplane of $\mathbb{F}_q$. Collecting one such hyperplane for every element of W we get a family of $|W|$ proper-affine hyperplanes containing every element of S. □

Combinatorially nice subsets of $\mathbb{F}_q^*$. Lemma 18 gives us some insight into the structure of algebraically nice subsets of $\mathbb{F}_q$. Our next goal is to develop an insight into the structure of combinatorially nice subsets. We start by reviewing some relations between tensor and dot products of vectors. For vectors $u \in \mathbb{F}_q^m$ and $v \in \mathbb{F}_q^n$ let $u \otimes v \in \mathbb{F}_q^{mn}$ denote the tensor product of u and v. Coordinates of $u \otimes v$ are labelled by all possible elements of $[m] \times [n]$ and $(u \otimes v)_{i,j} = u_i v_j$. Also, let $u^{\otimes l}$ denote the lth tensor power of u and let $u \circ v \in \mathbb{F}_q^{m+n}$ denote the concatenation of u and v. The following identity is standard. For any $u, x \in \mathbb{F}_q^m$ and $v, y \in \mathbb{F}_q^n$,

$$(u \otimes v, x \otimes y) = \sum_{i \in [m],\, j \in [n]} u_i v_j x_i y_j = \left(\sum_{i \in [m]} u_i x_i\right)\left(\sum_{j \in [n]} v_j y_j\right) = (u, x)(v, y). \tag{34}$$

In what follows we need a generalization of identity (34). First we let $f(x_1, \dots, x_h) = \sum_i c_i x_1^{\alpha_1^{(i)}} \dots x_h^{\alpha_h^{(i)}}$ be a polynomial in $\mathbb{F}_q[x_1, \dots, x_h]$. Then we define $\bar{f} \in \mathbb{F}_q[x_1, \dots, x_h]$ by $\bar{f} = \sum_{i : c_i \ne 0} x_1^{\alpha_1^{(i)}} \dots x_h^{\alpha_h^{(i)}}$; i.e., we simply set all nonzero coefficients of f to 1. Assume that f has no free term. For vectors $u_1, \dots, u_h$ in $\mathbb{F}_q^m$ define

$$f(u_1, \dots, u_h) = \dots \circ \left(c_i u_1^{\otimes \alpha_1^{(i)}} \otimes \dots \otimes u_h^{\otimes \alpha_h^{(i)}}\right) \circ \dots, \tag{35}$$

where the concatenation is over all values of i. Note that to obtain $f(u_1, \dots, u_h)$ we replaced products in f by tensor products and replaced addition by concatenation. Clearly, $f(u_1, \dots, u_h)$ is a vector whose length may be larger than m.

LEMMA 19. *For every $f \in \mathbb{F}_q[x_1, \dots, x_h]$ and $u_1, \dots, u_h, v_1, \dots, v_h \in \mathbb{F}_q^m$,*

$$\left(f(u_1, \dots, u_h), \bar{f}(v_1, \dots, v_h)\right) = f((u_1, v_1), \dots, (u_h, v_h)). \tag{36}$$

Proof. Let $\mathbf{u} = (u_1, \dots, u_h)$ and $\mathbf{v} = (v_1, \dots, v_h)$. Observe that if (36) holds for polynomials f_1 and f_2 defined over disjoint sets of monomials, then it also holds for $f = f_1 + f_2$:

$$\begin{aligned} \left(f(\mathbf{u}), \bar{f}(\mathbf{v})\right) &= \left((f_1 + f_2)(\mathbf{u}), (\bar{f}_1 + \bar{f}_2)(\mathbf{v})\right) = \left(f_1(\mathbf{u}) \circ f_2(\mathbf{u}), \bar{f}_1(\mathbf{v}) \circ \bar{f}_2(\mathbf{v})\right) \\ &= f_1\left((u_1, v_1), \dots, (u_h, v_h)\right) + f_2\left((u_1, v_1), \dots, (u_h, v_h)\right) = f\left((u_1, v_1), \dots, (u_h, v_h)\right). \end{aligned}$$

Therefore it suffices to prove (36) for monomials $f = c x_1^{\alpha_1} \dots x_h^{\alpha_h}$. It remains to notice that identity (36) for monomials $f = c x_1^{\alpha_1} \dots x_h^{\alpha_h}$ follows immediately from formula (34) using induction on $\sum_{i=1}^h \alpha_i$. □

The next lemma bounds combinatorial niceness of certain subsets of $\mathbb{F}_q^*$.

LEMMA 20. *Let $\mathbb{F}_q = \mathbb{F}_{p^l}$, where p is prime. Let $S \subseteq \mathbb{F}_q^*$. Suppose there exist h proper-affine hyperplanes $\{\pi_{\gamma_r,c_r}\}_{1\le r\le h}$ of $\mathbb{F}_q$ such that $S \subseteq \bigcup_{r=1}^{h} \pi_{\gamma_r,c_r}$; then S is at most $h(p-1)$-combinatorially nice.*

Proof. Assume S is t-combinatorially nice. This implies that for some $c > 0$ and every m there exist two $n = \lfloor cm^t \rfloor$-sized collections of vectors $\{u_i\}_{i\in[n]}$ and $\{v_i\}_{i\in[n]}$ in $\mathbb{F}_q^m$, such that

- for all $i \in [n]$, $(u_i, v_i) = 0$;
- for all $i, j \in [n]$ such that $i \neq j$, $(u_j, v_i) \in S$.

For a vector $u \in \mathbb{F}_q^m$ and integer e let u^e denote the vector resulting from raising every coordinate of u to the power e. For every $i \in [n]$ and $r \in [h]$ define vectors $u_i^{(r)}$ and $v_i^{(r)}$ in $\mathbb{F}_q^{ml}$ by

$$u_i^{(r)} = (\gamma_r u_i) \circ (\gamma_r u_i)^p \circ \cdots \circ (\gamma_r u_i)^{p^{l-1}} \quad \text{and} \quad v_i^{(r)} = v_i \circ v_i^p \circ \cdots \circ v_i^{p^{l-1}}. \tag{37}$$

Note that for every $r_1, r_2 \in [h]$, $v_i^{(r_1)} = v_i^{(r_2)}$. It is straightforward to verify that for every $i, j \in [n]$ and $r \in [h]$,

$$\left(u_j^{(r)}, v_i^{(r)}\right) = \mathrm{Tr}(\gamma_r(u_j, v_i)). \tag{38}$$

Combining formula (38) with $\mathrm{Tr}(0) = 0$ and using the fact that the set S is covered by proper-affine hyperplanes $\{\pi_{\gamma_r,c_r}\}_{r\in[h]}$ we conclude that

- for all $i \in [n]$ and $r \in [h]$, $(u_i^{(r)}, v_i^{(r)}) = 0$;
- for all $i, j \in [n]$ such that $i \neq j$, there exists $r \in [h]$ such that $(u_j^{(r)}, v_i^{(r)}) \in \mathbb{F}_p^*$.

Pick $g(x_1, \ldots, x_h) \in \mathbb{F}_p[x_1, \ldots, x_h]$ to be a homogeneous degree h polynomial such that for $\mathbf{a} = (a_1, \ldots, a_h) \in \mathbb{F}_p^h : g(\mathbf{a}) = 0$ if and only if $\mathbf{a}$ is the all-zeros vector. The existence of such a polynomial g follows from [20, Example 6.7]. Set $f = g^{p-1}$. Note that for $\mathbf{a} \in \mathbb{F}_p^h : f(a) = 0$ if $\mathbf{a}$ is the all-zeros vector, and $f(a) = 1$ otherwise. For all $i \in [n]$ define

$$u_i' = f\left(u_i^{(1)}, \ldots, u_i^{(h)}\right) \circ (1) \quad \text{and} \quad v_i' = \bar{f}\left(v_i^{(1)}, \ldots, v_i^{(h)}\right) \circ (-1). \tag{39}$$

Note that f and $\bar{f}$ are homogeneous degree $(p-1)h$ polynomials in h variables. Therefore (35) implies that, for all i vectors, u_i' and v_i' have $m' \le h^{(p-1)h}(ml)^{(p-1)h} + 1$ coordinates. Combining identities (39) and (36) and using the properties of dot products between vectors $\{u_i^{(r)}\}$ and $\{v_i^{(r)}\}$ discussed above, we conclude that for every m there exist two $n = \lfloor cm^t \rfloor$-sized collections of vectors $\{u_i'\}_{i\in[n]}$ and $\{v_i'\}_{i\in[n]}$ in $\mathbb{F}_q^{m'}$ such that

- for all $i \in [n]$, $(u_i', v_i') = -1$;
- for all $i, j \in [n]$ such that $i \neq j$, $(u_j', v_i') = 0$.

It remains to notice that a family of vectors with such properties exists only if $n \le m'$, i.e.,

$$\lfloor cm^t \rfloor \le h^{(p-1)h}(ml)^{(p-1)h} + 1.$$

Given that we can pick m to be arbitrarily large, this implies that $t \le (p-1)h$. ∎

Summary. The next lemma presents the main result of this section.

LEMMA 21. *Let k be an odd integer. Suppose there exists a k-nice sequence; then for infinitely many primes p some k elements of C_p sum to zero.*

Proof. Assume that $\{S_i \subseteq \mathbb{F}_{q_i}^*\}_{i\geq 1}$ is k-nice. Let p be a fixed prime. Combining Lemmas 18 and 20 we conclude that every k-algebraically nice subset $S \subseteq \mathbb{F}_{p^l}^*$ is at most $(p-1)p^k$-combinatorially nice. Note that our bound on combinatorial niceness is independent of l. Therefore there are only finitely many extensions of the field $\mathbb{F}_p$ in the sequence $\{\mathbb{F}_{q_i}\}_{i\geq 1}$, and the set $\mathbb{P} = \{\operatorname{char}\mathbb{F}_{q_i}\}_{i\geq 1}$ is infinite. It remains to notice that according to Lemma 17 for every $p \in \mathbb{P}$ there exist k elements of C_p that sum to zero. ∎

In what follows we present necessary conditions for the existence of k-tuples of pth roots of unity in $\overline{\mathbb{F}}_2$ that sum to zero. We treat the $k = 3$ case separately since in that case we can use a specialized argument to derive a slightly stronger conclusion.

5.2. A necessary condition for the existence of k pth roots of unity summing to zero. We start the section with the following lemma.

LEMMA 22. *Let $k \geq 3$ be odd and p be a prime. Suppose there exist $\zeta_1, \dots, \zeta_k \in C_p$ such that $\sum_{i=1}^k \zeta_i = 0$; then*

$$\operatorname{ord}_p(2) \leq 2p^{1-1/(k-1)}. \tag{40}$$

Proof. Let $t = \operatorname{ord}_p(2)$. Note that $C_p \subseteq \mathbb{F}_{2^t}$. Note also that no element of C_p other than the multiplicative identity falls into a proper subfield of $\mathbb{F}_{2^t}$. Therefore, for every $\zeta \in C_p$ where $\zeta \neq 1$ and every nonzero $f(x) \in \mathbb{F}_2[x]$ such that $\deg f \leq t-1$, we have $f(\zeta) \neq 0$.

By multiplying $\sum_{i=1}^k \zeta_i = 0$ through by ζ_k^{-1}, we may reduce to the case $\zeta_k = 1$. Let ζ be a generator of C_p. For every $i \in [k-1]$ pick $w_i \in \mathbb{Z}_p$ such that $\zeta_i = \zeta^{w_i}$. We now have $\sum_{i=1}^{k-1} \zeta^{w_i} + 1 = 0$. Set $h = \lfloor (t-1)/2 \rfloor$. In what follows we interchangeably treat variables $\{i_l, i_l', j_l\}_{l\in[k-1]}$ that take integer values in the range $[-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$ as taking values in $\mathbb{Z}$ or in $\mathbb{Z}_p$. Consider the $(k-1)$-tuples

$$(mw_1 + i_1, \dots, mw_{k-1} + i_{k-1}) \in \mathbb{Z}_p^{k-1} \quad \text{for} \quad m \in \mathbb{Z}_p \quad \text{and} \quad i_1, \dots, i_{k-1} \in [0, h]. \tag{41}$$

Suppose two of these coincide, say

$$(mw_1 + i_1, \dots, mw_{k-1} + i_{k-1}) = (m'w_1 + i_1', \dots, m'w_{k-1} + i_{k-1}')$$

with $(m, i_1, \dots, i_{k-1}) \neq (m', i_1', \dots, i_{k-1}')$. Set $n = m - m'$ and $j_l = i_l' - i_l$ for $l \in [k-1]$. We now have

$$(nw_1, \dots, nw_{k-1}) = (j_1, \dots, j_{k-1})$$

with $-h \leq j_1, \dots, j_{k-1} \leq h$. Observe that $n = 0$ would imply $m = m'$ and $i_l = i_l'$ for all $l \in [k-1]$. Therefore $n \neq 0$, and there exists a $g \in \mathbb{Z}_p$ such that $ng = 1$ modulo p. Consider a polynomial

$$P(z) = z^{j_1+h} + \dots + z^{j_{k-1}+h} + z^h \in \mathbb{F}_2[z].$$

Note that $\deg P \leq 2h \leq t-1$. Note also that $P(1) = 1$ and $P(\zeta^g) = 0$. The latter identity contradicts the fact that ζ^g is a proper element of $\mathbb{F}_{2^t}$. This contradiction implies that all $(k-1)$-tuples in (41) are distinct. This yields

$$p^{k-1} \geq p\left(\frac{t}{2}\right)^{k-1},$$

which is equivalent to (40). ∎

5.3. A necessary condition for the existence of three pth roots of unity summing to zero. In this section we slightly strengthen Lemma 22 in the special case when $k = 3$. Our argument is loosely inspired by the Agrawal–Kayal–Saxena deterministic primality test [1].

LEMMA 23. *Let p be a prime. Suppose there exist $\zeta_1, \zeta_2, \zeta_3 \in C_p$ that sum up to zero; then*

$$\operatorname{ord}_p(2) \le ((4/3)p)^{1/2}. \tag{42}$$

Proof. Let $t = \operatorname{ord}_p(2)$. Note that $C_p \subseteq \mathbb{F}_{2^t}$. Note also that no element of C_p other than the multiplicative identity falls into a proper subfield of $\mathbb{F}_{2^t}$. Therefore, for every $\zeta \in C_p$ where $\zeta \neq 1$ and every nonzero $f(x) \in \mathbb{F}_2[x]$ such that $\deg f \le t-1$, we have $f(\zeta) \neq 0$.

Observe that $\zeta_1+\zeta_2+\zeta_3 = 0$ implies $\zeta_1\zeta_2^{-1}+1 = \zeta_3\zeta_2^{-1}$. This yields $(\zeta_1\zeta_2^{-1}+1)^p = 1$. Put $\zeta = \zeta_1\zeta_2^{-1}$. Note that $\zeta \neq 1$ and $\zeta, 1+\zeta \in C_p$. Consider the products $\pi_{i,j} = \zeta^i(1+\zeta)^j \in C_p$ for $0 \le i, j \le t-1$. Note that $\pi_{i,j}, \pi_{k,l}$ cannot be the same if $i \ge k$, $l \ge j$, and $(i,j) \neq (k,l)$ as then ζ is a root of a polynomial

$$f(z) = z^{i-k} - (1+z)^{l-j},$$

which has degree less than t. In other words, if $\pi_{i,j} = \pi_{k,l}$ and $(i,j) \neq (k,l)$, then the pairs (i,j) and (k,l) are comparable under termwise comparison. In particular, either $(k,l) = (i+a,\, j+b)$ or $(i,j) = (k+a,\, l+b)$ for some pair (a,b) with $\pi_{a,b} = 1$.

We next check that there cannot be two distinct nonzero pairs $(a,b), (a',b')$ with $\pi_{a,b} = \pi_{a',b'} = 1$. As shown above, these pairs must be comparable; we may assume without loss of generality that $a \le a'$, $b \le b'$. The equations $\pi_{a,b} = 1$ and $\pi_{a'-a,\,b'-b} = 1$ force $a+b \ge t$ and $(a'-a)+(b'-b) \ge t$, so $a'+b' \ge 2t$. But $a', b' \le t-1$, a contradiction.

If there is no nonzero pair (a,b) with $0 \le a, b \le t-1$ and $\pi_{a,b} = 1$, then all $\pi_{i,j}$ are distinct, so $p \ge t^2$. Otherwise, as above, the pair (a,b) is unique, and the products $\pi_{i,j}$ with $0 \le i, j \le t-1$ and $(i,j) \not\ge (a,b)$ are pairwise distinct. The number of pairs excluded by the condition $(i,j) \not\ge (a,b)$ is $(t-a)(t-b)$; since $a+b \ge t$, $(t-a)(t-b) \le t^2/4$. Hence $p \ge t^2 - t^2/4 = 3t^2/4$ as desired. ∎

While the necessary condition given by Lemma 23 is quite far away from the sufficient condition given by Lemma 8, it nonetheless suffices for checking that for most primes p, there do not exist three pth roots of unity summing to zero. For instance, among the 664578 odd primes $p \le 10^8$, all but 550 are ruled out by Lemma 23. (There is an easy argument that t must be odd if $p > 3$; this cuts the list down to 273 primes.) Each remaining p can be tested by computing $\gcd(x^p+1,\, (x+1)^p+1)$; the only examples we found that did not satisfy the condition of Lemma 8 were $(p,t) = (73,9), (262657,27), (599479,33), (121369,39)$.

5.4. Summary. In the beginning of section 5 we argued that in order to use the method of [38] (i.e., Proposition 5) to obtain k-query LDCs of length $\exp(n^\epsilon)$ for some fixed k and all $\epsilon > 0$, one needs to exhibit a k-nice sequence of subsets of finite fields. In what follows we use technical results of the previous subsections to show that the existence of a k-nice sequence implies that infinitely many Mersenne numbers have large prime factors.

THEOREM 24. *Let k be odd. Suppose there exists a k-nice sequence of subsets of finite fields; then for infinitely many values of t we have*

$$P(2^t - 1) \ge (t/2)^{1+1/(k-2)}. \tag{43}$$

Proof. Using Lemmas 21 and 22 we conclude that a k-nice sequence yields infinitely many primes p such that $\mathrm{ord}_p(2) \le 2p^{1-1/(k-1)}$. Let p be such a prime and $t = \mathrm{ord}_p(2)$. Then $P(2^t - 1) \ge (t/2)^{1+1/(k-2)}$. □

A combination of Lemmas 21 and 23 yields a slightly stronger bound for the special case of 3-nice sequences.

THEOREM 25. *Suppose there exists a 3-nice sequence of subsets; then for infinitely many values of t we have*

$$P(2^t - 1) \ge (3/4)t^2. \tag{44}$$

We would like to remind the reader that although the lower bounds for $P(2^t - 1)$ given by (43) and (44) are extremely weak in light of the widely accepted conjecture stating that the number of Mersenne primes is infinite, they are substantially stronger than what is currently known unconditionally (2).

Acknowledgments. Sergey Yekhanin would like to thank Swastik Kopparty for providing the reference [5] and outlining the proof of Lemma 9. He would also like to thank Henryk Iwaniec, Carl Pomerance, and Peter Sarnak for their feedback regarding the number theory problems discussed in this paper. The authors would like to thank the reviewers for valuable comments.

REFERENCES

[1] M. AGRAWAL, N. KAYAL, AND N. SAXENA, *PRIMES is in P*, Ann. of Math. (2), 160 (2004), pp. 781–793.

[2] L. BABAI, L. FORTNOW, L. LEVIN, AND M. SZEGEDY, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd ACM Symposium on Theory of Computing (STOC), 1991, pp. 21–31.

[3] A. BEIMEL, Y. ISHAI, AND E. KUSHILEVITZ, *General constructions for information-theoretic private information retrieval*, J. Comput. System Sci., 71 (2005), pp. 213–247.

[4] A. BEIMEL, Y. ISHAI, E. KUSHILEVITZ, AND J. F. RAYMOND, *Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS), 2002, pp. 261–270.

[5] J. BOURGAIN AND M. CHANG, *A Gauss sum estimate in arbitrary finite fields*, C. R. Math. Acad. Sci. Paris, 342 (2006), pp. 643–646.

[6] B. CHOR, O. GOLDREICH, E. KUSHILEVITZ, AND M. SUDAN, *Private information retrieval*, J. ACM, 45 (1998), pp. 965–982.

[7] C. COOPER AND S. BOONE, http://www.mersenne.org/32582657.htm.

[8] D. COX, J. LITTLE, AND D. O'SHEA, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer-Verlag, New York, 1992.

[9] H. DAVENPORT AND H. HASSE, *Die Nullstellen der Kongruenz Zetafunktion in gewissen zyklischen Fallen*, J. Reine Angew. Math., 172 (1935), pp. 151–182.

[10] A. DESHPANDE, R. JAIN, T. KAVITHA, S. LOKAM, AND J. RADHAKRISHNAN, *Better lower bounds for locally decodable codes*, in Proceedings of the 17th Annual IEEE Conference on Computational Complexity (CCC), 2002, pp. 184–193.

[11] P. ERDOS AND T. SHOREY, *On the greatest prime factor of $2^p - 1$ for a prime p and other expressions*, Acta. Arith., 30 (1976), pp. 257–265.

[12] K. FORD, F. LUCA, AND I. SHPARLINSKI, *On the Largest Prime Factor of the Mersenne Numbers*, preprint, 2007; available online from http://www.arxiv.org/abs/0704.1327.

[13] W. GASARCH, *A survey on private information retrieval*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 82 (2004), pp. 72–107.

[14] O. GOLDREICH, *Short Locally Testable Codes and Proofs*, Report TR05-014, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[15] O. GOLDREICH, H. KARLOFF, L. SCHULMAN, AND L. TREVISAN, *Lower bounds for locally decodable codes and private information retrieval*, in Proceedings of the 17th Annual IEEE Conference on Computational Complexity (CCC), 2002, pp. 175–183.

[16] B. Hemenway and R. Ostrovsky, *Public Key Encryption Which Is Simultaneously a Locally-decodable Error-correcting Code*, Cryptology ePrint Archive, Report 2007/083.
[17] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error-correcting codes*, in Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 80–86.
[18] K. Kedlaya and S. Yekhanin, *Locally decodable codes from nice subsets of finite fields and prime factors of Mersenne numbers*, in Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC), 2008, pp. 175–186.
[19] I. Kerenidis and R. de Wolf, *Exponential lower bound for 2-query locally decodable codes via a quantum argument*, J. Comput. System Sci., 69 (2004), pp. 395–420.
[20] R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge University Press, Cambridge, UK, 1983.
[21] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, North–Holland, Amsterdam, 1977.
[22] L. Murata and C. Pomerance, *On the largest prime factor of a Mersenne number*, in Number Theory, CRM Proc. Lecture Notes 36, AMS, Providence, RI, 2004, pp. 209–218.
[23] M. Murty and S. Wong, *The ABC conjecture and prime divisors of the Lucas and Lehmer sequences*, in Proceedings of the Millennial Conference on Number Theory, III (Urbana, IL, 2000), A. K. Peters, Natick, MA, 2002, pp. 43–54.
[24] K. Obata, *Optimal lower bounds for 2-query locally decodable linear codes*, in Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 2483, Springer-Verlag, Berlin, 2002, pp. 39–50.
[25] A. Polishchuk and D. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC), 1994, pp. 194–203.
[26] C. Pomerance, *Recent developments in primality testing*, Math. Intelligencer, 3 (1980/81), pp. 97–105.
[27] P. Raghavendra, *A Note on Yekhanin's Locally Decodable Codes*, Report TR07-016, Electronic Colloquium on Computational Complexity, 2007.
[28] A. Romashchenko, *Reliable computations based on locally decodable codes*, in Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 3884, Springer-Verlag, Berlin, 2006, pp. 537–548.
[29] A. Schinzel, *On primitive prime factors of $a^n - b^n$*, Proc. Cambridge Philos. Soc., 58 (1962), pp. 555–562.
[30] C. Stewart, *The greatest prime factor of $a^n - b^n$*, Acta Arith., 26 (1974/75), pp. 427–433.
[31] C. Stewart, *On divisors of Fermat, Fibonacci, Lucas, and Lehmer numbers*, Proc. London Math. Soc. (3), 35 (1977), pp. 425–447.
[32] M. Sudan, *Efficient Checking of Polynomials and the Hardness of Approximation Problems*, Springer-Verlag, New York, 1995.
[33] L. Trevisan, *Some applications of coding theory in computational complexity*, in Complexity of Computations and Proofs, Quad. Mat. 13, Department of Mathematics, Seconda Univ. Napoli, Caserta, Italy, 2004, pp. 347–424.
[34] S. Wagstaff, *Divisors of Mersenne numbers*, Math. Comp., 40 (1983), pp. 385–397.
[35] S. Wehner and R. de Wolf, *Improved lower bounds for locally decodable codes and private information retrieval*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), Lecture Notes in Comput. Sci. 3580, Springer-Verlag, Berlin, pp. 1424–1436.
[36] D. Woodruff, *Corruption and recovery-efficient locally decodable codes*, in Proceedings of the 12th RANDOM, 2008, pp. 584–595.
[37] D. Woodruff, *New Lower Bounds for General Locally Decodable Codes*, Report TR07-006, Electronic Colloquium on Computational Complexity, 2007.
[38] S. Yekhanin, *Towards 3-query locally decodable codes of subexponential length*, J. ACM, 55 (2008), pp. 1–14.
[39] S. Yekhanin, *Locally Decodable Codes and Private Information Retrieval Schemes*, Ph.D. thesis, MIT, Cambridge, MA, 2007.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1970–1986

THE COMPLEXITY OF WEIGHTED BOOLEAN #CSP*

MARTIN DYER†, LESLIE ANN GOLDBERG‡, AND MARK JERRUM§

Abstract. This paper gives a dichotomy theorem for the complexity of computing the partition function of an instance of a weighted Boolean constraint satisfaction problem. The problem is parameterized by a finite set $\mathcal{F}$ of nonnegative functions that may be used to assign weights to the configurations (feasible solutions) of a problem instance. Classical constraint satisfaction problems correspond to the special case of 0,1-valued functions. We show that computing the partition function, i.e., the sum of the weights of all configurations, is $\mathsf{FP}^{\#\mathsf{P}}$-complete unless either (1) every function in $\mathcal{F}$ is of "product type," or (2) every function in $\mathcal{F}$ is "pure affine." In the remaining cases, computing the partition function is in P.

Key words. complexity theory, counting, #P, constraint satisfaction

AMS subject classifications. Primary, 68Q25; Secondary, 05C15, 68T27

DOI. 10.1137/070690201

1. Introduction.

This paper gives a dichotomy theorem for the complexity of the partition function of weighted Boolean constraint satisfaction problems. Such problems are parameterized by a set $\mathcal{F}$ of nonnegative functions that may be used to assign weights to configurations (solutions) of the instance. These functions take the place of the allowed constraint relations in classical constraint satisfaction problems (CSPs). Indeed, the classical setting may be recovered by restricting $\mathcal{F}$ to functions with range $\{0, 1\}$. The key problem associated with an instance of a weighted CSP is to compute its partition function, i.e., the sum of weights of all its configurations. Computing the partition function of a weighted CSP may be viewed as a generalization of counting the number of satisfying solutions of a classical CSP. Many partition functions from statistical physics may be expressed as weighted CSPs. For example, the *Potts model* [22] is naturally expressible as a weighted CSP, whereas in the classical framework only the "hard-core" versions may be directly expressed. (The hard-core version of the *antiferromagnetic* Potts model corresponds to graph coloring, and the hard-core version of the *ferromagnetic* Potts model is trivial—acceptable configurations color the entire graph with a single color.) A corresponding weighted version of the decision CSP was investigated by Cohen et al. [3]. This results in optimization problems.

We use $\#\mathsf{CSP}(\mathcal{F})$ to denote the problem of computing the partition function of weighted CSP instances that can be expressed using only functions from $\mathcal{F}$. We show in Theorem 4 below that if every function $f \in \mathcal{F}$ is "of product type," then computing the partition function $Z(I)$ of an instance I can be done in polynomial time. Formal definitions are given later, but the condition of being of product type is easily checked—it essentially means that the partition function factors. We show further in

*Received by the editors May 1, 2007; accepted for publication (in revised form) August 25, 2008; published electronically January 14, 2009. This work was partly funded by the EPSRC grant "The complexity of counting in constraint satisfaction problems."

http://www.siam.org/journals/sicomp/38-5/69020.html

†School of Computing, University of Leeds, Leeds LS2 9JT, UK (dyer@comp.leeds.ac.uk).

‡Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK (l.a.goldberg@liverpool.ac.uk).

§School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London E1 4NS, UK (m.jerrum@qmul.ac.uk).

Theorem 4 that if every function $f \in \mathcal{F}$ is "pure affine," then the partition function of $Z(I)$ can be computed in polynomial time. Once again, there is an algorithm to check whether $\mathcal{F}$ is pure affine. For each other set $\mathcal{F}$, we show in Theorem 4 that computing the partition function of a $\#\mathsf{CSP}(\mathcal{F})$ instance is complete for the class $\mathsf{FP}^{\#\mathsf{P}}$. The existence of algorithms for testing the properties of being purely affine or of product type means that the dichotomy is effectively decidable.

1.1. Constraint satisfaction. Constraint satisfaction, which originated in artificial intelligence, provides a general framework for modeling decision problems and has many practical applications. (See, for example, [17].) Decisions are modelled by *variables*, which are subject to *constraints*, modelling logical and resource restrictions. The paradigm is sufficiently broad that many interesting problems can be modelled, from satisfiability problems to scheduling problems and graph-theory problems. Understanding the complexity of CSPs has become a major and active area within computational complexity [7, 13].

A CSP typically has a finite *domain*, which we will denote by $[q] = \{0, 1, \ldots, q-1\}$ for a positive integer q.[1] A *constraint language* Γ with domain $[q]$ is a set of relations on $[q]$. For example, take $q = 2$. The relation $R = \{(0,0,1), (0,1,0), (1,0,0), (1,1,1)\}$ is a 3-ary relation on the domain $\{0,1\}$, with four tuples.

Once we have fixed a constraint language Γ, an *instance* of the CSP is a set of *variables* $V = \{v_1, \ldots, v_n\}$ and a set of *constraints*. Each constraint has a *scope,* which is a tuple of variables (for example, (v_4, v_5, v_1)) and a relation from Γ of the same arity, which constrains the variables in the scope. A *configuration* σ is a function from V to $[q]$. The configuration σ is *satisfying* if the scope of every constraint is mapped to a tuple that is in the corresponding relation. In our example above, a configuration σ satisfies the constraint with scope (v_4, v_5, v_1) and relation R if and only if it maps an odd number of the variables in $\{v_1, v_4, v_5\}$ to the value 1. Given an instance of a CSP with constraint language Γ, the *decision problem* $\mathsf{CSP}(\Gamma)$ asks us to determine whether any configuration is satisfying. The *counting problem* $\#\mathsf{CSP}(\Gamma)$ asks us to determine the *number* of (distinct) satisfying configurations.

Varying the constraint language Γ defines the classes CSP and $\#\mathsf{CSP}$ of decision and counting problems. These contain problems of different computational complexities. For example, if $\Gamma = \{R_1, R_2, R_3\}$, where R_1, R_2, and R_3 are the three binary relations defined by $R_1 = \{(0,1), (1,0), (1,1)\}$, $R_2 = \{(0,0), (0,1), (1,1)\}$, and $R_3 = \{(0,0), (0,1), (1,0)\}$, then $\mathsf{CSP}(\Gamma)$ is the classical 2-satisfiability problem, which is in P. On the other hand, there is a similar constraint language Γ' with four relations of arity 3 such that 3-satisfiability (which is NP-complete) can be represented in $\mathsf{CSP}(\Gamma')$. It may happen that the counting problem is harder than the decision problem. If Γ is the constraint language of 2-satisfiability above, then $\#\mathsf{CSP}(\Gamma)$ contains the problem of counting independent sets in graph, and is $\#\mathsf{P}$-complete [21] even if restricted to 3-regular graphs [12].

Any decision problem $\mathsf{CSP}(\Gamma)$ is in NP, but not every problem in NP can be represented as a CSP. For example, the question "Is G Hamiltonian?" cannot naturally be expressed as a CSP, because the property of being Hamiltonian cannot be captured by relations of bounded size. This limitation of the class CSP has an important advantage. If $\mathsf{P} \neq \mathsf{NP}$, then there are problems which are neither in P nor NP-complete [15]. But, for well-behaved smaller classes of decision problems, the situation can be simpler. We may have a *dichotomy theorem*, partitioning all problems in the class into those

[1] Usually $[q]$ is defined to be $\{1, 2, \ldots, q\}$, but it is more convenient here to start the enumeration of domain elements at 0 rather than 1.

which are in P and those which are NP-complete. There are no "leftover" problems of intermediate complexity. It has been conjectured that there is a dichotomy theorem for CSP. The conjecture is that CSP(Γ) is in P for some constraint languages Γ, and CSP(Γ) is NP-complete for all other constraint languages Γ. This conjecture appeared in a seminal paper of Feder and Vardi [10] but has not yet been proved.

A similar dichotomy, between FP- and #P-complete, is conjectured for #CSP [2]. The complexity classes FP and #P are the analogues of P and NP for counting problems. FP is simply the class of functions computable in deterministic polynomial time. #P is the class of integer functions that can be expressed as the number of accepting computations of a polynomial-time nondeterministic Turing machine. Completeness in #P is defined with respect to polynomial-time Turing reducibility [16, Chap. 18]. Bulatov and Dalmau [2] have shown in one direction that, if #CSP(Γ) is solvable in polynomial time, then the constraints in Γ must have certain algebraic properties (assuming #P $\not\subseteq$ FP). In particular, they must have a so-called *Mal'tsev polymorphism*. The converse is known to be false, though it remains possible that the dichotomy (if it exists) does have an algebraic characterization.

The conjectured dichotomies for CSP and #CSP are major open problems for computational complexity theory. There have been many important results for subclasses of CSP and #CSP. We mention the most relevant to our paper here. The first decision dichotomy was that of Schaefer [18], for the Boolean domain $\{0,1\}$. Schaefer's result is as follows.

THEOREM 1 (Schaefer [18]). *Let Γ be a constraint language with domain $\{0,1\}$. The problem* CSP(Γ) *is in* P *if Γ satisfies one of the conditions below. Otherwise,* CSP(Γ) *is* NP*-complete.*

1. *Γ is 0-valid or 1-valid.*
2. *Γ is weakly positive or weakly negative.*
3. *Γ is affine.*
4. *Γ is bijunctive.*

We will not give detailed definitions of the conditions in Theorem 1, but the interested reader is referred to the paper [18] or to Theorem 6.2 of the textbook [7]. An interesting feature is that the conditions in [7, Thm. 6.2] are all checkable. That is, there is an algorithm to determine whether CSP(Γ) is in P or NP-complete, given a constraint language Γ with domain $\{0,1\}$. Creignou and Hermann [6] adapted Schaefer's decision dichotomy to obtain a counting dichotomy for the Boolean domain. Their result is as follows.

THEOREM 2 (Creignou and Hermann [6]). *Let Γ be a constraint language with domain $\{0,1\}$. The problem* #CSP(Γ) *is in* FP *if Γ is affine. Otherwise,* #CSP(Γ) *is* #P*-complete.*

A constraint language Γ with domain $\{0,1\}$ is *affine* if every relation $R \in \Gamma$ is affine. A relation R is affine if the set of tuples $x \in R$ is the set of solutions to a system of linear equations over GF(2). These equations are of the form $v_1 \oplus \cdots \oplus v_n = 0$ and $v_1 \oplus \cdots \oplus v_n = 1$, where $\oplus$ is the *exclusive or* operator. It is well known (see, for example, Lemma 4.10 of [7]) that a relation R is affine if and only if $a, b, c \in R$ implies $d = a \oplus b \oplus c \in R$. (We will use this characterization below.) There is an algorithm for determining whether a Boolean constraint language Γ is affine, so there is an algorithm for determining whether #CSP(Γ) is in FP or #P-complete.

1.2. Weighted #CSP. The weighted graph homomorphism framework of [4] extends naturally to CSPs. Fix the domain $[q]$. Instead of constraining a length-k scope with an arity-k relation on $[q]$, we give a weight to the configuration on this scope by

applying a function f from $[q]^k$ to the nonnegative rationals. Let $\mathcal{F}_q = \{f : [q]^k \to \mathbb{Q}^+ \mid k \in \mathbb{N}\}$ be the set of all such functions (of all arities).[2] Given a function $f \in \mathcal{F}_q$ of arity k, the *underlying relation* of f is given by $R_f = \{x \in [q]^k \mid f(x) \neq 0\}$. It is often helpful to think of R_f as a table, with k columns corresponding to the positions of a k-tuple. Each row corresponds to a tuple $x = (x_1, \ldots, x_k) \in R_f$. The entry in row x and column j is x_j, which is a value in $[q]$.

A *weighted* #CSP problem is parameterized by a finite subset $\mathcal{F}$ of $\mathcal{F}_q$ and will be denoted by $\#\mathsf{CSP}(\mathcal{F})$. An instance I of $\#\mathsf{CSP}(\mathcal{F})$ consists of a set V of *variables* and a set $\mathcal{C}$ of *constraints*. Each constraint $C \in \mathcal{C}$ consists of a function $f_C \in \mathcal{F}$ (say of arity k_C) and a *scope*, which is a sequence $s_C = (v_{C,1}, \ldots, v_{C,k_C})$ of variables from V. The variables $v_{C,1}, \ldots, v_{C,k_C}$ need not be distinct. As in the unweighted case, a *configuration* σ for the instance I is a function from V to $[q]$. The *weight* of the configuration σ is given by

$$w(\sigma) = \prod_{C \in \mathcal{C}} f_C(\sigma(v_{C,1}), \ldots, \sigma(v_{C,k_C})).$$

Finally, the *partition function* $Z(I)$ is given, for instance I, by

$$Z(I) = \sum_{\sigma : V \to [q]} w(\sigma). \tag{1}$$

In the computational problem $\#\mathsf{CSP}(\mathcal{F})$, the goal is to compute $Z(I)$, given an instance I.

Note that an (unweighted) CSP counting problem $\#\mathsf{CSP}(\Gamma)$ can be represented naturally as a weighted CSP counting problem. For each relation $R \in \Gamma$, let f^R be the indicator function for membership in R. That is, if $x \in R$, we set $f^R(x) = 1$. Otherwise we set $f^R(x) = 0$. Let $\mathcal{F} = \{f^R \mid R \in \Gamma\}$. Then for any instance I of $\#\mathsf{CSP}(\Gamma)$ the number of satisfying configurations for I is given by the (weighted) partition function $Z(I)$ from (1).

This framework has been employed previously in connection with *graph homomorphisms* [1]. Suppose $H = (H_{ij})$ is any symmetric $q \times q$ matrix H of rational numbers. We view H as being an edge-weighting of an undirected graph $\mathcal{H}$, where a zero weight in H means that the corresponding edge is absent from $\mathcal{H}$. Given a (simple) graph $G = (V, E)$, we consider computing the partition function

$$Z_H(G) = \sum_{\sigma : V \to [q]} w(\sigma), \quad \text{where } w(\sigma) = \prod_{\{u,v\} \in E} H_{\sigma(u)\sigma(v)}.$$

Within our framework above, we view H as the binary function $h : [q]^2 \to \mathbb{R}$, and the problem is then computing the partition function of $\#CSP(\{h\})$.

Bulatov and Grohe [4] call H *connected* if $\mathcal{H}$ is connected and *bipartite* if $\mathcal{H}$ is bipartite. They give the following dichotomy theorem for nonnegative H.[3]

THEOREM 3 (Bulatov and Grohe [4]). *Let H be a symmetric matrix with nonnegative rational entries. Then we have the following:*

[2] We assume $0 \in \mathbb{N}$, so we allow nonnegative constants.

[3] This is not quite the original statement of the theorem. We have chosen here to restrict all inputs to be rational, in order to avoid issues of how to represent, and compute with, arbitrary real numbers.

1. *If H is connected and not bipartite, then computing Z_H is in* FP *if the rank of H is at most* 1*; otherwise computing Z_H is* #P*-hard.*
2. *If H is connected and bipartite, then computing Z_H is in* FP *if the rank of H is at most* 2*; otherwise computing Z_H is* #P*-hard.*
3. *If H is not connected, then computing Z_H is in* FP *if each of its connected components satisfies the corresponding conditions stated in* 1 *or* 2*; otherwise computing Z_H is* #P*-hard.*

Many partition functions arising in statistical physics may be viewed as weighted #CSP problems. An example is the q-state Potts model (which is, in fact, a weighted graph homomorphism problem). In general, weighted #CSP is very closely related to the problem of computing the partition function of a Gibbs measure in the framework of Dobrushin, Lanford, and Ruelle (see [1]). See also the framework of Scott and Sorkin [19].

1.3. Some notation. We will call the class of (rational) weighted #CSP problems *weighted* #CSP. The subclass having domain size $q = 2$ will be called weighted *Boolean* #CSP and will be the main focus of this paper. We will give a dichotomy theorem for weighted Boolean #CSP.

Since weights can be arbitrary nonnegative rational numbers, the solution to these problems is not an integer in general. Therefore $\#\mathsf{CSP}(\mathcal{F})$ is not necessarily in the class #P. However, Goldberg and Jerrum [11] have observed that $Z(I) = \tilde{Z}(I)/K(I)$, where $\tilde{Z}$ is a function in #P and $K(I)$ is a positive integer computable in FP. This follows because, for all $f \in \mathcal{F}$, we can ensure that $f(\cdot) = \tilde{f}(\cdot)/K(I)$, where $\tilde{f}(\cdot) \in \mathbb{N}$, by "clearing denominators." The denominator $K(I)$ can obviously be computed in polynomial time, and it is straightforward to show that computing $\tilde{Z}(I)$ is in #P, so the characterization of [11] follows. The resulting complexity class, comprising functions which are a function in #P divided by a function in FP, is named $\#\mathsf{P}_{\mathbb{Q}}$ in [11], where it is used in the context of approximate counting. Clearly we have

$$\mathsf{weighted}\,\#\mathsf{CSP} \subseteq \#\mathsf{P}_{\mathbb{Q}} \subseteq \mathsf{FP}^{\#\mathsf{P}}.$$

On the other hand, if $Z(I) \in \mathsf{weighted}\,\#\mathsf{CSP}$ is #P-hard, then, using an oracle for computing $Z(I)$, we can construct a #P oracle $\tilde{Z}(I)$ as outlined above. (Note that $Z(I) \notin \#\mathsf{P}$ in general.) Using this oracle, we can compute any function in $\mathsf{FP}^{\#\mathsf{P}}$ with a polynomial time-bounded oracle Turing machine. Thus any #P-hard function in $\mathsf{weighted}\,\#\mathsf{CSP}$ is complete for $\mathsf{FP}^{\#\mathsf{P}}$. We will use this observation to state our main result in terms of completeness for the class $\mathsf{FP}^{\#\mathsf{P}}$.

We make the following definition, which relates to the discussion above. We will say that $\mathcal{F} \subseteq \mathcal{F}_q$ *simulates* $f \in \mathcal{F}_q$ if, for each instance I of $\#\mathsf{CSP}(\mathcal{F} \cup \{f\})$, there is a polynomial time computable instance I' of $\#\mathsf{CSP}(\mathcal{F})$ such that $Z(I) = \varphi(I)Z(I')$ for some $\varphi(I) \in \mathbb{Q}$ which is FP-computable. This generalizes the notion of *parsimonious reduction* [16] among problems in #P. We will use $\leq_{\mathrm{T}}$ to denote the relation "is polynomial-time Turing-reducible to" between computational problems. Clearly, if $\mathcal{F}$ simulates f, we have $\#\mathsf{CSP}(\mathcal{F} \cup \{f\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\mathcal{F})$. Note also that, if $\tilde{f} = Kf$, for some constant $K > 0$, then $\{f\}$ simulates $\tilde{f}$. Thus there is no need to distinguish between "proportional" functions.

We use the following terminology for certain functions. Let $\chi_=$ be the binary *equality* function defined on $[q]$ as follows. For any element $c \in [q]$, $\chi_=(c, c) = 1$, and for any pair (c, d) of distinct elements of $[q]$, $\chi_=(c, d) = 0$. Let $\chi_{\neq}$ be the binary

disequality function given by $\chi_{\neq}(c,d) = 1 - \chi_{=}(c,d)$ for all $c, d \in [q]$.[4] We say that a function f is of *product type* if f can be expressed as a product of unary functions and binary functions of the form $\chi_{=}$ and $\chi_{\neq}$.

We focus attention in this paper on the Boolean case, $q = 2$. In this case, we say that a function $f \in \mathcal{F}_2$ has *affine support* if its underlying relation R_f, defined earlier, is affine. We say that f is *pure affine* if it has affine support and range $\{0, w\}$ for some $w > 0$. Thus a function is pure affine if and only if it is a positive real multiple of some (0,1-valued) function which is affine over GF(2).

1.4. Our result. Our main result is the following.

THEOREM 4. *Suppose* $\mathcal{F} \subseteq \mathcal{F}_2 = \{f : \{0,1\}^k \to \mathbb{Q}^+ \mid k \in \mathbb{N}\}$. *If every function in* $\mathcal{F}$ *is of product type, then* $\#\mathsf{CSP}(\mathcal{F})$ *is in* FP. *If every function in* $\mathcal{F}$ *is pure affine, then* $\#\mathsf{CSP}(\mathcal{F})$ *is in* FP. *Otherwise,* $\#\mathsf{CSP}(\mathcal{F})$ *is* $\mathsf{FP}^{\#\mathsf{P}}$*-complete.*

Proof. Suppose first that $\mathcal{F}$ is of product type. In this case the partition function $Z(I)$ of an instance I with variable set V is easy to evaluate because it can be factored into easy-to-evaluate pieces: Partition the variables in V into equivalence classes according to whether or not they are related by an equality or disequality function. (The equivalence relation on variables here is "depends linearly on.") An equivalence class consists of two (possibly empty) sets of variables U_1 and U_2. All of the variables in U_1 must be assigned the same value by a configuration σ of nonzero weight, and all variables in U_2 must be assigned the other value. Variables in $U_1 \cup U_2$ are not related by equality or disequality to variables in $V \setminus (U_1 \cup U_2)$. The equivalence class contributes one weight, say α, to the partition function if variables in U_1 are given value "0" by σ, and it contributes another weight, say β, to the partition function if variables in U_1 are given value "1" by σ. Thus, $Z(I) = (\alpha + \beta)Z(I')$, where I' is the instance formed from I by removing this equivalence class. Therefore, suppose we choose any equivalence class and remove its variables. Since $\mathcal{F}$ contains only unary, equality, or binary disequality constraints, we can also remove all functions involving variables in $U_1 \cup U_2$ to give $\mathcal{F}'$. Then I' is of product type with fewer variables, so we may compute $Z(I')$ recursively.

Suppose second that $\mathcal{F}$ if pure affine. Then $Z(I) = \prod_{f \in \mathcal{F}} w_f^{k_f} Z(I')$, where $\{0, w_f\}$ is the range of f, k_f is the number of constraints involving f in I, and I' is the instance obtained from I by replacing every function f by its underlying relation R_f (viewed as a function with range $\{0,1\}$). $Z(I')$ is easy to evaluate, because this is just counting solutions to a linear system over GF(2), as Creignou and Hermann have observed [6].

Finally, the #P-hardness in Theorem 4 follows from Lemma 5 below. □

LEMMA 5. *If* $f \in \mathcal{F}_2$ *is not of product type and* $g \in \mathcal{F}_2$ *is not pure affine, then* $\#\mathsf{CSP}(\{f, g\})$ *is* #P*-hard.*

Note that the functions f and g in Lemma 5 may be one and the same function. So $\#\mathsf{CSP}(\{f\})$ is #P-hard when f is not of product type nor pure affine. The rest of this article gives the proof of Lemma 5.

2. Useful tools for proving hardness of #CSP.

2.1. Notation. For any sequence $u_1, \ldots, u_k$ of variables of I and any sequence $c_1, \ldots, c_k$ of elements of the domain $[q]$, we will let $Z(I \mid \sigma(u_1) = c_1, \ldots, \sigma(u_k) = c_k)$ denote the contribution to $Z(I)$ from assignments σ with $\sigma(u_1) = c_1, \ldots, \sigma(u_k) = c_k$.

2.2. Projection. The first tool that we study is projection, which is referred to as "integrating out" in the statistical physics literature.

[4] A more general disequality function is defined in the appendix.

Let f be a function of arity k, and let $J = \{j_1, \ldots, j_r\}$ be a size-r subset of $\{1, \ldots, k\}$, where $j_1 < \cdots < j_r$.[5] We say that a k-tuple $x' \in [q]^k$ *extends* an r-tuple $x \in [q]^r$ on J (written $x' \sqsupseteq_J x$) if x' agrees with x on indices in J; that is to say, $x'_{j_i} = x_i$ for all $1 \le i \le r$. The *projection* g of f onto J is defined as follows. For every $x \in [q]^r$, $g(x) = \sum_{x' \sqsupseteq_J x} f(x')$.

The following lemma may be viewed as a weighted version of Proposition 2 of [2], where it is proved for the unweighted case. It is expressed somewhat differently in [2], in terms of counting the number of solutions to an existential formula.

LEMMA 6. *Suppose $\mathcal{F} \subseteq \mathcal{F}_q$. Let g be a projection of a function $f \in \mathcal{F}$ onto a subset of its indices. Then $\#\mathsf{CSP}(\mathcal{F} \cup \{g\}) \le_{\mathrm{T}} \#\mathsf{CSP}(\mathcal{F})$.*

Proof. Let k be the arity of f, and let g be the projection of f onto the subset J of its indices. Let I be an instance of $\#\mathsf{CSP}(\mathcal{F} \cup \{g\})$. We will construct an instance I' of $\#\mathsf{CSP}(\mathcal{F})$ such that $Z(I) = Z(I')$. The instance I' is identical to I except that every constraint C of I involving g is replaced with a new constraint C' of I' involving f. The corresponding scope $(v_{C',1}, \ldots, v_{C',k})$ is constructed as follows. If j_ℓ is the ℓth element of J, then $v'_{C',j_\ell} = v_{C,\ell}$. The other variables, $v_{C',j}$ $(j \notin J)$, are distinct new variables. We have shown that $\mathcal{F}$ simulates g with $\phi(I) = 1$. □

2.3. Pinning. For $c \in [q]$, δ_c denotes the unary function with $\delta_c(c) = 1$ and $\delta_c(d) = 0$ for $d \ne c$. The following lemma, which allows "pinning" CSP variables to specific values in hardness proofs, generalizes Theorem 8 of [2], which does the unweighted case. Again [2] employs different terminology, and its theorem is a statement about the full idempotent reduct of a finite algebra. The idea of pinning was used previously by Bulatov and Grohe of [4] in the context of counting weighted graph homomorphisms (see Lemma 32 of [4]). A similar idea was used by Dyer and Greenhill in the context of counting *unweighted* graph homomorphisms—in that context, Theorem 4.1 of [8] allows pinning all variables to a particular *component* of the target graph H.

LEMMA 7. *For every $\mathcal{F} \subseteq \mathcal{F}_q$, $\#\mathsf{CSP}(\mathcal{F} \cup \bigcup_{c \in [q]} \delta_c) \le_{\mathrm{T}} \#\mathsf{CSP}(\mathcal{F})$.*

The proof of Lemma 7 is deferred to the appendix. Since we use only the case $q = 2$ in this paper, we provide the (simpler) proof for the Boolean case here.

LEMMA 8. *For every $\mathcal{F} \subseteq \mathcal{F}_2$, $\#\mathsf{CSP}(\mathcal{F} \cup \{\delta_0, \delta_1\}) \le_{\mathrm{T}} \#\mathsf{CSP}(\mathcal{F})$.*

Proof. For $x \in [2]^k$, let $\overline{x}$ be the k-tuple whose ith component, $\overline{x}_i$, is $x_i \oplus 1$ for all i. Say that $\mathcal{F}$ is *symmetric* if it is the case that for every arity-k function $f \in \mathcal{F}$ and every $x \in [2]^k$, $f(\overline{x}) = f(x)$.

Given an instance I of $\#\mathsf{CSP}(\mathcal{F} \cup \{\delta_0, \delta_1\})$ with variable set V, we consider two instances I' and I'' of $\#\mathsf{CSP}(\mathcal{F})$. Let V_0 be the set of variables v of I to which the constraint $\delta_0(v)$ is applied. Let V_1 be the set of variables v of I to which the constraint $\delta_1(v)$ is applied. We can assume without loss of generality that V_0 and V_1 do not intersect. (Otherwise, $Z(I) = 0$ and we can determine this without using an oracle for $\#\mathsf{CSP}(\mathcal{F})$.) Let $V_2 = V \setminus (V_0 \cup V_1)$. The instance I' has variables $V_2 \cup \{t_0, t_1\}$, where t_0 and t_1 are distinct new variables that are not in V. Every constraint C of I involving a function $f \in \mathcal{F}$ corresponds to a constraint C' of I'. C' is the same as C except that variables in V_0 are replaced with t_0 and variables in V_1 are replaced with t_1. Similarly, the instance I'' has variables $V_2 \cup \{t\}$, where t is a new variable that is not in V. Every constraint C of I involving a function $f \in \mathcal{F}$ corresponds to a constraint C'' of I''. The constraint C'' is the same as C except that variables in $V_0 \cup V_1$ are replaced with t.

[5] It is not necessary to choose this particular ordering for J, but it is convenient to do so.

Case 1. $\mathcal{F}$ is symmetric: By construction,

$$Z(I') - Z(I'') = Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) + Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 0).$$

By symmetry, the summands are the same, so

$$Z(I') - Z(I'') = 2Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) = 2Z(I).$$

Case 2. $\mathcal{F}$ is not symmetric: Let f be an arity-k function in $\mathcal{F}$, and let $x \in [2]^k$ so that $f(x) > f(\overline{x}) \geq 0$. Let $s = (t_{x_1}, \ldots, t_{x_k})$, and let I'_x be the instance derived from I' by adding a new constraint with function f and scope s. Similarly, let I''_x be the instance derived from I'' by adding a new constraint with function f and scope $(t, \ldots, t)$. Now

$$\begin{aligned} Z(I'_x) &= Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) f(x) + Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 0) f(\overline{x}) \\ &\quad + Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 0) f(0, \ldots, 0) + Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 1) f(1, \ldots, 1) \\ &= Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) f(x) + Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 0) f(\overline{x}) + Z(I''_x). \end{aligned}$$

Thus we have two independent equations,

$$\begin{aligned} Z(I'_x) - Z(I''_x) &= Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) f(x) + Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 0) f(\overline{x}), \\ Z(I') - Z(I'') &= Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) \qquad + Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 0) \qquad , \end{aligned}$$

in the unknowns $Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1)$ and $Z(I' \mid \sigma(t_0) = 1, \sigma(t_1) = 0)$. Solving these, we obtain the value of $Z(I' \mid \sigma(t_0) = 0, \sigma(t_1) = 1) = Z(I)$. □

2.4. #P-hard problems. To prove Lemma 5, we will give reductions from some known #P-hard problems. The first of these is the problem of counting homomorphisms from simple graphs to 2-vertex multigraphs. We use the following special case of Bulatov and Grohe's Theorem 3.

COROLLARY 9 (Bulatov and Grohe [4]). *Let H be a symmetric 2×2 matrix with nonnegative real entries. If H has rank 2 and at most one entry of H is 0, then* EVAL(H) *is #P-hard.*

We will also use the problem of computing the *weight enumerator* of a linear code. Given a *generating matrix* $A \in \{0,1\}^{r \times C}$ of rank r, a *code word* c is any vector in the linear subspace Υ generated by the rows of A over GF(2). For any real number λ, the *weight enumerator* of the code is given by $W_A(\lambda) = \sum_{c \in \Upsilon} \lambda^{\|c\|}$, where $\|c\|$ is the number of 1's in c. The problem of computing the weight enumerator of a linear code is in FP for $\lambda \in \{-1, 0, 1\}$ and is known to be #P-hard for every other fixed $\lambda \in \mathbb{Q}$ (see [22]). We could not find a proof, so we provide one here. We restrict our attention to positive λ, since that is adequate for our purposes.

LEMMA 10. *Computing the weight enumerator of a linear code is #P-hard for any fixed positive rational number $\lambda \neq 1$.*

Proof. We will prove hardness by reduction from a problem EVAL(H), for some appropriate H, using Corollary 9. Let the input to EVAL(H) be a connected graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. Let B be the $n \times m$ incidence matrix of G, with $b_{ij} = 1$ if $v_i \in e_j$ and $b_{ij} = 0$ otherwise. Let A be the $(n-1) \times m$ matrix which is B with the row for v_n deleted. A will be the generating matrix of the weight enumerator instance, with $r = n - 1$ and $C = m$. It has rank $(n-1)$ since G contains a spanning tree. A code word c has $c_j = \bigoplus_{i \in U} b_{ij}$, where $U \subseteq V \setminus \{v_n\}$. Thus $c_j = 1$ if and only if e_j has exactly one endpoint in U, and the weight of c is λ^k, where k is the number of edges in the cut $U, V \setminus U$. Thus $W_A(\lambda) = \frac{1}{2} Z_H(G)$, where

H is the symmetric weight matrix with $H_{11} = H_{22} = 1$ and $H_{12} = H_{21} = \lambda$. The $\frac{1}{2}$ arises because we fixed which side of the cut contains v_n. Now H has rank 2 unless $\lambda = 1$, so this problem is #P-hard by Corollary 9. Note, by the way, that $Z_H(G)$ is the partition function of the Ising model in statistical physics [5]. □

3. The proof of Lemma 5. Throughout this section, we assume $q = 2$. The following lemma is a generalization of a result of Creignou and Hermann [6], which deals with the case in which f is a relation (or, in our setting, a function with range $\{0, 1\}$). The inductive technique used in the proof of Lemma 11 (combined with the follow-up in Lemma 12) is good for showing that $\#\mathsf{CSP}(\mathcal{F})$ is #P-hard when $\mathcal{F}$ contains a *single* function. A very different situation arises when $\#\mathsf{CSP}(\{f\})$ and $\#\mathsf{CSP}(\{g\})$ are in FP but $\#\mathsf{CSP}(\{f, g\})$ is #P-hard due to *interactions* between f and g—we deal with that problem later.

LEMMA 11. *Suppose that $f \in \mathcal{F}_2$ does not have affine support. Then $\#\mathsf{CSP}(\{f\})$ is #P-hard.*

Proof. Let k be the arity of f, and let us denote the ith component of k-tuple $a \in R_f$ by a_i. The proof is by induction on k. The lemma is trivially true for $k = 1$, since all functions of arity 1 have affine support.

For $k = 2$, we note that since R_f is not affine, it is of the form $R_f = \{(\alpha, \beta), (\bar{\alpha}, \beta), (\bar{\alpha}, \bar{\beta})\}$ for some $\alpha \in \{0, 1\}$ and $\beta \in \{0, 1\}$. We can show that $\#\mathsf{CSP}(\{f\})$ is #P-hard by reduction from $\text{EVAL}(H)$ using

$$H = \begin{pmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{pmatrix},$$

which has rank 2 and exactly one entry that is 0. Given an instance $G = (V, E)$ of $\text{EVAL}(H)$, we construct an instance I of $\#\mathsf{CSP}(\{f\})$ as follows. The variables of I are the vertices of G. For each edge $e = (u, v)$ of G, add a constraint with function f and variable sequence u, v. Corollary 9 now tells us that $\text{EVAL}(H)$ is #P-hard, so $\#\mathsf{CSP}(\{f\})$ is #P-hard.

Suppose $k > 2$. We start with some general arguments and notation. For any $i \in \{1, \ldots, k\}$ and any $\alpha \in \{0, 1\}$ let $f^{i=\alpha}$ be the function of arity $k-1$ derived from f by pinning the ith position to α. That is, $f^{i=\alpha}(x_1, \ldots, x_{k-1}) = f(x_1, \ldots, x_{i-1}, \alpha, x_{i+1}, \ldots, x_k)$. Also, let $f^{i=*}$ be the projection of f onto all positions apart from position i (see section 2.2). Note that $\#\mathsf{CSP}(\{f^{i=\alpha}\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\{f, \delta_0, \delta_1\})$, since $f^{i=\alpha}$ can obviously be simulated by $\{f, \delta_0, \delta_1\}$. Furthermore, by Lemma 8, $\#\mathsf{CSP}(\{f, \delta_0, \delta_1\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\{f\})$. Thus, we can assume that $f^{i=\alpha}$ has affine support—otherwise, we are finished by induction. Similarly, by Lemma 6, $\#\mathsf{CSP}(\{f^{i=*}\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\{f\})$. Thus we can assume that $f^{i=*}$ has affine support—otherwise, we are finished by induction.

Now, recall that R_f is not affine. Consider any $a, b, c \in R_f$ such that $d = a \oplus b \oplus c \notin R_f$. We have four cases.

Case 1. There are indices $1 \leq i < j \leq k$ such that $(a_i, b_i, c_i) = (a_j, b_j, c_j)$. Without loss of generality, suppose $i = 1$ and $j = 2$. Define the function f' of arity $(k - 1)$ by $f'(r_2, \ldots, r_k) = f(r_2, r_2, \ldots, r_k)$. Note that $R_{f'}$ is not affine since the condition $a \oplus b \oplus c \notin R_f$ is inherited by $R_{f'}$. So, by induction, $\#\mathsf{CSP}(\{f'\})$ is #P-hard. Now note that $\#\mathsf{CSP}(\{f'\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\{f\})$. To see this, note that any instance I_1 of $\#\mathsf{CSP}(\{f'\})$ can be turned into an instance I of $\#\mathsf{CSP}(\{f\})$ by repeating the first variable in the sequence of variables for each constraint.

Case 2. There is an index $1 \le i \le k$ such that $a_i = b_i = c_i$. Since d is not in R_f and $d_i = a_i$, we find that $f^{i=a_i}$ does not have affine support, contrary to earlier assumptions.

Having finished Cases 1 and 2, we may assume without loss of generality that we are in Case 3 or 4 below, where $\{\alpha, \beta\} \in \{0,1\}$, $\bar{\alpha} = 1 - \alpha$, $\bar{\beta} = 1 - \beta$, and $a', b', c' \in \{0,1\}^{k-2}$.

Case 3. $a = (\bar{\alpha}, \bar{\beta}, a')$, $b = (\bar{\alpha}, \beta, b')$, $c = (\alpha, \bar{\beta}, c')$. Since $R_{f^{1=*}}$ is affine and a, b, and c are in R_f, we must have either $d = (\alpha, \beta, d') \in R_f$ or $e = (\bar{\alpha}, \beta, d') \in R_f$, where $d' = a' \oplus b' \oplus c'$. In the first case, we are done (we have contradicted the assumption that $d \notin R_f$), so assume that $e \in R_f$ but $d \notin R_f$. Similarly, since $R_{f^{2=*}}$ is affine, we may assume that $g = (\alpha, \bar{\beta}, d') \in R_f$. Since $R_{f^{1=\bar{\alpha}}}$ is affine and a, b, and e are in R_f, we find that $h = a \oplus b \oplus e = (\bar{\alpha}, \bar{\beta}, c') \in R_f$. Since $R_{f^{2=\bar{\beta}}}$ is affine and a, c, and g are in R_f, we find that $i = (\bar{\alpha}, \bar{\beta}, b') \in R_f$. Also, since $R_{f^{2=\bar{\beta}}}$ is affine and a, h, and i are in R_f, we find that $j = (\bar{\alpha}, \bar{\beta}, d') \in R_f$. Let $f'(r_1, r_2) = f(r_1, r_2, d_3, \ldots, d_k)$. Since e, g, and j are in R_f but d is not, we have $(\bar{\alpha}, \beta), (\alpha, \bar{\beta}), (\bar{\alpha}, \bar{\beta}) \in R_{f'}$, but $(\alpha, \beta) \notin R_{f'}$. Thus, f' does not have affine support and $\#\mathsf{CSP}(\{f'\})$ is $\#\mathsf{P}$-hard by induction. Also, $\#\mathsf{CSP}(\{f'\}) \le_{\mathrm{T}} \#\mathsf{CSP}(\{f\})$ by Lemma 8.

Case 4. $a = (\bar{\alpha}, \alpha, a')$, $b = (\bar{\alpha}, \alpha, b')$, $c = (\alpha, \bar{\alpha}, c')$. Since $R_{f^{1=*}}$ is affine and a, b, and c are in R_f but d is not, we have $e = (\bar{\alpha}, \bar{\alpha}, d') \in R_f$. Similarly, since $R_{f^{2=*}}$ is affine and a, b, and c are in R_f but d is not, we have $g = (\alpha, \alpha, d') \in R_f$. Now since $R_{f^{1=\bar{\alpha}}}$ is affine and a, b, and e are in R_f, we have $h = (\bar{\alpha}, \bar{\alpha}, c') \in R_f$. Also, since $R_{f^{2=\alpha}}$ is affine and a, b, and g are in R_f, we have $i = (\alpha, \alpha, c') \in R_f$.

Let $f'(r_1, r_2) = f(r_1, r_2, c_3, \ldots, c_k)$. If $j = (\bar{\alpha}, \alpha, c') \notin R_f$, then f' does not have affine support (since c, h, and i are in R_f), so we finish by induction as in Case 3. Suppose $j \in R_f$. Since $R_{f^{1=\bar{\alpha}}}$ is affine and a, b, and j are in R_f, we have $\ell = (\bar{\alpha}, \alpha, d') \in R_f$. Let $f''(r_1, r_2) = f(r_1, r_2, d_3, \ldots, d_k)$. Then f'' does not have affine support (since e, g, and ℓ are in R_f but d is not), so we finish by induction as in Case 3. □

Lemma 11 showed that $\#\mathsf{CSP}(\{f\})$ is $\#\mathsf{P}$-hard when f does not have affine support. The following lemma gives another (rather technical, but useful) condition which implies that $\#\mathsf{CSP}(\{f\})$ is $\#\mathsf{P}$-hard. We start with some notation. Let f be an arity-k function. For a value $b \in \{0,1\}$, an index $i \in \{1, \ldots, k\}$, and a tuple $y \in \{0,1\}^{k-1}$, let $y^{i=b}$ denote the tuple $x \in \{0,1\}^k$ formed by setting $x_i = b$ and $x_j = y_j$ $(j \in \{1, \ldots, k\} \setminus \{i\})$.

We say that index i of f is *useful*, if there is a tuple y such that $f(y^{i=0}) > 0$ and $f(y^{i=1}) > 0$. We say that f is *product-like* if, for every useful index i, there is a rational number λ_i such that, for all $y \in \{0,1\}^{k-1}$,

$$f(y^{i=0}) = \lambda_i f(y^{i=1}). \tag{2}$$

If every position i of f is useful, then being product-like is the same as being of product type. However, being product-like is less demanding because it does not restrict indices that are not useful.

LEMMA 12. *If $f \in \mathcal{F}_2$ is not product-like, then $\#\mathrm{CSP}(\{f\})$ is $\#\mathsf{P}$-hard.*

Proof. We will use Corollary 9 to prove hardness, following an argument from [9]. Choose a useful index i so that there is no λ_i satisfying (2).

Suppose f has arity k. Let A be the $2 \times 2^{k-1}$ matrix such that for $b \in \{0,1\}$ and $y \in \{0,1\}^{k-1}$, $A_{b,y} = f(y^{i=b})$. Let $A' = AA^T$.

First, we show that $\text{EVAL}(A')$ is #P-hard. Note that A' is the following symmetric 2×2 matrix with nonnegative rational entries:

$$\begin{pmatrix} \sum_y A_{0,y}^2 & \sum_y A_{0,y}A_{1,y} \\ \sum_y A_{0,y}A_{1,y} & \sum_y A_{1,y}^2 \end{pmatrix} = \begin{pmatrix} \sum_y f(y^{i=0})^2 & \sum_y f(y^{i=0})f(y^{i=1}) \\ \sum_y f(y^{i=0})f(y^{i=1}) & \sum_y f(y^{i=1})^2 \end{pmatrix}.$$

Since index i is useful, all four entries of A' are positive. To show that $\text{EVAL}(A')$ is #P-hard by Corollary 9, we just need to show that its determinant is nonzero. By the Cauchy–Schwarz equation, the determinant is nonnegative and is zero only if λ_i exists, which we have assumed not to be the case. Thus $\text{EVAL}(A')$ is #P-hard by Corollary 9.

Now we reduce $\text{EVAL}(A')$ to $\#\text{CSP}(\{f\})$. To do this, take an undirected graph G which is an instance of $\text{EVAL}(A')$. Construct an instance Y of $\#\text{CSP}(\{f\})$. For every vertex v of G we introduce a variable x_v of Y. Also, for every edge e of G we introduce $k-1$ variables $x_{e,1}, \ldots, x_{e,k-1}$ of Y. We introduce constraints in Y as follows. For each edge $e = (v, v')$ of G we introduce constraints $f(x_v, x_{e,1}, \ldots, x_{e,k-1})$ and $f(x_{v'}, x_{e,1}, \ldots, x_{e,k-1})$ into Y, where we have assumed, without loss of generality, that the first index is useful.

It is clear that $\text{EVAL}(A')$ is exactly equal to the partition function of the $\#\text{CSP}(\{f\})$ instance Y. □

For $w \in \mathbb{Q}^+$, let U_w denote the unary function mapping 0 to 1 and 1 to w. Note that $U_0 = \delta_0$, and U_1 gives the constant (0-ary function) 1, occurrences of which leave the partition function unchanged. So, by Lemma 8, we can discard these constraints since they do not add to the complexity of the problem. Note, by the observation above about proportional functions, that the functions U_w include all unary functions except for δ_1 and the constant 0. We can discard δ_1 by Lemma 8, and if the constant 0 function is in $\mathcal{F}$, any instance I where it appears as a constraint has $Z(I) = 0$. So again we can discard these constraints since they do not add to the complexity of the problem.

Thus U_w will be called *nontrivial* if $w \notin \{0, 1\}$. Let $\oplus_k : \{0,1\}^k \to \{0,1\}$ be the arity-k parity function that is 1 if and only if its argument has an odd number of 1's. Let $\neg\oplus_k : \{0,1\}^k \to \{0,1\}$ be the function $1 - \oplus_k$. The following lemma shows that even a simple function like $\oplus_3$ can lead to intractable #CSP instances when it is combined with a nontrivial weight function U_λ.

LEMMA 13. *$\#\text{CSP}(\oplus_3, U_\lambda, \delta_0, \delta_1)$ and $\#\text{CSP}(\neg\oplus_3, U_\lambda, \delta_0, \delta_1)$ are both #P-hard, for any positive $\lambda \neq 1$.*

Proof. We give a reduction from computing the weight enumerator of a linear code, which was shown to be #P-hard in Lemma 10. In what follows, it is sometimes convenient to view $\oplus_k$, δ_0, etc., as relations as well as functions to $\{0, 1\}$.

We first argue that, for any k, the relation $\oplus_k$ can be simulated by $\{\oplus_3, \delta_0, \delta_1\}$. For example, to simulate $x_1 \oplus \cdots \oplus x_k$ for $k > 3$, take new variables y, z, and w and let $m = \lceil k/2 \rceil$ and use $x_1 \oplus \cdots \oplus x_m \oplus y$ and $x_{m+1} \oplus \cdots \oplus x_k \oplus z$ and $y \oplus z \oplus w$ and $\delta_0(w)$.

Since $\{\oplus_3, \delta_0, \delta_1\}$ can be used to simulate any relation $\oplus_k$, we can use $\{\oplus_3, \delta_0, \delta_1\}$ to simulate an arbitrary system of linear equations over GF(2). In particular, we can use them to simulate the subspace Υ of code words for a given generating matrix A.

Finally, we can use U_λ to simulate the function which evaluates the weight enumerator on Υ. Then, since $\lambda \neq 0, 1$, we can apply Lemma 10 to complete the argument. The same proof, with minor modifications, applies to $\neg\oplus_3$. □

LEMMA 14. *Suppose that $f \in \mathcal{F}_2$ is not of product type. Then, for any positive $\lambda \neq 1$, there exists a constant c, depending on f, such that $\#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda, U_c\})$ is $\#\mathsf{P}$-hard.*

Proof. If f does not have affine support, the result follows by Lemma 11. So suppose f has affine support. Consider the underlying relation R_f, viewed as a table. The rows of the table represent the tuples of the relation. Let J be the set of columns on which the relation is not constant. That is, if $i \in J$, then there is a row x with $x_i = 0$ and a row y with $y_i = 1$. Group the columns in J into equivalence classes: two columns are equivalent if and only if they are equal or complementary. Let k be the number of equivalence classes. Take one column from each of the k equivalence classes as a representative, and focus on the arity-k relation R induced by those columns.

Case 1. Suppose that R is the complete relation of arity k. Let f^* be the projection of f onto the k columns of R. By Lemma 6,

$$\#\mathsf{CSP}(\{f^*\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\{f\}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda, U_c\}).$$

We will argue that $\#\mathsf{CSP}(\{f^*\})$ is $\#\mathsf{P}$-hard. To see this, note that every column of f^* is useful. Thus, if f^* were product-like, we could conclude that f^* was of product type. But this would imply that f is of product type, which is not the case by assumption. So f^* is not product-like, and hardness follows from Lemma 12.

Case 2. Suppose that R is not the complete relation of arity k. We had assumed that R_f is affine. This means that, given three vectors, x, y, and z in R_f, $x \oplus y \oplus z$ is in R_f as well. The arity-k relation R inherits this property, so is also affine.

Choose a minimal set of columns of R that do not induce the complete relation. This exists by assumption. Suppose that there are j columns in this minimal set. Observe that $j \neq 1$ because there are no constant columns in J. Also $j \neq 2$, since otherwise the two columns would be related by equality or disequality, contradicting the preprocessing step. The argument here is that on two columns, R cannot have exactly three tuples because it is affine, and having tuples x, y, and z in would require the fourth tuple $x \oplus y \oplus z$. But if it has two tuples, then, because there are no constant columns, the only possibilities are either $(0,0)$ and $(1,1)$ or $(0,1)$ and $(1,0)$. Both contradict the preprocessing step, so $j \geq 3$.

Let R' be the restriction of R to the j columns. Now R' of course has fewer than 2^j rows, and at least 2^{j-1} by minimality. It is affine, and hence must be $\oplus_j$ or $\neg\oplus_j$. To see this, first note that the size of R' has to be a power of 2 since R' is the solution to a system of linear equations. Hence the size of R' must be 2^{j-1}. Then, since there are j variables, there can only be one defining equation. And, since every subset of $j-1$ variables induces a complete relation, this single equation must involve all variables. Therefore, the equation is $\oplus_j$ or $\neg\oplus_j$.

Let f' be the projection of f onto the j columns just identified. Let f'' be further obtained by pinning all but three of the j variables to 0. Pinning $j-3$ variables to 0 leaves a single equation involving all three remaining variables. Thus $R_{f''}$ must be $\oplus_3$ or $\neg\oplus_3$.

Now define the symmetric function f''' by

$$f'''(a,b,c) = f''(a,b,c)f''(a,c,b)f''(b,a,c)f''(b,c,a)f''(c,a,b)f''(c,b,a).$$

Note that $R_{f'''}$ is $\oplus_3$ or $\neg\oplus_3$, since $R_{f''}$ is symmetric and hence $R_{f'''} = R_{f''}$.

To summarize: using f and the constant functions δ_0 and δ_1, we have simulated a function f''' such that its underlying relation $R_{f'''}$ is either $\oplus_3$ or $\neg\oplus_3$. Furthermore, if triples x and y have the same number of 1's, then $f'''(x) = f'''(y)$.

We can now simulate an unweighted version of $\oplus_3$ or $\neg\oplus_3$ using f''' and a unary function U_c, with c set to a conveniently chosen value. There are two cases. Suppose

first that the affine support of f''' is $\neg\oplus_3$. Then let w_0 denote the value of f''' when applied to the 3-tuple $(0,0,0)$, and let w_2 denote $f'''(0,1,1) = f'''(1,0,1) = f'''(1,1,0)$. Recall that $f'''(x) = 0$ for any other 3-tuple x. Now let $c = (w_0/w_2)^{1/2}$. Note from the definition of f''' that w_0 and w_2 are squares of rational numbers, so c is also rational. Define a function g of arity 3 by $g(\alpha,\beta,\gamma) = U_c(\alpha)U_c(\beta)U_c(\gamma)f'''(\alpha,\beta,\gamma)$. Note that $g(0,0,0) = w_0$ and $g(0,1,1) = g(1,0,1) = g(1,1,0) = c^2 w_2 = w_0$. Thus, g is a pure affine function with affine support $\neg\oplus_3$ and range $\{0, w_0\}$. The other case, in which the affine support of f''' is $\oplus_3$, is similar.

We have established a reduction from either $\#\mathsf{CSP}(\oplus_3, U_\lambda, \delta_0, \delta_1)$ or $\#\mathsf{CSP}(\neg\oplus_3, U_\lambda, \delta_0, \delta_1)$, which are both #P-hard by Lemma 13. □

LEMMA 15. *If $f \in \mathcal{F}_2$ is not of product type, then $\#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda\})$ is #P-hard for any positive $\lambda \neq 1$.*

Proof. Take an instance I of $\#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda, U_c\})$, from Lemma 14, with n variables $x_1, x_2, \ldots, x_n$. We want to compute the partition function $Z(I)$ using only instances of $\#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda\})$, that is, instances which avoid using constraints U_c. For each i, let m_i denote the number of copies of U_c that are applied to x_i, and let $m = \sum_{i=1}^n m_i$. Then we can write the partition function as $Z(I) = Z(I; c)$, where

$$Z(I; w) = \sum_{\sigma \in \{0,1\}^n} \hat{Z}(\sigma) \prod_{i:\sigma_i = 1} w^{m_i} = \sum_{\sigma \in \{0,1\}^n} \hat{Z}(\sigma) w^{\sum_{i=1}^n m_i \sigma_i},$$

where $\hat{Z}(\sigma)$ denotes the value corresponding to the assignment $\sigma(x_i) = \sigma_i$, ignoring constraints applying U_c, and w is a variable. So $\hat{Z}(\sigma)$ is the weight of σ, taken over all constraints other than those applying U_c. Note also that $Z(I; w)$ is a polynomial of degree m in w. We can evaluate $Z(I; w)$ at the point $w = \lambda^j$ by replacing each U_c constraint with j copies of a U_λ constraint. This evaluation is an instance of $\#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda\})$. So, using m different values of j and interpolating, we learn the coefficients of the polynomial $Z(I; w)$. Then we can set $w = c$ to evaluate $Z(I)$. □

LEMMA 16. *Suppose that $f \in \mathcal{F}_2$ is not of product type and $g \in \mathcal{F}_2$ is not pure affine. Then $\#\mathsf{CSP}(\{f, g, \delta_0, \delta_1\})$ is #P-hard.*

Proof. If g does not have affine support, we are done by Lemma 11. So suppose that g has affine support. Since g is not pure affine, the range of g contains at least two nonzero values.

The high-level idea will be to use pinning and bisection to extract a nontrivial unary weight function U_λ from g. Then we can reduce from $\#\mathsf{CSP}(\{f, \delta_0, \delta_1, U_\lambda\})$, which we proved #P-hard in Lemma 15.

Look at the relation R_g, viewed as a table. If every column were constant, then g would be pure affine, so this is not the case. Select a nonconstant column with index h. If there are two nonzero values in the range of g amongst the rows of R_g that are 0 in column h, then we derive a new function g' by pinning column h to 0. The new function g' is not pure affine, since the two nonzero values prevent this. So we will show inductively that $\#\mathsf{CSP}(\{f, g', \delta_0, \delta_1\})$ is #P-hard. This will give the result since $\#\mathsf{CSP}(\{f, g', \delta_0, \delta_1\})$ trivially reduces to $\#\mathsf{CSP}(\{f, g, \delta_0, \delta_1\})$.

If we don't finish this way, or symmetrically by pinning column h to 1, then we know that there are distinct positive values w_0 and w_1 such that, for every row x of R_g with 0 in column h, $g(x) = w_0$ and, for every row x of R_g with 1 in column h, $g(x) = w_1$. Now note that, because the underlying relation R_g is affine, it has the same number of 0's in column h as 1's. This is because R_g is the solution of a set of linear equations. Adding the equation $x_h = 0$ or $x_h = 1$ exactly halves the set of solutions in either case. We now project onto the index set $\{h\}$. We obtain the

unary weight function U_λ, with $\lambda = w_1/w_0$, on using the earlier observation about proportional functions. This was our goal and completes the proof. □

Lemma 5 now follows from Lemmas 8 and 16, completing the proof of Theorem 4.

Appendix. The purpose of this appendix is to prove Lemma 7 for an arbitrary fixed domain $[q]$. We used only the special case $q = 2$, which we stated and proved as Lemma 8. However, pinning appears to be a useful technique for studying the complexity of #CSP, so we give a proof of the general Lemma 7, which we believe will be applicable elsewhere.

In order to prove the lemma, we introduce a useful, but less natural, variant of #CSP. Suppose $\mathcal{F} \subseteq \mathcal{F}_q$. An instance I of $\#\mathsf{CSP}^{\neq}(\mathcal{F})$ consists of a set V of variables and a set $\mathcal{C}$ of constraints, just like an instance of $\#\mathsf{CSP}(\mathcal{F})$. In addition, the instance may contain a *single* extra constraint C applying the arity-q *disequality* relation $\chi_{\neq}$ with scope $(v_{C,1}, \ldots, v_{C,q})$.

The disequality relation $\chi_{\neq}$ is defined by $\chi_{\neq}(x_1, \ldots, x_q) = 1$ if $x_1, \ldots, x_q \in [q]$ are pairwise distinct, that is, if they are a permutation of the domain $[q]$. Otherwise, $\chi_{\neq}(x_1, \ldots, x_q) = 0$.

Lemma 7 follows immediately from Lemmas 17 and 18 below.

LEMMA 17. *For every* $\mathcal{F} \subseteq \mathcal{F}_q$, $\#\mathsf{CSP}(\mathcal{F} \cup \bigcup_{c\in[q]} \delta_c) \leq_{\mathrm{T}} \#\mathsf{CSP}^{\neq}(\mathcal{F})$.

Proof. We follow the proof lines of Lemma 8, but instead of subtracting the contribution corresponding to configurations in which some t_i's get the same value, we use the disequality relation to restrict the partition function to configurations in which they get distinct values.

Say that $\mathcal{F}$ is *symmetric* if it is the case that for every arity-k function $f \in \mathcal{F}$, every tuple $x \in [q]^k$, and every permutation $\pi : [q] \to [q]$, $f(x_1, \ldots, x_k) = f(\pi(x_1), \ldots, \pi(x_k))$.

Let I be an instance of $\#\mathsf{CSP}(\mathcal{F} \cup \bigcup_{c\in[q]} \delta_c)$ with variable set V. Let V_c be the set of variables $v \in V$ to which the constraint $\delta_c(v)$ is applied. Assume without loss of generality that the sets V_c are pairwise disjoint. Let $V_q = V \setminus \bigcup_{c\in[q]} V_c$. We construct an instance I' of $\#\mathsf{CSP}^{\neq}(\mathcal{F})$. The instance has variables $V_q \cup \{t_0, \ldots, t_{q-1}\}$. Every constraint C of I involving a function $f \in \mathcal{F}$ corresponds to a constraint C' of I'. Here C' is the same as C except that variables in V_c are replaced with t_c, for each $c \in [q]$. Also, we add a new disequality constraint to the new variables $t_0, \ldots, t_{q-1}$.

Case 1. $\mathcal{F}$ is symmetric. By construction, $Z(I') = \sum_{y_0,\ldots,y_{q-1}} Z(I' \mid \sigma(t_0) = y_0, \ldots, \sigma(t_{q-1}) = y_{q-1})$, where the sum is over all permutations $y_0, \ldots, y_{q-1}$ of $[q]$. By symmetry, the summands are all the same, so $Z(I') = q!Z(I' \mid \sigma(t_0) = 0, \ldots, \sigma(t_{q-1}) = q-1) = q!Z(I)$.

Case 2. $\mathcal{F}$ is not symmetric. Say that two permutations $\pi_1 : [q] \to [q]$ and $\pi_2 : [q] \to [q]$ are *equivalent* if, for every $f \in \mathcal{F}$ and every tuple $x \in [q]^k$, $f(\pi_1(x_1), \ldots, \pi_1(x_k)) = f(\pi_2(x_1), \ldots, \pi_2(x_k))$. Partition the permutations $\pi : [q] \to [q]$ into equivalence classes. Let h be the number of equivalence classes and n_i be the size of the ith equivalence class, so $n_1 + \cdots + n_h = q!$.[6] Let $\{\pi_1, \ldots, \pi_h\}$ be a set of representatives of the equivalence classes with π_1 being the identity. We know that $n_1 \neq q!$ since $\mathcal{F}$ is not symmetric.

For a positive integer ℓ we will now build an instance I'_ℓ by adding new constraints to I'. For each π_i other than π_1 we add constraints as follows. Choose a function $f_i \in \mathcal{F}$

[6] In fact, it can be shown that these equivalence classes are cosets of the symmetry group of f, and hence are of equal size, though we do not use this fact here.

and a tuple y such that $f_i(y_1,\ldots,y_k) \neq f_i(\pi_i(y_1),\ldots,\pi_i(y_k))$. If $f_i(y_1,\ldots,y_k) > f_i(\pi_i(y_1),\ldots,\pi_i(y_k))$, then define the k-tuple x^i by $(x^i_1,\ldots,x^i_k) = (y_1,\ldots,y_k)$. Otherwise, let n be the order of the permutation π_i and let g_r denote $f_i(\pi_i^r(y_1),\ldots,\pi_i^r(y_k))$. Since $g_0 < g_1$ and $g_n = g_0$ there exists an $\xi \in \{1,\ldots,n-1\}$ such that $g_\xi > g_{\xi+1}$. Let $(x^i_1,\ldots,x^i_k) = (\pi^\xi(y_1),\ldots,\pi^\xi(y_k))$ so $f_i(x^i_1,\ldots,x^i_k) > f_i(\pi_i(x^i_1),\ldots,\pi_i(x^i_k))$.

Let w_{ij} denote $f_i(\pi_j(x^i_1),\ldots,\pi_j(x^i_k))$ so, since π_1 is the identity, we have just ensured that $w_{i1} > w_{ii}$. Let $s^i = (t_{x^i_1},\ldots,t_{x^i_k})$, and let $0 \leq z_i \leq h$ $(i = 2,\ldots,h)$ be positive integers, which we will determine below. Add ℓz_i new constraints to I'_ℓ with relation f_i and scope s^i. Let $\lambda_i = \prod_{\gamma=2}^h w_{\gamma i}^{z_\gamma}$. Note that, given $\sigma(t_0) = \pi_i(0),\ldots,\sigma(t_{q-1}) = \pi_i(q-1)$, the contribution to $Z(I'_\ell)$ for the new constraints is

$$\begin{aligned}\prod_{\gamma=2}^h f_\gamma(\sigma(t_{x^\gamma_1}),\ldots,\sigma(t_{x^\gamma_k}))^{z_\gamma \ell} &= \prod_{\gamma=2}^h f_\gamma(\pi_i(x^\gamma_1),\ldots,\pi_i(x^\gamma_k))^{z_\gamma \ell} \\ &= \prod_{\gamma=2}^h w_{\gamma,i}^{z_\gamma \ell} = \Bigg(\prod_{\gamma=2}^h w_{\gamma,i}^{z_\gamma}\Bigg)^\ell = {\lambda_i}^\ell.\end{aligned}$$

So

$$Z(I'_\ell) = \sum_{i=1}^h n_i\, Z(\, I' \mid \sigma(t_0) = \pi_i(0),\ldots,\sigma(t_{q-1}) = \pi_i(q-1))\, \lambda_i^\ell.$$

We have ensured that $\lambda_1 > 0$, since $w_{i1} > w_{ii} \geq 0$, so $w_{i1} > 0$ for all $i = 2,\ldots,h$. We now choose the z_i's so that $\lambda_i \neq \lambda_1$ for all $i = 2,\ldots,h$. If $w_{\gamma i} = 0$ for any $\gamma = 2,\ldots,h$, we have $\lambda_i = 0$ and hence $\lambda_i \neq \lambda_1$. Thus we will assume, without loss of generality, that $w_{\gamma i} > 0$ for all $\gamma = 2,\ldots,h$ and $i = 2,\ldots,h'$, where $h' \leq h$. Then we have

$$\frac{\lambda_i}{\lambda_1} \;=\; \prod_{\gamma=2}^h \Big(\frac{w_{\gamma i}}{w_{\gamma 1}}\Big)^{z_\gamma} \;=\; e^{\sum_{\gamma=2}^h \alpha_{\gamma i} z_\gamma} \qquad (i = 2,\ldots,h'),$$

where $\alpha_{\gamma i} = \ln(w_{\gamma i}/w_{\gamma 1})$. Note that $\alpha_{ii} < 0$, since $w_{ii} < w_{i1}$. We need to find an integer vector $z = (z_2,\ldots,z_h)$ so that none of the linear forms $\mathcal{L}_i(z) = \sum_{\gamma=2}^h \alpha_{\gamma i} z_\gamma$ is zero, for $i = 2,\ldots,h'$. We do this using a proof method similar to the Schwartz–Zippel lemma. (See, for example, [20].) None of the $\mathcal{L}_i(z)$ is identically zero, since $\alpha_{ii} \neq 0$. Consider the integer vectors $z \in [h]^{h-1}$. At most h^{h-2} of these can make $\mathcal{L}_i(z)$ zero for any i, since the equation $\mathcal{L}_i(z) = 0$ makes z_i a linear function of z_γ $(\gamma \neq i)$. Therefore there are at most $(h'-1)h^{h-2} < h^{h-1}$ such z which make any $\mathcal{L}_i(z)$ zero. Therefore there must be a vector $z \in [h]^{h-1}$ for which none of the $\mathcal{L}_i(z)$ is zero, and this is the vector we require.

Now, by combining terms with equal λ_i and ignoring terms with $\lambda_i = 0$, we can view $Z(I'_\ell)$ as a sum $Z(I'_\ell) = \sum_i c_i \lambda_i^\ell$, where the λ_i's are positive and pairwise distinct and

$$c_1 = n_1 Z(I' \mid \sigma(t_0) = 0,\ldots,\sigma(t_{q-1}) = q-1).$$

Thus, by Lemma 3.2 of [8] we can interpolate to recover c_1. Dividing by n_1, we get

$$Z(I' \mid \sigma(t_0) = 0,\ldots,\sigma(t_{q-1}) = q-1) = Z(I). \qquad \square$$

LEMMA 18. *For every* $\mathcal{F} \subseteq \mathcal{F}_q$, $\#\mathsf{CSP}^{\neq}(\mathcal{F}) \leq_{\mathrm{T}} \#\mathsf{CSP}(\mathcal{F})$.

Proof. We use Möbius inversion for posets, following the lines of the proof of [2, Theorem 8].[7] Consider the set of partitions of $[q]$. Let $\underline{0}$ denote the partition with q singleton classes. Consider the partial order in which $\eta \leq \theta$ if and only if every class of η is a subset of some class of θ. Define $\mu(\underline{0}) = 1$, and for any $\theta \neq \underline{0}$ define $\mu(\theta) = -\sum_{\eta \leq \theta, \eta \neq \theta} \mu(\eta)$. Consider the sum $\sum_{\eta \leq \theta} \mu(\eta)$. Clearly, this sum is 1 if $\theta = \underline{0}$. From the definition of μ, it is also easy to see that the sum is 0 otherwise, since

$$\sum_{\eta \leq \theta} \mu(\eta) = \mu(\theta) + \sum_{\eta \leq \theta, \eta \neq \theta} \mu(\eta) = 0.$$

Now let I be an instance of $\#\mathsf{CSP}^{\neq}(\mathcal{F})$ with a disequality constraint applied to variables $t_0, \ldots, t_{q-1}$. Let V be the set of variables of I. Given a configuration $\sigma : V \to [q]$, let $\vartheta(\sigma)$ be the partition of $[q]$ induced by of $(\sigma(t_0), \ldots, \sigma(t_{q-1}))$. Thus i and j in $[q]$ are in the same class of $\vartheta(\sigma)$ if and only if $\sigma(t_i) = \sigma(t_j)$. We say that a partition η is consistent with σ (written $\eta \preccurlyeq \sigma$) if $\eta \leq \vartheta(\sigma)$. Note that $\eta \preccurlyeq \sigma$ means that for any i and j in the same class of η, $\sigma(t_i) = \sigma(t_j)$.

Let Ω be the set of configurations σ that satisfy all constraints in I except possibly the disequality constraint. Then $Z(I) = \sum_{\sigma \in \Omega} w(\sigma) \mathbb{1}_\sigma$, where $\mathbb{1}_\sigma = 1$ if σ respects the disequality constraint, meaning that $\vartheta(\sigma) = \underline{0}$, and $\mathbb{1}_\sigma = 0$ otherwise. By the Möbius inversion formula derived above,

$$Z(I) = \sum_{\sigma \in \Omega} w(\sigma) \sum_{\eta \leq \vartheta(\sigma)} \mu(\eta).$$

Changing the order of summation, we get

$$Z(I) = \sum_{\eta} \mu(\eta) \sum_{\eta \leq \theta} \sum_{\sigma \in \Omega : \vartheta(\sigma) = \theta} w(\sigma) = \sum_{\eta} \mu(\eta) \sum_{\sigma \in \Omega : \eta \preccurlyeq \sigma} w(\sigma).$$

Now note that $\sum_{\sigma : \eta \preccurlyeq \sigma} w(\sigma)$ is the partition function $Z(I_\eta)$ of an instance I_η of $\#\mathsf{CSP}(\mathcal{F})$. The instance I_η is formed from I by ignoring the disequality constraint and identifying variables in $t_0, \ldots, t_{q-1}$ whose indices are in the same class of η. Thus we can compute all the $Z(I_\eta)$ in $\#\mathsf{CSP}(\mathcal{F})$. Finally, $Z(I) = \sum_\eta \mu(\eta) Z(I_\eta)$, completing the reduction. □

REFERENCES

[1] G. BRIGHTWELL AND P. WINKLER, *Graph homomorphisms and phase transitions*, J. Combin. Theory Ser. B, 77 (1999), pp. 221–262.

[2] A. BULATOV AND V. DALMAU, *Towards a dichotomy theorem for the counting constraint satisfaction problem*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 2003, pp. 562–573.

[3] D. COHEN, M. COOPER, P. JEAVONS, AND A. KROKHIN, *The complexity of soft constraint satisfaction*, Artificial Intelligence, 170 (2006), pp. 983–1016.

[4] A. BULATOV AND M. GROHE, *The complexity of partition functions*, Theoret. Comput. Sci., 348 (2005), pp. 148–186.

[5] B. CIPRA, *An introduction to the Ising model*, Amer. Math. Monthly, 94 (1987), pp. 937–959.

[6] N. CREIGNOU AND M. HERMANN, *Complexity of generalized satisfiability counting problems*, Inform. and Comput., 125 (1996), pp. 1–12.

[7] N. CREIGNOU, S. KHANNA, AND M. SUDAN, *Complexity Classifications of Boolean Constraint Satisfaction Problems*, SIAM Monogr. Discrete Math. Appl. 7, SIAM, Philadelphia, 2001.

[7] Lovász [14] had previously used Möbius inversion in a similar context.

[8] M. Dyer and C. Greenhill, *The complexity of counting graph homomorphisms*, Random Structures Algorithms, 17 (2000), pp. 260–289.
[9] M. Dyer, L. A. Goldberg, and M. Paterson, *On counting homomorphisms to directed acyclic graphs*, in Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 4051, Springer, New York, 2006, pp. 38–49.
[10] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.
[11] L. A. Goldberg and M. Jerrum, *Inapproximability of the Tutte polynomial*, Inform. Comput., 206 (2008), pp. 908–929.
[12] C. Greenhill, *The complexity of counting colourings and independent sets in sparse graphs and hypergraphs*, Comput. Complexity, 9 (2000), pp. 52–72.
[13] P. Hell and J. Nešetřil, *Graphs and Homomorphisms*, Oxford University Press, London, 2004.
[14] L. Lovász, *Operations with structures*, Acta Math. Hungarica, 18 (1967), pp. 321–328.
[15] R. Ladner, *On the structure of polynomial time reducibility*, J. ACM, 22 (1975), pp. 155–171.
[16] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.
[17] F. Rossi, P. van Beek, and T. Walsh, eds., *Handbook of Constraint Programming*, Elsevier, New York, 2006.
[18] T. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1978, pp. 216–226.
[19] A. Scott and G. Sorkin, *Polynomial Constraint Satisfaction: A Framework for Counting and Sampling CSPs and Other Problems*, online at http://arxiv.org/abs/cs/0604079.
[20] J. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.
[21] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.
[22] D. Welsh, *Complexity: Knots, Colourings and Counting*, LMS Lecture Note Ser. 186, Cambridge University Press, Cambridge, UK, 1993.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1987–2006

ON THE COMPLEXITY OF NUMERICAL ANALYSIS*

ERIC ALLENDER†, PETER BÜRGISSER‡, JOHAN KJELDGAARD-PEDERSEN§, AND PETER BRO MILTERSEN¶

Abstract. We study two quite different approaches to understanding the complexity of fundamental problems in numerical analysis: (a) the Blum–Shub–Smale model of computation over the reals; and (b) a problem we call the "generic task of numerical computation," which captures an aspect of doing numerical computation in floating point, similar to the "long exponent model" that has been studied in the numerical computing community. We show that both of these approaches hinge on the question of understanding the complexity of the following problem, which we call PosSLP: Given a division-free straight-line program producing an integer N, decide whether $N > 0$. In the Blum–Shub–Smale model, polynomial-time computation over the reals (on discrete inputs) is polynomial-time equivalent to PosSLP when there are only algebraic constants. We conjecture that using transcendental constants provides no additional power, beyond *nonuniform* reductions to PosSLP, and we present some preliminary results supporting this conjecture. The generic task of numerical computation is also polynomial-time equivalent to PosSLP. We prove that PosSLP lies in the counting hierarchy. Combining this with work of Tiwari, we obtain that the Euclidean traveling salesman problem lies in the counting hierarchy—the previous best upper bound for this important problem (in terms of classical complexity classes) being PSPACE. In the course of developing the context for our results on arithmetic circuits, we present some new observations on the complexity of the arithmetic circuit identity testing (ACIT) problem. In particular, we show that if $n!$ is not ultimately easy, then ACIT has subexponential complexity.

Key words. Blum–Shub–Smale model, arithmetic circuits, counting hierarchy, BPP, sum of square roots problem, straight-line programs

AMS subject classifications. 68Q15, 68Q17

DOI. 10.1137/070697926

1. Introduction.

The original motivation for this paper comes from a desire to understand the complexity of computation over the reals in the Blum–Shub–Smale model. In section 1.1 we give a brief introduction to this model, and we introduce the problem PosSLP and explain its importance in understanding the Blum–Shub–Smale model.

In section 1.2 we present yet another reason to be interested in PosSLP. We isolate a computational problem that lies at the root of the task of designing numerically stable algorithms. We show that this task is computationally equivalent to PosSLP. The material in sections 1.1 and 1.2 provides motivation for studying PosSLP and for attempting to place it within the framework of traditional complexity classes.

In section 1.3 we discuss our main technical contributions: proving upper and

*Received by the editors July 23, 2007; accepted for publication (in revised form) September 5, 2008; published electronically January 14, 2009.

http://www.siam.org/journals/sicomp/38-5/69792.html

†Department of Computer Science, Rutgers University, Piscataway, NJ 08854-8019 (allender@cs.rutgers.edu). The first author's work was supported by NSF grants CCF-0514155, CCF-0830133, and CCF-0832787.

‡Institute of Mathematics, Paderborn University, DE-33095 Paderborn, Germany (pbuerg@upb.de). This author's research was supported by DFG grant BU 1371.

§Decision Sciences Practice, PA Consulting Group, Tuborg Blvd. 5, DK 2900 Hellerup, Denmark (johan.kjeldgaard-pedersen@paconsulting.com).

¶Department of Computer Science, University of Aarhus, IT-parken, DK 8200 Aarhus N, Denmark (bromille@cs.au.dk). This author's research was supported by SNF grants 21-04-0551 and 272-07-0440 and by the Center for Algorithmic Game Theory, funded by the Carlsberg Foundation.

lower bounds on the complexity of PosSLP. In section 1.4 we present applications of our main result with respect to the Euclidean traveling salesman problem and the sum-of-square-roots problem.

1.1. Polynomial time over the reals. The Blum–Shub–Smale model of computation over the reals provides a very well-studied complexity-theoretic setting in which to study the computational problems of numerical analysis. We refer the reader to Blum et al. [12] for detailed definitions and background material related to this model; here, we will recall only a few salient facts. In the Blum–Shub–Smale model, each machine computing over the reals has associated with it a finite set S of real *machine constants*. The inputs to a machine are elements of $\bigcup_n \mathbb{R}^n = \mathbb{R}^\infty$, and thus each polynomial-time machine over $\mathbb{R}$ accepts a "decision problem" $L \subseteq \mathbb{R}^\infty$. The set of decision problems accepted by polynomial-time machines over $\mathbb{R}$ using only constants from $S \cup \{0,1\}$ is denoted $\mathrm{P}_{\mathbb{R}}^S$. The union of the classes $\mathrm{P}_{\mathbb{R}}^S$ over all S is called *polynomial time over* $\mathbb{R}$ and is denoted $\mathrm{P}_{\mathbb{R}}$. The subclass $\mathrm{P}_{\mathbb{R}}^\emptyset$ of "constant-free polynomial time" is commonly denoted by $\mathrm{P}_{\mathbb{R}}^0$; cf. Bürgisser and Cucker [20].

There has been considerable interest in relating computation over $\mathbb{R}$ to the classical Boolean complexity classes such as P, NP, PSPACE, etc. This is accomplished by considering the *Boolean part* of decision problems over the reals. That is, given a problem $L \subseteq \mathbb{R}^\infty$, the Boolean part of L is defined as $\mathrm{BP}(L) := L \cap \{0,1\}^\infty$. (Here, we follow the notation of [12]; $\{0,1\}^\infty = \bigcup_n \{0,1\}^n$, which is identical to $\{0,1\}^*$.) The Boolean part of $\mathrm{P}_{\mathbb{R}}$, denoted $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$, is defined as $\{\mathrm{BP}(L) \mid L \in \mathrm{P}_{\mathbb{R}}\}$.

By encoding the advice function in a single real constant as in Koiran [44], one can show that $\mathsf{P}/\mathrm{poly} \subseteq \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$. The best upper bound on the complexity of problems in $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ that is currently known was obtained by Cucker and Grigoriev [27]:

$$\mathrm{BP}(\mathrm{P}_{\mathbb{R}}) \subseteq \mathsf{PSPACE}/\mathrm{poly}. \tag{1.1}$$

There has been *no* work pointing to lower bounds on the complexity of $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$; nobody has presented any compelling evidence that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ is not equal to P/poly. There has also been some suggestion that perhaps $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ is equal to $\mathsf{PSPACE}/\mathrm{poly}$. For instance, certain variants of the RAM model that provide for unit-cost arithmetic can simulate all of PSPACE in polynomial time [10, 37]. Since the Blum–Shub–Smale model also provides for unit-time multiplication on "large" numbers, Cucker and Grigoriev [27] mention that researchers have raised the possibility that similar arguments might show that polynomial-time computation over $\mathbb{R}$ might be able to simulate PSPACE. Cucker and Grigoriev also observe that certain naïve approaches to providing such a simulation must fail.

One of our goals is to provide evidence that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ lies properly between P/poly and $\mathsf{PSPACE}/\mathrm{poly}$. Towards this goal, it is crucial to understand a certain decision problem PosSLP: *Decide, for a given division-free straight-line program, whether it represents a positive integer.* More generally, for a fixed finite subset $S \subset \mathbb{R}$, $\mathrm{PosSLP}(S)$ is the problem of deciding for a given division-free straight-line program, using constants from $S \cup \{0,1\}$, whether the real number represented by it is positive. (For precise definitions, see the next section.)

The immediate relationship between the Blum–Shub–Smale model and the problems $\mathrm{PosSLP}(S)$ is given by the proposition below.

PROPOSITION 1.1. *We have* $\mathsf{P}^{\mathrm{PosSLP}(S)} = \mathrm{BP}(\mathrm{P}_{\mathbb{R}}^S)$ *for all finite subsets* $S \subset \mathbb{R}$. *In particular,* $\mathsf{P}^{\mathrm{PosSLP}} = \mathrm{BP}(\mathrm{P}_{\mathbb{R}}^0)$.

Proof. It is clear that $\mathrm{PosSLP}(S)$ is in $\mathrm{BP}(\mathrm{P}_{\mathbb{R}}^S)$, since we can implement a standard SLP (straight-line program) interpreter in the real Turing machine framework

and evaluate the result in linear time using unit-cost instructions. The result is then obtained by one sign test. To show the other direction, assume that we have a polynomial-time machine over $\mathbb{R}$ using only the constants in $S \cup \{0, 1\}$. By a usual argument (separate computation of numerator and denominator), we may assume without loss of generality that the machine does not use divisions. Given a bit string as input, we simulate the computation by storing the straight-line program representation of the intermediate results instead of their values. Branch instructions can be simulated by using the oracle PosSLP(S) to determine if the contents of a given register (represented by a straight-line program) are greater than zero. □

It was shown by Chapuis and Koiran [24] that algebraic constants do not help. More specifically, $\mathrm{P}_{\mathbb{R}}^0$ is equal to the class of decision problems over the reals decided by polynomial-time Blum–Shub–Smale machines using real algebraic numbers as constants.

As already mentioned, by encoding the advice function in a single real constant, one can show that $\mathsf{P}/\mathrm{poly} \subseteq \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$. The proof in fact shows even $\mathsf{P}^{\mathrm{PosSLP}}/\mathrm{poly} \subseteq \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$. The real constant encoding the advice function, will, of course, in general be transcendental. Thus, there is a strong relationship between nonuniformity in the classical model of computation and the use of transcendental constants in the Blum–Shub–Smale model. We conjecture that this relationship can be further strengthened, as follows.

Conjecture 1. $\mathsf{P}^{\mathrm{PosSLP}}/\mathrm{poly} = \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$.

In section 3 we present some preliminary results toward proving this conjecture. For instance, we prove that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}}^{\{\alpha\}}) \subseteq \mathsf{P}^{\mathrm{PosSLP}}/\mathrm{poly}$ for almost all $\alpha \in \mathbb{R}$, in the sense of Lebesgue measure. We also show that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}}^{\{\alpha\}}) \subseteq \mathsf{P}^{\mathrm{PosSLP}}/1$ (one bit of advice) if α is the value of an elementary function on a rational number. This is the case, for instance, for the well-known transcendental numbers e or π.

1.2. The task of a numerical analyst. The Blum–Shub–Smale model is a very elegant one, but it does not take into account the fact that actual numerical computations have to deal with *finitely* represented values. We next observe that even if we take this into account, the PosSLP problem still captures the complexity of numerical computation.

Let $u \neq 0$ be a dyadic rational number. The *floating point* representation of u is obtained by writing $u = v2^m$, where m is an integer and $\frac{1}{2} \leq |v| < 1$. The floating point representation is then given by the sign of v and the usual binary representations of the numbers $|v|$ and m. The floating point representation of 0 is the string 0 itself. We shall abuse notation and identify the floating point representation of a number with the number itself, using the term "floating point number" for the number as well as its representation.

Let $u \neq 0$ be a real number. We may write u as $u = u'2^m$, where $\frac{1}{2} \leq |u'| < 1$ and m is an integer. Then, we define a *floating point approximation of u with k significant bits* to be a floating point number $v2^m$ so that $|v - u'| \leq 2^{-(k+1)}$.

We will focus on one part of the job that is done by numerical analysts: the design of numerically stable algorithms. In our scenario, the numerical analyst starts out with a known function f, and the task is to design a "good" algorithm for it. When we say that the function f is "known," we mean that the analyst starts out with some method of computing (or at least approximating) f; we restrict attention to the "easy" case where the method for computing f uses only the arithmetic operations $+, -, *, \div$, and thus the description of f that the analyst is given can be presented as an arithmetic circuit with operations $+, -, *, \div$. Usually, the analyst also has to worry about the

problems that are caused by the fact that the inputs to f are not known precisely but are only given as floating point numbers that are approximations to the "true" inputs—but again we will focus on the "easy" case where the analyst will merely try to compute a good approximation for $f(x_1, \ldots, x_n)$ on the exact floating point numbers $x_1, \ldots, x_n$ that are presented as input, as follows.

THE GENERIC TASK OF NUMERICAL COMPUTATION (GTNC). *Given a straight-line program P (with $\div$), and given inputs $x_1, \ldots, x_n$ for P (as floating point numbers) and an integer k in unary, along with a promise that $P(x_1, \ldots, x_n)$ neither evaluates to zero nor does division by zero, compute a floating point approximation of the value of the output $P(x_1, \ldots, x_n)$ with k significant bits.*

The traditional approach that numerical analysts have followed in trying to solve problems of this sort is to study the numerical stability of the algorithm represented by the circuit and, in case of instability, to attempt to devise an equivalent computation that is numerically stable. Although stable algorithms have been found for a great many important functions, the task of devising such algorithms frequently involves some highly nontrivial mathematics and algorithmic ingenuity. There seems to be no expectation that there will ever be a purely automatic way to solve this problem, and indeed there seems to be no expectation that a numerically stable algorithm will exist in general. To summarize, there is substantial empirical evidence that the generic task of numerical computation is intractable. It would be of significant practical interest if, contrary to expectation, it should turn out to be very easy to solve (say, solvable in linear time).

We show that the generic task of numerical computation is equivalent in power to PosSLP.

PROPOSITION 1.2. *The GTNC is polynomial-time Turing equivalent to* PosSLP.

Proof. We first reduce PosSLP to the GTNC. Given a division-free straight-line program representing the number N, we construct a straight-line program computing the value $v = 2N - 1$. The only inputs $0, 1$ of this program can be considered to be floating point numbers, and this circuit clearly satisfies the promise of the GTNC. Then $N > 0$ if $v \geq 1$, and $N \leq 0$ if $v \leq -1$. Determining an approximation of v to one significant bit is enough to distinguish between these cases.

Conversely, suppose we have an oracle solving PosSLP. Given a straight-line program with inputs being floating point numbers, we first convert it to a straight-line program having only input 1; it is easy to see that this can be done in polynomial time. By standard techniques we move all $\div$ gates to the top, so that the program computes a value $v = v_1/v_2$, where v_1, v_2 are given by division-free straight-line programs. We can use the oracle to determine the signs of v_1 and v_2. Without loss of generality assume that v is positive. Next we use the oracle to determine whether $v_1 \geq v_2$. Suppose that this is indeed the case (the opposite case is handled similarly).

We then find the least r so that $2^{r-1} \leq v < 2^r$, by first comparing v_1 with $v_2 2^{2^i}$ for $i = 0, 1, 2, 3, \ldots$, using the oracle, thus finding the minimum i so that $v < 2^{2^i}$, and afterwards doing a binary search, again using the oracle to compare v_1 to $v_2 2^r$ for various values of r. This takes polynomial time.

The desired output is a floating point number $u = u'2^r$, where $|v - u'| \leq 2^{-(k+1)}$. To obtain u' we first want to find the integer w between 2^k and $2^{k+1} - 1$ so that $w/2^{k+1} \leq v/2^r < (w+1)/2^{k+1}$. Since $w/2^{k+1} \leq v/2^r < (w+1)/2^{k+1}$ iff $w2^r v_2 \leq v_1 2^{k+1} < (w+1)2^r v_2$, we can determine this by another binary search, using $O(k)$ calls to the oracle. We then output the sign of v, the binary representation of the rational $w/2^{k+1}$, and the binary representation of r, together forming the desired floating point approximation of v. ∎

The reader may wonder how GTNC fits into the numerical analysis literature. The long exponent model (LEM) of Demmel [31, 30] offers the closest parallel. Demmel considers the classic problem of computation of the determinant, and he identifies three ways of modeling the problem, which he calls the traditional model, the short exponent model (SEM), and the LEM. Computing determinants is easy in the SEM, while in the LEM the problem is equivalent to a special case of GTNC. Namely, it is equivalent to instances of GTNC where the circuit C that is provided as input is the polynomial-size SLP for determinants given by Berkowitz [9].

Demmel goes so far as to conjecture that, in the LEM, the problem of deciding whether the determinant is zero is NP-hard [31]. Since this problem is actually a special case of EquSLP and thus lies in BPP, Demmel's conjecture is almost certainly false. However, we agree with his underlying intuition, in that we believe that the problem of deciding whether the determinant is *positive* in the LEM very likely is intractable (even if we see no evidence that it is NP-hard) [57]. That is, this special case of PosSLP is recognized as a difficult problem by the numerical analysis community.

1.3. The complexity of PosSLP. We consider Proposition 1.2 to be evidence for the computational intractability of PosSLP. If PosSLP is in P/poly, then there is a polynomial-sized "cookbook" that can be used in place of the creative task of devising numerically stable computations. This seems unlikely.

We wish to emphasize that the generic task of numerical computation models the *discrete* computational problem that underlies an important class of computational problems. Thus it differs quite fundamentally from the approach taken in the Blum–Shub–Smale model.

We also wish to emphasize that, in defining the generic task of numerical computation, we are *not* engaging in the debate over which real functions are "efficiently computable." There is by now a large literature comparing and contrasting the relative merits of the Blum–Shub–Smale model with the so-called bit model of computing, and there are various competing approaches to defining what it means for a real-valued function to be feasible to compute; see [13, 17, 16, 67, 68] among others. Our concerns here are orthogonal to that debate. We are not trying to determine which real-valued functions are feasible; we are studying a discrete computational problem that is relevant to numerical analysis, with the goal of proving upper and lower bounds on its complexity.

The generic task of numerical computation is one way of formulating the notion of what is feasible to compute in a world where *arbitrary precision* arithmetic is available for free. In contrast, the Blum–Shub–Smale model can be interpreted as formulating the notion of feasibility in a world where *infinite precision* arithmetic is available for free. According to Proposition 1.2, both of these approaches are *equivalent* (and captured by $\mathsf{P}^{\mathrm{PosSLP}}$) when only algebraic constants are allowed in the Blum–Shub–Smale model. Conjecture 1 claims that this is also true when allowing arbitrary real constants.

As another demonstration of the computational power of PosSLP, we show in section 2 that the problem of determining the total degree of a multivariate polynomial over the integers given as a straight-line program reduces to PosSLP.

The above discussion suggests that PosSLP is not an easy problem. Can more formal evidence of this be given? Although it would be preferable to show that PosSLP is hard for some well-studied complexity class, the best that we can do is observe that a somewhat stronger problem (BitSLP) is hard for $\#\mathsf{P}$. This will be done in section 2.

The above discussion also suggests that nontrivial upper bounds for PosSLP are

of great interest. Prior to this paper, the best upper bound was PSPACE. Our main technical result is an improved upper bound: We show, based on results on the uniform circuit complexity of integer division and the relationship between constant depth circuits and subclasses of PSPACE [6, 39], that PosSLP lies in the counting hierarchy CH, a well-studied subclass of PSPACE that bears more or less the same relationship to #P as the polynomial hierarchy bears to NP [64, 66].

THEOREM 1.3. PosSLP *is in* $\mathsf{P}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$.

Another interesting upper bound for PosSLP was recently discovered by Tarasov and Vyalyi [61], who give a reduction from PosSLP to the *semidefinite feasibility problem* (SDFP), i.e., the feasibility version of the optimization problem *semidefinite programming*. Their result can be seen as a lower bound for SDFP. SDFP is known to reduce to its complement and to lie in $\mathrm{NP}_{\mathbb{R}}$ [54]; also it is easy to see that SDFP reduces to the existential theory of the reals (for instance, see the discussion in [54]), and thus SDFP $\in$ PSPACE.

We suspect that PosSLP lies at an even lower level of CH. Note that, in presenting our upper bound, we do not exploit some powerful techniques that have been proved useful in computing certain bits of exponentially large numbers [40]. We leave as major open problems the question of providing better upper bounds for PosSLP and the question of providing any sort of hardness theorem, reducing a supposedly intractable problem to PosSLP.

Theorem 1.3, together with Proposition 1.1, implies that $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}}) \subseteq \mathsf{CH}$. It is reasonable to conjecture that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}}) \subseteq \mathsf{CH}/\mathrm{poly}$—and indeed that would follow from Conjecture 1—but as yet we are not able to improve the upper bound of $\mathrm{BP}(\mathrm{P}_{\mathbb{R}}) \subseteq \mathsf{PSPACE}/\mathrm{poly}$ that was presented by Cucker and Grigoriev [27].

We believe that it would be very interesting to verify Conjecture 1, as this would give a characterization of $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ in terms of classical complexity classes. But in fact, it would be equally interesting to refute it under some plausible complexity-theoretic assumption, as this would give evidence that the power of using transcendental constants in the sequential Blum–Shub–Smale model goes beyond the power of nonuniformity in classical computation.

1.4. Applications. The *sum-of-square-roots problem* is a well-known problem with many applications to computational geometry and elsewhere. The input to the problem is a list of integers $(d_1, \ldots, d_n)$ and an integer k, and the problem is to decide whether $\sum_i \sqrt{d_i} \geq k$. The complexity of this problem is posed as an open question by Garey, Graham, and Johnson [36] in connection with the Euclidean traveling salesman problem, which is not known to be in NP but which is easily seen to be solvable in NP relative to the sum-of-square-roots problem. See also O'Rourke [53, 52] and Etessami and Yannakakis [34] for additional information. Although it has been conjectured [51] that the problem lies in P, it seems that no classical complexity class smaller than PSPACE has been known to contain this problem. On the other hand, by observing that one can construct a polynomial-sized straight-line program with division that approximates the square root of any given integer with exponentially high precision, using Newton iteration, Tiwari [62] showed that this problem can be decided in polynomial time on an "algebraic random-access machine." In fact, it is easy to see that the set of decision problems decided by such machines in polynomial time is exactly $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$. Thus by Proposition 1.1 we see that the sum-of-square-roots problem reduces to PosSLP. Theorem 1.3 thus yields the following corollary.

COROLLARY 1.4. *The sum-of-square-roots problem and the Euclidean traveling salesman problem are in* CH.

2. Preliminaries. Our definitions of arithmetic circuits and straight-line programs are standard. An *arithmetic circuit* is a directed acyclic graph with input nodes labeled with the constants $0, 1$ or with indeterminates $X_1, \ldots, X_k$ for some k. Internal nodes are labeled with one of the operations $+, -, *, \div$. A *straight-line program* is a sequence of instructions corresponding to a sequential evaluation of an arithmetic circuit. If it contains no $\div$ operation, it is said to be *division-free*. Unless otherwise stated, all the straight-line programs considered will be division-free. Thus straight-line programs can be seen as very compact representations of a polynomial over the integers. In many cases, we will be interested in division-free straight-line programs using no indeterminates, which thus represent an integer.

By the n-bit binary representation of an integer N such that $|N| < 2^n$ we understand a bit string of length $n+1$ consisting of a *sign bit* followed by n bits encoding $|N|$ (padded with leading zeroes, if needed).

We consider the following problems:

EquSLP. Given a straight-line program representing an integer N, decide whether $N = 0$.

ACIT. Given a straight-line program representing a polynomial $f \in \mathbb{Z}[X_1, \ldots, X_k]$, decide whether $f = 0$.

DegSLP. Given a straight-line program representing a polynomial $f \in \mathbb{Z}[X_1, \ldots, X_k]$ and given a natural number d in binary, decide whether $\deg f \leq d$.

PosSLP. Given a straight-line program representing $N \in \mathbb{Z}$, decide whether $N > 0$.

BitSLP. Given a straight-line program representing N, and given $n, i \in \mathbb{N}$ in binary, decide whether the ith bit of the n-bit binary representation of N is 1.

It is not clear that any of these problems is in P, since straight-line program representations of integers can be exponentially smaller than ordinary binary representation.

There is an immediate relationship between the Blum–Shub–Smale model over the complex numbers $\mathbb{C}$ and the problem EquSLP. Let $\mathrm{P}_{\mathbb{C}}^0$ denote the class of decision problems over $\mathbb{C}$ decided by polynomial-time Blum–Shub–Smale machines using only the constants $0, 1$. Similarly as for Proposition 1.1 one can show that $\mathsf{P}^{\mathrm{EquSLP}} = \mathrm{BP}(\mathrm{P}_{\mathbb{C}}^0)$. On the other hand, it is known that constants can be eliminated in this setting [11, 45], and hence $\mathrm{BP}(\mathrm{P}_{\mathbb{C}}) = \mathrm{BP}(\mathrm{P}_{\mathbb{C}}^0)$. We therefore have the following result.

PROPOSITION 2.1. $\mathsf{P}^{\mathrm{EquSLP}} = \mathrm{BP}(\mathrm{P}_{\mathbb{C}})$.

Clearly, EquSLP is a special case of ACIT. Schönhage [57] showed that EquSLP is in coRP, using computation modulo a randomly chosen prime. Ibarra and Moran [41], building on DeMillo and Lipton [29], Schwartz [58], and Zippel [69], extended this to show that ACIT lies in coRP. In the spirit of Adleman's observation [1], Heintz and Schnorr [38] established the existence of nonuniform polynomial-time algorithms for an algebraic variant of the ACIT problem (allowing any field elements as constants). The problem ACIT has recently attracted much attention due to the work of Kabanets and Impagliazzo [42] who showed that a deterministic algorithm for ACIT would yield circuit lower bounds. (See [47] for some progress on finding deterministic algorithms for certain versions of the problem.) As far as we know, although the proof technique that we use in Proposition 2.2 is well known and has been applied various times over the years [3, 60], it has not been pointed out before that ACIT is actually polynomial-time equivalent to EquSLP. In other words, disallowing indeterminates in the straight-line program given as input does not make ACIT easier. Or more optimistically: It is enough to find a deterministic algorithm for this special case in order to have circuit lower bounds.

PROPOSITION 2.2. ACIT *is polynomial-time equivalent to* EquSLP.

Proof. We are given a straight-line program of size n with m indeterminates $X_1, \ldots, X_m$, computing the polynomial $p(X_1, \ldots, X_m)$. Define $B_{n,i} = 2^{2^{in^2}}$. Straight-line programs computing these numbers using iterated squaring can easily be constructed in polynomial time, so given a straight-line-program for p, we can easily construct a straight-line program for $p(B_{n,1}, \ldots, B_{n,m})$. We shall show that for $n \geq 3$, p is identically zero iff $p(B_{n,1}, \ldots, B_{n,m})$ evaluates to zero.

To see this, first note that the "only if" part is trivial, so we only have to show the "if" part. Thus, assume that $p(X_1, \ldots, X_m)$ is not the zero-polynomial. Let $q(X_1, \ldots, X_m)$ be the largest monomial occurring in p with respect to inverse lexicographic order,[1] and let k be the number of monomials. We can write $p = \alpha q + \sum_{i=1}^{k-1} \alpha_i q_i$, where $(q_i)_{i=1,\ldots,k-1}$ are the remaining monomials. An easy induction in the size of the straight-line program shows that $|\alpha_i| \leq 2^{2^{2n}}$, $k \leq 2^{2^n}$ and that the degree of any variable in any q_i is at most 2^n.

Now, our claim is that the absolute value $|\alpha q(B_{n,1}, \ldots, B_{n,m})|$ is strictly bigger than $|\sum_{i=1}^{k-1} \alpha_i q_i(B_{n,1}, \ldots, B_{n,m})|$, and thus we cannot have that $p(B_{n,1}, \ldots, B_{n,m}) = 0$.

Indeed, since the monomial q was the biggest in the inverse lexicographic ordering, we have that for any other monomial q_i there is an index j so that

$$\frac{q(B_{n,1}, \ldots, B_{n,m})}{q_i(B_{n,1}, \ldots, B_{n,m})} \geq \frac{2^{2^{jn^2}}}{\prod_{l=1}^{j-1} 2^{2^{ln^2} \cdot 2^n}} > 2^{2^{n^2-1}},$$

so we can bound

$$\left| \sum_{i=1}^{k-1} \alpha_i q_i(B_{n,1}, \ldots, B_{n,m}) \right| \leq 2^{2^n} 2^{2^{2n}} \left| \max_{i=1}^{k-1} q_i(B_{n,1}, \ldots, B_{n,m}) \right|$$

$$\leq 2^{2^n} 2^{2^{2n}} 2^{-2^{n^2-1}} |q(B_{n,1}, \ldots, B_{n,m})| < q(B_{n,1}, \ldots, B_{n,m}) \leq |\alpha q(B_{n,1}, \ldots, B_{n,m})|,$$

which proves the claim. □

We believe that Proposition 2.2 could be a useful tool for devising deterministic algorithms for ACIT. (See section 5 for one modest application in this direction.) Of course, it must also be acknowledged that multivariate polynomials exhibit a great deal of structure that is not so apparent in computation over the integers (as embodied by EquSLP), and algorithmic attacks on ACIT should also attempt to exploit this structure.

The problem DegSLP is not known to lie in BPP, even for the special case of univariate polynomials. Here, we show that it reduces to PosSLP.

PROPOSITION 2.3. DegSLP *polynomial-time many-one reduces to* PosSLP.

Proof. We first show the reduction for the case of univariate polynomials (i.e., straight-line programs with a single indeterminate), and afterwards we reduce the multivariate case to the univariate case.

Let $f \in \mathbb{Z}[X]$ be given by a straight-line program of length n. To avoid having to deal with the zero polynomial of degree $-\infty$ and to ensure that the image of the polynomial is a subset of the nonnegative integers, we first change the straight-line program computing f into a straight-line program computing $f_1(X) = (Xf(X) + 1)^2$

[1] $X_1^{\alpha_1} \cdots X_m^{\alpha_m}$ is greater than $X_1^{\beta_1} \cdots X_m^{\beta_m}$ in this order iff the right-most nonzero component of $\alpha - \beta$ is positive; cf. Cox, Little, and O'Shea [26, p. 59].

by adding a few extra lines. We can check whether the degree of f is at most d by checking whether the degree of f_1 is at most $D = 2(d+1)$ (except for $d = -\infty$, in which case we check whether the degree of f_1 is at most $D = 0$).

Let B_n be the integer $2^{2^{n^2}}$. As in the proof of Proposition 2.2, we can easily construct a straight-line program computing B_n and from this a straight-line program computing $f_1(B_n)$.

Now, suppose that $\deg f_1 \le D$. Using the same bounds on sizes of the coefficients as in the proof of Proposition 2.2 and assuming without loss of generality that $n \ge 3$, we then have

$$f_1(B_n) \le \sum_{i=0}^{D} 2^{2^{2n}} B_n^i < (2^n+1)2^{2^{2n}} B_n^D \le (2^{2^n}+1)2^{2^{2n}-2^{n^2}} B_n^{D+1} < \frac{B_n^{D+1}}{2}.$$

On the other hand, suppose that $\deg f_1 \ge D+1$. Then we have

$$f_1(B_n) \ge (B_n)^{D+1} - \sum_{i=0}^{D} 2^{2^{2n}} B_n^i \ge B_n^{D+1} - 2^{2^n} 2^{2^{2n}} 2^{-2^{n^2}} B_n^{D+1} > \frac{B_n^{D+1}}{2}.$$

Thus, to check whether $\deg f_1 \le D$, we just need to construct a straight-line-program for $2f_1(B_n) - B_n^{D+1}$ and check whether it computes a positive integer. This completes the reduction for the univariate case.

We next reduce the multivariate case to the univariate case. Thus, let f in $\mathbb{Z}[X_1, \ldots, X_m]$ be given by a straight-line program of length n. We define f^* in $\mathbb{Z}[X_1, \ldots, X_m, Y]$ by $f^*(X_1, \ldots, X_m, Y) = f(X_1 Y, \ldots, X_m Y)$. We claim that if we let $B_{n,i} = 2^{2^{in^2}}$ as in the proof of Proposition 2.2, then, for $n \ge 3$, the degree of the univariate polynomial $f^*(B_{n,1}, \ldots, B_{n,m}, Y)$ is equal to the total degree of f. Indeed, we can write f^* as a polynomial in Y with coefficients in $\mathbb{Z}[X_1, \ldots, X_m]$:

$$f^*(X_1, \ldots, X_m, Y) = \sum_{j=0}^{d^*} g_j(X_1, \ldots, X_m) Y^j,$$

where d^* is the degree of variable Y in the polynomial f^*. Note that this is also the total degree of the polynomial f. Now, the same argument as used in the proof of Proportion 2.2 shows that since g_{d^*} is not the zero-polynomial, $g_{d^*}(B_{n,1}, B_{n,2}, \ldots, B_{n,m})$ is different from 0. □

As PosSLP easily reduces to BitSLP, we obtain the chain of reductions

$$\text{EquSLP} \equiv \text{ACIT} \le_{\text{m}}^{\text{p}} \text{DegSLP} \le_{\text{m}}^{\text{p}} \text{PosSLP} \le_{\text{m}}^{\text{p}} \text{BitSLP}.$$

In section 4 we will show that all the above problems in fact lie in the counting hierarchy CH.

The complexity of BitSLP contrasts sharply with that of EquSLP.

PROPOSITION 2.4. BitSLP *is hard for* $\#\mathsf{P}$.

Proof. A similar result is stated without proof in [31]. The proof that we present is quite similar to that of Bürgisser [22, Proposition 5.3], which in turn is based on ideas of Valiant [65]. We show that computing the permanent of matrices with entries from {0,1} is reducible to BitSLP.

Given a matrix X with entries $x_{i,j} \in \{0, 1\}$, consider the univariate polynomial

$$f_n = \sum_i f_{n,i} Y^i = \prod_{i=1}^{n} \left(\sum_{j=1}^{n} x_{i,j} Y^{2^{j-1}} \right),$$

which can be represented by a straight-line program of size $O(n^2)$. Then $f_{n,2^n-1}$ equals the permanent of X. Let N be the number that is represented by the straight-line program that results from replacing the indeterminate Y with 2^{n^3}. It is easy to see that the binary representation of $f_{n,2^n-1}$ appears as a sequence of consecutive bits in the binary representation of N. □

Triggered by an earlier version of this paper [5], Koiran and Perifel [43] studied the variant of the problem DegSLP for computations of polynomials over the finite field $\mathbb{F}_p$. They proved a considerably better upper bound on this problem than what is currently known for DegSLP. Also, Koiran and Perifel [43] investigated the problem *zero monomial coefficient* of deciding for a polynomial f given by a straight-line program and a given monomial x^α whether the coefficient of this monomial in f equals zero. For computations of polynomials over the finite field $\mathbb{F}_p$, they managed to show that zero monomial coefficient is $\mathrm{Mod}_p P$ complete. However, for characteristic zero, no improvements upon the results in this paper were made in that respect.

3. Transcendental constants. We present here some first results toward establishing our Conjecture 1.

Let S denote a fixed finite subset of $\mathbb{R}$. By an *SLP over* S we shall understand a division-free straight-line program using constants from $S \cup \{0, 1\}$. Recall the following problem:

PosSLP(S). Given an SLP over S, decide whether the real number represented by it is positive.

Remark 1. We could have defined a variant of PosSLP(S) by allowing divisions in the straight-line programs. However, this variant is easily seen to be polynomial-time equivalent to PosSLP(S). Indeed, by computing separately with numerators and denominators we can transform an SLP representing α into two division-free SLPs representing numbers A, B such that $\alpha = A/B$. Hereby, the length of the SLPs increases at most by a factor of four. Now α is positive iff AB is positive.

A result by Chapuis and Koiran [24] implies that algebraic constants can be eliminated. It can be stated as follows.

PROPOSITION 3.1. *Let $S \subseteq \mathbb{R}$ be finite and $\alpha \in \mathbb{R}$ be algebraic over the field $\mathbb{Q}(S)$. Then $\mathsf{P}^{\mathrm{PosSLP}(S \cup \{\alpha\})} = \mathsf{P}^{\mathrm{PosSLP}(S)}$.*

Our first goal is to prove that almost all transcendental constants can be eliminated.

THEOREM 3.2. *For all $(\alpha_1, \alpha_2, \ldots, \alpha_k) \in \mathbb{R}^k$ except in a subset of Lebesgue measure zero we have $\mathsf{P}^{\mathrm{PosSLP}(\{\alpha_1,\ldots,\alpha_k\})}/\mathrm{poly} = \mathsf{P}^{\mathrm{PosSLP}}/\mathrm{poly}$.*

The proof will require some lemmas. The idea is to eliminate one by one the elements of such sets S, replacing each element with appropriate advice of polynomial size.

We denote by $R_n^S \subset \mathbb{R}$ the set of all real numbers that occur as a root of some nonzero univariate polynomial that is computed by an SLP of size n that uses constants in S. Note that $\mathbb{R} \setminus R_n^S$ consists of a collection of open intervals. Clearly, any univariate polynomial computed from S by an SLP of size n has constant sign on each of these intervals. For $\alpha \in \mathbb{R} \setminus R_n^S$, we denote by $I_n^S(\alpha)$ the unique interval containing α.

Remark 2. A real number α is transcendental over $\mathbb{Q}(S)$ iff $\alpha \notin R_n^S$ for all n (or equivalently, for infinitely many n).

DEFINITION 3.3. *We call a real number α* approximable with respect to *S if either α is algebraic over $\mathbb{Q}(S)$ or else α is transcendental over $\mathbb{Q}(S)$ and satisfies the following condition: there exists a polynomial p such that for all sufficiently large*

$n \in \mathbb{N}$ the interval $I_n^S(\alpha)$ contains an element x_n that can be represented by an SLP over S of size $p(n)$, possibly using divisions. (Note that this interval is well defined as $\alpha \notin R_n^S$; cf. Remark 2.) We say that α is approximable iff it is approximable with respect to the empty set.

LEMMA 3.4. *If $\alpha \in \mathbb{R}$ is approximable with respect to S, then* $\mathsf{P}^{\mathrm{PosSLP}(S\cup\{\alpha\})}/\mathrm{poly} = \mathsf{P}^{\mathrm{PosSLP}(S)}/\mathrm{poly}$.

Proof. Suppose that $\alpha \in \mathbb{R}$ is approximable with respect to S. By Proposition 3.1 we may assume that α is transcendental over $\mathbb{Q}(S)$. Then, for all sufficiently large n, there exist $x_n \in I_n^S(\alpha)$ computed by an SLP Γ_n over S (using divisions) of size polynomial in n.

It is sufficient to show that $\mathrm{PosSLP}(S \cup \{\alpha\})$ is contained in $\mathsf{P}^{\mathrm{PosSLP}(S)}/\mathrm{poly}$. Let C be an SLP (of size n) over $S \cup \{\alpha\}$ computing $v \in \mathbb{R}$. We want to decide whether v is positive. If we replace the constant α by the variable X, then this SLP computes a polynomial $f(X)$ and we have $v = f(\alpha)$. Since the sign of f is constant on the interval $I_{p(n)}^S(\alpha)$, v has the same sign as $f(x_{p(n)})$.

We interpret the SLP Γ_n over S as an advice of polynomial size. By concatenating Γ_n with the SLP for f, we obtain an SLP over S that computes $f(x_{p(n)})$. We eliminate the divisions in the concatenated SLP according to Remark 1. Then the sign of this number is obtained by one oracle call to $\mathrm{PosSLP}(S)$. ∎

LEMMA 3.5. *We have the following:*

1. $|R_n^S| \le (6(n + |S|))^n$.
2. *The minimal distance between two different elements of $R_n^\emptyset$ is at least $2^{-2^{N_n}}$ with $N_n = O(n \log n)$.*

Proof. Let F_n be the product of all nonzero univariate polynomials f that can be computed from the variable X by an SLP over S of size n. Note that such f have degree at most 2^n. Then R_n^S is the set of roots of F_n. There are at most $\prod_{i=1}^n 3(|S|+i-1)^2 \le (3(|S|+n)^2)^n$ SLPs over S. Therefore, $\deg F_n \le (6(|S|+n)^2)^n$, which shows the first assertion.

Before showing the second assertion we introduce a notation: let $\|g\|_1$ denote the sum of the absolute values of the coefficients of a univariate polynomial g. It is easy to see that $\|g \cdot h\|_1 \le \|g\|_1 \cdot \|h\|_1$.

Suppose now $S = \emptyset$. If $f(X)$ is computed by an SLP of size n over $\emptyset$ from the variable X, then one can show that $\log \|f\|_1 \le (n+1)2^n$; see, e.g., [21, Lemma 4.16]. By the submultiplicativity of $\| \ \|_1$ we conclude

$$\log \|F_n\|_1 \le (3n^2)^n \ (n+1)2^n \le 2^{O(n \log n)}.$$

Rump [55] showed that the distance between any two distinct real roots in a univariate polynomial P with integer coefficients and degree d is at least $2\sqrt{2}(d^{\frac{d}{2}+1}(\|P\|_1 + 1)^d)^{-1}$. The second assertion follows by applying this bound to the polynomial F_n. ∎

LEMMA 3.6. *For any finite $S \subset \mathbb{R}$, the set of real numbers that are not approximable with respect to S has Lebesgue measure zero.*

Proof. Let $\alpha \in \mathbb{R}$ and x_n be the binary approximation of α with a precision of n^2 digits, i.e., $|\alpha - x_n| < 2^{-n^2}$. Clearly, there is an SLP over $\{1/2\}$ of size $O(n^2)$ representing x_n. Furthermore, suppose that α has distance at least 2^{-n^2} from R_n^S for all sufficiently large n, say for $n \ge m$. Then x_n is contained in the interval $I_n^S(\alpha)$ for $n \ge m$. Hence, by definition, α is approximable with respect to S.

These reasonings show that for all $m \in \mathbb{N}$

$$B := \{\alpha \in \mathbb{R} \mid \alpha \text{ is not approximable w.r.t. } S\} \subseteq \bigcup_{n \geq m} U_n,$$

where $U_n := \{x \in \mathbb{R} \mid \exists \rho \in R_n^S \quad |x - \rho| < 2^{-n^2}\}$ denotes the 2^{-n^2}-neighborhood of R_n^S. Denoting by $\lambda(A)$ the Lebesgue measure of a set $A \subseteq \mathbb{R}$, we get from Lemma 3.5

$$\lambda(U_n) \leq 2\,|R_n^S|\; 2^{-n^2} \leq 2\,(6(n + |S|))^n 2^{-n^2} \leq 2^{-\frac{1}{2}n^2}$$

for sufficiently large n. Therefore, we conclude that for all sufficiently large m

$$\lambda(B) \leq \sum_{n=m}^{\infty} 2^{-\frac{1}{2}n^2}.$$

Since the series $\sum_n 2^{-\frac{1}{2}n^2}$ is convergent and m was arbitrary, we conclude that $\lambda(B) = 0$. □

Proof of Theorem 3.2. We consider for $\alpha := (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ and $0 \leq i \leq k$ the complexity classes $\mathcal{C}_i(\alpha) := \mathsf{P}^{\mathrm{PosSLP}(\{\alpha_1,\ldots,\alpha_i\})}/\mathrm{poly}$. Clearly, $\mathcal{C}_k(\alpha) \neq \mathcal{C}_0(\alpha)$ implies that $\mathcal{C}_s(\alpha) \neq \mathcal{C}_{s-1}(\alpha)$ for some index s. By applying Lemma 3.4 to the set of constants $S = \{\alpha_1, \ldots, \alpha_{s-1}\}$, we obtain that $\{\alpha \in \mathbb{R}^k \mid \mathcal{C}_k(\alpha) \neq \mathcal{C}_0(\alpha)\}$ is a subset of

$$\subseteq \bigcup_{s=1}^{k} \left\{\alpha \in \mathbb{R}^k \mid \alpha_s \text{ is not approximable w.r.t. } \{\alpha_1, \ldots, \alpha_{s-1}\}\right\}.$$

Lemma 3.6 says that, for fixed $\alpha_1, \ldots, \alpha_{s-1}$, the set

$$\{\alpha_s \in \mathbb{R} \mid \alpha_s \text{ is not approximable w.r.t. } \{\alpha_1, \ldots, \alpha_{s-1}\}\}$$

has Lebesgue measure zero. It follows from Fubini's theorem that the right-hand subset of $\mathbb{R}^k$ has measure zero as well, which shows the assertion. □

We can actually prove for many specific real numbers that they are approximable. Indeed, quite surprisingly, for any elementary function $f(X)$ there exists a sequence $(R_n(X))$ of rational functions such that $|R_n(x) - f(x)| < 2^{-n}$ for all $x \in [0, 1]$, and such that $R_n(X)$ can be computed by a straight-line program of polylogarithmic size (using divisions) from X. The elementary functions include the algebraic functions, the natural logarithm, and the exponential function. For algebraic functions, such approximating rational functions can be constructed with Newton's method; see Kung and Traub [46]. For the natural logarithm, the construction of such approximations relies on the AGM (arithmetic-geometric mean) iteration going back to Gauss, Lagrange, and Legendre, which, in particular, gives very good approximations of π. The latter algorithms were discovered by Brent [18] and Salamin [56]. The book by Borwein and Borwein [15] provides a complete and in-depth exposition of this subject.

More precisely, we shall understand by an *elementary function* a function built up from rational constants by finitely many arithmetic operations, applications of exp, ln, and the operation of taking a solution of a polynomial equation. (For a formal definition see [19].)

THEOREM 3.7. *Let α be the value of an elementary function at a rational number. Then the following hold:*

1. *α is approximable. In particular, $e = \exp(1)$ and π are approximable.*
2. *We have $\mathsf{P}^{\mathrm{PosSLP}(\{\alpha\})} \subseteq \mathsf{P}^{\mathrm{PosSLP}}/1$, where $/1$ means one bit of advice.*

Proof. 1. By Lemma 3.5 we know that $\epsilon_n = 2^{-2^{N_n}}$ with $N_n = O(n \log n)$ is a lower bound on the minimum distance between two different elements of $R_n^{\emptyset}$. Note that there is an SLP over $\{1/2\}$ of polynomial size computing ϵ_n (repeated squaring).

Let α be as in the statement of the theorem. Without loss of generality we may assume that α is transcendental. According to Borwein and Borwein [14, Table 1], for each n there is an SLP of size $n^{O(1)}$ (using divisions) computing an approximation a_n of α that satisfies $|a_n - \alpha| < \frac{1}{2}\epsilon_n$. By checking the proofs (cf. Borwein and Borwein [15]) one sees that these SLPs are uniform; i.e., they can be constructed in polynomial time in n.

We claim that there exist $b_n \in \{0, 1\}$ such that $x_n = a_n + b_n \frac{1}{2}\epsilon_n$ lies in the interval $I_n^{\emptyset}(\alpha)$ and thus satisfies the requirement in Definition 3.3. Hence α is approximable.

Indeed, let ℓ_n and r_n denote the left and right endpoints of the interval $I_n^{\emptyset}(\alpha)$, and denote by $m_n := \frac{1}{2}(\ell_n + r_n)$ its midpoint. Consider first the case where $\alpha < m_n$. If $\alpha \le a_n$, then $a_n < \alpha + \frac{1}{2}\epsilon_n < m_n + \frac{1}{2}\epsilon_n \le r_n$, and hence $x_n := a_n \in I_n^{\emptyset}(\alpha)$. Else if $a_n < \alpha$, then $\alpha < a_n + \frac{1}{2}\epsilon_n < \alpha + \frac{1}{2}\epsilon_n \le r_n$, and hence $x_n := a_n + \frac{1}{2}\epsilon_n \in I_n^{\emptyset}(\alpha)$ does the job. In the case where $\alpha \ge m_n$ one argues similarly.

2. We follow the proof of Lemma 3.4. However, since the SLPs computing the approximation a_n are polynomial-time uniform, only one bit of advice (corresponding to b_n) is in fact needed to emulate the computation with α. ∎

We have not been able to find a specific number that is *provably* nonapproximable. It is quite possible that there are no nonapproximable numbers at all.

4. PosSLP lies in CH. The counting hierarchy CH was defined by Wagner [66] and was studied further by Torán [64]; see also [8, 6]. A problem lies in CH if it lies in one of the classes in the sequence $\mathsf{PP}, \mathsf{PP}^{\mathsf{PP}}$, etc.

THEOREM 4.1. BitSLP *is in* CH.

Proof. It was shown by Hesse, Allender, and Mix Barrington [39] that there are Dlogtime-uniform threshold circuits of polynomial size and constant depth that compute the following function:

Input: A number X in Chinese remainder representation. That is, a sequence of values indexed (p, j) giving the jth bit of X mod p, for each prime $p < n^2$, where $0 \le X \le 2^n$ (thus we view n as an appropriate "size" measure of the input).

Output: The binary representation of the unique natural number $X < \prod_{p \text{ prime}, p<n^2} p$ whose value modulo each small prime is encoded in the input.

Let this circuit family be denoted $\{D_n\}$.

Now, as in the proof of [6, Lemma 5], we consider the following exponentially big circuit family $\{E_n\}$, which computes BitSLP.

Given as input an encoding of a straight-line program representing integer W, we first build a new program computing the positive integer $X = W + 2^{2^n}$. Note that the bits of the binary representation of W (including the sign bit) can easily be obtained from the bits of X.

Level 1 of the circuit E_n consists of gates labeled (p, j) for each prime p such that $p < 2^{2n}$ and for each $j : 1 \le j \le \lceil \log p \rceil$. The output of this gate records the jth bit of X mod p. (Observe that there are exponentially many gates on level 1, and also note that the output of each gate (p, j) can be computed in time polynomial in the size of the binary encoding of p and the size of the given straight-line program representing X. Note also that the gates on Level 1 correspond to the gates on the input level of the circuit $D_{2^{2n}}$.)

The higher levels of the circuit are simply the gates of $D_{2^{2n}}$.

Now, similar to the proof of [6, Lemma 5], we claim that for each constant d the following language is in the counting hierarchy: $L_d = \{(F, P, b) : F$ is the name of a gate on level d of E_n and F evaluates to b when given straight-line program P as input$\}$.

We have already observed that this is true when $d = 1$. For the inductive step, assume that $L_d \in \mathsf{CH}$. Here is an algorithm to solve L_{d+1} using oracle access to L_d. On input (F, P, b), we need to determine whether the gate F is a gate of E_n, and if so, we need to determine whether it evaluates to b on input P. F is a gate of E_n iff it is connected to some gate G such that, for some b', $(G, P, b') \in L_d$. This can be determined in $\mathsf{NP}^{L_d} \subseteq \mathsf{PP}^{L_d}$, since D_n is Dlogtime-uniform. That is, we can guess a gate G, check that G is connected to F (this takes only linear time because of the uniformity condition), and then use our oracle for L_d. If F is a gate of E_n, we need to determine whether the majority of the gates that feed into it evaluate to 1. (Note that all of the gates in D_n are MAJORITY gates.) That is, we need to determine whether it is the case that for most bit strings G such that G is the name of a gate that is connected to F, $(G, P, 1)$ is in L_d. This is clearly computable in PP^{L_d}.

Thus in order to compute BitSLP, given program P and index i, compute the name F of the output bit of E_n that produces the ith bit of N (which is easy because of the uniformity of the circuits $D_{2^{2^n}}$) and determine if $(F, P, 1) \in L_d$, where d is determined by the depth of the constant-depth family of circuits presented in [39]. □

Theorem 4.1 shows that $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$ lies in CH. A similar argument can be applied to an analogous restriction of "digital" $\mathrm{NP}_{\mathbb{R}}$ (i.e., where nondeterministic machines over the reals can guess "bits" but cannot guess arbitrary real numbers). Bürgisser and Cucker [20] present some problems in PSPACE that are related to *counting* problems over $\mathbb{R}$. It would be interesting to know whether these problems lie in CH.

Although Theorem 4.1 shows that BitSLP and PosSLP both lie in CH, some additional effort is required in order to determine the level of CH where these problems reside. We present a more detailed analysis for PosSLP, since it is our main concern in this paper. (A similar analysis can be carried out for BitSLP, showing that it lies in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}}$ [7].)

The following result implies Theorem 1.3, since Toda's theorem [63] shows that $\mathsf{PP}^{\mathsf{PH}^A} \subseteq \mathsf{P}^{\mathsf{PP}^A}$ for every oracle A.

THEOREM 4.2. $\mathrm{PosSLP} \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$.

Proof. We will use the Chinese remaindering algorithm of [39] to obtain our upper bound on PosSLP. (Related algorithms, which do not lead directly to the bound reported here, have been used on several occasions [2, 28, 33, 48, 49].) Let us introduce some notation relating to Chinese remaindering.

For $n \in \mathbb{N}$ let M_n be the product of all odd primes p less than 2^{n^2}. By the prime number theorem, $2^{2^n} < M_n < 2^{2^{n^2+1}}$ for n sufficiently large. For such primes p let $h_{p,n}$ denote the inverse of M_n/p mod p.

Any integer $0 \leq X < M_n$ can be represented uniquely as a list (x_p), where p runs over the odd primes $p < 2^{n^2}$ and $x_p = X \bmod p$. Moreover, X is congruent to $\sum_p x_p h_{p,n} M_n/p$ modulo M_n. Hence X/M_n is the fractional part of $\sum_p x_p h_{p,n}/p$.

Define the family of approximation functions $app_n(X)$ to be $\sum_p B_p$, where $B_p = x_p h_{p,n} \sigma_{p,n}$ and $\sigma_{p,n}$ is the result of truncating the binary expansion of $1/p$ after 2^{n^4} bits. Note that for n sufficiently large and $X < M_n$, $app_n(X)$ is within $2^{-2^{n^3}}$ of X/M_n.

Let the input to PosSLP be a program P of size n representing the integer W,

and set $Y_n = 2^{2^n}$. Since $|W| \le Y_n$, the number $X := W + Y_n$ is nonnegative, and we can easily transform P into a program of size $2n+2$ representing X. Clearly, $W > 0$ iff $X > Y_n$. Note that if $X > Y_n$, then X/M_n and Y_n/M_n differ by at least $1/M_n > 2^{-2^{n^2+1}}$, which implies that it is enough to compare the binary expansions of $app_n(X)$ and $app_n(Y_n)$. (Interestingly, this seems to be somewhat easier than computing the bits of X directly.)

We can determine whether $X > Y_n$ in PH relative to the following oracle A: $A = \{(P, j, b, 1^n) :$ the jth bit of the binary expansion of $app_n(X)$ is b, where X is the number represented by straight-line program P and j is given in binary$\}$.

Lemma 4.3 completes the proof by showing that $A \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$. □

LEMMA 4.3. $A \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$.

Proof. Assume for the moment that we can show that $B \in \mathsf{PH}^{\mathsf{PP}}$, where $B := \{(P, j, b, p, 1^n) :$ the jth bit of the binary expansion of B_p $(= x_p h_{p,n} \sigma_{p,n})$ is b, where $p < 2^{n^2}$ is an odd prime, $x_p = X \bmod p$, X is the number represented by the straight-line program P, and j is given in binary$\}$. In order to recognize the set A, it clearly suffices to compute 2^{n^4} bits of the binary representation of the sum of the numbers B_p. A uniform circuit family for the iterated sum is presented by Maciel and Thérien in [50, Corollary 3.4.2] consisting of MAJORITY gates on the bottom (input) level, with three levels of AND and OR gates above. As in the proof of Theorem 4.1, the construction of Maciel and Thérien immediately yields a $\mathsf{PH}^{\mathsf{PP}^B}$ algorithm for A, by simulating the MAJORITY gates by PP^B computation, simulating the OR gates above the MAJORITY gates by $\mathsf{NP}^{\mathsf{PP}^B}$ computation, etc. The claim follows, since by Toda's theorem [63], $\mathsf{PH}^{\mathsf{PP}^B} \subseteq \mathsf{PH}^{\mathsf{PP}^{\mathsf{PH}^{\mathsf{PP}}}} = \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$. It remains only to show that $B \in \mathsf{PH}^{\mathsf{PP}}$. □

LEMMA 4.4. $B \in \mathsf{PH}^{\mathsf{PP}}$.

Proof. Observe that, given (P, j, b, p), we can determine in polynomial time whether p is prime [4], and we can compute x_p.

In $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$ we can find the least generator g_p of the multiplicative group of the integers mod p. The set $C = \{(q, g_p, i, p) : p \ne q$ are primes and i is the least number for which $g_p^i \equiv q \mod p\}$ is easily seen to lie in PH. We can compute the discrete log base g_p of the number $M_n/p \bmod p$ in $\#\mathsf{P}^C \subseteq \mathsf{P}^{\mathsf{PP}}$ by the algorithm that nondeterministically guesses q and i, verifies that $(q, g_p, i, p) \in C$, and if so generates i accepting paths. Thus we can compute the number $M_n/p \bmod p$ itself in P^{PP} by first computing its discrete log, and then computing g_p to that power, mod p. The inverse $h_{p,n}$ is now easy to compute in P^{PP}, by finding the inverse of $M_n/p \bmod p$.

Our goal is to compute the jth bit of the binary expansion of $x_p h_{p,n} \sigma_{p,n}$. We have already computed x_p and $h_{p,n}$ in P^{PP}, so it is easy to compute $x_p h_{p,n}$. The jth bit of $1/p$ is 1 iff $2^j \bmod p$ is odd, so bits of $\sigma_{p,n}$ are easy to compute in polynomial time. (Note that j is exponentially large.)

Thus our task is to obtain the jth bit of the product of $x_p h_{p,n}$ and $\sigma_{p,n}$, or (equivalently) adding $\sigma_{p,n}$ to itself $x_p h_{p,n}$ times. The problem of adding $\log^{O(1)} n$ many n-bit numbers lies in uniform AC^0 [32]. Simulating these AC^0 circuits leads to the desired $\mathsf{PH}^{\mathsf{PP}}$ algorithm for B. □

5. An observation on derandomizing ACIT. The connections between algebraic complexity and the counting hierarchy in the preceding section were first introduced in an earlier version of this paper [5]. Recently, these connections have led to further developments. Bürgisser shows in [23] that the counting hierarchy provides

a useful tool for showing implications among several hypotheses in algebraic complexity theory that were not previously known to be related. In that same paper, he also improves a theorem of Koiran, relating the arithmetic circuit complexity of the permanent to a frequently studied question about the complexity of expressing $n!$. We have some new observations to present on this topic, and we start by recalling some background and definitions.

We will follow the terminology of Shub and Smale [59], and say that $n!$ is *"easy"* if there is a sequence of SLPs C_n of size $\log^{O(1)} n$, where C_n represents the number $n!$. Following the same convention, we say that $n!$ is *"ultimately easy"* if there is a sequence of SLPs C_n of size $\log^{O(1)} n$, where C_n represents a nonzero multiple of the number $n!$. (It does not matter which multiple is represented.) Shub and Smale conjectured that $n!$ is not ultimately easy, and they showed that this condition implies that $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$. It is also pointed out in [12] that if factoring is sufficiently hard to compute, it implies that $n!$ is not easy. There are a number of papers that touch on the questions of whether or not $n!$ is easy or ultimately easy. The reader is referred to [25, 23, 12] for more references. A goal of this section is to relate these questions to the complexity of ACIT.

Note that if $n!$ is not ultimately easy, it says merely that there are *infinitely many* n for which multiples of $n!$ require large circuits. It may be useful also to consider the hypothesis that this condition holds for *all* large n; that is, for all k there is an m such that for all $n > m$ there is no SLP of size $\log^k n$ representing a nonzero multiple of $n!$ Let us call this condition "$n!$ is ultimately hard."

The following implications are known to hold:

$$n! \text{ is ultimately hard} \Rightarrow n! \text{ is not ultimately easy} \Rightarrow n! \text{ is not easy}$$
$$\Rightarrow \text{the permanent requires arithmetic circuits of superpolynomial size} \Rightarrow \text{AFIT} \in \bigcap_{\epsilon>0} \text{io-[DTIME}(2^{n^\epsilon})],$$

where io-[DTIME$(t(n))$] denotes the class of problems A for which there is a deterministic algorithm running in time $t(n)$ that solves A correctly for all instances of length n, for infinitely many n, and where AFIT denotes arithmetic formula identity testing: a special case of ACIT. The third implication is from [23], the fourth is from [42, Theorem 7.7]. Derandomization results such as those of [42] usually come in two flavors. If one assumes that a particular function (such as the permanent) is hard on infinitely many input lengths, then one obtains only algorithms that work correctly on infinitely many input lengths. One can also obtain an algorithm that works correctly on all input lengths if one starts with a stronger assumption, such as that the permanent requires large circuits on all input lengths.

It has not been known whether any of these hypotheses are sufficiently strong to derandomize ACIT itself, although it is known that if ACIT is in $\bigcap_{\epsilon>0}\mathrm{DTIME}(2^{n^\epsilon})$ (or even in $\bigcap_{\epsilon>0}\mathrm{NTIME}(2^{n^\epsilon})$), then either the permanent requires arithmetic circuits of superpolynomial size or $\mathsf{NEXP} \not\subseteq \mathsf{P}/\mathrm{poly}$ [42]. We observe now that the following implication holds.

PROPOSITION 5.1. *We have the following:*

1. *If $n!$ is ultimately hard, then* ACIT $\in \bigcap_{\epsilon>0}\mathrm{DTIME}(2^{n^\epsilon})$.
2. *If $n!$ is not ultimately easy, then* ACIT $\in \bigcap_{\epsilon>0}\text{io-[DTIME}(2^{n^\epsilon})]$.

Proof. We prove only the second claim. The first is easier and follows by the same method.

First note that, by Proposition 2.2, it is sufficient to prove the implication for EquSLP instead of ACIT. Assume that $n!$ is not ultimately easy. Then for every k

there is an infinite set $I(k)$ of numbers such that for all $m \in I(k)$ no SLP of size at most $\log^k m$ can produce a nonzero multiple of $m!$.

Given $\epsilon > 0$, pick any γ such that $0 < \gamma < \epsilon$. Choose k to be the least integer larger than $1/\gamma$. For any $m \in I(k)$ set $n = \lfloor \log^k m \rfloor$.

Suppose we are given as input an SLP C of size n. Note that the binary encoding of m has length at most $n^{1/k}$—but we do not know what m is. Thus we try all numbers z having binary encoding of length at most $n^{1/k}$ (one of which will be m). We then compute the binary representation of $z!$ with the obvious algorithm, which takes time at most $z^2 \log^{O(1)} z$, which is less than 2^{n^γ} for sufficiently large n. Then we evaluate the SLP C modulo $z!$; we accept iff the result is zero for all of the numbers z. This algorithm works correctly, since by our assumption, the SLP C cannot produce a nonzero multiple of $m!$. The running time is $2^{O(n^\gamma)} 2^{O(n^\gamma)}$, which is less than 2^{n^ϵ} for all large n. □

We remark that this proof makes use of no special properties of the factorial function. As one of the referees has pointed out to us, the same upper bound follows if there is any sequence of numbers $(a(n))$ such that the binary representation of $a(n)$ can be computed from n in time polynomial in m, the length of the binary representation of $a(n)$, such that no multiple of $a(n)$ can be represented by an arithmetic circuit of size $\log^{O(1)} m$. If we drop the requirement that $a(n)$ be computable in time polynomial in m, then a simple counting argument (or Kolmogorov complexity argument) shows that *most* numbers $a(n)$ have this property.

6. Closing remarks. NP-hardness is firmly established as a useful tool for providing evidence of intractibility. We believe that PosSLP can become a useful tool for providing evidence of intractibility for problems that do not appear to be NP-hard and for providing evidence that certain problems do not lie in NP or reduce to NP. Indeed, results of this flavor have already started to appear: Etessami and Yannakakis have recently shown that PosSLP reduces to the problem of finding mixed strategy profiles close to exact Nash equilibria in three-person games [35]; i.e., they show that this fundamental but very challenging numerical problem of computational game theory is PosSLP-hard. We may regard this PosSLP-hardness result as evidence that the problem is not NP-easy. In contrast, the related, weaker, and much less elusive notion of computing a strategy profile approximately satisfying the equilibrium conditions is trivially NP-easy. It would be most interesting to establish similar PosSLP-hardness results for other natural numerical problems in the computational sciences (e.g., computational physics) for which no efficient and "fool-proof" methods, even nondeterministic ones, are known but which also do not appear to be NP-hard.

There are several directions for further research suggested by the results that we have presented. We would very much like to see a resolution of our Conjecture 1, and we think that it is likely that PosSLP lies at a lower level of the counting hierarchy than is proved in Theorem 1.3. Perhaps better upper bounds can be presented at least for the sum-of-square-roots problem. Can better evidence be presented for the intractibility of PosSLP? Can some important problems in $\mathrm{NP}_{\mathbb{R}}$ (such as the existential theory of the reals) be shown to lie in the counting hierarchy?

Acknowledgments. We acknowledge helpful conversations with Kousha Etessami, Sambuddha Roy, Felipe Cucker, Lenore Blum, Richard Lipton, Parikshit Gopalan, Mark Braverman, Madhu Sudan, Klaus Meer, Pascal Koiran, Qi Cheng, James Demmel, Salil Vadhan, Fengming Wang, Valentine Kabanets, Neeraj Kayal, and Kristoffer Arnsfelt Hansen.

REFERENCES

[1] L. Adleman, *Two theorems on random polynomial time*, in Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 1978, pp. 75–83.

[2] M. Agrawal, E. Allender, and S. Datta, *On TC^0, AC^0, and arithmetic circuits*, J. Comput. System Sci., 60 (2000), pp. 395–421.

[3] M. Agrawal and S. Biswas, *Primality and identity testing via chinese remaindering*, J. ACM, 50 (2003), pp. 429–443.

[4] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math., 160 (2004), pp. 781–793.

[5] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen, *On the complexity of numerical analysis*, in Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC '06), IEEE Press, Piscataway, NJ, 2006, pp. 331–339.

[6] E. Allender, M. Koucký, D. Ronneburger, S. Roy, and V. Vinay, *Time-space tradeoffs in the counting hierarchy*, in Proceedings of the 16th Annual IEEE Conference on Computational Complexity (CCC '01), IEEE Press, Piscataway, NJ, 2001, pp. 295–302.

[7] E. Allender and H. Schnorr, *The Complexity of the BitSLP Problem*, manuscript, Department of Computer Science, Rutgers University, 2005.

[8] E. Allender and K. W. Wagner, *Counting hierarchies: Polynomial time and constant depth circuits*, in Current Trends in Theoretical Computer Science, G. Rozenberg and A. Salomaa, eds., World Scientific, River Edge, NJ, 1993, pp. 469–483.

[9] S. Berkowitz, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150.

[10] A. Bertoni, G. Mauri, and N. Sabadini, *Simulations among classes of random access machines and equivalence among numbers succinctly represented*, Ann. Discrete Math., 25 (1985), pp. 65–90.

[11] L. Blum, F. Cucker, M. Shub, and S. Smale, *Algebraic settings for the problem "$P \neq NP$?"*, in The Mathematics of Numerical Analysis, Lectures in Appl. Math. 32, AMS, Providence, RI, 1996, pp. 125–144.

[12] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, Springer, New York, 1998.

[13] L. Blum, *Computing over the reals: Where Turing meets Newton*, Notices Amer. Math. Soc., 51 (2004), pp. 1024–1034.

[14] J. M. Borwein and P. B. Borwein, *On the complexity of familiar functions and numbers*, SIAM Rev., 30 (1988), pp. 589–601.

[15] J. M. Borwein and P. B. Borwein, *Pi and the AGM, A study in analytic number theory and computational complexity*, Canadian Mathematical Society Series of Monographs and Advanced Texts, John Wiley & Sons Inc., New York, 1987.

[16] M. Braverman and S. Cook, *Computing over the reals: Foundations for scientific computing*, Notices Amer. Math. Soc., 55 (2006), pp. 318–329.

[17] M. Braverman, *On the complexity of real functions*, in Proceedings of the 46th Annual IEEE Symposium on Fundations of Computer Science, IEEE Press, Piscataway, NJ, 2005, pp. 155–164.

[18] R. P. Brent, *Fast multiple-precision evaluation of elementary functions*, J. ACM, 23 (1976), pp. 242–251.

[19] M. Bronstein, *Symbolic integration.* I, *Transcendental functions*, Algorithms Comput. Math. 1, Springer-Verlag, Berlin, 1997.

[20] P. Bürgisser and F. Cucker, *Counting complexity classes for numeric computations* II: *Algebraic and semialgebraic sets*, J. Complexity, 22 (2006), pp. 147–191.

[21] P. Bürgisser, *Completeness and Reduction in Algebraic Complexity Theory*, Algorithms Comput. Math. 7, Springer-Verlag, Berlin, 2000.

[22] P. Bürgisser, *The complexity of factors of multivariate polynomials*, Found. Comput. Math., 4 (2004), pp. 369–396.

[23] P. Bürgisser, *On defining integers in the counting hierarchy and proving lower bounds in algebraic complexity*, in Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07), W. Thomas and P. Weil, eds., Lecture Notes in Comput. Sci. 44394, Springer, New York, 2007, pp. 133–144.

[24] O. Chapuis and P. Koiran, *Saturation and stability in the theory of computation over the reals*, Ann. Pure Appl. Logic, 99 (1999), pp. 1–49.

[25] Q. Cheng, *On the ultimate complexity of factorials*, Theoret. Comput. Sci., 326 (2004), pp. 419–429.

[26] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, Springer, New York, 1991.
[27] F. Cucker and D. Grigoriev, *On the power of real Turing machines over binary inputs*, SIAM J. Comput., 26 (1997), pp. 243–254.
[28] G. I. Davida and B. Litow, *Fast parallel arithmetic via modular representation*, SIAM J. Comput., 20 (1991), pp. 756–765.
[29] R. DeMillo and R. Lipton, *A probabilistic remark on algebraic program testing*, Inform. Process. Lett., 7 (1978), pp. 193–195.
[30] J. Demmel and P. Koev, *Accurate and efficient algorithms for floating point computation*, in Applied Mathematics Entering the 21st Century: Invited Talks from the ICIAM 2003 Congress, J. M. Hill and R. Moore, eds., Proc. Appl. Math. 116, SIAM, Philadelphia, 2004, pp. 73–90.
[31] J. Demmel, *The complexity of accurate floating point computation*, in Proceedings of the 2002 International Congress of Mathematicians, 2002, vol. 3, pp. 697–706.
[32] L. Denenberg, Y. Gurevich, and S. Shelah, *Definability by constant-depth polynomial-size circuits*, Inform. and Control, 70 (1986), pp. 216–240.
[33] P. F. Dietz, I. I. Macarie, and J. I. Seiferas, *Bits and relative order from residues, space efficiently*, Inform. Process. Lett., 50 (1994), pp. 123–127.
[34] K. Etessami and M. Yannakakis, *Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations*, in Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'05), V. Diekert and B. Durand, eds., Lecture Notes in Comput. Sci. 3404, Springer, New York, 2005, pp. 340–352.
[35] K. Etessami and M. Yannakakis, *On the complexity of Nash equilibria and other fixed points*, in Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 2007, pp. 113–123.
[36] M. Garey, R. L. Graham, and D. S. Johnson, *Some NP-complete geometric problems*, in Proceedings of the ACM Symposium on Theory Computations, ACM, New York, 1976, pp. 10–22.
[37] J. Hartmanis and J. Simon, *On the power of multiplication in random-access machines*, in Proceedings of the 15th Annual IEEE Symposium on Switching Automata Theory, IEEE Press, Piscataway, NJ, 1974, pp. 13–23.
[38] J. Heintz and C. P. Schnorr, *Testing polynomials which are hard to compute*, in Logic and Algorithmic: An International Symposium Held in Honor of Ernst Specker, Monogr. No. 30 de l'Enseign. Math., University of Geneva, Geneva, Switzerland, 1982, pp. 237–254.
[39] W. Hesse, E. Allender, and D. A. Mix Barrington, *Uniform constant-depth threshold circuits for division and iterated multiplication*, J. Comput. System Sci., 65 (2002), pp. 695–716.
[40] M. Hirvensalo and J. Karhumäki, *Computing partial information out of intractable one—The first digit of 2^n at base 3 as an example*, in Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 2420, Springer, New York, 2002, pp. 319–327.
[41] O. H. Ibarra and S. Moran, *Equivalence of straight-line programs*, J. ACM, 30 (1983), pp. 217–228.
[42] V. Kabanets and R. Impagliazzo, *Derandomizing polynomial identity tests means proving circuit lower bounds*, Comput. Complexity, 13 (2004), pp. 1–46.
[43] P. Koiran and S. Perifel, *The complexity of two problems on arithmetic circuits*, Theoret. Comput. Sci., 389 (2007), pp. 172–181.
[44] P. Koiran, *Computing over the reals with addition and order*, Theoret. Comput. Sci., 133 (1994), pp. 35–47.
[45] P. Koiran, *Elimination of constants from machines over algebraically closed fields*, J. Complexity, 13 (1997), pp. 65–82.
[46] H. T. Kung and J. F. Traub, *All algebraic functions can be computed fast*, J. ACM, 25 (1978), pp. 245–260.
[47] R. Lipton and N. Vishnoi, *Deterministic identity testing for multivariate polynomials*, in Proceedings of the Fourteenth ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 756–760.
[48] B. Litow, *On iterated integer product*, Inform. Process. Lett., 42 (1992), pp. 269–272.
[49] I. I. Macarie, *Space-efficient deterministic simulation of probabilistic automata*, SIAM J. Comput., 27 (1998), pp. 448–465.
[50] A. Maciel and D. Thérien, *Threshold circuits of small majority-depth*, Inform. and Comput., 146 (1998), pp. 55–83.

[51] G. Malajovich, *An Effective Version of Kronecker's Theorem on Simultaneous Diophantine Approximation*, Technical report, Department of Mathematics, City University of Hong Kong, Hong Kong, 1996.
[52] J. O'Rourke, *Advanced problem* 6369, Amer. Math. Monthly, 88 (1981), p. 769.
[53] J. O'Rourke, The Open Problems Project, webpage at http://maven.smith.edu/~orourke/TOPP.
[54] M. Ramana, *An exact duality theory for semidefinite programming and its complexity implications*, Math. Program., 77 (1997), pp. 129–162.
[55] S. M. Rump, *Polynomial minimum root separation*, Math. Comp., 33 (1979), pp. 327–336.
[56] E. Salamin, *Computation of π using arithmetic-geometric mean*, Math. Comp., 30 (1976), pp. 565–570.
[57] A. Schönhage, *On the power of random access machines*, in Automata, Languages and Programming ICALP'79, H.A. Maurer, ed., Lecture Notes in Comput. Sci. 71, Springer, New York, 1979, pp. 520–529.
[58] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.
[59] M. Shub and S. Smale, *On the intractability of Hilbert's Nullstellensatz and an algebraic version of "P=NP"*, Duke Math. J., 81 (1996), pp. 47–54.
[60] V. Strassen, *Polynomials with rational coefficients which are hard to compute*, SIAM J. Comput., 3 (1974), pp. 128–149.
[61] S. P. Tarasov and M. N. Vyalyi, *Semidefinite programming and arithmetic circuit evaluation*, Discrete Appl. Math., 156 (2008), pp. 2070–2078.
[62] P. Tiwari, *A problem that is easier to solve on the unit-cost algebraic RAM*, J. Complexity, 8 (1992), pp. 393–397.
[63] S. Toda, PP *is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), pp. 865–877.
[64] J. Torán, *Complexity classes defined by counting quantifiers*, J. ACM, 38 (1991), pp. 753–774.
[65] L. G. Valiant, *Reducibility by Algebraic Projections*, in Logic and Algorithmic: An International Symposium Held in Honor of Ernst Specker, Monogr. No. 30 de l'Enseign. Math., University of Geneva, Geneva, Switzerland, 1982, pp. 365–380.
[66] K. W. Wagner, *The complexity of combinatorial problems with succinct input representation*, Acta Inform., 23 (1986), pp. 325–356.
[67] K. Weihrauch, *Computable Analysis*, Springer-Verlag, Berlin, 2000.
[68] H. Wozniakowski, *Why does information-based complexity use the real number model?*, Theoret. Comput. Sci., 219 (1999), pp. 451–465.
[69] R. E. B. Zippel, *Simplification of Radicals with Applications to Solving Polynomial Equations*, Master's thesis, Electrical Engineering and Computer Science Department, M.I.T., Cambridge, MA, 1977.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 2007–2020

INTERVAL COMPLETION IS FIXED PARAMETER TRACTABLE*

YNGVE VILLANGER†, PINAR HEGGERNES†, CHRISTOPHE PAUL‡, AND JAN ARNE TELLE†

Abstract. We present an algorithm with runtime $O(k^{2k}n^3m)$ for the following NP-complete problem [M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979, problem GT35]: Given an arbitrary graph G on n vertices and m edges, can we obtain an interval graph by adding at most k new edges to G? This resolves the long-standing open question [H. Kaplan, R. Shamir, and R. E. Tarjan, *SIAM J. Comput.*, 28 (1999), pp. 1906–1922; R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999; M. Serna and D. Thilikos, *Bull. Eur. Assoc. Theory Comput. Sci. EATCS*, 86 (2005), pp. 41–65; G. Gutin, S. Szeider, and A. Yeo, in *Proceedings IWPEC* 2006, Lecture Notes in Comput. Sci. 4169, Springer-Verlag, Berlin, 2006, pp. 60–71], first posed by Kaplan, Shamir, and Tarjan, of whether this problem was fixed parameter tractable. The problem has applications in profile minimization for sparse matrix computations [J. A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981; R. E. Tarjan, in *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, eds., Academic Press, 1976, pp. 3–22], and our results show tractability for the case of a small number k of zero elements in the envelope. Our algorithm performs bounded search among possible ways of adding edges to a graph to obtain an interval graph and combines this with a greedy algorithm when graphs of a certain structure are reached by the search.

Key words. interval graphs, profile minimization, edge completion, FPT algorithm, branching

AMS subject classifications. 05C85, 68Q25, 68R10, 68W40

DOI. 10.1137/070710913

1. Introduction and motivation.

Interval graphs are the intersection graphs of intervals of the real line and have a wide range of applications [13]. Connected with interval graphs is the following problem: Given an arbitrary graph G, what is the minimum number of edges that must be added to G in order to obtain an interval graph? This problem is NP-hard [18, 9]. The problem arises in sparse matrix computations, where one of the standard methods for reordering a matrix to get as few nonzero elements as possible during Gaussian elimination is to permute the rows and columns of the matrix so that nonzero elements are gathered close to the main diagonal [10]. The *profile* of a matrix is the smallest number of entries that can be enveloped within off-diagonal nonzero elements of the matrix. Translated to graphs, the profile of a graph G is exactly the minimum number of edges in an interval supergraph of G [26]. Originally, physical mapping of DNA was another motivation for this problem [12].

In this paper, we present an algorithm with runtime $O(k^{2k}n^3m)$ for the *k-interval completion* problem of deciding whether a graph on n vertices and m edges can be made into an interval graph by adding at most k edges. A parameterized problem

*Received by the editors December 14, 2007; accepted for publication (in revised form) May 22, 2008; published electronically January 30, 2009. This work was supported by the Research Council of Norway and the French ANR project "Graph decomposition and algorithm." An extended abstract of this paper was presented at STOC 2007.

http://www.siam.org/journals/sicomp/38-5/71091.html

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway (yngvev@ii.uib.no, pinar@ii.uib.no, telle@ii.uib.no).

‡LIRMM, Université Montpellier II, 34392 Montpellier Cedex 5, France, and School of Computer Science, McGill University, Montreal, QC, H3A 2A7, Canada (paul@lirmm.fr).

with parameter k and input size x that can be solved by an algorithm having runtime $f(k) \cdot x^{O(1)}$ is called *fixed parameter tractable (FPT)* (see [7] for an introduction to fixed parameter tractability and bounded search tree algorithms). The k-interval completion problem is thus FPT, which settles a long-standing open problem [17, 7, 25, 14]. An early paper (first appearance FOCS '94 [16]) in this line of research by Kaplan, Shamir and Tarjan [17] gives FPT algorithms for k-chordal completion, k-strongly chordal completion, and k-proper interval completion. In all these cases a bounded search tree algorithm identifies a subgraph which is a witness of nonmembership in the desired class of graphs, and branches recursively on all possible ways of adding an inclusion-minimal set of edges that gets rid of the witness. For example, since a graph is chordal if and only if it has no induced cycle on more than 3 vertices, the k-chordal completion algorithm [17, 2] takes as witness a subset C of vertices inducing a d-cycle, $4 \leq d \leq k+3$, and branches on all ways of adding the $d-3$ edges needed to make the subgraph induced by C chordal. The existence of an FPT algorithm for solving k-interval completion was left as an open problem by [17], with the following explanation for why a bounded search tree algorithm seemed unlikely: "An arbitrarily large obstruction X could exist in a graph that is not interval but could be made interval with the addition of any one out of $O(|X|)$ edges." Our FPT algorithm for this problem is nevertheless based heavily on applying the bounded search tree technique, supplemented with a greedy algorithm to circumvent the obstructions mentioned in the quote.

Let us mention some related work. Ravi, Agrawal, and Klein [24] gave an $O(\log^2 n)$-approximation algorithm for minimum interval completion, subsequently improved to an $O(\log n \log\log n)$-approximation by Even, Naor, Rao, and Schieber [8] and finally to an $O(\log n)$-approximation by Rao and Richa [23]. Heggernes, Suchan, Todinca, and Villanger showed that an inclusion-minimal interval completion can be found in polynomial time [15]. Kuo and Wang [20] gave an $O(n^{1.722})$ algorithm for minimum interval completion of a tree, subsequently improved to an $O(n)$ algorithm by Díaz, Gibbons, Paterson, and Torán [5]. Cai [2] proved that k-completion into any hereditary graph class having a finite set of forbidden subgraphs is FPT. Some researchers have been misled into thinking that this settled the complexity of k-interval completion; however, interval graphs do not have a finite set of forbidden subgraphs [21]. Gutin, Szeider, and Yeo [14] gave an FPT algorithm for deciding if a graph G has profile at most $k + |V(G)|$, but the more natural parameterization of the profile problem is to ask if G has profile at most $k + |E(G)|$, which is equivalent to the k-interval completion problem on G. Similar questions, asking if we can add/remove at most k vertices/edges to a graph such that a certain property is satisfied, have been investigated in the literature for various graph properties; see, e.g., [3].

Our algorithm for k-interval completion circumvents the problem of large obstructions (witnesses) by first getting rid of all small witnesses, in particular witnesses for the existence of an asteroidal triple (AT) of vertices. Three nonadjacent vertices a, b, c form an AT if there exists a path from any two of them avoiding the neighborhood of the third. Since a graph is an interval graph if and only if it is both chordal and AT-free [21], to complete into an interval graph we must destroy witnesses for nonchordality and witnesses for existence of an AT. Witnesses for nonchordality (chordless cycles of length > 3) must have size $O(k)$ and do not present a problem. Likewise, if an AT is witnessed by an induced subgraph S of size $O(k)$ it does not present a problem, as shown in section 3 of this paper. In section 4, we show that in every induced subgraph S witnessing the existence of an AT, one of the vertices of the AT is *shallow*, meaning that there is a short path from it to each of the other two vertices

of the AT. We give a branching rule for the case when G has no AT witnessed by a small subgraph, but it has at least $k+1$ shallow vertices. The most difficult case is when we have a chordal noninterval graph G with no AT having a small witness and with at most k shallow vertices. For this case we introduce *thick* AT-witnesses in section 5, consisting of an AT and all vertices on any chordless path between any two vertices of the AT avoiding the neighborhood of the third vertex of the AT. We define minimality for thick AT-witnesses and show that also in every minimal thick AT-witness one of the vertices of the AT is shallow. In section 6 we show how to carefully compute the set C of shallow vertices so that removing C from the graph gives an interval graph. Based on the cardinality of C, we show how to further continue branching in a bounded way. When no bounded branching is possible we show that the instance has enough structure that the best solution is a completion computed in a greedy manner. The presented algorithm consists of 4 branching rules in addition to the greedy completion.

2. Preliminaries. We work with simple and undirected graphs $G = (V, E)$, with vertex set $V(G) = V$ and edge set $E(G) = E$, and $n = |V|$, $m = |E|$. For $X \subset V$, $G[X]$ denotes the subgraph of G induced by the vertices in X. We will use $G - x$ to denote $G[V \setminus \{x\}]$ for $x \in V$, and $G - S$ to denote $G[V \setminus S]$ for $S \subseteq V$.

For neighborhoods, we use $N_G(x) = \{y \mid xy \in E\}$ and $N_G[x] = N_G(x) \cup \{x\}$. For $X \subseteq V$, $N_G[X] = \bigcup_{x \in X} N_G[x]$ and $N_G(X) = N_G[X] \setminus X$. We will omit the subscript when the graph is clear from the context. A vertex set X is a *clique* if $G[X]$ is complete, and a *maximal clique* if no superset of X is a clique. A vertex x is *simplicial* if $N(x)$ is a clique.

We will say that a path $P = v_1, v_2, \ldots, v_p$ is *between* v_1 and v_p, and we call it a *v_1, v_p-path.* The *length* of P is p. We will use $P - v_p$ and $P + v_{p+1}$ to denote the paths $v_1, v_2, \ldots, v_{p-1}$ and $v_1, v_2, \ldots, v_p, v_{p+1}$, respectively. We say that a path P *avoids* a vertex set S if P contains no vertex of S. A *chord* of a cycle (path) is an edge connecting two nonconsecutive vertices of the cycle (path). A *chordless* cycle (path) is an induced subgraph that is isomorphic to a cycle (path). A graph is *chordal* if it contains no chordless cycle of length at least 4.

A graph is an *interval graph* if intervals can be associated with its vertices such that two vertices are adjacent if and only if their corresponding intervals overlap. Three nonadjacent vertices form an *asteroidal triple (AT)* if there is a path between every two of them that does not contain a neighbor of the third. A graph is *AT-free* if it contains no AT. A graph is an interval graph if and only if it is chordal and AT-free [21]. A vertex set $S \subseteq V$ is called *dominating* if every vertex not contained in S is adjacent to some vertex in S. A pair of vertices $\{u, v\}$ is called a *dominating pair* if every u, v-path is dominating. Every interval graph has a dominating pair [4] and thus also a dominating chordless path.

A *clique tree* of a graph G is a tree T whose nodes (also called *bags*) are maximal cliques of G such that for every vertex v in G, the subtree T_v of T that is induced by the bags that contain v is connected. A graph is chordal if and only if it has a clique tree [1]. A *clique path* Q of a graph G is a clique tree that is a path. A graph G is an interval graph if and only if has a clique path [11]. An interval graph has at most n maximal cliques.

Given two vertices u and v in G, a vertex set S is a *u, v-separator* if u and v belong to different connected components of $G - S$. A u, v-separator S is *minimal* if no proper subset of S is a u, v-separator. S is a *minimal separator* of G if there exist two vertices u and v in G with S a minimal u, v-separator. For a chordal graph G, a

set of vertices S is a minimal separator of G if and only if S is the intersection of two neighboring bags in any clique tree of G [1].

An interval supergraph $H = (V, E \cup F)$ of a given graph $G = (V, E)$, with $E \cap F = \emptyset$, is called an *interval completion* of G. H is called a *k-interval completion* of G if $|F| \leq k$. The set F is called the set of *fill edges* of H. On input G and k, the *k-interval completion problem* asks whether there is an interval completion of G with at most k fill edges.

3. Nonchordality and small simple AT-witnesses: Rules 1, 2.

Branching Rule 1. If G is not chordal, find a chordless cycle C of length at least 4. If $|C| > k + 3$, answer no; otherwise:

- Branch on the at most $4^{|C|}$ different ways to add an inclusion minimal set of edges (of cardinality $|C| - 3$) between the vertices of C to make C chordal.

It is shown in [17, 2] that there are at most $4^{|C|}$ minimal sets of edges for making C chordal. If Rule 1 applies, we branch by creating at most $4^{|C|}$ recursive calls, each with new parameter value $k - (|C| - 3)$. The correctness of Rule 1 is well understood [17, 2]. Let us remark that each invocation of the recursive search tree subroutine will apply only one of four branching rules. Thus, if Rule 1 applies we apply it and branch, else if Rule 2 applies we apply it and branch, else if Rule 3 applies we apply it and branch, else apply Rule 4. Rules 2, 3, and 4 will branch on single fill edges, dropping the parameter by one. Also Rule 1 could have branched on single fill edges, simply by taking the set of nonedges of the induced cycle and branching on each nonedge separately. We continue with Rule 2.

Observation 3.1. Given a graph G, let $\{a, b, c\}$ be an AT in G. Let P'_{ab} be the set of vertices on a path between a and b in $G - N[c]$, let P'_{ac} be the set of vertices on a path between a and c in $G - N[b]$, and let P'_{bc} be the set of vertices on a path between b and c in $G - N[a]$. Then any interval completion of G contains at least one fill edge from the set $\{cx \mid x \in P'_{ab}\} \cup \{ax \mid x \in P'_{bc}\} \cup \{bx \mid x \in P'_{ac}\}$.

Proof. Otherwise $\{a, b, c\}$ would still be an independent set of vertices with a path between any two avoiding the neighborhood of the third; in other words it would be an AT. □

We introduce simple AT-witnesses and give a branching rule for small such witnesses.

DEFINITION 3.2. *Let $\{a, b, c\}$ be an AT in a graph G. There are three paths P_{ab}, P_{bc}, P_{bc}, where P_{ab} is the set of vertices on a* shortest *path between a and b in $G - N[c]$, P_{ac} is the set of vertices on a* shortest *path between a and c in $G - N[b]$, and P_{bc} is the set of vertices on a* shortest *path between b and c in $G - N[a]$. The induced subgraph $G_{abc} = G[P_{ab} \cup P_{bc} \cup P_{ac}]$ of G is called a* simple AT-witness *of this AT. We call $\{P_{ab}, P_{bc}, P_{bc}\}$ the* core *of G_{abc}.*

Observe that a simple AT-witness for $\{a, b, c\}$ and the mentioned shortest paths of its core exist if and only if $\{a, b, c\}$ is an AT. Furthermore, from the definition of an AT-witness G_{abc} for $\{a, b, c\}$, a, b, and c are vertices of G_{abc}.

Branching Rule 2. If G is chordal: For each triple $\{a, b, c\}$ check if $\{a, b, c\}$ is an AT. For each AT $\{a, b, c\}$, find a simple AT-witness G_{abc} for it with core $\{P_{ab}, P_{bc}, P_{bc}\}$. If there exists an AT $\{a, b, c\}$, such that $|\{cx \mid x \in P_{ab}\} \cup \{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\}| \leq k + 15$ for the simple AT-witness G_{abc}, then:

- Branch on each of the fill edges in the set $\{cx \mid x \in P_{ab}\} \cup \{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\}$.

By Observation 3.1, any interval completion contains at least one edge from the set branched on by Rule 2.

LEMMA 3.3. *Let G be a graph to which Rule* 1 *cannot be applied (i.e., G is chordal). There exists a polynomial time algorithm that finds a simple AT-witness G_{abc} with core $\{P_{ab}, P_{bc}, P_{bc}\}$, where $|\{cx \mid x \in P_{ab}\} \cup \{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\}| \leq k + 15$, if such an AT-witness exists.*

Proof. A simple AT-witness can be found in polynomial time: for a triple of vertices, check if there exists a shortest path between any two of them that avoids the neighborhood of the third vertex. Since shortest paths are used to define simple AT-witnesses and all shortest paths between a pair of vertices have the same length, then $|\{cx \mid x \in P_{ab}\} \cup \{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\}|$ will be the same for all cores $\{P_{ab}, P_{bc}, P_{bc}\}$ defining simple AT-witnesses for an AT $\{a, b, c\}$. □

4. Minimal simple AT-witnesses and shallow vertices: Rule 3. In this section we consider chordal graphs to which Rule 2 cannot be applied, which means chordal graphs containing no AT of small enough size. We introduce minimal simple AT-witnesses and show that they each have a shallow vertex.

DEFINITION 4.1. *A simple AT-witness G_{abc} for an AT $\{a, b, c\}$ is* minimal *if $G_{abc} - x$ is AT-free for any $x \in \{a, b, c\}$.*

Since the vertices of an AT belong to any AT-witness for that AT, it follows that not every AT has a minimal simple AT-witness. However, clearly, for each AT $\{a', b', c'\}$, we can find an AT $\{a, b, c\}$ that has a minimal simple AT-witness. Hence, as long as there is an AT in a chordal graph, there is also an AT that has a minimal simple AT-witness. Interestingly, by the following result of Lekkerkerker and Boland [21] and since minimal simple AT-witnesses are induced subgraphs, a minimal simple AT-witness of a chordal graph is either of constant size or it is a member of one of the two families of graphs shown in Figure 1.

THEOREM 4.2 (see ([21])). *Let G be a chordal graph with more than* 7 *vertices. Then G contains an AT, and no induced subgraph of G contains an AT, if and only if G belongs to the family of graphs shown in Figure* 1.

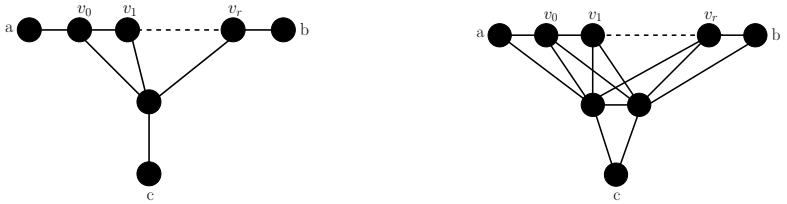


FIG. 1. *These two families of graphs ($r \geq 1$) are the only minimal simple AT-witnesses of nonconstant size that survive Rules* 1 *and* 2.

Since the chordal graphs that we study in this section contain no AT of size less than 15, the minimal simple AT-witnesses that we encounter from now on will always be one of the two types given in Figure 1. (The reader might be interested to know that, by the results of Lekkerkerker and Boland [21], any other possible minimal AT-witness in a chordal graph is of size exactly 7 and is one of two different graphs.)

DEFINITION 4.3. *Let $\{a, b, c\}$ be an AT in a chordal graph G. Vertex c is called* shallow *if shortest a, c-paths and shortest b, c-paths are of length at most* 4.

Observation 4.4. Let G be a graph to which neither Rule 1 (i.e., G is chordal) nor Rule 2 can be applied. Let G_{abc} be a minimal simple AT-witness for an AT $\{a, b, c\}$ in G. Then the following statements are true.

- Each of a, b, c is a simplicial vertex in G_{abc}.

- For any $x \in \{a, b, c\}$, $G_{abc} - x$ is an interval graph, and in this interval graph, $\{a, b, c\} \setminus \{x\}$ is a dominating pair.
- Let $\{P_{ab}, P_{bc}, P_{ac}\}$ be the core of G_{abc}, where $|P_{ab}| \geq |P_{bc}| \geq |P_{ac}|$. If $|P_{ab}| \geq 6$, then c is shallow.

Proof. Since neither Rule 1 nor Rule 2 apply to G and G_{abc} is minimal, then by Theorem 4.2 we know that G_{abc} belongs to one of the families in Figure 1. Each of a, b, c is either the end vertex of the long path or the vertex at the bottom for each of the graphs in Figure 1. Hence the observation follows. The vertex at the bottom is the shallow vertex. ∎

We are ready to give Branching Rule 3.

Branching Rule 3. Let $C(G)$ be the set of vertices of G each of which is shallow in some AT of G. This rule applies if Rules 1 and 2 do not apply and $|C(G)| > k$, in which case we let B be a subset of $C(G)$ with $|B| = k + 1$. For each $c \in B$ find a simple AT-witness G_{abc} with core $\{P_{ab}, P_{bc}, P_{ac}\}$, where c is shallow.

- For each $c \in B$, branch on the at most 8 fill edges $\{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\}$.
- Branch on the at most $|B|(|B| - 1)/2$ possible fill edges $\{uv \mid u, v \in B$ and $uv \notin E\}$.

Observe that Rule 3 only needs a subset of C of size $k+1$, and thus an algorithm can stop the computation of C when this size is reached.

LEMMA 4.5. *If Rule* 3 *applies to G, then any k-interval completion of G contains a fill edge which is branched on by Rule* 3.

Proof. In a k-interval completion we cannot add more than k fill edges. Thus, since $|B| = k+1$ any k-interval completion H of G either contains a fill edge between two vertices in B (and all these are branched on by Rule 3), or there exists a vertex $c \in B$ with no fill edge incident to it (since the opposite would require $k+1$ fill edges). If $c \in B$ does not have a fill edge incident to it, then by Observation 3.1 one of the edges in $\{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\}$ must be a fill edge (and all these are branched on for each $c \in B$ by Rule 3). ∎

LEMMA 4.6. *Let G be a graph to which Rule* 1 *and Rule* 2 *cannot be applied. There exists a polynomial time algorithm that finds the unique maximal set $C(G)$ of shallow vertices in G, and applies Rule* 3 *if $|C(G)| \geq k + 1$.*

Proof. A minimal simple AT-witness can be found in polynomial time: for a triple of vertices, check if there exists a shortest path between any two of them that avoids the neighborhood of the third vertex. If these three paths exist and induce a minimal simple AT-witness for the triple, add the shallow vertex to the set $C(G)$. Notice that every vertex that is shallow in some minimal simple AT-witness will be added to the set $C(G)$. Rule 3 is used on the set $C(G)$, when vertex number $k + 1$ is added. ∎

5. Thick AT-witnesses. In this section we introduce thick AT-witnesses and show that minimal thick AT-witnesses have a shallow vertex. These shallow vertices will be important for the fourth and final rule given in the next section.

We now consider graphs G to which none of the Rules 1, 2, or 3 can be applied. This means that G is chordal (Rule 1), the set $C(G)$ of shallow vertices in G has cardinality at most k (Rule 3), implying that (the connected components of) $G[C(G)]$ is an interval graph (Rule 2).

DEFINITION 5.1. *Let $\{a, b, c\}$ be an AT in a chordal graph G, and let $W = \{w \mid w$ be a vertex of a chordless a, b-path, a, c-path, or b, c-path in $G\}$. The graph $G_{Tabc} = G[W]$ is the (unique)* thick AT-witness *for the AT $\{a, b, c\}$.*

We denote the neighborhoods of a, b, and c in G_{Tabc} respectively by S_a, S_b, and

S_c, since these are minimal separators in G_{Tabc} and also in G by the following two observations.

Observation 5.2. Let G_{Tabc} be a thick AT-witness in a chordal graph G. For any $x \in \{a, b, c\}$, x is a simplicial vertex and $S_x = N_{G_{Tabc}}(x)$ is a minimal separator in G_{Tabc}.

Proof. We prove the observation for $x = a$; the other possibilities are symmetric. Because of the existence of a shortest b, c-path avoiding S_a, it follows that b and c are contained in the same connected component of $G_{Tabc} \setminus S_a$. Every neighbor of a in G_{Tabc} appears in a chordless path from a to either b or c or both. By the fact that b and c appear in the same connected component of $G_{Tabc} \setminus S_a$ and that neighbor of a in G_{Tabc} appears in a chordless path from a to either b or c, we can conclude that S_a is both a minimal a, b-separator and a minimal a, c-separator. In a chordal graph, every minimal separator is a clique [6]. Hence a is simplicial in G_{Tabc}. □

Observation 5.3. Let G_{Tabc} be a thick AT-witness in a chordal graph G. Then the set of minimal separators of G_{Tabc} are exactly the set of minimal a, b-separators, a, c-separators, and b, c-separators of G.

Proof. Let S be a minimal a, b-separator in G. Note that $S \subseteq G_{Tabc}$. There exist two connected components C_a and C_b of $G - S$, containing, respectively, a and b, such that $N_G(C_a) = N_G(C_b) = S$. For any vertex $z \in S$ we can now find a chordless shortest path in G from z to each of a and b, where every intermediate vertex is contained in, respectively, C_a and C_b. By joining these two paths, we get a chordless path from a to b that contains z. Since this holds for any vertex in S, it follows by the way we defined G_{Tabc} that any minimal a, b-separator of G is a minimal a, b-separator of G_{Tabc}. The argument can be repeated with a, c and b, c to show that every minimal a, c-separator or b, c-separator of G is also a minimal separator of G_{Tabc}.

Every minimal separator of G_{Tabc} separates two simplicial vertices appearing in two different leaf bags of any clique tree of G_{Tabc}. Since a, b, c are the only simplicial vertices in G_{Tabc} (every other vertex being an internal vertex of a chordless path), every minimal separator of G_{Tabc} is a minimal a, b-separator, b, c-separator, or a, c-separator. Let S be a minimal a, b-separator in G_{Tabc}. Vertex set S is a subset of a minimal a, b-separator of G, since the same chordless paths exist in G. But S cannot be a proper subset of a minimal a, b-separator of G, since every minimal a, b-separator of G is a minimal a, b-separator in G_{Tabc}, and thus S would not be a minimal separator in G_{Tabc} otherwise. The argument can be repeated with a, c and b, c. □

Definition 5.4. *A thick AT-witness G_{Tabc} is* minimal *if $G_{Tabc} - x$ is AT-free for every $x \in \{a, b, c\}$.*

Observation 5.5. Let G_{Tabc} be a minimal thick AT-witness in a chordal graph G. Then $G_{Tabc} - c$ is an interval graph, and in this interval graph $\{a, b\}$ is a dominating pair.

Proof. The graph $G' = G_{Tabc} - c$ is by definition an interval graph, since G_{Tabc} is a minimal thick AT-witness. For a contradiction assume that $\{a, b\}$ is not a dominating pair, and thus there exists a path P'_{ab} from a to b in $G' - N[y]$ for some vertex $y \in V(G') \setminus \{a, b\}$. Let Q be a clique path of G'. Vertex y does not appear in any bag of Q that contains a or b, and it does not appear in any bags between the subpaths Q_a and Q_b of Q. Let us without loss of generality assume that Q_a appears between Q_y and Q_b in Q. We show that y is then not in any chordless path between any pair of a, b, c, giving the contradiction. Due to the above assumptions, y is not contained in the component C_b of $G' - N[a]$ that contains b. Furthermore, a is a simplicial vertex by Observation 5.2, and P'_{ab} contains vertices from $N_{G'}(a)$; thus $y \notin N_{G'}(a)$ since P'_{ab}

would not avoid the neighborhood of y otherwise. The path $P_{bc} - c$ is contained in C_b since it contains no vertex of $N[a]$, and thus y is not adjacent to any vertex in $P_{bc} - c$. We know that $cy \notin E(G_{Tabc})$, since by Observation 5.2, $N_{G_{Tabc}}(c)$ is a clique, and thus y would be adjacent to the neighbor of c in P_{bc} if cy were an edge in $E(G_{Tabc})$. Now we have a contradiction since y is not in any chordless path between any pair of a, b, c. □

LEMMA 5.6. *Let G be a graph to which neither Rule* 1 *nor Rule* 2 *can be applied, and let G_{Tabc} be a minimal thick AT-witness in G where c is shallow. Then for every vertex $u \in S_c$ we have $S_a \cup S_b \subseteq N[u]$.*

Proof. Let $E' = E(G_{Tabc})$, and let us on the contrary and without loss of generality assume that $c'a' \notin E'$ for $c' \in S_c$ and $a' \in S_a$. Let $P_{ab} = (a = v_1, v_2, \ldots, v_r = b), P_{bc}$, and P_{ac} be the shortest paths used to define a simple AT-witness for $\{a, b, c\}$. We will show that either $\{a', b, c\}$ or $\{a, v_{r-1}, c\}$ is an AT in a subgraph of G_{Tabc}, contradicting its minimality.

Vertex set $\{a', b, c\}$ is an independent set since $cb \notin E'$, $a'b \notin E'$ due to $|P_{ab}| > 15 - 8$ (Rule 2), and $a'c \notin E'$ because c is simplicial in G_{Tabc}, and thus $c'a' \in E'$ if $a'c \in E'$. Either $v_2 = a'$, or $a'v_2 \in E'$ since a is simplicial in G_{Tabc}. $P_{ab} - a + a'$ is a path from a' to b that avoids the neighborhood of c. In the same way $P_{ac} - a + a'$ is a path from a' to c, and since we have already seen that $a'b \notin E'$ this path avoids the neighborhood of b. By Observation 5.5, c' is adjacent to some vertex on the path $P_{ab} = (a = v_1, v_2, \ldots, v_r = b)$. If c' is adjacent to some vertex v_i where $i > 3$, then there is a path $c, c', v_i, \ldots, v_r = b$ that avoids the neighborhood of a', and we have a contradiction since a', b, c would be an AT in $G_{Tabc} - a$. We can therefore assume that there is a $j \in \{2, 3\}$ such that $v_j c' \in E'$, and that there exists no $v_i c' \in E'$ for any $i > 3$. The set $\{a, v_{r-1}, c\}$ is an independent set, since $cv_{r-1}, av_{r-1} \notin E'$. The path $a, v_2, \ldots, v_{r-1}$ avoids the neighborhood of c, the path $c, c', v_j, \ldots, a$ avoids the neighborhood of v_{r-1}, and $P_{bc} - b + v_{r-1}$ is a path from c to v_{r-1} that avoids the neighborhood of a, since b is simplicial in G_{Tabc}. This is a contradiction since $G_{Tabc} - b$ contains the AT $\{a, v_{r-1}, c\}$. □

LEMMA 5.7. *Let $G = (V, E)$ be a graph to which neither Rule* 1 *nor Rule* 2 *can be applied. Let G_{Tabc} be a minimal thick AT-witness in G where c is shallow. Let C_c be the connected component of $G - S_c$ that contains c. Then every vertex of C_c has in G the same set of neighbors S_c outside C_c; in other words $\forall u \in C_c : N_G(u) \setminus C_c = S_c$.*

Proof. By definition $N_G(u) \setminus C_c \subseteq S_c$. Let us assume for a contradiction that $ux \notin E$ for some $x \in S_c$ and $u \in C_c$. Since C_c is a connected component there exists a path from u to c inside C_c. Let u', c' be two consecutive vertices on this path, such that $S_c \subseteq N_G(c')$ and $u'x' \notin E$ for some $x' \in S_c$. By Lemma 5.6 x' creates a short path from a to b that avoids the neighborhood of u', and by using $P_{ac} - c$ and $P_{bc} - c$ and the vertices c' and u' we can create short paths from a to u' and from b to u' that avoid the neighborhoods of b and a, respectively. This is now a contradiction, since $\{a, b, u'\}$ is an AT with a simple AT-witness where the number of branching fill edges are 5 for the path a, a', x', b', b, 5 for $P_{ac} - c$ and c', u', and 5 for $P_{bc} - c$ and c', u', giving a total of 15 branching edges. □

The following simple observations are needed for the proof of Lemma 5.10.

Observation 5.8. A vertex v is simplicial only if v is an end vertex of every chordless path that contains v.

Proof. Any vertex that appears as a nonend vertex in a chordless path has two neighbors that are not adjacent. □

Observation 5.9. Let G_{Tabc} be a minimal thick AT-witness in a graph G to which

neither Rule 1 nor Rule 2 can be applied. Then at least one of the vertices in the AT $\{a, b, c\}$ is shallow, and there exists a minimal simple AT-witness G_{abc} for $\{a, b, c\}$, where $V(G_{abc}) \subseteq V(G_{Tabc})$.

Proof. Let P_{ab}, P_{ac}, P_{bc} be shortest chordless paths contained in G_{Tabc}, and let G_{abc} be defined by P_{ab}, P_{ac}, P_{bc}. It is clear that G_{Tabc} is minimal only if G_{abc} is minimal. By Observation 4.4, Rule 2, and the fact that G_{abc} is a minimal AT-witness for $\{a, b, c\}$, we know that at least one of the vertices in $\{a, b, c\}$ is shallow. ∎

LEMMA 5.10. *Let G be a graph to which neither Rule 1 nor Rule 2 can be applied, and let G_{Tabc} be a thick AT-witness in G. Then there exists a minimal thick AT-witness G_{Txyz} in G, where $V(G_{Txyz}) \subseteq V(G_{Tabc})$ and z is shallow, such that $z \in \{a, b, c\}$.*

Proof. G_{Txyz} will be obtained from G_{Tabc} by deleting one of the simplicial vertices in the AT that defines G_{Tabc}, and repeat this until a minimal thick AT-witness G_{Txyz} is obtained. Note that only neighbors of the deleted vertex can become simplicial after each deletion, by Observation 5.8. As a result, the deleted vertices induce at most three connected components. Actually the number of components will be exactly three, since otherwise the connected components contain a chordless path between two of the vertices in $\{x, y, z\}$ which is a contradiction to the definition of a minimal thick AT-witness. Thus, each connected component is adjacent to one of the vertices x, y, z. By Observation 5.9 one of the vertices x, y, z is shallow. Let us without loss of generality assume that z is the shallow vertex in G_{Txyz}. By Lemma 5.3, minimal separators of G_{Txyz} are also minimal separators of G_{Tabc}, so let us assume without loss of generality that z and c are contained in the same connected component of $G_{Tabc} - N_{G_{Txyz}}(z)$. Notice that z and c might be the same vertex. By Lemma 5.7, c is shallow in the minimal thick AT-witness G_{Txyc}. ∎

Like Rules 2 and 3, Rule 4 will branch on single fill edges, but it will also consider minimal separators, based on the following two basic observations.

Observation 5.11. If G has a minimal thick AT-witness G_{Tabc} in which P_{ac} and P_{bc} are shortest a, c and b, c-paths avoiding $N(b)$ and $N(a)$, respectively, then any interval completion of G either contains a fill edge from the set $\{bx \mid x \in P_{ac}\} \cup \{ax \mid x \in P_{bc}\}$ or contains one of the edge sets $\{\{cx \mid x \in S\} \mid S$ is a minimal a, b-separator in $G_{Tabc}\}$.

Proof. By Observation 3.1, we know that at least one of the edges in $\{ax \mid x \in P_{bc}\} \cup \{bx \mid x \in P_{ac}\} \cup \{cx \mid x \in P_{ab}\}$ for the paths P_{ab}, P_{ac}, P_{bc} defined in the proof of Observation 5.9, is a fill edge of any interval completion of G. If an interval completion H does not contain any fill edge from the set $\{bx \mid x \in P_{ac}\} \cup \{ax \mid x \in P_{bc}\}$, then H contains at least one fill edge from the set $\{cx \mid x \in P'_{ab}\}$, where P'_{ab} is any chordless a, b-path in G that avoids the neighborhood of c. Thus, $N_H(c)$ contains a minimal a, b-separator in G (which by Observation 5.3 is also a minimal a, b-separator in G_{Tabc}) since every chordless and thus every a, b-path in $G - N[c]$ contains a vertex of $N_H(c)$. ∎

Observation 5.12. Let G be a graph to which neither Rule 1 nor Rule 2 can be applied, and let G_{Tabc} be a minimal thick AT-witness in G where c is shallow. Then $S_c \subset S$ for every minimal a, b-separator S different from S_a and S_b.

Proof. Let S be a minimal a, b-separator different from S_a and S_b. No minimal a, b-separator contains another minimal a, b-separator as a subset; thus there exists a vertex $a' \in S_a \setminus S$ and a vertex $b' \in S_b \setminus S$. S is then also a minimal a', b'-separator because of the edges aa' and bb'. It then follows from Lemma 5.6 that $S_c \subset N(a') \cap N(b')$, and thus $S_c \subset S$. ∎

6. Partitioning the shallow vertices: Rule 4. In this section we present the fourth and final rule and prove correctness of the resulting search tree algorithm. We start by detailing the computation of the set $C(G)$ of shallow vertices which will give us a partition of $C(G)$ that we will use in our Branching Rule 4.

DEFINITION 6.1. *Given a graph G to which Rules 1 and 2 do not apply, we compute a set $C(G) = C_1 \cup C_2 \cup \cdots \cup C_r$ of vertices that are shallow in some minimal thick AT-witness, with $G \setminus C(G) = R_r$ an interval graph, as follows:*
$R_0 := G$; $i := 0$; $C(G) := \emptyset$;
while *R_i is not an interval graph* **do**
  $i := i + 1$;
  Find $G_{Ta_ib_ic_i}$ a minimal thick AT-witness in R_{i-1} with c_i shallow;
  Let C_i be the connected component of $R_{i-1} - N_{G_{Ta_ib_ic_i}}(c_i)$ that contains c_i;
  for *each $c \in C_i$* **do** $G_{Ta_ib_ic} := G_{Ta_ib_ic_i} - c_i + c$;
  $R_i := R_{i-1} - C_i$;
  $C(G) := C(G) \cup C_i$;
end-while
$r := i$

The minimal thick AT-witness $G_{Ta_ib_ic_i}$ is found by first finding an AT $\{a, b, c\}$, then removing simplicial vertices different from a, b, c according to Observation 5.8 to get a thick AT-witness, and then applying the procedure in the proof of Lemma 5.10.

A priori we have no guarantee that there are no edges between a vertex in C_i and a vertex in C_j, for some $i \neq j$, but when $|C(G)| \leq k$ (which is ensured by Rule 3) this indeed holds, as shown in the following lemma.

LEMMA 6.2. *Let $G = (V, E)$ be a graph to which none of Rules $1, 2, 3$ can be applied, and let $C(G) = C_1 \cup C_2 \cup \cdots \cup C_r$ from Definition 6.1. Then C_i induces an interval graph that is a connected component of $G[C(G)]$, for each $1 \leq i \leq r$.*

Proof. First, $|C_i| \leq k$ since Rule 3 does not apply, and since Rules 1, 2 do not apply, it must induce an interval graph. To argue that it is a connected component, note first that by definition $G[C_i]$ is connected and $C_i \cap C_j = \emptyset$ for any $i \neq j$. For a contradiction we assume that $cz \in E$ for some $c \in C_i$ and $z \in C_j$ with $i < j$. Notice that $cz \in E$ implies that $z \in S_c$. Let G_{Tabc} be the minimal thick AT-witness in R_{i-1} with c the shallow vertex and $S_c = N_{G_{Tabc}}(c)$, and likewise let G_{Txyz} be the minimal thick AT-witness in R_{j-1} with z shallow and $S_z = N_{G_{Txyz}}(z)$. Let P_{ab} be a path from a to b in $G_{Tabc} \setminus N(c)$. There are now two cases.

Case I: There is a vertex $w \in P_{ab} \cap S_z$. Note that S_c and S_z are minimal separators in the chordal graphs R_{i-1} and R_{j-1}, respectively, and thus by Observation 5.3 S_c and S_z are cliques [6]. Thus, since $cw \notin E$ we must have $c \notin S_z$. But then we have c and z in the same component D_z of $G \setminus S_z$. By Lemma 5.7 c and z must therefore have the same neighbors outside D_z. But this contradicts the fact that $zw \in E$ while $cw \notin E$.

Case II: $P_{ab} \cap S_z = \emptyset$. Let D_z be the connected component of $G \setminus S_z$ that contains z. By Observation 5.5 $G_{Tabc} \setminus \{c\}$ is an interval graph where a, b is a dominating pair; thus $zw \in E$ for some $w \in P_{ab}$ since $z \in S_c$ and therefore $V(P_{ab}) \subseteq D_z$.

Since c is shallow we know that $|P_{ac}| + |P_{bc}| \leq 8$, and since Rule 2 cannot be applied we know that $|P_{ab}| + |P_{ac}| + |P_{bc}| \geq k + 16$. Thus we have at least $k + 16 - 8$ vertices in P_{ab} and thus $|D_z| \geq |P_{ab}| > k$. Assuming we can show the subset-property $D_z \subseteq C_1 \cup C_2 \cup \cdots \cup C_j$ we are done with the proof since this will lead to the contradiction $k < |D_z| \leq |C_1 \cup C_2 \cup \cdots \cup C_j| \leq |C(G)| \leq k$. Let us prove the subset-property. G has a perfect elimination ordering starting with the vertices of C_1, as

these vertices are a component resulting from removing a minimal separator from G. By induction, we have that G has a perfect elimination ordering α starting with the vertices in $C_1 \cup C_2 \cup \cdots \cup C_{j-1}$. For a contradiction assume there exists a vertex $t \in D_z \setminus (C_1 \cup C_2 \cup \cdots \cup C_j)$. As $t \in D_z$ there is a shortest chordless t,z-path P_{tz} in D_z. The edge $tz \notin E$ since this would make t a member of C_j (as $t \notin C_j$). Only vertices in $C_1 \cup C_2 \cup \cdots \cup C_{j-1}$ are removed from the graph, and these separate z from t in $G[D_z]$ (since $t \notin C_j$). Let s be the lowest numbered vertex in the ordering α that belongs to the path P_{tz}. This is now a contradiction, since a nonend vertex of a chordless path cannot be simplicial if two adjacent vertices eliminated later in the perfect elimination ordering are nonadjacent. ∎

Rule 4 will branch on a bounded number of single fill edges and it will also compute a greedy completion by choosing for each shallow vertex a minimal separator minimizing fill and making the shallow vertex adjacent to all vertices of that separator. We will prove that if none of the single fill edges branched on in Rule 4 are present in any k-interval completion, then the greedy completion gives an interval completion with the minimum number of edges. The greedy choices of separators are made as follows:

DEFINITION 6.3. *Let G be a graph to which none of Rules $1,2,3$ can be applied. Let Definition* 6.1 *give $C(G) = C_1 \cup C_2 \cup \cdots \cup C_r$, representative vertices $c_1, c_2, \ldots, c_r$ and minimal thick AT-witnesses $G_{Ta_ib_ic_i}$ and graphs $G = R_0 \supset R_1 \supset \cdots \supset R_r$, with R_r interval. Let M_i for $i = 1,2,\ldots,r$ be a minimal a_i,b_i-separator S in $G_{Ta_ib_ic_i}$ different from S_{a_i} and S_{b_i} and $N(C_j)$ for all $1 \le j \le r$, satisfying $S \cap C(G) = \emptyset$ and minimizing $|S \setminus N(C_i)|$. If no such S exists, define $M_i = null$.*

LEMMA 6.4. *If $M_i \neq null$, then M_i is a minimal separator in R_r.*

Proof. The vertex set M_i is a minimal separator in $G_{Ta_ib_ic_i}$ by construction, and since $G_{Ta_ib_ic_i}$ is a subgraph of the chordal graph R_i it is by Observation 5.3 also a minimal separator of R_i. We prove that M_i is also a minimal separator in R_j for any $i+1 \le j \le r$ by induction on j. Recall that R_j is obtained by removing C_j from R_{j-1}, where C_j is a component of $R_{j-1} \setminus S_{c_i}$ for a minimal separator S_{c_i} of R_{j-1}, and $S_{c_i} = N(C_j)$ by Lemma 5.7. Consider a clique tree of R_{j-1} and observe that any minimal separator of R_{j-1} that is not a minimal separator of R_j either is equal to $N(C_j)$ or contains a vertex of C_j. Finally, note that the minimal separator M_i has been chosen so that it is not of this type. ∎

Branching Rule 4. Rule 4 applies if none of Rules 1, 2, 3 apply, in which case we compute, as in Definitions 6.1 and 6.3, $C_1, C_2, \ldots, C_r$ (which are connected components of $G[C(G)]$ by Lemma 6.2), the minimal thick AT-witnesses $G_{Ta_ib_ic}$ with c shallow for each $c \in C_i$, and $M_1, \ldots, M_r$ (which are minimal separators of R_r by Lemma 6.4). For each $1 \le i \le r$ and each $c \in C_i$ choose $a'_i \in S_{a_i} \setminus S_c$ and $b'_i \in S_{b_i} \setminus S_c$ and find P_{a_ic} and P_{b_ic} (shortest paths in $G_{Ta_ib_ic}$ avoiding $N(b_i)$ and $N(a_i)$, respectively, of length at most 4 by Observation 5.9). For each pair $1 \le i \ne j \le r$, choose a vertex $v_{i,j} \in N(C_j) \setminus N(C_i)$ (if it exists).

- For $1 \le i \le r$ and $c \in C_i$, branch on the at most 8 fill edges $\{a_ix \mid x \in P_{b_ic}\} \cup \{b_ix \mid x \in P_{a_ic}\}$ and also on the 2 fill edges $\{ca'_i, cb'_i\}$.
- Branch on the at most $|C(G)|(|C(G)|-1)/2$ fill edges $\{uv \mid u,v \in C(G)$ and $uv \notin E\}$.
- Branch on the at most $|C(G)|r$ fill edges $\bigcup_{1 \le i \ne j \le r}\{cv_{i,j} \mid c \in C_i\}$.
- Finally, compute $H = (V,\ E \cup \bigcup_{1 \le i \le r}\{cx \mid c \in C_i$ and $x \in M_i\})$ and check if it is a k-interval completion of G (note that we do not branch on H).

LEMMA 6.5. *If G has a k-interval completion, and Rules $1,2,$ and 3 do not apply*

to G, and no k-interval completion of G contains any single fill edge branched on by Rule 4*, then the graph H, which Rule* 4 *obtains by adding fill edges from every vertex in C_i to every vertex in M_i for every $1 \leq i \leq r$, is a k-interval completion of G.*

Proof. By Observation 5.11, for each $c \in C_i$ either one of the edges in $\{a_i x \mid x \in P_{b_i c}\} \cup \{b_i x \mid x \in P_{a_i c}\}$ is a fill edge (and all these are branched on by Rule 4) or else the k-interval completion contains the edge set $\{cx \mid x \in S\}$ for some minimal a_i, b_i-separator S in $G_{Ta_i b_i c}$. Such an edge set in a k-interval completion is one of four types (listed below) depending on the separator S used to define it. For each type and any $c \in C_i$ we argue that Rule 4 considers it. Observe that $N(C_i) \setminus C_i = N(c) \setminus C_i$ by Lemma 5.7, and thus the fill edges from c will go to vertices in $S \setminus N(C_i)$, which is nonempty since there is an a_i, b_i-path avoiding $N(c)$. We now give the four types of minimal separators S and show that the first three are branched on by a single fill edge:

1. $S \cap C(G) \neq \emptyset$. Since $N(C_i) \cap C(G) = \emptyset$ by Lemma 6.2, we have in this case a fill edge between two vertices in $C(G)$ (between $c \in C_i$ and a vertex in $C(G) \cap S \setminus N(C_i)$), and all these are branched on by Rule 4.
2. $S = S_{a_i}$ or $S = S_{b_i}$, where S_{a_i}, S_{b_i}, S_c defined by $G_{Ta_i b_i c}$. We found in Rule 4 a pair of vertices $a'_i \in S_{a_i} \setminus S_c$ and $b'_i \in S_{b_i} \setminus S_c$ and branched on the fill edges ca'_i and cb'_i.
3. $S = N(C_j)$ for some $1 \leq j \leq r$. If $S = N(C_j)$, then $N(C_j) \setminus N(C_i) \neq \emptyset$ and we found in Rule 4 a vertex $v_{i,j} \in N(C_j) \setminus N(C_i)$ and branched on the fill edge $cv_{i,j}$.
4. S is none of the three types above. Note that M_i was chosen in Definition 6.3 by looping over all minimal a_i, b_i-separators S in $G_{Ta_i b_i c_i}$ (which by Lemma 5.7 are exactly the minimal a_i, b_i-separators of $G_{Ta_i b_i c}$) satisfying $S \cap C(G) = \emptyset$, $S \neq S_a$, $S \neq S_b$, and $S \neq N(C_j)$ for any j. Thus, of all separators of this fourth type, M_i is the one minimizing the fill.

The assumption is that G has a k-interval completion but no single edge branched on by Rule 4 is present in any k-interval completion. This means that only separators of the fourth type are used in any k-interval completion. Since H added the minimum possible number of fill edges while using only separators of the fourth type, any interval completion of G must add at least $|E(H) \setminus E(G)|$ edges. It remains to show that H is an interval graph. H is constructed from an interval graph R_r and the components $G[C_1], \ldots, G[C_r]$ of $G[C(G)]$, which are interval graphs by Lemma 6.2, and $M_1, \ldots, M_r$ which are minimal separators of R_r by Lemma 6.4. Since $M_i \neq S_{a_i}$ and $M_i \neq S_{b_i}$ we have by Observation 5.12 that $S_c = N(C_i) \subset M_i$ so that adding all edges between C_i and M_i for $1 \leq i \leq r$ gives the graph H. We show that H is an interval graph by induction on $0 \leq i \leq r$. Let $H_0 = R_r$ and let H_i for $i \geq 1$ be the graph we get from H_{i-1} and C_i by making all vertices of C_i adjacent to all vertices of the minimal separator M_i of R_r. H_0 is an interval graph by induction, and its minimal separators include all minimal separators of R_r. If $(K_1, K_2, \ldots, K_q)$ is a clique path of H_{i-1} with $M_i = K_j \cap K_{j+1}$, and $(K'_1, K'_2, \ldots, K'_p)$ is a clique path of $G[C_i]$, then $(K_1, K_2, \ldots, K_j, K'_1 \cup M_i, K'_2 \cup M_i, \ldots, K'_p \cup M_i, K_{j+1}, \ldots, K_q)$ is a clique path of H_i, and hence H_i is an interval graph. Finally, observe that the minimal separators of H_{i-1} and hence of R_r are also minimal separators of H_i. ∎

THEOREM 6.6. *The search tree algorithm applying Rules* $1, 2, 3,$ *and* 4 *in that order decides in $O(k^{2k} n^3 m)$ time whether an input graph G on n vertices and m edges can be completed into an interval graph by adding at most k edges.*

Proof. At least one of the rules will apply to any graph which is not interval.

The correctness of Rule 1 is well understood [17, 2]; that of Rules 2 and 3 follows by Observations 3.1 and 5.11 and of Rule 4 by Lemma 6.5. Each branching of Rules 2, 3, and 4 adds a single fill edge and drops k by one. As already mentioned, Rule 1 could also have added a single fill edge in each of its then at most k^2 branchings. The height of the tree is thus no more than k, before k reaches 0 and we can answer "no." If an interval graph is found we answer "yes."

Let us argue for the runtime. The graph we are working on never has more than $m + k$ edges. In Rule 1 we decide in linear time if the graph has a large induced cycle. In Rule 2 we may have to try all triples when searching for an AT with a small simple AT-witness, taking $O(n^3(m + k))$ time. In Rule 3 and 4 we need to find a minimal thick AT-witness at most $k + 1$ times. As observed earlier, the minimal thick AT-witness is found by first finding an AT $\{a, b, c\}$, which can be done in time $O(m + k)$ since G is a chordal graph [19], then remove simplicial vertices different from a, b, c to find the thick AT-witness, and then make it minimal. Using a clique tree we find in this way a single minimal thick AT-witness in time $O(n^3)$ and at most k of them in time $O(n^3k)$. Hence each rule takes time at most $O(n^3(m + k))$ and has branching factor at most k^2 (e.g., in Rule 1 and also in Rule 3 when branching on all fill edges between pairs of shallow vertices). The height of the search tree is at most k and the number of nodes therefore at most k^{2k}. We can assume $k \leq n \leq m$ since otherwise a brute-force algorithm easily solves minimum interval completion in n^{2n} steps. Thus each rule takes time $O(n^3m)$ for total runtime $O(k^{2k}n^3m)$. □

7. Concluding remarks. We have shown that k-interval completion is FPT. The runtime of our algorithm can probably be improved somewhat, at the expense of much more complicated data structures. In an earlier version of this paper, in the STOC 2007 proceedings, we asked if there was a hereditary graph class recognizable in polynomial time for which the k-completion problem into this graph class was not FPT. This question has been answered [22], since for the complements of wheel-free graphs the k-completion problem is $W[2]$-complete. It is still an open problem whether the complexity of k-completions into perfect graphs is FPT.

An alternative equivalent definition of the complexity class FPT relates to kernelization. In this formulation a parameterized problem is FPT if there exists a polynomial-time algorithm that for any instance outputs an equivalent "kernelized" instance whose size is a function of the parameter only. The quest for the smallest possible kernel size is orthogonal to the quest for the fastest possible FPT algorithm. The FPT algorithm given here for k-interval completion implies that this problem has an exponential-sized kernel. We leave as an open problem whether k-interval completion has a polynomial-sized kernel.

REFERENCES

[1] P. Buneman, *A characterization of rigid circuit graphs*, Discrete Math., 9 (1974), pp. 205–212.

[2] L. Cai, *Fixed-parameter tractability of graph modification problems for hereditary properties*, Inform. Process. Lett., 58 (1996), pp. 171–176.

[3] L. Cai, *Parameterized complexity of vertex colouring*, Discrete Appl. Math., 127 (2003), pp. 415–429.

[4] D. G. Corneil, S. Olariu, and L. Stewart, *Asteroidal triple-free graphs*, SIAM J. Discrete Math., 10 (1997), pp. 399–430.

[5] J. Díaz, A. M. Gibbons, M. S. Paterson, and J. Torán, *The minsumcut problem*, in Proceedings WADS, 1991, pp. 65–79.

[6] G. A. Dirac, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.

[7] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[8] G. Even, J. Naor, S. Rao, and B. Scheiber, *Divide-and-conquer approximation algorithms via spreading metrics*, in Proceedings FOCS, 1995, pp. 62–71.

[9] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979.

[10] J. A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[11] P. C. Gilmore and A. J. Hoffman, *A characterization of comparability graphs and of interval graphs*, Canad. J. Math., 16 (1964), pp. 539–548.

[12] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir, *Four strikes against physical mapping of DNA*, J. Comput. Bio., 2 (1995), pp. 139–152.

[13] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, 2nd ed., Annals of Discrete Mathematics 57, Elsevier, Amsterdam, 2004.

[14] G. Gutin, S. Szeider, and A. Yeo, *Fixed-parameter complexity of minimum profile problems*, in Proceedings IWPEC 2006, Lecture Notes in Comput. Sci. 4169, Springer-Verlag, Berlin, 2006, pp. 60–71.

[15] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger, *Minimal interval completions*, in Proceedings ESA 2005, Lecture Notes in Comput. Sci. 3669, Springer-Verlag, Berlin, 2005, pp. 403–414.

[16] H. Kaplan, R. Shamir, and R. E. Tarjan, *Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping*, in Proceedings FOCS, 1994, pp. 780–791.

[17] H. Kaplan, R. Shamir, and R. E. Tarjan, *Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs*, SIAM J. Comput., 28 (1999), pp. 1906–1922.

[18] T. Kashiwabara and T. Fujisawa, *An NP-complete problem on interval graphs*, IEEE Symposium of Circuits and Systems, 1979, pp. 82–83.

[19] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad, *Certifying algorithms for recognizing interval graphs and permutation graphs*, SIAM J. Comput., 36 (2006), pp. 326–353.

[20] D. Kuo and G. J. Chang, *The profile minimization problem in trees*, SIAM J. Comput., 23 (1994), pp. 71–81.

[21] C. G. Lekkerkerker and J. C. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.

[22] D. Lokshtanov, *Wheel-free deletion is $W2$-hard*, in Proceedings IWPEC 2008, Lecture Notes in Comput. Sci. 5018, Springer-Verlag, Berlin, 2008, pp. 141–147.

[23] S. Rao and A. W. Richa, *New approximation techniques for some linear ordering problems*, SIAM J. Comput., 34 (2004), pp. 388–404.

[24] R. Ravi, A. Agrawal, and P. Klein, *Ordering problems approximated: single processor scheduling and interval graph completion*, in Proceedings ICALP 1991, pp. 751–762.

[25] M. Serna and D. Thilikos, *Parameterized complexity for graph layout problems*, Bull. Eur. Assoc. Theor. Comput. Sci., 86 (2005), pp. 41–65.

[26] R. E. Tarjan, *Graph theory and Gaussian elimination*, Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, 1976, pp. 3–22.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 2021–2043

OPTIMIZING SCHEMA LANGUAGES FOR XML: NUMERICAL CONSTRAINTS AND INTERLEAVING*

WOUTER GELADE†, WIM MARTENS‡, AND FRANK NEVEN§

Abstract. The presence of a schema offers many advantages in processing, translating, querying, and storage of XML data. Basic decision problems such as equivalence, inclusion, and nonemptiness of intersection of schemas form the basic building blocks for schema optimization and integration, and algorithms for static analysis of transformations. It is thereby paramount to establish the exact complexity of these problems. Most common schema languages for XML can be adequately modeled by some kind of grammar with regular expressions at right-hand sides. In this paper, we observe that, apart from the usual regular operators of union, concatenation, and Kleene-star, schema languages also allow numerical occurrence constraints and interleaving operators. Although the expressiveness of these operators remains within the regular languages, the presence or absence of these operators has a significant impact on the complexity of the basic decision problems. We present a complete overview of the complexity of the basic decision problems for DTDs, XSDs, and Relax NG with regular expressions incorporating numerical occurrence constraints and interleaving. We also discuss chain regular expressions and the complexity of the schema simplification problem incorporating the new operators.

Key words. XML schema languages, complexity, optimization, regular expressions

AMS subject classifications. 68P15, 68Q45, 68Q17

DOI. 10.1137/070697367

1. Introduction. XML is the lingua franca for data exchange on the Internet [1]. Within applications or communities, XML data is usually not arbitrary but adheres to some structure imposed by a schema. The presence of such a schema not only provides users with a global view on the anatomy of the data, but, far more importantly, it enables automation and optimization of standard tasks such as (i) searching, integration, and processing of XML data (cf., e.g., [12, 23, 26, 44]), and (ii) static analysis of transformations (cf., e.g., [2, 17, 29, 34]). Decision problems such as equivalence, inclusion, and nonemptiness of intersection of schemas, hereafter referred to as *the basic decision problems*, constitute essential building blocks in solutions for the just mentioned optimization and static analysis problems. Additionally, the basic decision problems are fundamental for schema minimization (cf., e.g., [10, 30]). Because of their widespread applicability, it is therefore important to establish the exact complexity of the basic decision problems for the various XML schema languages.

The most common schema languages for XML are DTD, XML Schema [39], and Relax NG [9], and these can be modeled by grammar formalisms [33]. In particular, DTDs correspond to context-free grammars with regular expressions (REs) at right-hand sides, while Relax NG is abstracted by extended DTDs (EDTDs) [35]

*Received by the editors July 16, 2007; accepted for publication (in revised form) August 8, 2008; published electronically January 30, 2009. A preliminary version of this work was presented at the 11th International Conference on Database Theory, Barcelona, Spain, 2007.
http://www.siam.org/journals/sicomp/38-5/69736.html

†School for Information Technology, Hasselt University and Transnational University of Limburg, B-3590 Limburg, Belgium, and Fund for Scientific Research - Flanders, B-1000 Brussels, Belgium (wouter.gelade@uhasselt.be).

‡Department of Computer Science, Universität Dortmund, D-44221 Dortmund, Germany (wim.martens@udo.edu).

§School for Information Technology, Hasselt University and Transnational University of Limburg, B-3590 Limburg, Belgium (frank.neven@uhasselt.be).

shop	$\rightarrow$	regular* & discount-box*
regular	$\rightarrow$	cd
discount-box	$\rightarrow$	cd$^{[10,12]}$ price
cd	$\rightarrow$	artist & title & price

FIG. 1.1. *A sample schema using the numerical occurrence and interleave operators. The schema defines a shop that sells CDs and offers a special price for boxes of* 10–12 *CDs.*

or, equivalently, unranked tree automata [5], defining the regular unranked tree languages. While XML Schema is usually abstracted by unranked tree automata as well, recent results indicate that XSDs correspond to a strict subclass of the regular tree languages and are much closer to DTDs than to tree automata [27]. In fact, they can be abstracted by single-type EDTDs. As detailed in [28], the relationship between schema formalisms and grammars provides direct upper and lower bounds for the complexity of the basic decision problems.

A closer inspection of the various schema specifications reveals that the above abstractions in terms of grammars with regular expressions is too coarse. Indeed, in addition to the conventional regular expression operators such as concatenation, union, and Kleene-star, the XML Schema and the Relax NG specifications allow two other operators as well:

(1) Both the XML Schema and the Relax NG specifications allow a certain form of unordered concatenation: the `ALL` and the `interleave` operator, respectively. This operator is actually the resurrection of the &-operator from SGML DTDs that was excluded from the definition of XML DTDs. Although there are restrictions on the use of `ALL` and `interleave`, we consider the operator in its unrestricted form. We refer by RE(&) to such regular expressions with the interleaving operator.
(2) The XML Schema specification allows us to express numerical occurrence constraints which define the minimal and maximal number of times a regular construct can be repeated. We refer by RE(#) to such regular expressions with numerical occurrence constraints.

We illustrate these additional operators in Figure 1.1. Their formal definition is given in section 2. Although the new operators can be expressed by the conventional regular operators, they cannot do so succinctly [14], which has severe implications for the complexity of the basic decision problems.

The goal of this paper is to study the complexity of the basic decision problems for DTDs, XSDs, and Relax NG with regular expressions extended with interleaving and numerical occurrence constraints. The latter class of regular expressions is denoted by RE(#, &). As observed in section 5, the complexity of inclusion and equivalence of RE(#, &) expressions (and subclasses thereof) carries over to DTDs and single-type EDTDs. We therefore first establish the complexity of the basic decision problems for RE(#, &) expressions and frequently occurring subclasses. These results are summarized in Table 1.1 and Table 4.1. Of independent interest, we introduce NFA(#, &)s, an extension of NFAs with counter and split/merge states for dealing with numerical occurrence constraints and interleaving operators. Finally, we revisit the simplification problem introduced in [27] for schemas with RE(#, &) expressions. This problem is defined as follows: given an extended DTD, can it be rewritten into an equivalent DTD or a single-type EDTD?

TABLE 1.1
Overview of new and known complexity results. All results are completeness results. The new results are printed in bold.

	INCLUSION	EQUIVALENCE	INTERSECTION
RE	PSPACE [41]	PSPACE [41]	PSPACE [24]
RE(&)	EXPSPACE [31]	EXPSPACE [31]	**PSPACE**
RE(#)	EXPSPACE [32]	EXPSPACE [32]	**PSPACE**
RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
NFA(#), NFA(&), and NFA(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
DTDs with RE	PSPACE [41]	PSPACE [41]	PSPACE [24]
DTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**PSPACE**
single-type EDTDs with RE	PSPACE [28]	PSPACE [28]	EXPTIME [28]
single-type EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**EXPTIME**
EDTDs with RE	EXPTIME [37]	EXPTIME [37]	EXPTIME [38]
EDTDs with RE(#), RE(&), or RE(#, &)	**EXPSPACE**	**EXPSPACE**	**EXPTIME**

In this paper, we do not consider deterministic or one-unambiguous regular expressions which form a strict subclass of the regular expressions [6]. The reason is twofold. First, one-unambiguity is a highly debatable constraint (cf., e.g., [42, p. 98] and [25, 40]) which is required only for DTDs and XML Schema, not for Relax NG. Actually, the only direct advantage of one-unambiguity is that it gives rise to PTIME algorithms for some of the basic decision problems for standard regular expressions. The latter does not hold anymore for RE(#, &) expressions rendering the notion even less attractive. Indeed, already intersection for one-unambiguous regular expressions is PSPACE-hard [28], and inclusion for one-unambiguous RE(#) expressions is coNP-hard [22]. A second reason is that, in contrast to conventional regular expressions, one-unambiguity is not yet fully understood for regular expressions with numerical occurrence constraints and interleaving operators. Some initial results are provided by Brüggemann-Klein [7] and Kilpeläinen and Tuhkanen [21] who give algorithms for deciding one-unambiguity of RE(&) and RE(#) expressions, respectively. However, the results of Brüggemann-Klein are on the SGML interleaving operator, which is not the same as the Relax NG interleaving operator considered here. Furthermore, no study investigating the properties of these one-unambiguous languages has been undertaken. Such a study, although definitely relevant, is outside the scope of this paper.

Outline. In section 2, we provide the necessary definitions. In section 3, we define NFA(#, &). In sections 4 and 5, we establish the complexity of the basic decision problems for regular expressions and schema languages, respectively. We discuss simplification in section 6. We conclude in section 7.

2. Definitions.

2.1. Regular expressions with counting and interleaving. For the rest of the paper, Σ always denotes a finite alphabet. A Σ-*symbol* (or simply symbol) is an element of Σ, and a Σ-*string* (or simply string) is a finite sequence $w = a_1 \cdots a_n$ of Σ-symbols. We define the length of w, denoted by $|w|$, to be n. We denote the

empty string by ε. The set of *positions of* w is $\{1, \ldots, n\}$, and the *symbol of* w *at position* i is a_i. By $w_1 \cdot w_2$ we denote the *concatenation* of two strings w_1 and w_2. For readability, we usually denote the concatenation of w_1 and w_2 by $w_1 w_2$. The set of all strings is denoted by Σ^*. A *string language* is a subset of Σ^*. For two string languages $L, L' \subseteq \Sigma^*$, we define their concatenation $L \cdot L'$ to be the set $\{ww' \mid w \in L, w' \in L'\}$. We abbreviate $L \cdot L \cdots L$ (i times) by L^i. By $w_1 \,\&\, w_2$ we denote the set of strings that is obtained by *interleaving* or *shuffling* w_1 and w_2 in every possible way. That is, for $w, w_1, w_2 \in \Sigma^*$ and $a, b \in \Sigma$, $w \,\&\, \varepsilon = \varepsilon \,\&\, w = \{w\}$, and $a \cdot w_1 \,\&\, b \cdot w_2 = (\{a\} \cdot (w_1 \,\&\, b \cdot w_2)) \cup (\{b\} \cdot (a \cdot w_1 \,\&\, w_2))$. Here, $\cdot$ has precedence over &. The operator & is then extended to languages in the canonical way.

The set of *regular expressions* over Σ, denoted by RE, is defined in the usual way, ε, and every Σ-symbol is a regular expression; when r and s are regular expressions, then rs, $r + s$, and r^* are also regular expressions. By RE(#, &) we denote RE extended with two new operators: *interleaving* and *numerical occurrence constraints*. That is, when r and s are RE(#, &) expressions, so are $r \,\&\, s$ and $r^{[k,\ell]}$ for $k, \ell \in \mathbb{N}$ with $k \leq \ell$ and $\ell > 0$. By RE(#) and RE(&), we denote RE extended only with counting and interleaving, respectively. Notice that we disallow $\emptyset$ as it does not occur in practical schema languages.

The language defined by a regular expression r, denoted by $L(r)$, is inductively defined as follows: $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L(rs) = L(r) \cdot L(s)$; $L(r+s) = L(r) \cup L(s)$; $L(r^*) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$; $L(r^{[k,\ell]}) = \bigcup_{i=k}^{\ell} L(r)^i$; and $L(r \,\&\, s) = L(r) \,\&\, L(s)$. The *size* of a regular expression r over Σ, denoted by $|r|$, is the number of Σ-symbols and operators occurring in r plus the sizes of the binary representations of the integers. By $r?$ and r^+, we abbreviate the expression $r + \varepsilon$ and rr^*, respectively. We assume familiarity with finite automata such as nondeterministic finite automata (NFAs) and deterministic finite automata (DFAs) [16].

2.2. Schema languages for XML. The set of *unranked* Σ*-trees*, denoted by $\mathcal{T}_\Sigma$, is the smallest set of strings over Σ and the parenthesis symbols "(" and ")" such that, for $a \in \Sigma$ and $w \in (\mathcal{T}_\Sigma)^*$, $a(w)$ is in $\mathcal{T}_\Sigma$. So, a tree is either ε (empty) or is of the form $a(t_1 \cdots t_n)$, where each t_i is a tree. In the tree $a(t_1 \cdots t_n)$, the subtrees $t_1, \ldots, t_n$ are attached to the root labeled a. We write a rather than $a()$. Notice that there is no a priori bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. For every $t \in \mathcal{T}_\Sigma$, the *set of nodes* of t, denoted by $\mathrm{Dom}(t)$, is the set defined as follows: (i) if $t = \varepsilon$, then $\mathrm{Dom}(t) = \emptyset$; and (ii) if $t = a(t_1 \cdots t_n)$, where each $t_i \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iu \mid u \in \mathrm{Dom}(t_i)\}$. In what follows, whenever we say tree, we always mean Σ-tree. A *tree language* is a set of trees.

We make use of the following definitions to abstract from the commonly used schema languages.

DEFINITION 2.1. *Let $\mathcal{R}$ be a class of regular expressions over Σ.*

1. *A* DTD($\mathcal{R}$) *over Σ is a tuple (Σ, d, s_d), where d is a function that maps Σ-symbols to elements of $\mathcal{R}$ and $s_d \in \Sigma$ is the start symbol. For convenience of notation, we denote (Σ, d, s_d) by d and leave the start symbol s_d implicit whenever this cannot give rise to confusion.*
   A tree t satisfies *d if* (i) *$lab^t(\varepsilon) = s_d$ and,* (ii) *for every $u \in Dom(t)$ with n children, $lab^t(u1) \cdots lab^t(un) \in L(d(lab^t(u)))$. By $L(d)$ we denote the set of trees satisfying d.*
2. *An* extended DTD (EDTD($\mathcal{R}$)) *over Σ is a 5-tuple $D = (\Sigma, \Sigma', d, s, \mu)$, where Σ' is an alphabet of* types*, (Σ', d, s) is a DTD($\mathcal{R}$) over Σ', and μ is a mapping from Σ' to Σ.*

A tree t then satisfies *an extended DTD if $t = \mu(t')$ for some $t' \in L(d)$. Here we abuse notation and let μ also denote its extension to define a homomorphism on trees. Again, we denote by $L(D)$ the set of trees satisfying D. For ease of exposition, we always take $\Sigma' = \{a^i \mid 1 \leq i \leq k_a, a \in \Sigma, i \in \mathbb{N}\}$ for some natural numbers k_a, and we set $\mu(a^i) = a$.*

3. *A* single-type EDTD (EDTD$^{\text{st}}(\mathcal{R})$) *over Σ is an EDTD($\mathcal{R}$) $D = (\Sigma, \Sigma', d, s, \mu)$ with the property that, for every $a \in \Sigma'$, in the regular expression $d(a)$ no two types b^i and b^j with $i \neq j$ occur.*

We denote by EDTD, EDTD(#), EDTD(&), and EDTD(#, &) the classes EDTD(RE), EDTD(RE(#)), EDTD(RE(&)), and EDTD(RE(#, &)), respectively. The same notation is used for EDTD$^{\text{st}}$s and DTDs.

For clarity, we sometimes write $a \to r$ rather than $d(a) = r$ in examples and proofs. Following this notation, a simple example of an EDTD is the following:

shop^1	$\to$	$(\text{cd}^1 + \text{cd}^2)^*\text{cd}^2(\text{cd}^1 + \text{cd}^2)^*$	title^1	$\to$	ε
cd^1	$\to$	$\text{title}^1\ \text{price}^1$	price^1	$\to$	ε
cd^2	$\to$	$\text{title}^1\ \text{price}^1\ \text{discount}^1$	discount^1	$\to$	ε

Here, cd^1 defines ordinary CDs, while cd^2 defines CDs on sale. The rule for shop^1 specifies that there should be at least one CD on sale. Notice that the above EDTD is not a single-type EDTD as cd^1 and cd^2 occur in the same rule.

As explained in [33, 27], EDTDs and single-type EDTDs correspond to Relax NG and XML Schema, respectively.

2.3. Decision problems. The following problems are fundamental to this paper.

DEFINITION 2.2. *Let $\mathcal{M}$ be a class of regular expressions, string automata, or extended DTDs. We define the following problems:*

- INCLUSION *for $\mathcal{M}$: Given two elements $e, e' \in \mathcal{M}$, is $L(e) \subseteq L(e')$?*
- EQUIVALENCE *for $\mathcal{M}$: Given two elements $e, e' \in \mathcal{M}$, is $L(e) = L(e')$?*
- INTERSECTION *for $\mathcal{M}$: Given an arbitrary number of elements $e_1, \ldots, e_n \in \mathcal{M}$, is $\bigcap_{i=1}^n L(e_i) \neq \emptyset$?*
- MEMBERSHIP *for $\mathcal{M}$: Given an element $e \in \mathcal{M}$ and a string or a tree f, is $f \in L(e)$?*

We recall the known results concerning the complexity of REs and EDTDs.

THEOREM 2.3.

(1) INCLUSION, EQUIVALENCE, *and* INTERSECTION *for REs are* PSPACE*-complete* [24, 41].
(2) INCLUSION *and* EQUIVALENCE *for RE(&) and RE(#) are* EXPSPACE*-complete* [31, 32].
(3) INCLUSION *and* EQUIVALENCE *for EDTDst are* PSPACE*-complete* [28]; INTERSECTION *for EDTDst is* EXPTIME*-complete* [28].
(4) INCLUSION, EQUIVALENCE, *and* INTERSECTION *for EDTDs are* EXPTIME*-complete* [37, 38].
(5) MEMBERSHIP *for RE(&) is* NP*-complete* [31].
(6) MEMBERSHIP *for RE(#) is in* PTIME [19].

2.4. Relating decision problems for regular expressions to DTDs and single-type EDTDs. In [28] it was shown for any subclass of the REs that the complexity of INCLUSION and EQUIVALENCE is the same as the complexity of the corresponding problem for DTDs and single-type EDTDs. The same holds for INTERSECTION and DTDs. The proofs of these theorems carry over literally to RE(#, &).

We call a complexity class $\mathcal{C}$ *closed under positive reductions* if the following holds for every $O \in \mathcal{C}$. Let L' be accepted by a deterministic polynomial-time Turing machine M with oracle O (denoted $L' = L(M^O)$). Let M further have the property that $L(M^A) \subseteq L(M^B)$ whenever $L(A) \subseteq L(B)$. Then L' is also in $\mathcal{C}$. For a more precise definition of this notion we refer the reader to [15]. For our purposes, it is sufficient that important complexity classes like PTIME, NP, coNP, PSPACE, and EXPSPACE have this property, and that every such class contains PTIME.

PROPOSITION 2.4 (see [28]). *Let $\mathcal{R}$ be a subclass of $RE(\#, \&)$ and let $\mathcal{C}$ be a complexity class closed under positive reductions. Then the following are equivalent:*

(a) INCLUSION *for $\mathcal{R}$ expressions is in $\mathcal{C}$.*
(b) INCLUSION *for $DTD(\mathcal{R})$ is in $\mathcal{C}$.*
(c) INCLUSION *for $EDTD^{st}(\mathcal{R})$ is in $\mathcal{C}$.*

The corresponding statement holds for EQUIVALENCE.

The previous proposition can be generalized to INTERSECTION of DTDs as well.

PROPOSITION 2.5 (see [28]). *Let $\mathcal{R}$ be a subclass of $RE(\#, \&)$ and let $\mathcal{C}$ be a complexity class which is closed under positive reductions. Then the following are equivalent:*

(a) INTERSECTION *for $\mathcal{R}$ expressions is in $\mathcal{C}$.*
(b) INTERSECTION *for $DTD(\mathcal{R})$ is in $\mathcal{C}$.*

The above proposition does not hold for single-type EDTDs. Indeed, there is a class of regular expressions $\mathcal{R}'$ for which INTERSECTION is NP-complete while INTERSECTION for $\text{EDTD}^{\text{st}}(\mathcal{R}')$ is EXPTIME-complete [28].

3. Automata for occurrence constraints and interleaving. We introduce the automaton model NFA$(\#, \&)$. In brief, an NFA$(\#, \&)$ is an NFA with two additional features: (i) split and merge transitions to handle interleaving; and (ii) counting states and transitions to deal with numerical occurrence constraints. The idea of split and merge transitions stems from Jędrzejowicz and Szepietowski [18]. Their automata are more general as they can express shuffle-closure which is not regular. Counting states are also used in the counter automata of Kilpeläinen and Tuhkanen [20] and Reuter [36] although these counter automata operate quite differently from NFA$(\#)$s. Dal-Zilio and Lugiez [11] also proposed an automaton model that incorporates counting and interleaving by means of Presburger formulas. None of the cited papers considers the complexity of the basic decision problems of their model. We will use NFA$(\#, \&)$s to obtain complexity upper bounds in sections 4 and 5.

For readability, we denote $\Sigma \cup \{\varepsilon\}$ by Σ_ε. We then define an NFA$(\#, \&)$ as follows.

DEFINITION 3.1. *An NFA$(\#, \&)$ is a 5-tuple $A = (Q, \Sigma, s, f, \delta)$, where the following hold:*

- *Q is a finite set of states. To every $q \in Q$, we associate a lower bound $\min(q) \in \mathbb{N}$ and an upper bound $\max(q) \in \mathbb{N}$.*
- *$s, f \in Q$ are the start and final states, respectively.*
- *δ is the transition relation and is a subset of the union of the following sets:*

(1)	$Q \times \Sigma_\varepsilon \times Q$	*ordinary transition (resets the counter)*
(2)	$Q \times \{store\} \times Q$	*transition that does not reset the counter*
(3)	$Q \times \{split\} \times Q \times Q$	*split transition*
(4)	$Q \times Q \times \{merge\} \times Q$	*merge transition*

Let $\max(A) = \max\{\max(q) \mid q \in Q\}$ be the largest upper bound occurring in A. A *configuration* γ is a pair (P, α), where $P \subseteq Q$ is a set of states and $\alpha : Q \to \{0, \ldots, \max(A)\}$ is the value function mapping states to the value of their counter. For a state $q \in Q$, we denote by α_q the value function mapping q to 1 and every

other state to 0. The initial configuration γ_s is $(\{s\}, \alpha_s)$. The final configuration γ_f is $(\{f\}, \alpha_f)$. When α is a value function, $\alpha[q = 0]$ and $\alpha[q^{++}]$ denote the functions obtained from α by setting the value of q to 0 and incrementing the value of q by 1, respectively, while leaving all other values unchanged.

We now define the transition relation between configurations. Intuitively, the value of the state at which the automaton arrives is always incremented by one. When exiting a state, the state's counter is always reset to zero, except when we exit through a *counting transition*, in which case the counter remains the same. In addition, exiting a state through a noncounting transition is allowed only when the value of the counter lies between the allowed minimum and maximum. The latter, hence, ensures that the occurrence constraints are satisfied. *Split* and *merge transitions* start and close a parallel composition.

A configuration $\gamma' = (P', \alpha')$ *immediately follows* a configuration $\gamma = (P, \alpha)$ by reading $\sigma \in \Sigma_\varepsilon$, denoted $\gamma \to_{A,\sigma} \gamma'$, when one of the following conditions hold:

1. *Ordinary transition.* There are a $q \in P$ and $(q, \sigma, q') \in \delta$ such that $\min(q) \leq \alpha(q) \leq \max(q)$, $P' = (P - \{q\}) \cup \{q'\}$, and $\alpha' = \alpha[q = 0][q'^{++}]$. That is, A is in state q and moves to state q' by reading σ (note that σ can be ε). The latter is allowed only when the counter value of q is between the lower and upper bounds. The state q is replaced in P by q'. The counter of q is reset to zero, and the counter of q' is incremented by one.
2. *Counting transition.* There are a $q \in P$ and $(q, \text{store}, q') \in \delta$ such that $\alpha(q) < \max(q)$, $P' = (P - \{q\}) \cup \{q'\}$, and $\alpha' = \alpha[q'^{++}]$. That is, A is in state q and moves to state q' by reading ε when the counter of q has not reached its maximal value yet. The state q is replaced in P by q'. The counter of q is not reset but remains the same. The counter of q' is incremented by one.
3. *Split transition.* There are a $q \in P$ and $(q, \text{split}, q_1, q_2) \in \delta$ such that $\min(q) \leq \alpha(q) \leq \max(q)$, $P' = (P - \{q\}) \cup \{q_1, q_2\}$, and $\alpha' = \alpha[q = 0][q_1{}^{++}][q_2{}^{++}]$. That is, A is in state q and splits into states q_1 and q_2 by reading ε when the counter value of q is between the lower and upper bounds. The state q in P is replaced by (split into) q_1 and q_2. The counter of q is reset to zero, and the counters of q_1 and q_2 are incremented by one.
4. *Merge transition.* There are $q_1, q_2 \in P$ and $(q_1, q_2, \text{merge}, q) \in \delta$ such that, for each $j = 1, 2$, $\min(q_j) \leq \alpha(q_j) \leq \max(q_j)$, $P' = (P - \{q_1, q_2\}) \cup \{q\}$, and $\alpha' = \alpha[q_1 = 0][q_2 = 0][q^{++}]$. That is, A is in states q_1 and q_2 and moves to state q by reading ε when the respective counter values of q_1 and q_2 are between the lower and upper bounds. The states q_1 and q_2 in P are replaced by (merged into) q, the counters of q_1 and q_2 are reset to zero, and the counter of q is incremented by one.

For a string w and two configurations γ, γ', we denote by $\gamma \Rightarrow_{A,w} \gamma'$ when there is a sequence of configurations $\gamma \to_{A,\sigma_1} \cdots \to_{A,\sigma_n} \gamma'$ such that $w = \sigma_1 \cdots \sigma_n$. The latter sequence is called a *run* when γ is the initial configuration γ_s. A string w is *accepted* by A if and only if $\gamma_s \Rightarrow_{A,w} \gamma_f$ with γ_f the final configuration. We usually denote $\Rightarrow_{A,w}$ simply by $\Rightarrow_w$ when A is clear from the context. We denote by $L(A)$ the set of strings accepted by A. The size of A, denoted by $|A|$, is $|Q| + |\delta| + \Sigma_{q \in Q} \log(\max(q))$. Thus, each $\max(q)$ is represented in binary.

An example of an NFA(#, &) defining $dvd^{[10,12]}$ & $cd^{[10,12]}$ is shown in Figure 3.1.

An NFA(#) is an NFA(#, &) without split and merge transitions. An NFA(&) is an NFA(#, &) without counting transitions. An NFA is an NFA(#) without counting transitions.

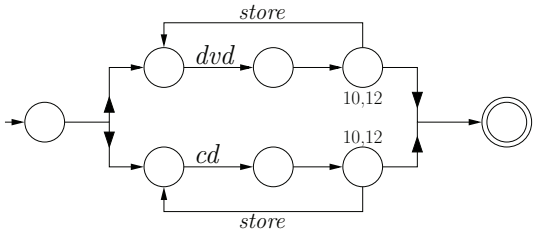


FIG. 3.1. *An NFA(#, &) for the language* $dvd^{[10,12]} \& cd^{[10,12]}$. *For readability, we only displayed the alphabet symbol on nonepsilon transitions and counters for states* q, *where* $\min(q)$ *and* $\max(q)$ *are different from one. The arrows from the initial state and to the final state are split and merge transitions, respectively. The arrows labeled* store *represent counting transitions.*

Clearly NFA(#, &)s accept all regular languages. The next theorem shows the complexity of translating between RE(#, &) and NFA(#, &), and NFA(#, &) and NFA.

THEOREM 3.2.

(1) *Given an RE(#, &) expression* r, *an equivalent NFA(#, &) can be constructed in time linear in the size of* r.
(2) *Given an NFA(#, &)* A, *an equivalent NFA can be constructed in time exponential in the size of* A.

Proof. (1) We prove the theorem by induction on the structure of RE(#, &)-expressions. For every r we define a corresponding NFA(#, &), $A(r) = (Q_r, \Sigma, s_r, f_r, \delta_r)$, such that $L(r) = L(A(r))$.

For r of the form ε, a, $r_1 \cdot r_2$, $r_1 + r_2$, and r_1^* these are the usual RE to NFA with ε-transition constructions as displayed in textbooks such as [16].

We perform the following steps for the numerical occurrence and interleaving operator which are graphically illustrated in Figure 3.2. The construction for the interleaving operator comes from [18].

(i) If $r = (r_1)^{[k,\ell]}$ and $A(r_1) = (Q_1, \Sigma, s_1, f_1, \delta_1)$, then
- $Q_r := Q_{r_1} \uplus \{s_r, f_r, q_r\}$;
- $\min(s_r) = \max(s_r) = \min(f_r) = \max(f_r) = 1$, $\min(q_r) = k$, and $\max(q_r) = \ell$;
- if $k \neq 0$, then $\delta_r := \delta_{r_1} \uplus \{(s_r, \varepsilon, s_{r_1}), (f_{r_1}, \varepsilon, q_r), (q_r, \text{store}, s_{r_1}), (q_r, \varepsilon, f_r)\}$; and,
- if $k = 0$, then $\delta_r := \delta_{r_1} \uplus \{(s_r, \varepsilon, s_{r_1}), (f_{r_1}, \varepsilon, q_r), (q_r, \text{store}, s_{r_1}), (q_r, \varepsilon, f_r), (s_r, \varepsilon, f_r)\}$.

(ii) If $r = r_1 \,\&\, r_2$, $A(r_1) = (Q_{r_1}, \Sigma, s_{r_1}, f_{r_1}, \delta_{r_1})$ and $A(r_2) = (Q_{r_2}, \Sigma, s_{r_2}, f_{r_2}, \delta_{r_2})$, then
- $Q_r := Q_{r_1} \uplus Q_{r_2} \uplus \{s_r, f_r\}$;
- $\min(s_r) = \max(s_r) = \min(f_r) = \max(f_r) = 1$;
- $\delta_r := \delta_{r_1} \uplus \delta_{r_2} \uplus \{(s_r, \text{split}, s_{r_1}, s_{r_2}), (f_{r_1}, f_{r_2}, \text{merge}, f_r)\}$.

Notice that in each step of the construction, a constant number of states are added to the automaton. Moreover, the constructed counters are linear in the size of r. It follows that the size of $A(r)$ is linear in the size of r. The correctness of the construction can easily be proved by induction on the structure of r. □

We next turn to the complexity of the basic decision problems for NFA(#, &).

THEOREM 3.3.

(1) EQUIVALENCE *and* INCLUSION *for NFA(#, &) are* EXPSPACE*-complete;*

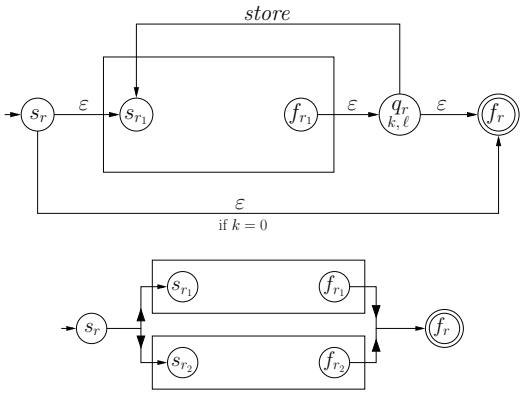


FIG. 3.2. *From RE(#, &) to NFA(#, &).*

(2) INTERSECTION *for NFA(#, &) is* PSPACE*-complete; and*

(3) MEMBERSHIP *for NFA(#) is* NP*-hard, and* MEMBERSHIP *for NFA(&) and NFA(#,&) are* PSPACE*-complete.*

Proof. (1) EXPSPACE-hardness follows from Theorem 3.2(1) and the EXPSPACE-hardness of EQUIVALENCE for RE(&) [31]. Membership in EXPSPACE follows from Theorem 3.2(2) and the fact that INCLUSION for NFAs is in PSPACE [41].

(2) The lower bound follows from [24]. We show that the problem is in PSPACE. For $j \in \{1, \ldots, n\}$, let $A_j = (Q_j, \Sigma, s_j, f_j, \delta_j)$ be an NFA(#, &). The algorithm proceeds by guessing a Σ-string w such that $w \in \bigcap_{j=1}^{n} L(A_j)$. Instead of guessing w at once, we guess it symbol by symbol and keep for each A_j one current configuration γ_j on the tape. More precisely, at each time instant, the tape contains for each A_j a configuration $\gamma_j = (P_j, \alpha_j)$ such that $\gamma_{s_j} \Rightarrow_{A_j, w_i} (P_j, \alpha_j)$, where $w_i = a_1 \cdots a_i$ is the prefix of w guessed up to now. The algorithm accepts when each γ_j is a final configuration. Formally, the algorithm operates as follows.

1. Set $\gamma_j = (\{s_j\}, \alpha_{s_j})$ for $j \in \{1, \ldots, n\}$;
2. While not every γ_j is a final configuration
   (i) Guess an $a \in \Sigma$.
   (ii) Nondeterministically replace each γ_j by a (P'_j, α'_j) such that $(P_j, \alpha_j) \Rightarrow_{A_j, a} (P'_j, \alpha'_j)$.

As the algorithm only uses space polynomial in the size of the NFA(#, &) and step 2(ii) can be done in PSPACE, the overall algorithm operates in PSPACE.

(3) The MEMBERSHIP problem for NFA(#, &)s is easily seen to be in PSPACE by an on-the-fly implementation of the construction in Theorem 3.2(2). Indeed, as a configuration of an NFA(#, &) $A = (Q, \Sigma, s, f, \delta)$ has size at most $|Q| + |Q| \cdot \log(\max(A))$, we can store a configuration using only polynomial space.

We show that the MEMBERSHIP problem for NFA(#)s is NP-hard by a reduction from a modification of INTEGER KNAPSACK. We define this problem as follows. Given a set of natural numbers $W = \{w_1, \ldots, w_k\}$ and two integers m and n, all in binary notation, the problem asks whether there exists a mapping $\tau : W \to \mathbb{N}$ such that $m \leq \sum_{w \in W} \tau(w) \times w \leq n$. The latter mapping is called a solution. This problem is known to be NP-complete [13].

We construct an NFA(#) $A = (Q, \Sigma, s, f, \delta)$ such that $L(A) = \{\varepsilon\}$ if W, m, n has a solution, and $L(A) = \emptyset$ otherwise.

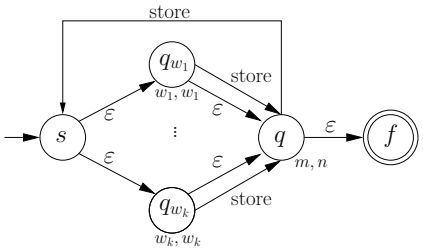


FIG. 3.3. NP-*hardness of* MEMBERSHIP *for NFA(#).*

The state set Q consists of the start and final states s and f, a state q_{w_i} for each weight w_i, and a state q. Intuitively, a successful computation of A loops at least m and at most n times through state q. In each iteration, A also visits one of the states q_{w_i}. Using numerical occurrence constraints, we can ensure that a computation accepts if and only if it passes at least m and at most n times through q and a multiple of w_i times through each q_{w_i}. Hence, an accepting computation exists if and only if there is a mapping τ such that $m \leq \sum_{w \in W} \tau(w) \times w \leq n$.

Formally, the transitions of A are the following:

- $(s, \varepsilon, q_{w_i})$ for each $i \in \{1, \ldots, k\}$;
- $(q_{w_i}, \text{store}, q)$ for each $i \in \{1, \ldots, k\}$;
- $(q_{w_i}, \varepsilon, q)$ for each $i \in \{1, \ldots, k\}$;
- (q, store, s); and,
- (q, ε, f).

We set $\min(s) = \max(s) = \min(f) = \max(f) = 1$, $\min(q) = m$, $\max(q) = n$, and $\min(q_{w_i}) = \max(q_{w_i}) = w_i$ for each q_{w_i}. The automaton is graphically illustrated in Figure 3.3.

Finally, we show that MEMBERSHIP for NFA(&)s is PSPACE-hard. Before giving the proof, we describe some n-ary merge and split transitions which can be rewritten in function of the regular binary split and merge transitions.

1. $(q_1, q_2, \text{merge-split}, q_1', q_2')$: States q_1 and q_2 are read and immediately split into states q_1' and q_2'.
2. $(q_1, q_2, q_3, \text{merge-split}, q_1', q_2', q_3')$: States q_1, q_2 and q_3 are read and immediately split into states q_1', q_2' and q_3'.
3. $(q_1, \text{split}, q_1', \ldots, q_n')$: State q_1 is read and is immediately split into states $q_1', \ldots, q_n'$.
4. $(q_1, \ldots, q_n, \text{merge}, q_1')$: States $q_1, \ldots, q_n$ are read and are merged into state q_1'.

Transitions of type 1 (resp., 2) can be rewritten using 2 (resp., 4) *regular* transitions and 1 (resp., 3) new auxiliary states. Transitions of types 3 and 4 can be rewritten using $(n-1)$ regular transitions and $(n-1)$ new auxiliary states. For example, the transition $(q_1, q_2, \text{merge-split}, q_1', q_2')$ is equal to the transitions $(q_1, q_2, \text{merge}, q_h)$ and $(q_h, \text{split}, q_1', q_2')$, where q_h is a new auxiliary state.

To show that MEMBERSHIP for NFA(&)s is PSPACE-hard, we reduce from CORRIDOR TILING. A *tiling instance* is a tuple $T = (X, H, V, \overline{b}, \overline{t}, n)$, where X is a finite set of tiles, $H, V \subseteq X \times X$ are the horizontal and vertical constraints, n is an integer in unary notation, and $\overline{b}, \overline{t}$ are n-tuples of tiles ($\overline{b}$ and $\overline{t}$ stand for *bottom row* and *top row*, respectively).

A *correct corridor tiling for* T is a mapping $\lambda : \{1, \ldots, m\} \times \{1, \ldots, n\} \to X$ for some $m \in \mathbb{N}$ such that the following constraints are satisfied:

- the bottom row is $\overline{b}$: $\overline{b} = (\lambda(1,1), \ldots, \lambda(1,n))$;
- the top row is $\overline{t}$: $\overline{t} = (\lambda(m,1), \ldots, \lambda(m,n))$;
- all vertical constraints are satisfied: $\forall i < m$, $\forall j \leq n$, $(\lambda(i,j), \lambda(i+1,j)) \in V$; and,
- all horizontal constraints are satisfied: $\forall i \leq m$, $\forall j < n$, $(\lambda(i,j), \lambda(i,j+1)) \in H$.

The CORRIDOR TILING problem asks, given a tiling instance, whether there exists a correct corridor tiling. The latter problem is PSPACE-complete [43].

Given a tiling instance $T = (X, H, V, \overline{b}, \overline{t}, n)$, we construct an NFA(&) A over the empty alphabet ($\Sigma = \emptyset$) which accepts ε if and only if there exists a correct corridor tiling for T.

The automaton constructs the tiling row by row. Therefore, A must at any time reflect the current row in its state set (recall that an NFA(&) can be in more than one state at once). To do this, A contains, for every tile x, a set of states $x^1, \ldots, x^n$, where n is the length of each row. If A is in state x^i, this means that the ith tile of the current row is x. For example, if $\overline{b} = x_1x_3x_1$ and $\overline{t} = x_2x_2x_1$, then the initial state set is $\{x_1^1, x_3^2, x_1^3\}$, and A can accept when the state set is $\{x_2^1, x_2^2, x_1^3\}$.

It remains to define how A can transform the current row ("state set") into a state set which describes a valid row on top of the current row. This transformation proceeds on a tile by tile basis and begins with the first tile, say x_i, in the current row which is represented by x_i^1 in the state set. Now, for every tile x_j for which $(x_i, x_j) \in V$, we allow x_i^1 to be replaced by x_j^1, since x_j can be the first tile of the row on top of the current row. For the second tile of the next row, we have to replace the second tile of the current row, say x_k, by a new tile, say x_ℓ, such that the vertical constraints between x_k and x_ℓ are satisfied and such that the horizontal constraints between x_ℓ and the tile we just placed at the first position of the first row, x_j, are satisfied as well.

The automaton proceeds in this manner for the remainder of the row. For this, the automaton needs to know at any time at which position a tile must be updated. Therefore, an extra set of states $p_1, \ldots, p_n$ is created, where the state p_i says that the tile at position i has to be updated. Thus, the state set always consists of one state p_i and a number of states which represent the current and next rows. Here, the states up to position i already represent the tiles of the next row, the states from position i still represent the current row, and i is the next position to be updated.

We can now formally construct an NFA(&) $A = (Q, \Sigma, s, f, \delta)$ which accepts ε if and only if there exists a correct corridor tiling for a tiling instance $T = (X, H, V, \overline{b}, \overline{t}, n)$ as follows:

- $Q = \{x^j \mid x \in X, 1 \leq j \leq n\} \cup \{p_i \mid 1 \leq i \leq n\} \cup \{s, f\}$.
- $\Sigma = \emptyset$.
- δ is the union of the following transitions:
  - $(s, \text{split}, p_1, \overline{b}_1^1, \ldots, \overline{b}_n^n)$. From the initial state the automaton immediately goes to the states which represent the bottom row.
  - $(p_1, \overline{t}_1^1, \ldots, \overline{t}_n^n, \text{merge}, f)$. When the state set represents a full row (the automaton is in state p_1) and the current row is the accepting row, all states are merged to the accepting state.
  - $\forall x_i, x_j \in X, (x_j, x_i) \in V$: $(p_1, x_j^1, \text{merge-split}, p_2, x_i^1)$. When the first tile has to be updated, the automaton only has to check the vertical constraints with the first tile of the previous row.
  - $\forall x_i, x_j, x_k \in X, m \in \mathbb{N}, 2 \leq m \leq n, (x_k, x_i) \in V, (x_j, x_i) \in H$:

$(p_m, x_k^m, x_j^{m-1}, \text{merge-split}, p_{(m \mod n)+1}, x_i^m, x_j^{m-1})$. When a tile at the mth ($m \neq 1$) position has to be updated, the automaton has to check the vertical constraint with the mth tile at the previous row, and the horizontal constraint with the $(m-1)$th tile of the new row.

Clearly, if there exists a correct corridor tiling for T, there exists a run of A accepting ε. Conversely, the construction of our automaton, in which the updates are always determined by the position p_i and the horizontal and vertical constraints, assures that when there is an accepting run of A on ε, this run simulates a correct corridor tiling for T. □

4. Complexity of regular expressions. Before we turn to schemas, we first deal with the complexity of regular expressions and frequently used subclasses as these are directly related to the complexities of DTDs and single-type EDTDs.

Mayer and Stockmeyer [31] and Meyer and Stockmeyer [32] already established the EXPSPACE-completeness of INCLUSION and EQUIVALENCE for RE(&) and RE(#), respectively. From Theorems 3.2(1) and 3.3(1) it then directly follows that allowing both operators does not increase the complexity. It further follows from Theorems 3.2(1) and 3.3(2) that INTERSECTION for RE(#, &) is in PSPACE. We stress that the latter results could also have been obtained without making use of NFA(#, &)s but by translating RE(#, &)s directly to NFAs. However, in the case of INTERSECTION such a construction should be done in an on-the-fly fashion to not go beyond PSPACE. Although such an approach is certainly possible, we prefer the shorter and more elegant construction using NFA(#, &)s.

THEOREM 4.1.

(1) EQUIVALENCE *and* INCLUSION *for RE(#, &) are in* EXPSPACE*; and*

(2) INTERSECTION *for RE(#, &) is* PSPACE*-complete.*

Proof. (1) The proof follows directly from Theorems 3.2(1) and 3.3(1).

(2) The upper bound follows directly from Theorems 3.2(1) and 3.3(2). The lower bound is already known for ordinary regular expressions. □

Bex, Neven, and Van den Bussche [4] established that the vast majority of regular expressions occurring in practical DTDs and XSDs are of a very restricted form as defined next. The class of *chain regular expressions (CHAREs)* are those REs consisting of a sequence of factors $f_1 \cdots f_n$, where every factor is an expression of the form $(a_1 + \cdots + a_n)$, $(a_1 + \cdots + a_n)?$, $(a_1 + \cdots + a_n)^+$, or $(a_1 + \cdots + a_n)^*$, where $n \geq 1$ and every a_i is an alphabet symbol. For instance, the expression $a(b + c)^* d^+ (e + f)?$ is a CHARE, while $(ab + c)^*$ and $(a^* + b?)^*$ are not.[1]

We introduce some additional notation to define subclasses and extensions of CHAREs. By CHARE(#) we denote the class of CHAREs where factors of the form $(a_1 + \cdots + a_n)^{[k,\ell]}$ also are allowed. For the following fragments, we list the admissible types of factors. Here, a, $a?$, and a^* denote the factors $(a_1 + \cdots + a_n)$, $(a_1 + \cdots + a_n)?$, and $(a_1 + \cdots + a_n)^*$, respectively, with $n = 1$, while $a\#$ denotes $a^{[k,\ell]}$, and $a\#^{>0}$ denotes $a^{[k,\ell]}$ with $k > 0$.

Table 4.1 lists the new and the relevant known results. We first show that adding numerical occurrence constraints to CHAREs increases the complexity of INCLUSION by one exponential. We reduce from EXP-CORRIDOR TILING.

THEOREM 4.2. INCLUSION *for CHARE(#) is* EXPSPACE*-complete.*

Proof. The EXPSPACE upper bound already follows from Theorem 4.1(1).

[1] We disregard here the additional restriction used in [3] that every symbol can occur only once.

TABLE 4.1

Overview of new and known complexity results concerning chain regular expressions. All results are completeness results, unless otherwise mentioned. The new results are printed in bold.

	INCLUSION	EQUIVALENCE	INTERSECTION
CHARE	PSPACE [28]	in PSPACE [41]	PSPACE [28]
CHARE(#)	**EXPSPACE**	**in EXPSPACE**	**PSPACE**
CHARE($a, a?$)	coNP [28]	in PTIME [28]	NP [28]
CHARE(a, a^*)	coNP [28]	in PTIME [28]	NP [28]
CHARE($a, a?, a\#$)	**coNP**	**in PTIME**	**NP**
CHARE($a, a\#^{>0}$)	**in PTIME**	**in PTIME**	**in PTIME**

The proof for the EXPSPACE lower bound is similar to the proof for PSPACE-hardness of INCLUSION for CHAREs in [28]. The main difference is that the numerical occurrence operator allows us to compare tiles over a distance exponential in the size of the tiling instance.

The proof is a reduction from EXP-CORRIDOR TILING. A *tiling instance* is a tuple $T = (X, H, V, x_\bot, x_\top, n)$, where X is a finite set of tiles, $H, V \subseteq X \times X$ are the horizontal and vertical constraints, $x_\bot, x_\top \in X$, and n is a natural number in unary notation. A *correct exponential corridor tiling for* T is a mapping $\lambda : \{1, \ldots, m\} \times \{1, \ldots, 2^n\} \to X$ for some $m \in \mathbb{N}$ such that the following constraints are satisfied:

- the first tile of the first row is $x_\bot$: $\lambda(1,1) = x_\bot$;
- the first tile of the mth row is $x_\top$: $\lambda(m,1) = x_\top$;
- all vertical constraints are satisfied: $\forall i < m, \forall j \leq 2^n, (\lambda(i,j), \lambda(i+1,j)) \in V$; and
- all horizontal constraints are satisfied: $\forall i \leq m, \forall j < 2^n, (\lambda(i,j), \lambda(i,j+1)) \in H$.

The EXP-CORRIDOR TILING problem asks, given a tiling instance, whether there exists a correct exponential corridor tiling. The latter problem is easily shown to be EXPSPACE-complete [43].

We proceed with the reduction from EXP-CORRIDOR TILING. Thereto, let $T = (X, H, V, x_\bot, x_\top, n)$ be a tiling instance. Without loss of generality (w.l.o.g.), we assume that $n \geq 2$. We construct two CHARE(#) expressions r_1 and r_2 such that

$$L(r_1) \subseteq L(r_2) \text{ if and only if } \text{there exists no correct exponential corridor tiling for } T.$$

As EXPSPACE is closed under complement, the EXPSPACE-hardness of INCLUSION for CHARE(#) follows.

Set $\Sigma = X \uplus \{\$, \triangle\}$. For ease of exposition, we denote $X \cup \{\triangle\}$ by $X_\triangle$ and $X \cup \{\triangle, \$\}$ by $X_{\triangle,\$}$. We encode candidates for a correct tiling by a string in which the rows are separated by the symbol $\triangle$, that is, by strings of the form

$$(\dagger) \qquad \triangle R_0 \triangle R_1 \triangle \cdots \triangle R_m \triangle,$$

in which each R_i represents a row, that is, belongs to X^{2^n}. Moreover, R_0 is the bottom row, and R_m is the top row. The following regular expressions detect strings of this form which do not encode a correct tiling for T:

- $X_\triangle^* \triangle X^{[0,2^n-1]} \triangle X_\triangle^*$. This expression detects rows that are too short, that is, contain fewer than 2^n symbols.

- $X^*_\triangle \triangle X^{[2^n+1,2^n+1]} X^* \triangle X^*_\triangle$. This expression detects rows that are too long, that is, contain more than 2^n symbols.
- $X^*_\triangle x_1 x_2 X^*_\triangle$ for every $x_1, x_2 \in X$, $(x_1, x_2) \notin H$. These expressions detect all violations of horizontal constraints.
- $X^*_\triangle x_1 X^{[2^n,2^n]}_\triangle x_2 X^*_\triangle$ for every $x_1, x_2 \in X$, $(x_1, x_2) \notin V$. These expressions detect all violations of vertical constraints.

Let $e_1, \ldots, e_k$ be an enumeration of the above expressions. Notice that $k = \mathcal{O}(|X|^2)$. It is straightforward that a string w in (†) does not match $\bigcup_{i=1}^k e_i$ if and only if w encodes a correct tiling.

Let $e = e_1 \cdots e_k$. Because of leading and trailing $X^*_\triangle$ expressions, $L(e) \subseteq L(e_i)$ for every $i \in \{1, \ldots, k\}$. We are now ready to define r_1 and r_2:

$$r_1 = \overbrace{\$e\$e\$\cdots\$e\$}^{k \text{ times } e} \triangle x_\perp X^{[2^n-1,2^n-1]} \triangle X^*_\triangle \triangle x_\top X^{[2^n-1,2^n-1]} \triangle \overbrace{\$e\$e\$\cdots\$e\$}^{k \text{ times } e},$$

$$r_2 = \$X^*_{\triangle,\$} \$e_1\$e_2\$\cdots\$e_k\$ X^*_{\triangle,\$}\$.$$

Notice that both r_1 and r_2 are in CHARE(#) and can be constructed in polynomial time. It remains to show that $L(r_1) \subseteq L(r_2)$ if and only if there is no correct tiling for T.

We first show the implication from left to right. Thereto, let $L(r_1) \subseteq L(r_2)$. Let uwu' be an arbitrary string in $L(r_1)$ such that $u, u' \in L(\$e\$e\$\cdots\$e\$)$ and $w \in \triangle x_\perp X^{[2^n-1,2^n-1]} \triangle X^*_\triangle \triangle x_\top X^{[2^n-1,2^n-1]} \triangle$. By assumption, $uwu' \in L(r_2)$.

Notice that uwu' contains $2k+2$ times the symbol "\$." Moreover, the first and the last "\$" of uwu' are always matched onto the first and last "\$" of r_2. This means that $k+1$ consecutive \$-symbols of the remaining $2k$ \$-symbols in uwu' must be matched onto the \$-symbols in $\$e_1\$e_2\$\cdots\$e_k\$$. Hence, w is matched onto some e_i. Thus, w does not encode a correct tiling. As the subexpression $\triangle x_\perp X^{[2^n-1,2^n-1]} \triangle X^*_\triangle \triangle x_\top X^{[2^n-1,2^n-1]} \triangle$ of r_1 defines all candidate tilings, the system T has no solution.

To show the implication from right to left, assume that there is a string $uwu' \in L(r_1)$ that is not in r_2, where $u, u' \in L(\$e\$e\$\cdots\$e\$)$. Then $w \notin \bigcup_{i=1}^k L(e_i)$, and, hence, w encodes a correct tiling. □

Adding numerical occurrence constraints to the fragment CHARE($a, a?$) keeps EQUIVALENCE in PTIME, INTERSECTION in NP, and INCLUSION in coNP.

THEOREM 4.3.

(1) EQUIVALENCE *for* CHARE($a, a?, a\#$) *is in* PTIME.
(2) INCLUSION *for* CHARE($a, a?, a\#$) *is* coNP*-complete.*[2]
(3) INTERSECTION *for* CHARE($a, a?, a\#$) *is* NP*-complete.*

Proof. (1) It is shown in [28] that two CHARE($a, a?$) expressions are equivalent if and only if they have the same *sequence normal form* (which is defined below). As $a^{[k,\ell]}$ is equivalent to $a^k(a?)^{\ell-k}$, it follows that two CHARE($a, a?, a\#$) expressions are equivalent if and only if they have the same sequence normal form. It remains to argue that the sequence normal form of CHARE($a, a?, a\#$) expressions can be computed in polynomial time. To this end, let $r = f_1 \cdots f_n$ be a CHARE($a, a?, a\#$) expression with factors $f_1, \ldots, f_n$. The *sequence normal form* is then obtained in the following way. First, we replace every factor of the form

- a by $a[1,1]$;
- $a?$ by $a[0,1]$; and
- $a^{[k,\ell]}$ by $a[k,\ell]$,

[2] In the previous version of this article presented at ICDT'07 the complexity was wrongly attributed to lie between PSPACE and EXPSPACE.

where a is an alphabet symbol. We call a the *base symbol* of the factor $a[i,j]$. Then, we replace successive subexpressions $a[i_1,j_1]$ and $a[i_2,j_2]$ carrying the same base symbol a by $a[i_1+i_2, j_1+j_2]$ until no further replacements can be made. For instance, the sequence normal form of $aa?a^{[2,5]}a?bb?b?b^{[1,7]}$ is $a[3,8]b[2,10]$. Obviously, the above algorithm to compute the sequence normal form of CHARE$(a, a?, a\#)$ expressions can be implemented in polynomial time. It can then be tested in linear time whether two sequence normal forms are the same.

(2) coNP-hardness is immediate since INCLUSION is already coNP-complete for CHARE$(a, a?)$ expressions [28].

We show that the problem remains in coNP. To this end, we represent strings w by their sequence normal form as discussed above, where we take each string w as the regular expression defining w. We call such strings *compressed*. Let r_1 and r_2 be two CHARE$(a, a?, a\#)$s. We can assume that they are in sequence normal form.

To show that $L(r_1) \not\subseteq L(r_2)$, we guess a compressed string w of polynomial size for which $w \in L(r_1)$, but $w \notin L(r_2)$. We guess $w \in L(r_1)$ in the following manner. We iterate from left to right over the factors of r_1. For each factor $a[k,\ell]$ we guess an h such that $k \leq h \leq \ell$, and add a^h to the compressed string w. This algorithm gives a compressed string of polynomial size which is defined by r_1. Furthermore, this algorithm is capable of guessing every possible string defined by r_1. It is, however, possible that in the compressed string there are two consecutive *elements* a^i, a^j with the same *base symbol* a. If this is the case we merge these elements to a^{i+j} which gives a proper compressed string.

The following lemma shows that testing $w \notin L(r_2)$ can be done in PTIME.

LEMMA 4.4. *Given a compressed string w and an expression r in sequence normal form, deciding whether $w \in L(r)$ is in* PTIME.

Proof. Let $w = a_1^{p_1} \cdots a_n^{p_n}$, and let $r = b_1[k_1,\ell_1] \cdots b_m[k_m,\ell_m]$. Denote $b_i[k_i,\ell_i]$ by f_i. For every position i of w $(0 < i \leq n)$, we define C_i as a set of factors $b[k,\ell]$ of r. Formally, $f_j \in C_i$ when $a_1^{p_1} \cdots a_{i-1}^{p_{i-1}} \in L(f_1 \cdots f_{j-1})$ and $a_i = b_j$. We compute the C_i as follows.

- C_1 is the set of all $b_j[k_j,\ell_j]$ such that $a_1 = b_j$ and $\forall h < j, k_h = 0$. These are all the factors of r which can match the first symbol of w.
- Then, $\forall\ i \in \{2,\ldots,n\}$, we compute C_i from C_{i-1}. In particular, $f_h = b_h[k_h,\ell_h] \in C_i$ when there is an $f_j = b_j[k_j,\ell_j] \in C_{i-1}$ such that $a_{i-1}^{p_{i-1}} \in f_j \cdots f_{h-1}$ and $a_i = b_h$. That is, the following conditions should hold:
  - $j < h$: f_h occurs after f_j in r.
  - $b_h = a_i$: f_h can match the first symbol of $a_i^{p_i}$.
  - $\forall e \in \{j,\ldots,h-1\}$, if $b_e \neq a_{i-1}$, then $k_e = 0$: in between factors f_j and f_h it is possible to match only symbols a_{i-1}.
  - Let $\min = \sum_{e\in\{j,\ldots,h-1\}, b_e = a_{i-1}} k_e$ and $\max = \sum_{e\in\{j,\ldots,h-1\}, b_e = a_{i-1}} \ell_e$. Then $\min \leq p_{i-1} \leq \max$. That is, p_{i-1} symbols a_{i-1} should be matched from f_j to f_{h-1}.

Then, $w \in L(r)$ if and only if there is an $f_j \in C_n$ such that $a_n^{p_n} \in L(f_j \cdots f_n)$. As the latter test and the computation of $C_1,\ldots,C_n$ can be done in PTIME, the lemma follows. □

(3) NP-hardness is immediate since INTERSECTION is already NP-complete for CHARE$(a, a?)$ expressions [28].

We show that the problem remains in NP. As in the proof of Theorem 4.3(2) we represent a string w as a compressed string. Let $r_1,\ldots,r_n$ be CHARE$(a, a?, a\#)$ expressions.

LEMMA 4.5. *If* $\bigcap_{i=1}^{n} L(r_i) \neq \emptyset$, *then there exists a string* $w = a_1^{p_1} \cdots a_m^{p_m} \in \bigcap_{i=1}^{n} L(r_i)$ *such that* $m \leq \min\{|r_i| \mid i \in \{1, \ldots, n\}\}$ *and, for each* $i \in \{1, \ldots, n\}$, j_i *is not larger than the largest integer occurring in* $r_1, \ldots, r_n$.

Proof. Suppose that there exists a string $w = a_1^{p_1} \cdots a_m^{p_m} \in \bigcap_{i=1}^{n} L(r_i)$, with $a_i \neq a_{i+1}$ for every $i \in \{1, \ldots, m-1\}$. Since w is matched by every expression $r_1, \ldots, r_n$, and since a factor of a CHARE$(a, a?, a\#)$ expression can never match a strict superstring of $a_i^{p_i}$ for $i \in \{1, \ldots, n\}$, we have that $m \leq \min\{|r_i| \mid i \in \{1, \ldots, n\}\}$.

Furthermore, since w is matched by every expression $r_1, \ldots, r_n$, no j_i can be larger than the largest integer occurring in $r_1, \ldots, r_n$. ∎

The NP algorithm then consists of guessing a compressed string w of polynomial size and verifying whether $w \in \bigcap_{i=1}^{n} L(r_i)$. If we represent $r_1, \ldots, r_n$ by their sequence normal form, this verification step can be done in polynomial time by Lemma 4.4. ∎

Finally, we exhibit a tractable subclass with numerical occurrence constraints.

THEOREM 4.6. INCLUSION, EQUIVALENCE, *and* INTERSECTION *for CHARE*$(a, a\#^{>0})$ *are in* PTIME.

Proof. The upper bound for EQUIVALENCE is immediate from Theorem 4.3(2).

For INCLUSION, let r_1 and r_2 be two CHARE$(a, a\#^{>0})$s in sequence normal form (as defined in the proof of Theorem 4.3). Let $r_1 = a_1[k_1, \ell_1] \cdots a_n[k_n, \ell_n]$ and $r_2 = a'_1[k'_1, \ell'_1] \cdots a'_{n'}[k'_{n'}, \ell'_{n'}]$. Notice that every number k_i and k'_j is greater than zero. We claim that $L(r_1) \subseteq L(r_2)$ if and only if $n = n'$ and for every $i \in \{1, \ldots, n\}$, $a_i = a'_i$, $k_i \geq k'_i$, and $\ell_i \leq \ell'_i$.

Indeed, if $n \neq n'$, or if there exists an i such that $a_i \neq a'_i$ or $k_i < k'_i$, then $a_1^{k_1} \cdots a_n^{k_n} \in L(r_1) - L(r_2)$. If there exists an i such that $\ell_i > \ell'_i$, then $a_1^{\ell_1} \cdots a_n^{\ell_n} \in L(r_1) - L(r_2)$. Conversely, it is immediate that every string in $L(r_1)$ is also in $L(r_2)$. It is straightforward to test these conditions in linear time.

For INTERSECTION, let, for every $i \in \{1, \ldots, n\}$, $r_i = a_{i,1}[k_{i,1}, \ell_{i,1}] \cdots a_{i,m_i}[k_{i,m_i}, \ell_{i,m_i}]$ be a CHARE$(a, a\#^{>0})$ in sequence normal form. Notice that every number $k_{i,j}$ is greater than zero. We claim that $\bigcap_{i=1}^{n} L(r_i) \neq \emptyset$ if and only if

(i) $m_1 = m_2 = \cdots = m_n$;
(ii) for every $i, j \in \{1, \ldots, n\}$ and $x \in \{1, \ldots, m_1\}$, $a_{i,x} = a_{j,x}$; and
(iii) for every $x \in \{1, \ldots, m_1\}$, $\max\{k_{i,x} \mid 1 \leq i \leq n\} \leq \min\{\ell_{i,x} \mid 1 \leq i \leq n\}$.

Indeed, if the above conditions hold, we have that $a_{1,1}^{K_1} \cdots a_{1,m_1}^{K_{m_1}}$ is in $\bigcap_{i=1}^{n} L(r_i)$, where $K_x = \max\{k_{i,x} \mid 1 \leq i \leq n\}$ for every $x \in \{1, \ldots, m_1\}$. If $m_i \neq m_j$ for some $i, j \in \{1, \ldots, n\}$, then the intersection between r_i and r_j is empty. So assume that condition (i) holds. If $a_{i,x} \neq a_{j,x}$ for some $i, j \in \{1, \ldots, n\}$ and $x \in \{1, \ldots, m_1\}$, then we also have that the intersection between r_i and r_j is empty. Finally, if condition (iii) does not hold, take i, j, and x such that $k_{i,x} = \max\{k_{i,x} \mid 1 \leq i \leq n\}$ and $\ell_{j,x} = \min\{\ell_{i,x} \mid 1 \leq i \leq n\}$. Then the intersection between r_i and r_j is empty.

Finally, testing conditions (i)–(iii) can be done in linear time. ∎

5. Complexity of schemas.

5.1. DTDs and single-type EDTDs. By Proposition 2.4 the results on the EQUIVALENCE and INCLUSION problem of the previous section carry over to DTDs and single-type EDTDs. For the INTERSECTION problem, the results carry over only to DTDs (Proposition 2.5). The only remaining problem is INTERSECTION for single-type EDTDs with counting and interleaving. However, INTERSECTION for EDTD$^{\text{st}}$(RE) is EXPTIME-hard, and in the next section we will see that even for EDTD$(\#, \&)$ INTERSECTION remains in EXPTIME. It immediately follows that INTERSECTION for EDTD$^{\text{st}}(\#)$, EDTD$^{\text{st}}(\&)$, and EDTD$^{\text{st}}(\#,\&)$ is also EXPTIME-complete.

5.2. Extended DTDs. We next consider the complexity of the basic decision problems for EDTDs with numerical occurrence constraints and interleaving. As the basic decision problems are EXPTIME-complete for EDTD(RE), the straightforward approach of translating every RE(#, &) expression into an NFA and then applying the standard algorithms gives rise to a double exponential time complexity. By using NFA(#, &), we can do better: EXPSPACE for INCLUSION and EQUIVALENCE and, more surprisingly, EXPTIME for INTERSECTION.

THEOREM 5.1.

(1) EQUIVALENCE *and* INCLUSION *for EDTD*(#, &) *are in* EXPSPACE*;*
(2) EQUIVALENCE *and* INCLUSION *for EDTD*(#) *and EDTD*(&) *are* EXPSPACE*-hard; and*
(3) INTERSECTION *for EDTD*(#, &) *is* EXPTIME*-complete.*

Proof. (1) We show that INCLUSION is in EXPSPACE. The upper bound for EQUIVALENCE then immediately follows.

First, we introduce some notation. For an EDTD $D = (\Sigma, \Sigma', d, s, \mu)$, we will denote elements of Σ', i.e., types, by τ. We denote by (D, τ) the EDTD D with start symbol τ. We define the *depth* of a tree t, denoted by $\text{depth}(t)$, as follows: if $t = \varepsilon$, then $\text{depth}(t) = 0$, and if $t = \sigma(t_1 \cdots t_n)$, then $\text{depth}(t) = \max\{\text{depth}(t_i) \mid i \in \{1, \ldots, n\}\} + 1$.

Suppose that we have two EDTDs $D_1 = (\Sigma, \Sigma'_1, d_1, s_1, \mu_1)$ and $D_2 = (\Sigma, \Sigma'_2, d_2, s_2, \mu_2)$. We provide an EXPSPACE algorithm that decides whether $L(D_1) \not\subseteq L(D_2)$. As EXPSPACE is closed under complement, the theorem follows. The algorithm computes a set E of pairs $(C_1, C_2) \in 2^{\Sigma'_1} \times 2^{\Sigma'_2}$, where $(C_1, C_2) \in E$ if and only if there exists a tree t such that $C_j = \{\tau \in \Sigma'_j \mid t \in L((D_j, \tau))\}$ for each $j = 1, 2$. That is, every C_j is the set of types that can be assigned by D_j to the root of t. Or, when viewing D_j as a tree automaton, C_j is the set of states that can be assigned to the root in a run on t. Therefore, we say that t *is a witness for* (C_1, C_2). Notice that $t \in L(D_1)$ (resp., $t \in L(D_2)$) if $s_1 \in C_1$ (resp., $s_2 \in C_2$). Hence, $L(D_1) \not\subseteq L(D_2)$ if and only if there exists a pair $(C_1, C_2) \in E$ with $s_1 \in C_1$ and $s_2 \notin C_2$.

We compute the set E in a bottom-up manner as follows:

1. Initially, set $E_1 := \{(C_1, C_2) \mid \exists a \in \Sigma, \tau_1 \in \Sigma'_1, \tau_2 \in \Sigma'_2$ such that $\mu_1(\tau_1) = \mu_2(\tau_2) = a$ and, for $i = 1, 2$, $C_i = \{\tau \in \Sigma'_i \mid \varepsilon \in d_i(\tau) \wedge \mu_i(\tau) = a\}\}$.
2. For every $k > 1$, E_k is the union of E_{k-1} and the pairs (C_1, C_2) for which there are $a \in \Sigma$, $n \in \mathbb{N}$, and a string $(C_{1,1}, C_{2,1}) \cdots (C_{1,n}, C_{2,n})$ in E^*_{k-1} such that

$$C_j = \{\tau \in \Sigma'_j \mid \mu_j(\tau) = a, \exists b_{j,1} \in C_{j,1}, \ldots, b_{j,n} \in C_{j,n} \text{ with } b_{j,1} \cdots b_{j,n} \in d_j(\tau)\} \text{ for each } j = 1, 2.$$

Let $E := E_\ell$ for $\ell = 2^{|\Sigma'_1|} \cdot 2^{|\Sigma'_2|}$. The algorithm then accepts when there is a pair $(C_1, C_2) \in E$ with $s_1 \in C_1$ and $s_2 \notin C_2$ and rejects otherwise.

We argue that the algorithm is correct. As $E_k \subseteq E_{k+1}$, for every k, it follows that $E_\ell = E_{\ell+1}$. Hence, the algorithm computes the largest set of pairs. The following lemma then shows that the algorithm decides whether $L(D_1) \not\subseteq L(D_2)$. The lemma can be proved by induction on k.

LEMMA 5.2. *For every* $k \geq 1$, $(C_1, C_2) \in E_k$ *if and only if there exists a witness tree for* (C_1, C_2) *of depth at most* k.

It remains to show that the algorithm can be carried out using exponential space. Step 1 reduces to a linear number of tests $\varepsilon \in L(r)$ for some RE(#, &) expressions r

which is in PTIME by [19]. For step 2, it suffices to argue that, when E_{k-1} is known, it is decidable in EXPSPACE whether a pair (C_1, C_2) is in E_k. As there are only an exponential number of such possible pairs, the result follows. To this end, we need to verify that there exists a string $W = (C_{1,1}, C_{2,1}) \cdots (C_{1,n}, C_{2,n})$ in E_{k-1}^* such that, for each $j = 1, 2$,

(A) for every $\tau \in C_j$, there exist $b_{j,1} \in C_{j,1}, \ldots, b_{j,n} \in C_{j,n}$ with $b_{j,1} \cdots b_{j,n} \in d_j(\tau)$; and
(B) for every $\tau \in \Sigma'_j \setminus C_j$, there do *not* exist $b_{j,1} \in C_{j,1}, \ldots, b_{j,n} \in C_{j,n}$ with $b_{j,1} \cdots b_{j,n} \in d_j(\tau)$.

Assume that $\Sigma'_1 \cap \Sigma'_2 = \emptyset$. Let, for each $j = 1, 2$ and $\tau \in \Sigma'_j$, $N(\tau)$ be the NFA(#, &) accepting $d_j(\tau)$. Intuitively, we guess the string W one symbol at a time and compute the set of reachable configurations Γ_τ for each $N(\tau)$.

Initially, Γ_τ is the singleton set containing the initial configuration of $N(\tau)$. Suppose that we have guessed a prefix $(C_{1,1}, C_{2,1}) \cdots (C_{1,m-1}, C_{2,m-1})$ of W and that we guess a new symbol $(C_{1,m}, C_{2,m})$. Then, we compute the set $\Gamma'_\tau = \{\gamma' \mid \exists b \in C_{j,m}, \gamma \in \Gamma_\tau \text{ such that } \gamma \Rightarrow_{N(\tau),b} \gamma'\}$ and set Γ_τ to Γ'_τ. Each set Γ'_τ can be computed in exponential space from Γ_τ. We accept (C_1, C_2) when, for every $\tau \in \Sigma'_j$, $\tau \in C_j$ if and only if Γ_τ contains an accepting configuration.

(2) It is shown by Mayer and Stockmeyer [31] and Meyer and Stockmeyer [32] that EQUIVALENCE and INCLUSION are EXPSPACE-hard for RE(&)s and RE(#), respectively. Hence, EQUIVALENCE and INCLUSION are also EXPSPACE-hard for EDTD(&) and EDTD(#).

(3) The lower bound follows from [38]. We argue that the problem is in EXPTIME. Thereto, let, for each $i \in \{1, \ldots, n\}$, $D_i = (\Sigma, \Sigma'_i, d_i, s_i, \mu_i)$ be an EDTD(#, &). We assume w.l.o.g. that the sets Σ'_i are pairwise disjoint. We also assume that the start type s_i never appears at the right-hand side of a rule. Finally, we assume that no derivation tree consists of only the root. For each type $\tau \in \Sigma'_i$, let $N(\tau)$ denote an NFA(#, &) for $d_i(\tau)$. According to Theorem 3.2, $N(\tau)$ can be computed from $d_i(\tau)$ in polynomial time. We provide an alternating polynomial space algorithm that guesses a tree t and accepts if $t \in L(D_1) \cap \cdots \cap L(D_n)$. As APSPACE = EXPTIME [8], this shows the theorem.

We guess t node by node in a top-down manner. For every guessed node v, the following information is written on the tape of the TM: for every $i \in \{1, \ldots, n\}$, the triple $c_i = (\tau_v^i, \tau_p^i, \gamma^i)$, where τ_v^i is the type assigned to v by grammar D_i, τ_p^i is the type of the parent assigned by D_i, and γ^i is the current configuration of $N(\tau_p^i)$ after reading the string formed by the left siblings of v. In the following, we say that $\tau \in \Sigma'_i$ is an a-type when $\mu_i(\tau) = a$.

The algorithm proceeds as follows:

1. As for each grammar the types of the roots are given, we start by guessing the first child of the root. That is, we guess an $a \in \Sigma$, and, for each $i \in \{1, \ldots, n\}$, we guess an a-type τ^i and write the triple $c_i = (\tau^i, s_i, \gamma_s^i)$ on the tape where γ_s^i is the start configuration of $N(s_i)$.
2. For $i \in \{1, \ldots, n\}$, let $c_i = (\tau^i, \tau_p^i, \gamma^i)$ be the triples on the tape. The algorithm now universally splits into two parallel branches as follows:
   (a) *Downward extension.* When for every i, $\varepsilon \in d_i(\tau^i)$, the current node can be a leaf node and the branch accepts. Otherwise, guess an $a \in \Sigma$ and for each i, guess an a-type θ^i. Replace every c_i by the triple $(\theta^i, \tau^i, \gamma_s^i)$ and proceed to step 2. Here, γ_s^i is the start configuration of $N(\tau^i)$.

(b) *Extension to the right.* For every $i \in \{1, \ldots, n\}$, compute a configuration γ'^i for which $\gamma^i \Rightarrow_{N(\tau_p^i),\tau^i} \gamma'^i$. When every γ'^i is a final configuration, we do not need to extend to the right anymore and the algorithm accepts. Otherwise, guess an $a \in \Sigma$ and for each i, guess an a-type θ^i. Replace every c_i by the triple $(\theta^i, \tau^i, \gamma'^i)$ and proceed to step 2.

We argue that the algorithm is correct. If the algorithm accepts, we have guessed a tree t and, for every $i \in \{1, \ldots, n\}$, a tree t'_i with $\mu_i(t'_i) = t$ and $t'_i \in L(d_i)$. Therefore, $t \in \bigcap_{i=1}^n L(D_i)$. For the other direction, suppose that there exists a tree $t \in \bigcap_{i=1}^n L(D_i)$ and t is minimal in the sense that no subtree t_0 of t is in $\bigcap_{i=1}^n L(D_i)$. Then, there is a run of the above algorithm that guesses t and guesses trees t'_i with $\mu_i(t'_i) = t$. The tree t must be minimal since the algorithm stops extending the tree as soon as possible.

The algorithm obviously uses only polynomial space. □

6. Simplification. The simplification problem is defined as follows: Given an EDTD, check whether it has an equivalent EDTD of a restricted type, i.e., an equivalent DTD or single-type EDTD. In [27], this problem was shown to be EXPTIME-complete for EDTDs with standard regular expressions. We revisit this problem in the context of RE(#, &).

We need a bit of terminology. Let t be a tree and v be a node. By anc-str$^t(v)$ we denote the string formed by the labels on the path from the root to v, i.e., $\text{lab}^t(\varepsilon)\text{lab}^t(i_1)\text{lab}^t(i_1 i_2)\cdots\text{lab}^t(i_1 i_2 \cdots i_k)$, where $v = i_1 i_2 \cdots i_k$.

We say that a tree language L is *closed under ancestor-guarded subtree exchange* if the following holds: whenever for two trees $t_1, t_2 \in L$ with nodes $u_1 \in \text{Dom}(t_1)$ and $u_2 \in \text{Dom}(t_2)$, anc-str$^{t_1}(u_1) =$ anc-str$^{t_2}(u_2)$ implies $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in L$. Here, $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)]$ denotes the tree obtained from t_1 by replacing its subtree rooted at u_1 by the subtree rooted at u_2 in t_2.

We recall the following theorem from [27].

THEOREM 6.1 (Theorem 7.1 in [27]). *Let L be a tree language defined by an EDTD. Then the following conditions are equivalent.*

(a) *L is definable by a single-type EDTD.*

(b) *L is closed under ancestor-guarded subtree exchange.*

We are now ready for the following theorem.

THEOREM 6.2. *Given an EDTD(#, &), deciding whether it is equivalent to an EDTDst(#, &) or DTD(#, &) is* EXPSPACE*-complete.*

Proof. We first show that the problem is hard for EXPSPACE. We use a reduction from EQUIVALENCE of RE(#), which is EXPSPACE-complete [32].

Let r_1, r_2 be RE(#) expressions over Σ and let b and s be two symbols not occurring in Σ. By definition $L(r_j) \neq \emptyset$, for $j = 1, 2$. Define $D = (\Sigma \cup \{b, s\}, \Sigma \cup \{s, b^1, b^2\}, d, s, \mu)$ as the EDTD with the following rules:

$$\begin{array}{lcl} s & \to & b^1 b^2, \\ b^1 & \to & r_1, \\ b^2 & \to & r_2, \end{array}$$

where for every $\tau \in \Sigma \cup \{s\}$, $\mu(\tau) = \tau$ and $\mu(b^1) = \mu(b^2) = b$. We claim that D is equivalent to a single-type DTD or a DTD if and only if $L(r_1) = L(r_2)$. Clearly, if r_1 is equivalent to r_2, then D is equivalent to the DTD (and therefore also to a single-type EDTD)

$$\begin{array}{lcl} s & \to & bb, \\ b & \to & r_1. \end{array}$$

Conversely, suppose that there exists an EDTD$^{\text{st}}$ which defines the language $L(D)$. Toward a contradiction, assume that r_1 is not equivalent to r_2. Thus, there exists a string w_1 such that $w_1 \in L(r_1)$ and $w_1 \notin L(r_2)$, or $w_1 \notin L(r_1)$ and $w_1 \in L(r_2)$. We consider only the first case; the second is identical. Now, let w_2 be a string in $L(r_2)$ and consider the tree $t = s(b(w_1)b(w_2))$. Clearly, t is in $L(D)$. However, the tree $t' = s(b(w_2)b(w_1))$ obtained from t by switching its left and right subtree is not in $L(D)$. According to Theorem 6.1, every tree language defined by a single-type EDTD is closed under such an exchange of subtrees. So, this means that $L(D)$ cannot be defined by an EDTD$^{\text{st}}$, which leads to the desired contradiction.

We now proceed with the upper bounds. The following algorithms are along the same lines as the EXPTIME algorithms in [27] for the simplification problem without numerical occurrence or interleaving operators. We first give an EXPSPACE algorithm which decides whether an EDTD is equivalent to an EDTD$^{\text{st}}$. Let $D = (\Sigma, \Sigma', d, s, \mu)$ be an EDTD. Intuitively, we compute an EDTD$^{\text{st}}$ $D_0 = (\Sigma, \Sigma'_0, d_0, s, \mu_0)$ which is the closure of D under the single-type property. The EDTD$^{\text{st}}$ D_0 has the following properties:

(a) Σ'_0 is in general exponentially larger than Σ';
(b) the RE$(\#, \&)$ expressions in the definition of d_0 are only polynomially larger than the RE$(\#, \&)$ expressions in the definition of d;
(c) $L(D) \subseteq L(D_0)$; and
(d) $L(D_0) = L(D) \Leftrightarrow D$ is equivalent to an EDTD$^{\text{st}}$.

Hence, we have that D is equivalent to an EDTD$^{\text{st}}$ if and only if $L(D_0) \subseteq L(D)$.

We first show how D_0 can be constructed. We can assume w.l.o.g. that, for each type $a^i \in \Sigma'$, there exists a tree $t' \in L(d)$ such that a^i is a label in t'. Indeed, every *useless* type can be removed from D in a simple preprocessing step. Then, for a string $w \in \Sigma^*$ and $a \in \Sigma$, let types(wa) be the set of all types $a^i \in \Sigma'$ for which there are a tree t and a tree $t' \in L(d)$ with $\mu(t') = t$, and a node v in t such that anc-str$^t(v) = wa$ and the type of v in t' is a^i. We show how to compute types(wa) in exponential time. To this end, we enumerate all sets types(w). Let $s = c^1$. Initially, set $W := \{c\}$, Types$(c) := \{c^1\}$, and $R := \{\{c^1\}\}$. Repeat the following until W becomes empty:

(1) Remove a string wa from W.
(2) For every $b \in \Sigma$, let Types(wab) contain all b^i for which there exist an a^j in Types(wa) and a string in $d(a^j)$ containing b^i. If Types(wab) is not empty and not already in R, then add it to R and add wab to W.

Since we add every set only once to R, the algorithm runs in time exponential in the size of D. Moreover, we have that Types$(w) =$ types(w) for every w and that $R = \Sigma'_0$.

For each $a \in \Sigma$, let types(D, a) be the set of all nonempty sets types(wa), with $w \in \Sigma^*$. Clearly, each types(D, a) is finite. We next define $D_0 = (\Sigma, \Sigma'_0, d_0, s, \mu_0)$. Its set of types is $\Sigma'_0 := \bigcup_{a \in \Sigma}$ types(D, a). Note that $s \in \Sigma'_0$. For every $\tau \in$ types(D, a), set $\mu_0(\tau) = a$. In d_0, the right-hand side of the rule for each types(wa) is the disjunction of all $d(a^i)$ for $a^i \in$ types(wa), with each b^j in $d(a^i)$ replaced by types(wab).

We show that properties (a)–(d) hold. Since $\Sigma'_0 \subseteq 2^{\Sigma'}$, we immediately have that (a) holds. The RE$(\#, \&)$ expressions that we constructed in D_0 are unions of a linear number of RE$(\#, \&)$ expressions in D, but have types in $2^{\Sigma'}$ rather than in Σ'. Hence, the size of the RE$(\#, \&)$ expressions in D_0 is at most quadratic in the size of D. Finally, we note that it has been shown in Theorem 7.1 in [27] that (c) and (d) also hold.

It remains to argue that it can be decided in EXPSPACE that $L(D_0) \subseteq L(D)$. A direct application of the EXPSPACE algorithm in Theorem 5.1(1) leads to a 2EX-

PSPACE algorithm to test whether $L(D_0) \subseteq L(D)$, due to the computation of C_1. Indeed, the algorithm remembers, given the EDTDs $D_0 = (\Sigma, \Sigma'_0, d_0, s_0, \mu_0)$ and $D = (\Sigma, \Sigma', d, s, \mu)$, all possible pairs (C_1, C_2) such that there exists a tree t with $C_1 = \{\tau \in \Sigma'_0 \mid t \in L((D_0, \tau))\}$ and $C_2 = \{\tau \in \Sigma' \mid t \in L((D, \tau))\}$. It then accepts if there exists such a pair (C_1, C_2) with $s_0 \in C_1$ and $s \notin C_2$. However, when we use nondeterminism, notice that it is not necessary to compute the entire set C_1. Indeed, as we test only whether there *exist* elements in C_1 in the entire course of the algorithm, we can adapt the algorithm to compute pairs (c_1, C_2), where c_1 is an element of C_1, rather than the entire set. Since NEXPSPACE = EXPSPACE, we can use this adaption to test whether $L(D_0) \subseteq L(D)$ in EXPSPACE.

Finally, we give the algorithm which decides whether an EDTD $D = (\Sigma, \Sigma', d, s, \mu)$ is equivalent to a DTD. We compute a DTD (Σ, d_0, s_d) which is equivalent to D if and only if $L(D)$ is definable by a DTD. Thereto, let, for each $a^i \in \Sigma'$, $r_{a,i}$ be the expression obtained from $d(a^i)$ by replacing each symbol b^j in $d(a^i)$ by b. For every $a \in \Sigma$, define $d_0(a) = \bigcup_{a^i \in \Sigma'} r_{a,i}$. Again, it is shown in [27] that $L(D) = L(d_0)$ if and only if $L(D)$ is definable by a DTD. By Theorem 5.1(1) and since d_0 is of size polynomial in the size of D, this can be tested in EXPSPACE. □

7. Conclusion. The present work gives an overview of the complexity of the basic decision problems for abstractions of several schema languages including numerical occurrence constraints and interleaving. W.r.t. INTERSECTION the complexity remains the same, while for INCLUSION and EQUIVALENCE the complexity increases by one exponential for DTDs and single-type EDTDs, and goes from EXPTIME to EXPSPACE for EDTDs. The results w.r.t. CHAREs also follow this pattern. We further showed that the complexity of simplification increases to EXPSPACE.

We emphasize that this is a theoretical study delineating the worst case complexity boundaries for the basic decision problems. Although these complexities must be studied, we note that the regular expressions used in the hardness proofs do not correspond at all to those employed in practice. Further, w.r.t. XSDs, our abstraction is not fully adequate as we do not consider the one-unambiguity (or unique particle attribution) constraint. However, it is doubtful that this constraint is the right one to get tractable complexities for the basic decision problems. Indeed, already intersection for unambiguous regular expressions is PSPACE-hard [28] and inclusion for one-unambiguous RE(#) expressions is coNP-hard [22]. It would therefore be desirable to find robust subclasses for which the basic decision problems are in PTIME.

REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, San Francisco, 1999.

[2] M. Benedikt, W. Fan, and F. Geerts, *XPath satisfiability in the presence of DTDs*, J. ACM, 55 (2008), article 8.

[3] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls, *Inference of concise DTDs from XML data*, in Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), 2006, pp. 115–126.

[4] G. J. Bex, F. Neven, and J. Van den Bussche, *DTDs versus XML schema: A practical study*, in Proceedings of the 7th International Workshop on the Web and Databases (WebDB), 2004, pp. 79–84.

[5] A. Brüggemann-Klein, M. Murata, and D. Wood, *Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1, April 3,* 2001, Technical report HKUST-TCSC-2001-0, The Hong Kong University of Science and Technology, Hong Kong, 2001.

[6] A. Brüggemann-Klein and D. Wood, *One-unambiguous regular languages*, Inform. and Comput., 142 (1998), pp. 182–206.

[7] A. Brüggemann-Klein, *Unambiguity of extended regular expressions in SGML document grammars*, in Proceedings of the First Annual European Symposium on Algorithms (ESA), Springer, London, 1993, pp. 73–84.
[8] A. K. Chandra, D. Kozen, and L. J. Stockmeyer, *Alternation*, J. ACM, 28 (1981), pp. 114–133.
[9] J. Clark and M. Murata, *RELAX NG Specification*, OASIS, 2001.
[10] J. Cristau, C. Löding, and W. Thomas, *Deterministic automata on unranked trees*, in Fundamentals of Computation Theory (FCT), Lecture Notes in Comput. Sci. 3623, M. Liskiewicz and R. Reischuk, eds., Springer, Berlin, 2005, pp. 68–79.
[11] S. Dal-Zilio and D. Lugiez, *XML schema, tree logic and sheaves automata*, in Rewriting Techniques and Applications (RTA), Lecture Notes in Comput. Sci. 2706, R. Nieuwenhuis, ed., Springer, Berlin, 2003, pp. 246–263.
[12] A. Deutsch, M. F. Fernandez, and D. Suciu, *Storing semistructured data with STORED*, in Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, pp. 431–442.
[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
[14] W. Gelade, *Succinctness of regular expressions with interleaving, intersection and counting*, in Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 5162, Springer, Berlin, 2008, pp. 363–374.
[15] L. Hemaspaandra and M. Ogihara, *The Complexity Theory Companion*, Springer, Berlin, 2002.
[16] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed., Addison–Wesley, Reading, MA, 2007.
[17] H. Hosoya and B. C. Pierce, *XDuce: A statically typed XML processing language*, ACM Trans. Internet Technol., 3 (2003), pp. 117–148.
[18] J. Jędrzejowicz and A. Szepietowski, *Shuffle languages are in P*, Theoret. Comput. Sci., 250 (2001), pp. 31–53.
[19] P. Kilpeläinen and R. Tuhkanen, *Regular expressions with numerical occurrence indicators—preliminary results*, in Proceedings of the Eighth Symposium on Programming Languages and Software Tools (SPLST), 2003, pp. 163–173.
[20] P. Kilpeläinen and R. Tuhkanen, *Towards efficient implementation of XML schema content models*, in Proceedings of the 2004 ACM Symposium on Document Engineering (DOCENG), ACM, New York, 2004, pp. 239–241.
[21] P. Kilpeläinen and R. Tuhkanen, *One-unambiguity of regular expressions with numeric occurrence indicators*, Inform. and Comput., 205 (2007), pp. 890–916.
[22] P. Kilpeläinen, *Inclusion of Unambiguous #REs Is NP-Hard*, unpublished note, University of Kuopio, Kuopio, Finland, 2004.
[23] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier, *Schema-based scheduling of event processors and buffer minimization for queries on structured data streams*, in Proceedings of the International Conference on Very Large Data Bases (VLDB), 2004, pp. 228–239.
[24] D. Kozen, *Lower bounds for natural proof systems*, in Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1977, pp. 254–266.
[25] M. Mani, *Keeping chess alive—Do we need* 1*-unambiguous content models?*, talk given at Extreme Markup Languages, Montreal, 2001.
[26] I. Manolescu, D. Florescu, and D. Kossmann, *Answering XML queries on heterogeneous data sources*, in Proceedings of the International Conference on Very Large Data Bases (VLDB), 2001, pp. 241–250.
[27] W. Martens, F. Neven, T. Schwentick, and G. J. Bex, *Expressiveness and complexity of XML schema*, ACM Trans. Database Syst., 31 (2006), pp. 770–813.
[28] W. Martens, F. Neven, and T. Schwentick, *Complexity of decision problems for simple regular expressions*, in Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science (MFCS), J. Fiala, V. Koubek, and J. Kratochvíl, eds., Lecture Notes in Comput. Sci. 3153, Springer, Berlin, 2004, pp. 889–900.
[29] W. Martens and F. Neven, *Frontiers of tractability for typechecking simple XML transformations*, J. Comput. System Sci., 73 (2007), pp. 362–390.
[30] W. Martens and J. Niehren, *On the minimization of XML schemas and tree automata for unranked trees*, J. Comput. System Sci., 73 (2007), pp. 550–583.
[31] A. J. Mayer and L. J. Stockmeyer, *The complexity of word problems—this time with interleaving*, Inform. and Comput., 115 (1994), pp. 293–311.

[32] A. R. Meyer and L. J. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, in Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory, 1972, pp. 125–129.
[33] M. Murata, D. Lee, M. Mani, and K. Kawaguchi, *Taxonomy of XML schema languages using formal language theory*, ACM Trans. Internet Technol., 5 (2005), pp. 1–45.
[34] F. Neven and T. Schwentick, *On the complexity of XPath containment in the presence of disjunction, DTDs, and variables*, Log. Methods Comput. Sci., 2 (2006), 3:1.
[35] Y. Papakonstantinou and V. Vianu, *DTD inference for views of XML data*, in Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), ACM Press, New York, 2000, pp. 35–46.
[36] F. Reuter, *An Enhanced W3C XML Schema-Based Language Binding for Object Oriented Programming Languages*, manuscript, 2006.
[37] H. Seidl, *Deciding equivalence of finite tree automata*, SIAM J. Comput., 19 (1990), pp. 424–437.
[38] H. Seidl, *Haskell overloading is DEXPTIME-complete*, Inform. Process. Lett., 52 (1994), pp. 57–60.
[39] C. M. Sperberg-McQueen and H. Thompson, *XML Schema*, http://www.w3.org/XML/Schema (2005).
[40] C. M. Sperberg-McQueen, *XML Schema* 1.0*: A language for document grammars*, in XML 2003, 2003; available online at http://www.idealliance.org/papers/dx_xml03/index.html.
[41] L. J. Stockmeyer and A. R. Meyer, *Word problems requiring exponential time: Preliminary report*, in Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC), 1973, pp. 1–9.
[42] E. van der Vlist, *XML Schema*, O'Reilly, Sebastopol, CA, 2002.
[43] P. van Emde Boas, *The convenience of tilings*, in Complexity, Logic, and Recursion Theory, Lect. Notes Pure Appl. Math. 187, Dekker, New York, 1997, pp. 331–363.
[44] G. Wang, M. Liu, J. X. Yu, B. Sun, G. Yu, J. Lv, and H. Lu, *Effective schema-based XML query optimization techniques*, in Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS), IEEE Computer Society, Los Alamitos, CA, 2003, pp. 230–235.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 2044–2059

STREAM ORDER AND ORDER STATISTICS: QUANTILE ESTIMATION IN RANDOM-ORDER STREAMS*

SUDIPTO GUHA† AND ANDREW MCGREGOR‡

Abstract. When trying to process a data stream in small space, how important is the order in which the data arrive? Are there problems that are unsolvable when the ordering is worst case, but that can be solved (with high probability) when the order is chosen uniformly at random? If we consider the stream as if ordered by an adversary, what happens if we restrict the power of the adversary? We study these questions in the context of quantile estimation, one of the most well studied problems in the data-stream model. Our results include an $O(\text{polylog } n)$-space, $O(\log \log n)$-pass algorithm for exact selection in a randomly ordered stream of n elements. This resolves an open question of Munro and Paterson [*Theoret. Comput. Sci.*, 23 (1980), pp. 315–323]. We then demonstrate an exponential separation between the random-order and adversarial-order models: using $O(\text{polylog } n)$ space, exact selection requires $\Omega(\log n/\log \log n)$ passes in the adversarial-order model. This lower bound, in contrast to previous results, applies to fully general randomized algorithms and is established via a new bound on the communication complexity of a natural pointer-chasing style problem. We also prove the first fully general lower bounds in the random-order model: finding an element with rank $n/2 \pm n^{\delta}$ in the single-pass random-order model with probability at least $9/10$ requires $\Omega(\sqrt{n^{1-3\delta}/\log n})$ space.

Key words. communication complexity, stochastically generated streams, stream computation

AMS subject classifications. 68Q05, 68Q17, 68Q25, 68W20, 68W25

DOI. 10.1137/07069328X

1. Introduction.

One of the principal theoretical motivations for studying the data-stream model is to understand the role played by the order in which a problem is revealed. While an algorithm in the RAM model can process the input data in an arbitrary order, the key constraint of the data-stream model is that the algorithm must process (in small space) the input data in the order in which it arrives. Parameterizing the number of passes that an algorithm may have over the data establishes a spectrum between the RAM model and the one-pass data-stream model. How does the computational power of the model vary along this spectrum? To what extent does it matter how the stream is ordered?

These issues date back to one of the earliest papers on the data-stream model in which Munro and Paterson considered the problems of sorting and selection in limited space [21]. They showed that $\tilde{O}(n^{1/p})$ space was sufficient to find the exact median of a sequence of n numbers given p passes over the data. However, if the data were randomly ordered, $\tilde{O}(n^{1/(2p)})$ space sufficed. Based on this result and other observations, it seemed plausible that any p-pass algorithm in the random-order model could be simulated by a $2p$-pass algorithm in the adversarial-order model. This was posed as an open problem by Kannan [17], and further support for this conjecture came via work initiated by Feigenbaum et al. [6] that considered the relationship

*Received by the editors May 30, 2007; accepted for publication (in revised form) August 19, 2008; published electronically January 30, 2009. Part of this work originally appeared in the proceedings of both PODS 2006 [11] and ICALP 2007 [12].

http://www.siam.org/journals/sicomp/38-5/69328.html

†Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (sudipto@cis.upenn.edu). This author's research was supported in part by an Alfred P. Sloan Research Fellowship and by NSF awards CCF-0430376 and CCF-0644119.

‡Department of Computer Science, University of Massachusetts, Amherst, MA 01003 (mcgregor@cs.umass.edu). Part of this work was done while the author was at the University of Pennsylvania.

between various property testing models and the data-stream model. It was shown by Guha, McGregor, and Venkatasubramanian [14] that several models of property testing can be simulated in the single-pass random-order data-stream model, while it appeared that a similar simulation in the adversarial-order model required two passes.

In this paper we resolve the conjecture and demonstrate the important role played by the stream order in the context of exact selection and quantile estimation. Before detailing our results, we first motivate the study of the random-order model. We believe that this motivation, coupled with the array of further questions that naturally arise, may establish a fruitful area of future research.

1.1. Motivation. In the literature to date, it is usually assumed that the stream to be processed is ordered by an omnipotent adversary that knows the algorithm and the set of elements in the stream. In contrast to the large body of work on adversarially ordered streams, the random-order model has received little explicit attention to date. Aside from the aforementioned work by Munro and Paterson [21] and Guha, McGregor, and Venkatasubramanian [14], the only other results were given by Demaine, López-Ortiz, and Munro [5] in a paper about frequency estimation. However, there are numerous motivations for considering this model.

First, the random-order model gives rise to a natural notion of *average-case analysis* which explains why certain data-stream problems may have prohibitive space lower bounds while being typically solvable in practice. When evaluating a permutation invariant function f on a stream, we observe that there are two orthogonal components to an instance: the set of data items in the stream, $\mathcal{O} = \{x_1, x_2, \ldots, x_n\}$, and π, the permutation of $\{1, 2, \ldots, n\}$ that determines the ordering of the stream. Since f is permutation invariant, $\mathcal{O}$ determines the value of f. One approach when designing algorithms is to make an assumption about $\mathcal{O}$ such as that the set of items is distributed according to a Gaussian distribution. While this approach has its merits, because we are trying to compute something about $\mathcal{O}$ it is often difficult to find a suitable assumption that would allow small-space computation while not directly implying the value of the function. We take an alternative view and, rather than making assumptions about $\mathcal{O}$, we consider which problems can be solved, with high probability, when the data items are ordered randomly. This approach is an average-case analysis, where π is chosen uniformly from all possible permutations while $\mathcal{O}$ is chosen worst case.

Second, if we consider π to be determined by an adversary, a natural complexity question is the relationship between the power of the adversary and the resources required to process a stream. If we impose certain computational constraints on the adversary, a popular idea in cryptography, how does this affect the space and time required to process the stream?

Lastly, there are situations in which it is reasonable to assume the stream is not ordered adversarially. These include the following scenarios, where the stream order is random either by design, by definition, or because of the semantics of data:

1. *Random by definition:* A natural setting in which a data stream would be ordered randomly is if each element of the stream is a sample drawn independently from some unknown distribution. Regardless of the source distribution, given the set of n samples, each of the $n!$ permutations of the sequence of samples was equally likely. Density estimation algorithms that capitalized on this were presented by Guha and McGregor [13].
2. *Random by semantics:* In other situations the semantics of the data in the stream may imply that the stream is randomly ordered. For example, consider

a database of employee records in which the records are sorted by surname. We wish to estimate some property of the employee salaries given a stream of ⟨surname, salary⟩ tuples. If there is no correlation between the lexicographic ordering of the surnames and the numerical ordering of salaries, then the salary values are ordered uniformly at random. We note that several query optimizers make such assumptions.

3. *Random by design:* Lastly, there are some scenarios in which we dictate the order of the stream. Naturally we can therefore ensure it is nonadversarial! An example is the "backing sample" architecture proposed by Gibbons and coworkers [7, 8] for maintaining accurate estimates of aggregate properties of a database. A large sample is stored on the disk and this sample is used to periodically correct estimates of the relevant properties.

1.2. Our contributions. We start with the following algorithmic results which are proved in section 3:

1. A single-pass algorithm using $O(\log n)$ space that, given any k, returns an element of rank $k \pm O(k^{1/2} \log^2 n \log \delta^{-1})$ with probability at least $1 - \delta$ if the stream is randomly ordered. The algorithm does not require prior knowledge of the length of the stream.
2. An algorithm using $O(\text{polylog}\, n)$ space that performs exact selection in only $O(\log \log n)$ passes. This was conjectured by Munro and Paterson [21] but has been unresolved for over 30 years.

In section 4, we introduce two notions of the order of the stream being "semi-random." The first is related to the computational power of an adversary ordering the stream, and the second is related to the random process that determines the order. We show how the performance of our algorithms degrades as the randomness of the order decreases according to either notion. These notions of semirandomness will also be critical for proving lower bounds in the random-order model. In sections 5 and 6, we prove the following lower bounds:

1. Any algorithm that returns an n^δ-approximate median, i.e., an element with rank $n/2 \pm n^\delta$, in the single-pass random-order model with probability at least $9/10$ requires $\Omega(\sqrt{n^{1-3\delta}/\log n})$ space. This is the first unqualified lower bound in this model. Previously, all that was known was that a single-pass algorithm that maintained a set of elements whose ranks (among the elements read thus far) are consecutive and as close to the current median as possible, required $\Omega(\sqrt{n})$ space to find the exact median in the random-order model [21]. Our result, which is fully general, uses a reduction from communication complexity but deviates significantly from the usual form of such reductions because of the novel challenges arising when proving a lower bound in the random-order model. We believe the techniques used will be useful in proving average-case lower bounds for a variety of data-stream problems.
2. Any algorithm that returns an n^δ-approximate median in p passes of an adversarially ordered stream requires $\Omega(n^{(1-\delta)/p} p^{-6})$ space. In particular, this implies that in the adversarial-order model any $O(\text{polylog}\, n)$-space algorithm for exact selection must use $\Omega(\log n / \log \log n)$ passes. This is established via a new bound on the communication complexity of a natural pointer-chasing style problem. The best previous result showed that any deterministic, comparison-based algorithm for exact selection required $\Omega(n^{1/p})$ space for constant p [21]. This resolves the conjecture of Kannan and establishes that existing multipass algorithms are optimal up to terms polynomial in p.

1.3. Related work on quantile estimation. Quantile estimation is perhaps the most extensively studied problem in the data-stream model [3,4,9,10,15,19,20,23]. Manku, Rajagopalan, and Lindsay [19,20] showed that we can find an element of rank $k \pm \epsilon n$ using $O(\epsilon^{-1} \log^2 \epsilon n)$ space, where n is the length of the stream and k is user specified. This was improved to a deterministic, $O(\epsilon^{-1} \log \epsilon n)$-space algorithm by Greenwald and Khanna [10]. Gilbert et al. [9] gave an algorithm for the model in which elements may also be "deleted" from the stream. Shrivastava et al. [23] presented another deterministic algorithm for insert-only streams that uses $O(\epsilon^{-1} \log U)$ space, where U is the size of the domain from which the input is drawn. Gupta and Zane [15] and Cormode et al. [3] presented algorithms for estimating *biased quantiles*, i.e., algorithms that return an element of rank $k \pm \epsilon k$ for any $k \in \{1, \ldots, n\}$. We note that all these algorithms are for the adversarial-order model and therefore are not designed to take advantage of a weak, or absent, adversary.

1.4. Recent developments. Since the initial submission of this paper, there has been follow-up work that presents lower bounds on the space required for multi-pass algorithms for randomly ordered data streams [1,2]. In particular, it was shown that any $O(\text{polylog}\, n)$-space algorithm that returns the median of a randomly ordered stream of length n with probability at least $9/10$ requires $\Omega(\log \log n)$ passes. Other problems were also considered in the random-order model, including estimating frequency moments, graph connectivity, and measuring information divergences [1].

2. Notation and preliminaries. Let $[n] = \{1, \ldots, n\}$. Let Sym_n be the set of all $n!$ permutations of $[n]$. We say $a = b \pm c$ if $|a - b| \leq c$ and write $a \in_R S$ to indicate that a is chosen, uniformly at random, from the set S. The next definition clarifies the rank of an element in a multiset.

DEFINITION 2.1 (rank and approximate selection). *The rank of an item x in a set S is defined as*

$$\text{RANK}_S(x) = |\{x' \in S | x' < x\}| + 1 \ .$$

Assuming there are no duplicate elements in S, we say x is an Υ-approximate k-rank element if $\text{RANK}_S(x) = k \pm \Upsilon$. If there are duplicate elements in S, we say x is an Υ-approximate k-rank element if there exists some way of breaking ties such that x is an Υ-approximate k-rank element.

At various points we will appeal to less common variants of Chernoff–Hoeffding bounds that pertain to sampling without replacement.

THEOREM 2.2 (Hoeffding [16]). *Consider a population C consisting of N values $\{c_1, \ldots, c_N\}$. Let the mean value of the population be $\mu = N^{-1} \sum_{i=1}^{N} c_i$ and let $c_{\max} = \max_{i \in N} c_i - \min_{i \in N} c_i$. Let $X_1, \ldots, X_n$ be a sequence of independent samples* without replacement *from C and $\bar{X} = n^{-1} \sum_{i=1}^{n} X_i$. Then,*

$$\Pr\left[\bar{X} \notin (\mu - a, \mu + b)\right] \leq \exp\left(-2na^2/{c_{\max}}^2\right) + \exp\left(-2nb^2/{c_{\max}}^2\right) \ .$$

The following corollary will also be useful. If $c_i = 1$ for $i \in [k]$ and 0 otherwise, then

$$\Pr\left[\bar{X} \notin (\mu - a, \mu + b)\right] \leq \exp\left(-2a^2n^2/k\right) + \exp\left(-2b^2n^2/k\right) \ .$$

3. Algorithms for random-order streams. In this section we show how to perform approximate selection of the kth smallest element in a single pass over a

Selection Algorithm:
1. Let $a = -\infty, b = +\infty$, and

$$\begin{aligned}\Upsilon &= 20\ln^2(n)\ln(\delta^{-1})\sqrt{k}\\ p &= 4(\log_{4/3}(n/\Upsilon) + (\ln(3/\delta)\log_{4/3}(n/\Upsilon))^{1/2})\\ l_1 &= n\Upsilon^{-1}\ln(3n^2p/\delta)\\ l_2 &= 2(n-1)\Upsilon^{-1}\sqrt{(k+\Upsilon)\ln(6np/\delta)}\end{aligned}$$

2. Partition stream as $S = \langle S_1, E_1, \ldots, S_p, E_p\rangle$ where $|S_i| = l_1$ and $|E_i| = l_2$
3. Phase $i \in [p]$:
(a) *Sample*: If $S_i \cap (a,b) = \emptyset$, return a, else let u be first element in $S_i \cap (a,b)$
(b) *Estimate*: Compute $r = \textsc{Rank}_{E_i}(u)$ and let $\tilde{r} = (n-1)(r-1)/l_2 + 1$
(c) *Update*: If $\tilde{r} < k - \Upsilon/2$, $a \leftarrow u$, $\tilde{r} > k + \Upsilon/2$, $b \leftarrow u$, else return u

FIG. 3.1. *The selection algorithm.*

randomly ordered stream of length n. As we are interested in massive data streams, we consider the space complexity and accuracy guarantees of the algorithm as n becomes large.

DEFINITION 3.1 (random order). *Consider a set of elements* $x_1, \ldots, x_n \in [\text{poly}(n)]$. *Then this set and* $\pi \in \text{Sym}_n$ *define a stream* $S = \langle x_{\pi(1)}, \ldots, x_{\pi(n)}\rangle$. *If* π *is chosen uniformly from* Sym_n, *then we say the stream is in* random order.

We will present the algorithm, assuming the exact value of the length of the stream, n, is known in advance. In a subsequent section, we will show that this assumption is not necessary. In what follows, we will assume that the stream contains distinct values. This can easily be achieved with probability at least $1-\delta$ by attaching a secondary value $y_i \in_R [n^2\delta^{-1}]$ to each item x_i in the stream. We say $(x_i, y_i) < (x_j, y_j)$ iff $x_i < x_j$ or $(x_i = x_j$ and $y_i < y_j)$. Note that breaking the ties arbitrarily results in a stream whose order is not random. We also may assume that $k \le n/2$ by symmetry.

3.0.1. Algorithm overview. Our algorithm proceeds in phases and each phase is composed of the following three distinct subphases: the *sample* subphase, the *estimate* subphase, and the *update* subphase. At all points, we maintain an open interval (a,b) such that we believe that the value of the element with rank k is between a and b. In each phase we aim to narrow the interval (a,b). The sample subphase finds a value $u \in (a,b)$. The estimate subphase estimates the rank of u. The update subphase replaces a or b by u depending on whether the rank of u is believed to be less than or greater than k. See Figure 3.1 for the algorithm.

3.0.2. Analysis. For the analysis we define the following quantity:

$$\Gamma(a,b) = |S \cap (a,b)| = |\{v \in S : a < v < b\}| \ .$$

LEMMA 3.2. *With probability* $1 - \delta/3$, *for all phases, if* $\Gamma(a,b) \ge \Upsilon$, *then there exists an element* u *in each sample subphase, i.e.,*

$$\Pr[\forall\ i \in [p] \text{ and } a, b \in S \text{ such that } \Gamma(a,b) \ge \Upsilon; S_i \cap (a,b) \ne \emptyset] \ge 1 - \delta/3 \ .$$

Proof. Fix $i \in [p]$ and $a, b \in S$ such that $\Gamma(a,b) \geq \Upsilon$. Then,

$$\Pr[S_i \cap (a,b) \neq \emptyset] \geq 1 - \left(1 - \frac{\Gamma(a,b)}{n}\right)^{l_1} \geq 1 - \exp\left(\frac{-\Upsilon l_1}{n}\right) = 1 - \frac{\delta}{3n^2p} \ .$$

The result follows by applying the union bound over all choices of i, a, and b. ∎

LEMMA 3.3. *With probability $1-\delta/3$, for all phases, we determine the rank of u with sufficient accuracy, i.e.,*

$$\Pr\left[\forall i \in [p],\ u \in S;\ \begin{array}{ll} \tilde{r} = \textsc{Rank}_S(u) \pm \Upsilon/2 & \text{if } \textsc{Rank}_S(u) < k + \Upsilon + 1 \\ \tilde{r} > k + \Upsilon/2 & \text{if } \textsc{Rank}_S(u) \geq k + \Upsilon + 1 \end{array}\right] \geq 1 - \delta/3 \ ,$$

where $\tilde{r} = (n-1)(\textsc{Rank}_{E_i}(u) - 1)/l_2 + 1$.

Proof. Fix $i \in [p]$ and $u \in S$. First, we consider u such that $\textsc{Rank}_S(u) < k+\Upsilon+1$. Let $X = \textsc{Rank}_{E_i}(u) - 1$ and note that $E[X] = l_2(\textsc{Rank}_S(u) - 1)/(n-1)$. Appealing to the second part of Theorem 2.2,

$$\begin{aligned} \Pr[\tilde{r} \neq \textsc{Rank}_S(u) \pm \Upsilon/2] = &\ \Pr\left[|X - E[X]| \geq \frac{l_2\Upsilon}{2(n-1)}\right] \\ \leq &\ 2\exp\left(\frac{-2(l_2\Upsilon/(2(n-1)))^2}{\textsc{Rank}_S(u) - 1}\right) \\ \leq &\ \frac{\delta}{3np} \ , \end{aligned}$$

where the last inequality follows because $(l_2\Upsilon/(2(n-1)))^2 = (k+\Upsilon)\ln(6np/\delta)$ (by definition of l_2) and $\textsc{Rank}_S(u) - 1 < k + \Upsilon$ (by assumption). Now assume that $\textsc{Rank}_S(u) \geq k + \Upsilon + 1$ and note that $\Pr[\tilde{r} \geq k + \Upsilon/2]$ is minimized for $\textsc{Rank}_S(u) = k + \Upsilon + 1$. Hence,

$$\begin{aligned} \Pr[\tilde{r} > k + \Upsilon/2] = &\ 1 - \Pr\left[E[X] - X \geq \frac{l_2\Upsilon}{2(n-1)}\right] \\ \geq &\ 1 - \exp\left(-\frac{(l_2\Upsilon)^2}{4(k+\Upsilon)(n-1)^2}\right) \\ = &\ 1 - \frac{\delta}{6np} \ . \end{aligned}$$

The result follows by applying the union bound over all choices of i and u. ∎

We now give the main theorem of this section.

THEOREM 3.4. *For $k \in [n]$, there exists a single-pass, $O(\log n)$-space algorithm in the random-order model that returns u such that $\textsc{Rank}_S(u) = k \pm 20\ln^2(n)\ln(\delta^{-1})\sqrt{k}$ with probability at least $1-\delta$.*

Proof. Consider $\Gamma(a,b) = |\{v \in S : a < v < b\}|$ in each phase of the algorithm. By Lemmas 3.2 and 3.3, with probability at least $1 - 2\delta/3$, in every phase, if we do not terminate, then $\Gamma(a,b)$ decreases and $\textsc{Rank}_S(a) \leq k \leq \textsc{Rank}_S(b)$. In particular, in each phase, with probability $1/4$, either we terminate or $\Gamma(a,b)$ decreases by at least a factor of $3/4$. Let Y be the number of phases in which $\Gamma(a,b)$ decreases by a factor of $3/4$. If the algorithm does not terminate, then $Y < \log_{4/3}(n/\Upsilon)$ since $\Gamma(a,b)$ is initially n and the algorithm will terminate if $\Gamma(a,b) < \Upsilon$. But,

$$\Pr\left[Y < \log_{4/3}(n/\Upsilon)\right] = \Pr\left[Y < E[Y] - \sqrt{\ln(3/\delta)\log_{4/3}(n/\Upsilon)}\right] \leq \delta/3 \ .$$

Hence with probability at least $1-\delta$ the algorithm returns a value with rank $k \pm \Upsilon$.

The space bound follows immediately from the fact that the algorithm only stores a constant number of polynomially sized values and maintains a counter that stores values in the range $[n]$. Finally, for sufficiently large n,

$$p(l_1+l_2) \leq 20\ln^2(n)\ln(\delta^{-1})n\Upsilon^{-1}\sqrt{k} = n,$$

and hence the stream is sufficiently long for all the phases to complete. □

3.1. Generalizing to unknown stream lengths. The algorithm in the previous section assumed prior knowledge of n, the length of the stream. We now discuss a simple way to remove this assumption. First we argue that, for our purposes, it is sufficient to look at only half the stream.

LEMMA 3.5. *Given a randomly ordered stream S of length n, let S' be a contiguous substream of length $\tilde{n} \geq n/2$. Then, with probability at least $1-\delta$, if u is the $\tilde{k}$th smallest element of S', then* $\text{RANK}_S(u) = \tilde{k}n/\tilde{n} \pm 2(8\tilde{k}\ln\delta^{-1})^{0.5}$.

Proof. Let $a = \tilde{k}/\tilde{n}$. Let the elements in the stream be $x_1 \leq \cdots \leq x_n$. Let $X = |\{x_1,\ldots,x_{an+b}\} \cap S'|$ and $Y = |\{x_1,\ldots,x_{an-b-1}\} \cap S'|$, where $b = 2(8\tilde{k}\ln\delta^{-1})^{0.5}$. The probability that the element of rank $\tilde{k} = a\tilde{n}$ in S' has rank in S outside the range $[an-b, an+b]$ is less than

$$\begin{aligned}\Pr[X < a\tilde{n} \text{ or } Y > a\tilde{n}] \leq\ & \Pr[X < E[X] - b/2 \text{ or } Y > E[Y] + b/2] \\ \leq\ & 2\exp\left(\frac{-(b/2)^2}{3(a\tilde{n}+b)}\right) \leq \delta\ .\end{aligned}$$

The lemma follows. □

To remove the assumption that we know n, we make multiple instantiations of the algorithm. Each instantiation corresponds to a guess of n. Let $\beta = 1.5$. Instantiation i guesses a length of $\lceil 4\beta^i \rceil - \lfloor \beta^i \rfloor + 1$ and is run on the stream starting with the $\lfloor \beta^i \rfloor$th data item and ending with the $\lceil 4\beta^i \rceil$th data item. We remember the result of the algorithm until the $2(\lceil 4\beta^i \rceil - \lfloor \beta^i \rfloor + 1)$th element arrives. We say the instantiation has been canceled at this point.

LEMMA 3.6. *At any time, there is only a constant number of instantiations. Furthermore, when the stream terminates, at least one instantiation has run on a substream of at least $n/2$.*

Proof. Consider the tth element of the data stream. By this point there have been $O(\log_\beta t)$ instantiations made. However, $\Omega(\log_\beta t/6)$ instantiations have been canceled. Hence $O(\log_\beta t - \log_\beta t/6) = O(1)$ instantiations are running. We now show that there always exists an instantiation that has been running on at least half the stream. The ith instantiation gives a useful result if the length of the stream $n \in U_i = \{\lfloor 4\beta^i \rfloor + 1, \ldots, 2(\lceil 4\beta^i \rceil - \lfloor \beta^i \rfloor + 1)\}$. But $\bigcup_{i\geq 0} U_i = \mathbb{N} \setminus \{0,1,2,3,4\}$ since for all $i > 1$, $\lfloor 4\beta^i + 1 \rfloor \leq 2(\lceil 4\beta^{i-1} \rceil - \lfloor \beta^{i-1} \rfloor + 1)$. □

We can therefore generalize Theorem 3.4 as follows.

THEOREM 3.7. *For $k \in [n]$, there exists a single-pass, $O(\log n)$-space algorithm in the random-order model that returns u such that* $\text{RANK}_S(u) = k \pm 11\ln^2(n)\ln(\delta^{-1})\sqrt{k}$ *with probability at least $1-\delta$. The algorithm need not know n in advance.*

3.2. Multipass exact selection. In this section we consider the problem of exact selection of an element of rank $k = \Omega(n)$. We will later show that this requires $\Omega(\sqrt{n})$ space if an algorithm is permitted only one pass over a stream in random order. However, if $O(\log\log n)$ passes are permitted, we now show that $O(\text{polylog}\, n)$

space is sufficient. We will again assume that the elements in the stream are distinct but we note that it is not difficult to avoid this assumption.

We use a slight variant of the single-pass algorithm in section 3 as a building block. Rather than returning a single candidate, we output the pair a and b. Using the analysis in section 3, it can be shown that, with probability $1-\delta$, $\textsc{Rank}_S(a) < k < \textsc{Rank}_S(b)$ and that

$$|\textsc{Rank}_S(a) - \textsc{Rank}_S(b)| \le O(\sqrt{n}\log^2 n \log \delta^{-1}) \ .$$

In one additional pass, $\textsc{Rank}_S(a)$ and $\textsc{Rank}_S(b)$ can be computed exactly. Hence, after two passes, by ignoring all elements outside the range (a,b), we have reduced the problem to that of finding an element of rank $k - \textsc{Rank}_S(a)$ in a stream of length $O(\sqrt{n}\log^3 n)$ if we assume that $\delta^{-1} = \text{poly}(n)$. If we repeat this process $O(\log\log n)$ times and then select the desired element by explicitly storing the remaining $O(\text{polylog}\, n)$-length stream, it would appear that we can perform exact selection in $O(\text{polylog}\, n)$ space and $O(\log\log n)$ passes. However, there is one crucial detail that needs to be addressed.

In the first pass, by assumption we are processing a data stream whose order is chosen uniformly from Sym_n. However, because the stream order is not rerandomized between each pass, it is possible that the previous analysis does not apply because of dependencies that may arise between different passes. Fortunately, the following straightforward, but necessary, observation demonstrates that this is not the case.

FACT 3.8. *Let a and b, respectively, be the lower and upper bound returned after a pass of the algorithm on the stream $\langle x_1, \ldots, x_n\rangle$. Let $\pi \in \text{Sym}_n$ satisfy $i = \pi(i)$ for all $i \in [n]$ such that $x_i \notin (a,b)$. Then the algorithm also would return the same bounds after processing the stream $\langle x_{\pi(1)}, \ldots, x_{\pi(n)}\rangle$.*

Therefore, conditioned on the algorithm returning a and b, the substream of elements in the range (a,b) are still ordered uniformly. This leads to the following theorem.

THEOREM 3.9. *For $k \in [n]$, there exists an $O(\text{polylog}\, n)$-space, $O(\log\log n)$-pass algorithm in the random-order model that returns the kth smallest value of a stream with probability $1 - 1/\text{poly}(n)$.*

3.3. Applications to equidepth histograms. In this section we briefly overview an application to constructing B-bucket equidepth histograms. Here, the histogram is defined by B buckets whose boundaries are defined by the items of rank $in/(B+1)$ for $i \in [B]$. Gibbons, Matias, and Poosala [8] consider the problem of constructing an approximate B-bucket equidepth histogram of data stored in a backing sample. The measure of "goodness of fit" they consider is

$$\mu = n^{-1}\sqrt{B^{-1}\sum_{i\in[B]}\epsilon_i^2} \ ,$$

where ϵ_i is the error in the rank of the boundary of the ith bucket. They show that μ can be made smaller than any $\epsilon > 0$ where the space used depends on ϵ. However, in their model it is possible to ensure that the data are stored in random order. As a consequence of the algorithm in section 3, we get the following theorem.

COROLLARY 3.10. *In a single pass over a backing sample of size n stored in random order, we can compute the B quantiles of the samples using $O(B\log n)$ memory with error $\tilde{O}(n^{-1/2})$. Since the error goes to zero as the sample size increases, we have the first consistent estimator for this problem.*

4. Semirandom order. In this section we consider two natural notions of "semirandom" ordering and explain how our algorithm can be adjusted to process streams whose order is semirandom under either definition. The first notion is stochastic in nature: we consider the distribution over orders which are "close" to the uniform order in terms of the variational distance. This will play a critical role when proving lower bounds.

DEFINITION 4.1 (ϵ-generated-random order). *Given set $\{x_1, \ldots, x_n\}$, $\pi \in \mathrm{Sym}_n$ defines a stream $\langle x_{\pi(1)}, \ldots, x_{\pi(n)} \rangle$. We say the order is* ϵ-generated random *(ϵ-GR) if π is chosen according to a distribution ν such that $\|\mu - \nu\|_1 \leq \epsilon$, where μ is the uniform distribution on* Sym_n.

The importance of this definition is captured in the following simple lemma.

LEMMA 4.2. *Let $\mathcal{A}$ be a randomized algorithm that succeeds (i.e., returns an estimate of some property with some accuracy guarantee) with probability at least $1 - \delta$ in the random-order model. Then $\mathcal{A}$ succeeds with probability at least $1 - \delta - \epsilon$ when the stream order is ϵ-GR.*

Proof. Let $\Pr_{\mu,\mathrm{coin}}[\cdot]$ denote the probability of an event over the internal coin tosses of $\mathcal{A}$ and the ordering of the stream when the stream order is chosen according to the uniform distribution μ. Similarly, define $\Pr_{\nu,\mathrm{coin}}[\cdot]$, where ν is any distribution satisfying $\|\mu - \nu\|_1 \leq \epsilon$:

$$\Pr_{\mu,\mathrm{coin}}[\mathcal{A} \text{ succeeds}] = \sum_{\pi \in \mathrm{Sym}_n} \Pr_{\mu}[\pi] \Pr_{\mathrm{coin}}[\mathcal{A} \text{ succeeds} | \pi] \leq \Pr_{\nu,\mathrm{coin}}[\mathcal{A} \text{ succeeds}] + \epsilon \ .$$

The lemma follows since $\Pr_{\mu,\mathrm{coin}}[\mathcal{A} \text{ succeeds}] \geq 1 - \delta$ by assumption. ∎

The next theorem follows immediately from Theorem 3.4 and Lemma 4.2.

THEOREM 4.3. *For $k \in [n]$, there exists a single-pass, $O(\log n)$-space algorithm in the ϵ-GR-order model that returns u such that $\mathrm{RANK}_S(u) = k \pm 11 \ln^2(n) \ln(\delta^{-1}) \sqrt{k}$ with probability at least $1 - \delta - \epsilon$.*

The second definition is computational in nature. We consider an adversary upstream of our algorithm that can reorder the elements subject to having limited memory to do this reordering.

DEFINITION 4.4 (t-bounded-adversary-random order). *A* t-bounded adversary *is an adversary that can only delay at most t elements at a time; i.e., when presented with a stream $\langle x_1, \ldots, x_n \rangle$, it can ensure that the received stream is $\langle x_{\pi(1)}, \ldots, x_{\pi(n)} \rangle$ if $\pi \in \mathrm{Sym}_n$ satisfies*

$$\forall i \in [n], |\{i < j \leq n : \pi(j) < \pi(i)\}| \leq t \ . \tag{4.1}$$

The order of a stream is t-bounded-adversary-random (t-BAR) if it is generated by a t-bounded adversary acting on a stream whose order is random.

For example, a 2-bounded adversary acting on the stream $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle$ can transform it into $\langle 3, 2, 1, 6, 5, 4, 9, 8, 7 \rangle$ or $\langle 3, 4, 5, 6, 7, 8, 9, 1, 2 \rangle$ but can not generate $\langle 9, 8, 7, 6, 5, 4, 3, 2, 1 \rangle$. In particular, in the adversarial-order model the stream order is $(n-1)$-BAR, while in the random-order model the order is 0-BAR.

LEMMA 4.5. *Consider streams $\langle x_1, \ldots, x_n \rangle$ and $\langle x_{\pi(1)}, \ldots, x_{\pi(n)} \rangle$, where π satisfies* (4.1). *Then for any $j, w \in [n]$, $|\{x_j, \ldots, x_{j+w-1}\} \cap \{x_{\pi(j)}, \ldots, x_{\pi(j+w-1)}\} \geq w - 2t$.*

We assume that $t \leq \sqrt{k}$. Given the above lemma, it is straightforward to transform the algorithm of the previous section into one that is correct (with prescribed probability) when processing a stream in t-BAR order. In particular, it is sufficient

to set $l_1 = O(n\Upsilon^{-1}\ln(3n^2p/\delta) + t\delta^{-1})$ and to choose a random u among $S_i \cap (a,b)$ in each sample phase. Note that $l_1 < l_2$ for $t \le \sqrt{k}$. In each estimate phase a t-bounded adversary can introduce an extra $2nt/l_2 \le t\Upsilon/\sqrt{k} \le \Upsilon$ error. Hence, the total error is at most 2Υ.

THEOREM 4.6. *For $k \in [n]$, there exists a single-pass, $O(\log n)$-space algorithm in the t-BAR-order model that returns u such that* $\text{RANK}_S(u) = k \pm 20\ln^2(n)\ln(\delta^{-1})\sqrt{k}$ *with probability at least $1-\delta$.*

5. Random-order lower bound. In this section we will prove a lower bound on the space required to n^δ approximate the median in the single-pass, random-order model. Our lower bound will be based on a reduction from the communication complexity of indexing [18]. However, the reduction is significantly more involved than typical reductions because different segments of a stream cannot be determined independently by different players if the stream is in random order.

Consider two players Alice and Bob, where Alice has a binary string σ of length s and Bob has an index $r \in [s]$, where s will be determined later. It is known that for Bob to determine $\text{INDEX}(\sigma, r) = \sigma_r$ after a single message from Alice with probability at least $4/5$, this message must consist of $\Omega(s)$ bits.

THEOREM 5.1 (see, e.g., [18]). $R_{1/5}^{1-\text{way}}(\text{INDEX}) \ge c^* s$ *for some constant $c^* > 0$.*

We start by assuming that there exists an algorithm $\mathcal{A}$ that computes an n^δ-approximate median in the single-pass, random-order model with probability at least $9/10$. We then use this to construct a one-way communication protocol that will allow Alice and Bob to solve their INDEX problem. They do this by simulating $\mathcal{A}$ on a stream of length n, where Alice determines a long prefix of the stream and Bob determines the remaining elements. For convenience we assume n is even and consider the median to be the element of rank $n/2$. The stream they construct consists of the union of the following sets of elements:

X: A size x set consisting of $n/2 + n^\delta - (2n^\delta + 1)r$ copies of 0.

Y: A size y set consisting of $n/2 - n^\delta - (2n^\delta + 1)(s - r)$ copies of $2s + 2$.

Z: A size $z = (2n^\delta + 1)s$ set consisting of $2n^\delta + 1$ copies of $\{2i + \sigma_i : i \in [s]\}$.

Note that any n^δ-approximate median of $U = S \cup X \cup Y$ is $2r + \sigma_r$. The difficulty we face is that we may only assume $\mathcal{A}$ returns an n^δ-approximate median of U if U is ordered randomly. Ensuring this seems to require a significant amount of communication between Alice and Bob. How else can Alice determine the balance of elements from X and Y in the prefix of the stream or can Bob know the elements of Z that should appear in the suffix of the stream?

In what follows we will argue that by carefully choosing the length of the prefix, suffix, and s, it is possible for Alice and Bob to ensure that the ordering of the stream is $1/20$-GR, while only communicating a sufficiently small number of bits with probability at least $19/20$. Then, by appealing to Lemma 4.2, we may assume that the protocol is correct with probability at least $4/5$.

5.1. Generating a stream in semirandom order. Let A be the set of elements in the prefix of the stream which is determined by Alice. Let $B = U \setminus A$ be the set of elements in the remaining part of the stream which is determined by Bob. Roughly speaking, A will consist of $n - \tilde{\Theta}(n^{1-\delta})$ of the stream elements, including most of the elements from Z. The number of elements from X and Y will be determined on the assumption that $x = y$. B will consist of the remaining $\tilde{\Theta}(n^{1-\delta})$ elements from $X \cup Y \cup Z$. The intuition is that if B is too large, then B will contain too many elements from Z whereas, if B is too small, the assumption that $x = y$ in

the determination of A will be problematic when it comes to arguing that the order of the stream is nearly random.

Let $p = c^*/(8n^\delta \log n)$ and consider the following protocol:

1. Alice determines $A \cap Z$ and $B \cap Z$ by placing an element from Z into B with probability p, and placing it in A otherwise. Alice picks t_0 according to $T_0 \sim \mathrm{Bin}(n/2-z, 1-p)$ and t_1 according to $T_1 \sim \mathrm{Bin}(n/2-z, 1-p)$. She places t_0 copies of 0 and t_1 copies of $2s+2$ into A. She sends a message encoding $B \cap Z$, t_0, t_1, and the memory state of $\mathcal{A}$ run on a random permutation of A.
2. Bob instantiates $\mathcal{A}$ with memory state sent by Alice and continues running it on a random permutation of $B = (B \cap Z) \cup \{x - t_0 \text{ copies of } 0\} \cup \{y - t_1 \text{ copies of } 2s+2\}$. Finally, Bob returns 1 if the output of the algorithm is odd, and 0 otherwise.

Let ν be the distribution over stream orders generated by the above protocol. The next lemma establishes that ν is almost uniform. This will be required to prove the correctness of the algorithm.

LEMMA 5.2. *If $z = 10^{-6}\sqrt{pn}$, then $\|\mu - \nu\|_1 \le 1/20$ where μ is the uniform distribution on* Sym_n.

Proof. Define the random variables $T_0' \sim \mathrm{Bin}(x, 1-p)$ and $T_1' \sim \mathrm{Bin}(y, 1-p)$ and let $a_0 = x - n/2 + z$ and $a_1 = y - n/2 + z$. Note that $a_0, a_1 \ge 0$ and $a_0 + a_1 = z$. We upper bound $\|\mu - \nu\|_1$ as follows:

$$\begin{aligned}\|\mu - \nu\|_1 &= \sum_{t_0,t_1} |\Pr[T_0 = t_0, T_1 = t_1] - \Pr[T_0' = t_0, T_1' = t_1]| \\ &\le \max_{\substack{t_0 \in (1-p)x \pm b^* \\ t_1 \in (1-p)y \pm b^*}} \left| \frac{\Pr[T_0 = t_0, T_1 = t_1]}{\Pr[T_0' = t_0, T_1' = t_1]} - 1 \right| \\ &\qquad + \Pr[\max\{|T_0 - E[T_0]|, |T_1 - E[T_1]|\} \ge b^* - pz] \\ &\qquad + \Pr[\max\{|T_0' - E[T_0']|, |T_1' - E[T_1']|\} \ge b^*],\end{aligned}$$

where $b^* = 10\sqrt{pn/2} + pz$. By the Chernoff bound,

$$\begin{aligned}&\Pr[\max\{|T_0 - E[T_0]|, |T_1 - E[T_1]|\} \ge b^* - pz] \\ &\quad + \Pr[\max\{|T_0' - E[T_0']|, |T_1' - E[T_1']|\} \ge b^*] \le 8\exp\left(-2(b^* - pz)^2/(3pn)\right),\end{aligned}$$

and hence the (sum of the) last two terms are upper bounded by $1/40$ for sufficiently large n.

Let $t_0 = (1-p)x + b_0$ and $t_1 = (1-p)x + b_1$ and assume that $|b_0|, |b_1| \le b^*$. Then,

$$\begin{aligned}\frac{\Pr[T_0 = t_0, T_1 = t_1]}{\Pr[T_0' = t_0, T_1' = t_1]} &= \frac{\binom{n/2-z}{t_0}\binom{n/2-z}{t_1}}{\binom{x}{t_0}\binom{y}{t_1}p^z} \\ &= \left(\prod_{i \in [a_0]} \frac{xp - i + 1 - b_0}{(x-i+1)p}\right)\left(\prod_{i \in [a_1]} \frac{yp - i + 1 - b_1}{(y-i+1)p}\right),\end{aligned}$$

and therefore

$$\exp\left(\frac{-zb^*}{p(x-z)} + \frac{-zb^*}{p(y-z)}\right) \le \frac{\Pr[T_0 = t_0, T_1 = t_1]}{\Pr[T_0' = t_0, T_1' = t_1]} \le \exp\left(\frac{2z^2 + zb^*}{p(x-z)} + \frac{2z^2 + zb^*}{p(y-z)}\right).$$

Substituting z establishes that $|\Pr[T_0 = t_0, T_1 = t_1] / \Pr[T_0' = t_0, T_1' = t_1] - 1| \le 1/40$ for sufficiently large n. The lemma follows. ∎

The next lemma will be necessary to bound the communication of the protocol.

LEMMA 5.3. $\Pr[|Z \cap B| \geq c^* s/(2\log n)] \leq 1/20$ *for* $s = \omega(\log n)$.

Proof. Note that $E[|Z \cap B|] = pz \leq 3c^* s/(8\log n)$. Then, by an application of the Chernoff bound,

$$\Pr[|Z \cap B| \geq c^* s/(2\log n)] = \Pr[|Z \cap B| \geq (4/3)E[|Z \cap B|]] \leq \exp(-c^* s/(72\log n)) \ . \quad \square$$

THEOREM 5.4. *Computing an* n^δ*-approximate median in the random-order model with probability at least* $9/10$ *requires* $\Omega(\sqrt{n^{1-3\delta}/\log n})$ *space.*

Proof. Let Alice and Bob follow the above protocol to solve their instance of INDEX using $\mathcal{A}$. Assume $\mathcal{A}$ uses M bits of space. By Lemmas 4.2 and 5.2, the protocol is correct with probability at least $9/10 - 1/20 = 17/20$. Furthermore, by Lemma 5.3, with probability at least $19/20$ the protocol requires at most $3c^* s/4 + M$ bits of communication (for sufficiently large n): $c^* s/2$ bits to transmit $Z \cap B$, $2\log n$ bits to transmit t_0 and t_1, and M bits for the memory state of $\mathcal{A}$. Therefore, there exists a protocol transmitting $3c^* s/4 + M$ bits that is correct with probability at least $17/20 - 1/20 = 4/5$. Hence, by Theorem 5.1, $M = \Omega(s) = \Omega(\sqrt{n^{1-3\delta}/\log n})$. $\square$

6. Adversarial-order lower bound. In this section we prove that any p-pass algorithm that returns an n^δ-approximate median in the adversarial-order model requires $\Omega(n^{(1-\delta)/p} p^{-6})$ space. This, coupled with the upper bound of Munro and Paterson [21], will resolve the space complexity of multipass algorithms for median finding up to polylogarithmic factors. The proof will use a reduction from the communication complexity of a generalized form of pointer chasing that we now describe.

DEFINITION 6.1 (generalized pointer chasing). *For* $i \in [p]$*, let* $f_i : [m] \to [m]$ *be an arbitrary function. Then* g_p *is defined by*

$$g_p(f_1, f_2, \ldots, f_p) = f_p(f_{p-1}(\ldots(f_1(1))\ldots)) \ .$$

Let the i*th player,* P_i*, have function* f_i*, and consider a protocol in which the players must speak in the reverse order, i.e.,* $P_p, P_{p-1}, \ldots, P_1, P_p, \ldots$*. We say the protocol has* r *rounds if* P_p *communicates* r *times. Let* $R^r_\delta(g_p)$ *be the total number of bits that must be communicated in an* r *round (randomized) protocol for* P_1 *to learn* g_p *with probability at least* $1 - \delta$*.*

Note that $R^p_0(g_p) = O(p\log m)$. We will be looking at $(p-1)$-round protocols. The proof of the next result will be deferred to the next section.

THEOREM 6.2. $R^{p-1}_{1/10}(g_p) = \Omega(m/p^4 - p^2\log m)$.

The next theorem is shown by reducing generalized pointer chasing to approximate selection.

THEOREM 6.3. *Finding an* n^δ*-approximate median in* p *passes with probability at least* $9/10$ *in the adversarial-order model requires* $\Omega(n^{(1-\delta)/p} p^{-6})$ *space.*

Proof. We will show how a p-pass algorithm $\mathcal{A}$ that computes a t-approximate median of a length n stream gives rise to a p-round protocol for computing g_{p+1} when $m = ((n/(2t+1))^{1/p} + 1)/2$. If $\mathcal{A}$ uses M bits of space, then the protocol uses at most $(p^2 + p - 1)M$ bits. Hence by Theorem 6.2, this implies that $M = \Omega(m/p^6) = \Omega((n/t)^{1/p} p^{-6})$.

The reduction proceeds as follows. Consider a $(p+1)$-level, m-ary tree T, where we say v has level j if the distance between v and the closest leaf is $j - 1$. We start by defining some notation:

1. For $j \in [p+1], i_p, \ldots, i_j \in [m]$, let $v[i_p, \ldots, i_j]$ denote the i_jth child of $v[i_p, \ldots, i_{j+1}]$, where $v[]$ is the root of the tree.

S_1	S_2	S_3
$(0,0,0)\times 5(3-f_1(1))$		
	$(1,0,0)\times(3-f_2(1))$	$(1,1,f_3(1)),(1,2,f_3(2)),(1,3,f_3(3))$
	$(1,4,0)\times(f_2(1)-1)$	
	$(2,0,0)\times(3-f_2(2))$	$(2,1,f_3(1)),(2,2,f_3(2)),(2,3,f_3(3))$
	$(2,4,0)\times(f_2(2)-1)$	
	$(3,0,0)\times(3-f_2(3))$	$(3,1,f_3(1)),(3,2,f_3(2)),(3,3,f_3(3))$
	$(4,4,0)\times(f_2(3)-1)$	
$(4,0,0)\times 5(f_1(1)-1)$		

FIG. 6.1. *Reduction from pointer chasing to exact median finding. A triple of the form* (x_2,x_1,x_0) *corresponds to the numerical value* $x_2\cdot 5^2+x_1\cdot 5^1+x_0\cdot 5^0$. *Note that* $\text{median}(S_1\cup S_2\cup S_3) = f_1(1)\cdot 5^2 + f_2(f_1(1))\cdot 5^1 + f_3(f_2(f_1(1)))\cdot 5^0$.

2. Let the $(p+1)$ tuple $(h_p,\ldots,h_0)$ denote $\sum_{i=0}^{p} h_i(m+2)^i$.
3. For each internal node of level j, e.g., $v = v[i_p,\ldots,i_j]$, we associated a multi-set of elements $S(v)$ of size a_j. Let $a_1 = 2t+1$ and $a_j = (m-1)b_{j-1}$, where

$$b_{j-1} = a_{j-1} + ma_{j-2} + m^2 a_{j-3} + \cdots + m^{j-2}a_1 \ .$$

Note that $|\cup_{v\in V(T)} S(v)| = b_{p+1} = (2m-1)^p(2t+1)$. $S(v)$ contains

$$b_{j-1}(m - f_{p+2-j}(i_j)) \text{ copies of } (i_p,\ldots,i_j,0,0,\ldots 0)$$
$$\text{and } b_{j-1}(f_{p+2-j}(i_j)-1) \text{ copies of } (i_p,\ldots,i_j,m+1,0,\ldots,0),$$

where we define $i_{p+1} = 1$.
4. For a leaf node, e.g., $v = v[i_p,\ldots,i_1]$, we generate $2t+1$ copies of

$$(i_p,\ldots,i_1,f_{p+1}(i_1)) \ .$$

It can be shown by induction that any t-approximate median of $\cup_{v\in V(T)}S(v)$ equals $(g_1,g_2,\ldots,g_{p+1})$. See Figure 6.1 for the case when $p=2, m=3$, and $t=0$.

Let S_j be the union of $S(v)$ over all v in the jth layer. Note that S_j can be determined by the $(p+2-j)$th player, P_{p+2-j}, who knows the function f_{p+2-j}. The players emulate $\mathcal{A}$ on the stream $\langle S_1,S_2,\ldots,S_{p+1}\rangle$ in the standard way: P_{p+1} runs $\mathcal{A}$ on S_1, transmits the memory state to P_p, who instantiates the algorithm with the transmitted memory state and continues running $\mathcal{A}$ on S_2, etc., until p passes of the algorithm have been emulated. Note that this is a p-round protocol in which $M(p(p+1)-1)$ bits are communicated. The result follows. □

6.1. Proof of Theorem 6.2. The proof is a generalization of a proof by Nisan and Widgerson [22]. We present the entire argument for completeness. In the proof we lower bound the $(p-1)$-round distributional complexity, $D^{p-1}_{1/20}(g_p)$; i.e., we will consider a deterministic protocol and an input chosen from some distribution. The theorem will then follow by Yao's lemma [24] since

$$D^{p-1}_{1/20}(g_p) \le 2R^{p-1}_{1/10}(g_p) \ .$$

Let T be the protocol tree of a deterministic p-round protocol. We consider the input distribution, where each f_i is chosen uniformly from F, the set of all m^m functions from $[m]$ to $[m]$. Note that this distribution over inputs gives rise to a distribution over paths from the root of T to the leaves. We will assume that in round j, P_i's message includes g_{j-1} if $i > j$ and g_j if $i \leq j$; e.g., for $p = 4$ the appended information is shown in the following table, where $g_0 = 1$.

	Round 1				Round 2				Round 3			
Player	4	3	2	1	4	3	2	1	4	3	2	1
Appended	g_0	g_0	g_0	g_1	g_1	g_1	g_2	g_2	g_2	g_3	g_3	-

This is possible with only $O(p^2 \log m)$ extra communication. Consequently we may assume that at each node, at least $\lg m$ bits are transmitted. We will assume that protocol T requires at most $\epsilon m/2$ bits of communication, where $\epsilon = 10^{-4}(p+1)^{-4}$, and derive a contradiction.

Consider a node z in the protocol tree of T corresponding to the jth round of the protocol when it is P_i's turn to speak. Let g_{t-1} be the appended information in the last transmission. Note that $g_0, g_1, \ldots, g_{t-1}$ are specified by the messages so far.

Denote the set of functions $f_1 \times \cdots \times f_p$ that are consistent with the messages already sent be $F_1^z \times \cdots \times F_p^z$. Note that the probability of arriving at node z is $|F|^{-p} \prod_{1 \leq j \leq p} |F_j^z|$. Also note that, conditioned on arriving at node z, $f_1 \times \cdots \times f_p$ is uniformly distributed over $F_1^z \times \cdots \times F_p^z$.

DEFINITION 6.4. *Let c_z be the total communication until z is reached. We say a node z in the protocol tree is* nice *if, for $\delta = \max\{4\sqrt{\epsilon}, 400\epsilon\}$, it satisfies the following two conditions:*

$$|F_j^z| \geq 2^{-2c_z}|F| \text{ for } j \in [p] \quad \text{and} \quad H(f_t^z(g_{t-1})) \geq \lg m - \delta \ ,$$

where $H(\cdot)$ is the binary entropy.

CLAIM 1. *Given the protocol reaches node z and z is nice,*

$$\Pr[\text{next node visited is nice}] \geq 1 - 4\sqrt{\epsilon} - 1/m \ .$$

Proof. Let w be a child of z and let $c_w = c_z + a_w$. For $l \neq i$ note that $|F_l^w| = |F_l^z|$ since P_l did not communicate at node z. Hence the probability that we reach node w given that we have reached z is $\prod_{1 \leq j \leq p} |F_j^w|/|F_j^z| = |F_i^w|/|F_i^z|$. Furthermore, since z is nice,

$$\Pr\left[|F_i^w| < 2^{-2c_w}|F|\right] \leq \Pr\left[\frac{|F_i^w|}{|F_i^z|} < 2^{-2a_w}\right] \leq \sum_w 2^{-2a_w} \leq \frac{1}{m} \sum_w 2^{-a_w} \leq \frac{1}{m},$$

where the second-to-last inequality follows from $a_w \geq \lg m$ and the last inequality follows by Kraft's inequality. Hence, with probability at least $1 - 1/m$, the next node in the protocol tree satisfies the first condition of being nice.

Proving the second condition is satisfied with high probability is more complicated. Consider two different cases, $i \neq t$ and $i = t$, corresponding to whether or not player i appended g_t. In the first case, since P_t did not communicate, $F_t^z = F_t^w$ and hence $H(f_t^w(g_{t-1})) = H(f_t^z(g_{t-1})) \geq \lg m - \delta$.

We now consider the second case. In this case we need to show that $H(f_{t+1}^w(g_t)) \geq \lg m - \delta$. Note that we can express f_{t+1}^w as the following vector of random variables, $(f_{t+1}^w(1), \ldots, f_{t+1}^w(m))$, where each $f_{t+1}^w(v)$ is a random variables in universe $[m]$. Note

there is no reason to believe that components of this vector are independent. By the subadditivity of entropy,

$$\sum_{v\in[m]} H(f^w_{t+1}(v)) \geq H(f^w_{t+1}) \geq \lg(2^{-2c_w}|F|) = \lg(|F|) - 2c_w \geq m\lg m - \epsilon m,$$

using the fact that f^w_{t+1} is uniformly distribution over F^w_{t+1}, $|F^w_{t+1}| \geq 2^{-2c_w}|F|$, and $c_w \leq \epsilon m/2$. Hence if v were chosen uniformly at random from $[m]$, then

$$\Pr\big[H(f^w_{t+1}(v)) \leq \lg m - \delta\big] \leq \epsilon/\delta$$

by Markov's inequality. However, we are not interested in a v chosen uniformly at random but rather $v = g_t = f^z_t(g_{t-1})$. But, since the entropy of $f^z_t(g_{t-1})$ is large, it is "almost" distributed uniformly. Specifically, since $H(f^z_t(g_{t-1})) \geq \lg m - \delta$,

$$\Pr\big[H(f^w_{t+1}(g_t)) \leq \lg m - \delta\big] \leq \frac{\epsilon}{\delta}\left(1 + \sqrt{\frac{4\delta}{\epsilon/\delta}}\right) \leq 4\sqrt{\epsilon}\ .$$

Hence, with probability at least $1 - 4\sqrt{\epsilon}$ the next node satisfies the second condition of being nice. The claim follows by the union bound. □

Note that the height of the protocol tree is $p(p-1)$ and that the root of the protocol tree is nice. Hence the probability of ending at a leaf that is not nice is at most $p(p-1)(1/m + 4\sqrt{\epsilon}) \leq 1/25$. If the final leaf node is nice, then $H(g_t)$ is at least $\lg m - \delta$, and hence the probability that g_t is guessed correctly is at most $(\delta+1)/\lg m$ using Fano's inequality. This is less than $1/100$ for sufficiently large m, and hence the total probability of P_1 guessing g_p correctly is at most $1 - 1/20$.

7. Conclusions. In this paper we motivated the study of random-order data streams and presented the first extensive study of the theoretical issues that arise in this model. We studied these issues in the context of quantile estimation, one of the most well studied problems in the data-stream model. Our results demonstrated some of the trade-offs that arise between space, passes, and accuracy in both the random-order and adversarial-order models. We resolved a long-standing open question of Munro and Paterson [21] by devising an $O(\text{polylog}\, n)$-space, $O(\log\log n)$-pass algorithm for exact selection in a randomly ordered stream of n elements. We also resolved an open question of Kannan [17] by demonstrating an exponential separation between the random-order and adversarial-order models: using $O(\text{polylog}\, n)$ space, exact selection requires $\Omega(\log n/\log\log n)$ passes in the adversarial-order model.

Acknowledgments. We would like to thank Anupam Gupta for suggesting that we consider biased quantiles in addition to median finding. We would also like to thank Joshua Brody, Amit Chakrabarti, and Piotr Indyk for helpful discussions.

REFERENCES

[1] A. Chakrabarti, G. Cormode, and A. McGregor, *Robust lower bounds for communication and stream computation*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing (British Columbia, Canada), 2008, pp. 641–650.

[2] A. Chakrabarti, T. S. Jayram, and M. Pătraşcu, *Tight lower bounds for selection in randomly ordered streams*, in Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA), 2008, pp. 720–729.

[3] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, *Space- and time-efficient deterministic algorithms for biased quantiles over data streams*, in Proceedings of the 25th ACM SIGMOD–SIGACT–SIGART Symposium on Principles of Database Systems (Chicago, IL), 2006, pp. 263–272.

[4] G. Cormode and S. Muthukrishnan, *An improved data stream summary: The count-min sketch and its applications*, J. Algorithms, 55 (2005), pp. 58–75.

[5] E. D. Demaine, A. López-Ortiz, and J. I. Munro, *Frequency estimation of internet packet streams with limited space*, in Proceedings of the 10th Annual European Symposium on Algorithms, Springer-Verlag, London, 2002, pp. 348–360.

[6] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, *Testing and spot-checking of data streams*, Algorithmica, 34 (2002), pp. 67–80.

[7] P. B. Gibbons and Y. Matia, *Synopsis data structures for massive data sets*, in External Memory Algorithms. Papers from the DIMACS Workshop on External Memory Algorithms (Rutgers University, Piscataway, NJ), J. M. Abello and J. S. Vitter, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 50, AMS, Providence, RI, 1999, pp. 39–70.

[8] P. B. Gibbons, Y. Matias, and V. Poosala, *Fast incremental maintenance of approximate histograms*, ACM Trans. Database Syst., 27 (2002), pp. 261–298.

[9] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *How to summarize the universe: Dynamic maintenance of quantiles*, in Proceedings of the 28th International Conference on Very Large Data Bases (Hong Kong), 2002, pp. 454–465.

[10] M. Greenwald and S. Khanna, *Efficient online computation of quantile summaries*, in Proceedings of the ACM SIGMOD International Conference on Management of Data (Santa Barbara, CA), 2001, pp. 58–66.

[11] S. Guha and A. McGregor, *Approximate quantiles and the order of the stream*, in Proceedings of the 25th ACM SIGMOD–SIGACT–SIGART Symposium on Principles of Database Systems (Chicago, IL), 2006, pp. 273–279.

[12] S. Guha and A. McGregor, *Lower bounds for quantile estimation in random-order and multi-pass streaming*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 4596, Springer, New York, 2007, pp. 704–715.

[13] S. Guha and A. McGregor, *Space-efficient sampling*, in Proceedings of the AISTATS, 2007, pp. 169–176. Available online at http://www.stat.umn.edu/~aistat/proceedings/start.htm

[14] S. Guha, A. McGregor, and S. Venkatasubramanian, *Streaming and sublinear approximation of entropy and information distances*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (Miami, FL), 2006, pp. 733–742.

[15] A. Gupta and F. Zane, *Counting inversions in lists*, Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD), 2003, pp. 253–254.

[16] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[17] S. Kannan, *Open problems in data stream algorithms*, in Proceedings of the DIMACS Workshop on Streaming Data Analysis and Mining (Rutgers University, Piscataway, NJ), available online at http://dimacs.rutgers.edu/Workshops/Streaming/abstracts.html, 2001.

[18] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[19] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, *Approximate medians and other quantiles in one pass and with limited memory*, in Proceedings of the ACM SIGMOD International Conference on Management of Data (Seattle, WA), 1998, pp. 426–435.

[20] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, *Random sampling techniques for space efficient online computation of order statistics of large datasets*, in Proceedings of the ACM SIGMOD International Conference on Management of Data (Philadelphia, PA), 1999, pp. 251–262.

[21] J. I. Munro and M. Paterson, *Selection and sorting with limited storage*, Theoret. Comput. Sci., 12 (1980), pp. 315–323.

[22] N. Nisan and A. Wigderson, *Rounds in communication complexity revisited*, SIAM J. Comput., 22 (1993), pp. 211–219.

[23] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, *Medians and beyond: New aggregation techniques for sensor networks*, in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (Baltimore, MD), 2004, pp. 239–249.

[24] A. C. Yao, *Lower bounds by probabilistic arguments*, in Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (Tuscon, AZ), 1980, pp. 420–428.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 2060–2078

SAMPLING ALGORITHMS AND CORESETS FOR ℓ_p REGRESSION*

ANIRBAN DASGUPTA†, PETROS DRINEAS‡, BOULOS HARB§, RAVI KUMAR†, AND MICHAEL W. MAHONEY¶

Abstract. The ℓ_p *regression problem* takes as input a matrix $A \in \mathbb{R}^{n\times d}$, a vector $b \in \mathbb{R}^n$, and a number $p \in [1,\infty)$, and it returns as output a number $\mathcal{Z}$ and a vector $x_{\text{OPT}} \in \mathbb{R}^d$ such that $\mathcal{Z} = \min_{x\in\mathbb{R}^d} \|Ax-b\|_p = \|Ax_{\text{OPT}}-b\|_p$. In this paper, we construct coresets and obtain an efficient two-stage sampling-based approximation algorithm for the very overconstrained ($n \gg d$) version of this classical problem, for all $p \in [1,\infty)$. The first stage of our algorithm nonuniformly samples $\hat{r}_1 = O(36^p d^{\max\{p/2+1,p\}+1})$ rows of A and the corresponding elements of b, and then it solves the ℓ_p regression problem on the sample; we prove this is an 8-approximation. The second stage of our algorithm uses the output of the first stage to resample $\hat{r}_1/\epsilon^2$ constraints, and then it solves the ℓ_p regression problem on the new sample; we prove this is a $(1+\epsilon)$-approximation. Our algorithm unifies, improves upon, and extends the existing algorithms for special cases of ℓ_p regression, namely, $p = 1, 2$ [K. L. Clarkson, in *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms*, ACM, New York, SIAM, Philadelphia, 2005, pp. 257–266; P. Drineas, M. W. Mahoney, and S. Muthukrishnan, in *Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms*, ACM, New York, SIAM, Philadelphia, 2006, pp. 1127–1136]. In the course of proving our result, we develop two concepts—well-conditioned bases and subspace-preserving sampling—that are of independent interest.

Key words. randomized algorithms, sampling algorithms, ℓ_p regression

AMS subject classification. 68W20

DOI. 10.1137/070696507

1. Introduction.

An important question in algorithmic theory is whether there exists a *small* subset of the input such that if computations are performed only on this subset, then the solution to the given problem can be *approximated* well. Such a subset is often known as a *coreset* for the problem. The concept of coresets has been extensively used in solving many problems in optimization and computational geometry; e.g., see the excellent survey by Agarwal, Har-Peled, and Varadarajan [2].

In this paper, we construct coresets and obtain efficient sampling algorithms for the classical ℓ_p regression problem, for all $p \in [1,\infty)$. Recall the ℓ_p *regression problem.*

PROBLEM 1 (ℓ_p regression problem). *Let $\|\cdot\|_p$ denote the p-norm of a vector. Given as input a matrix $A \in \mathbb{R}^{n\times m}$, a target vector $b \in \mathbb{R}^n$, and a real number $p \in [1,\infty)$, find a vector x_{OPT} and a number $\mathcal{Z}$ such that*

$$\mathcal{Z} = \min_{x\in\mathbb{R}^m} \|Ax-b\|_p = \|Ax_{\text{OPT}}-b\|_p . \tag{1}$$

In this paper, we will use the following ℓ_p *regression coreset* concept.

*Received by the editors July 10, 2007; accepted for publication (in revised form) November 5, 2008; published electronically February 6, 2009.
http://www.siam.org/journals/sicomp/38-5/69650.html
†Yahoo! Research, 701 First Ave., Sunnyvale, CA 94089 (anirban@yahoo-inc.com, ravikumar@yahoo-inc.com).
‡Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 (drinep@cs.rpi.edu).
§Google Inc., New York, NY 10011 (harb@google.com).
¶Department of Mathematics, Stanford University, Stanford, CA 94305 (mmahoney@cs.stanford.edu).

DEFINITION 2 (ℓ_p regression coreset). *Let $0 < \epsilon < 1$. A* coreset *for Problem 1 is a set of indices $\mathcal{I}$ such that the solution $\hat{x}_{\text{OPT}}$ to $\min_{x\in\mathbb{R}^m} ||\hat{A}x - \hat{b}||_p$, where $\hat{A}$ is composed of those rows of A whose indices are in $\mathcal{I}$ and $\hat{b}$ consists of the corresponding elements of b, satisfies $\|A\hat{x}_{\text{OPT}} - b\|_p \leq (1+\epsilon)\min_x \|Ax - b\|_p$.*

If $n \gg m$, i.e., if there are many more constraints than variables, then (1) is an *overconstrained ℓ_p regression problem*. In this case, there does not in general exist a vector x such that $Ax = b$, and thus $\mathcal{Z} > 0$. Overconstrained regression problems are fundamental in statistical data analysis and have numerous applications in applied mathematics, data mining, and machine learning [17, 10]. Even though convex programming methods can be used to solve the overconstrained regression problem in time $O((mn)^c)$ for $c > 1$, this is prohibitive if n is large.[1] This raises the natural question of developing more efficient algorithms that run in time $O(m^c n)$ for $c > 1$, while possibly relaxing the solution to (1). In particular, can we get a κ-approximation to the ℓ_p regression problem, i.e., a vector $\hat{x}$ such that $\|A\hat{x} - b\|_p \leq \kappa\mathcal{Z}$, where $\kappa > 1$? Note that a coreset of small size would strongly satisfy our requirements and result in an efficiently computed solution that is almost as good as the optimal. Thus, the question becomes: Do coresets exist for the ℓ_p regression problem, and if so, can we compute them efficiently?

Our main result is an efficient two-stage sampling-based approximation algorithm that constructs a coreset and thus achieves a $(1+\epsilon)$-approximation for the ℓ_p regression problem. The first stage of the algorithm is sufficient to obtain a (fixed) constant factor approximation. The second stage of the algorithm carefully uses the output of the first stage to construct a coreset and achieve arbitrary constant factor approximation.

1.1. Our contributions. Summary of results. For simplicity of presentation, we summarize the results for the case of $m = d = \text{rank}(A)$. Let $k = \max\{p/2+1,\ p\}$, and let $\phi(r, d)$ be the time required to solve an ℓ_p regression problem with r constraints and d variables. In the first stage of the algorithm, we compute a set of sampling probabilities $p_1, \ldots, p_n$ in time $O(nd^5 \log n)$, sample $\widehat{r_1} = O(36^p d^{k+1})$ rows of A and the corresponding elements of b according to the p_i's, and solve an ℓ_p regression problem on the (much smaller) sample; we prove this is an 8-approximation algorithm with a running time of $O\left(nd^5 \log n + \phi(\widehat{r_1}, d)\right)$. In the second stage of the algorithm, we use the residual from the first stage to compute a new set of sampling probabilities $q_1, \ldots, q_n$, sample an additional $\widehat{r_2} = O(\widehat{r_1}/\epsilon^2)$ rows of A and the corresponding elements of b according to the q_i's, and solve an ℓ_p regression problem on the (much smaller) sample; we prove this is a $(1+\epsilon)$-approximation algorithm with a total running time of $O\left(nd^5 \log n + \phi(\widehat{r_2}, d)\right)$ (section 4). We also show how to extend our basic algorithm to commonly encountered and more general settings of constrained, generalized, and weighted ℓ_p regression problems (section 5).

We note that the l_p regression problem for $p = 1, 2$ has been studied before. For $p = 1$, Clarkson [11] uses a subgradient-based algorithm to preprocess A and b and then samples the rows of the modified problem; these elegant techniques, however, depend crucially on the linear structure of the l_1 regression problem.[2] Furthermore, this algorithm does not yield coresets. For $p = 2$, Drineas, Mahoney, and Muthukrishnan [13] construct coresets by exploiting the singular value decomposition, a property

[1] For the special case of $p = 2$, vector space methods can solve the regression problem in time $O(m^2 n)$, and if $p = 1$ or ∞, linear programming methods can be used.

[2] Two ingredients of [11] use the linear structure: the subgradient-based preprocessing itself and the counting argument for the concentration bound.

peculiar to the l_2 space. Thus, in order to efficiently compute coresets for the ℓ_p regression problem for all $p \in [1, \infty)$, we need tools that capture the geometry of l_p-norms. In this paper we develop the following two tools that may be of independent interest (section 3).

(1) *Well-conditioned bases.* Informally speaking, if the the columns of matrix U form a well-conditioned basis for a d-dimensional subspace of $\mathbb{R}^n$, then for all $z \in \mathbb{R}^d$, $\|z\|_p$ should be close to $\|Uz\|_p$. We will formalize this by requiring[3] that for all $z \in \mathbb{R}^d$, $\|z\|_q$ multiplicatively approximates $\|Uz\|_p$ by a factor that can depend on d but is *independent* of n (where p and q are dual; i.e., $\frac{1}{q}+\frac{1}{p} = 1$). We show that these bases exist and can be constructed in time $O(nd^5 \log n)$. In fact, our notion of a well-conditioned basis can be interpreted as a computational analogue of the Auerbach and Lewis bases studied in functional analysis [28]. They are also related to the barycentric spanners recently introduced by Awerbuch and R. Kleinberg [5] (section 3.1). J. Kleinberg and Sandler [18] defined the notion of an ℓ_1-independent basis, and our well-conditioned basis can be used to obtain an exponentially better "condition number" than their construction. Further, Clarkson [11] defined the notion of an "ℓ_1-conditioned matrix," and he preprocessed the input matrix to an ℓ_1 regression problem so that it satisfies conditions similar to those satisfied by our bases.

(2) *Subspace-preserving sampling.* We show that sampling rows of A according to information in the rows of a well-conditioned basis of A minimizes the sampling variance, and, consequently, the rank of A is not lost by sampling. This is critical for our relative-error approximation guarantees. The notion of subspace-preserving sampling was used in [13] for $p = 2$, but we abstract and generalize this concept for all $p \in [1, \infty)$.

We note that for $p = 2$, our sampling complexity matches that of [13], which is $O(d^2/\epsilon^2)$; and for $p = 1$, it improves that of [11] from $O(d^{3.5}(\log d)/\epsilon^2)$ to $O(d^{2.5}/\epsilon^2)$.

Overview of our methods. Given an input matrix A, we first construct a well-conditioned basis for A and use that to obtain bounds on a slightly nonstandard notion of a p-norm condition number of a matrix. The use of this particular condition number is crucial since the variance in the subspace-preserving sampling can be upper-bounded in terms of it. An ε-net argument then shows that the first stage sampling gives us an 8-approximation. The next twist is to use the output of the first stage as a feedback to fine-tune the sampling probabilities. This is done so that the "positional information" of b with respect to A is also preserved in addition to the subspace. A more careful use of a different ε-net shows that the second stage sampling achieves a $(1 + \epsilon)$-approximation.

1.2. Related work. As mentioned earlier, in the course of providing a sampling-based approximation algorithm for ℓ_1 regression, Clarkson [11] shows that coresets exist and can be computed efficiently for a *controlled* ℓ_1 regression problem. Clarkson first preprocesses the input matrix A to make it well conditioned with respect to the ℓ_1-norm and then applies a subgradient-descent–based approximation algorithm to guarantee that the ℓ_1-norm of the target vector is conveniently bounded. Coresets of size $O(d^{3.5} \log d/\epsilon^2)$ are thereupon exhibited for this modified regression problem. For the ℓ_2 case, Drineas, Mahoney, and Muthukrishnan [13] designed sampling strategies to preserve the subspace information of A and proved the existence of a coreset of

[3]The requirement could equivalently be in terms of $\|z\|_p$, but the above form yields the tightest dependence on d, since we plan to use Hölder's inequality.

rows of size $O(d^2/\epsilon^2)$, for the *original* ℓ_2 regression problem; this leads to a $(1+\epsilon)$-approximation algorithm. Their algorithm used $O(nd^2)$ time to construct the coreset and solve the ℓ_2 regression problem, which is sufficient time to solve the regression problem without resorting to sampling. In a subsequent work, Sarlós [22] improved the running time for the $(1+\epsilon)$-approximation to $\tilde{O}(nd)$ by using random sketches based on the fast Johnson–Lindenstrauss transform of Ailon and Chazelle [3].

More generally, embedding d-dimensional subspaces of L_p into $\ell_p^{f(d)}$ using coordinate restrictions has been extensively studied [21, 23, 8, 25, 26, 24]. Using well-conditioned bases, one can provide a constructive analogue of Schechtman's existential L_1 embedding result [23] (see also [8]) that any d-dimensional subspace of $L_1[0,1]$ can be embedded in ℓ_1^r with distortion $(1+\epsilon)$ with $r = O(d^2/\epsilon^2)$, albeit with an extra factor of $\sqrt{d}$ in the sampling complexity. Coresets have been analyzed by the computational geometry community as a tool for efficiently approximating various extent measures [1, 2]; see also [16, 6, 14] for applications of coresets in combinatorial optimization. An important difference is that most of the coreset constructions are exponential in dimension and thus applicable only to low-dimensional problems, whereas our coresets are polynomial in dimension and thus applicable to high-dimensional problems.

2. Preliminaries. Given a vector $x \in \mathbb{R}^m$, its *p-norm* is $\|x\|_p = \sum_{i=1}^m (|x_i|^p)^{1/p}$, and the *dual norm* of $\|\cdot\|_p$ is denoted $\|\cdot\|_q$, where $1/p + 1/q = 1$. Given a matrix $A \in \mathbb{R}^{n\times m}$, its *generalized p-norm* is $|||A|||_p = (\sum_{i=1}^n \sum_{j=1}^m |A_{ij}|^p)^{1/p}$. This is a submultiplicative matrix norm that generalizes the Frobenius norm from $p = 2$ to all $p \in [1,\infty)$, but it is not a vector-induced matrix norm. The jth column of A is denoted $A_{\star j}$, and the ith row is denoted $A_{i\star}$. In this notation, $|||A|||_p = (\sum_j \|A_{\star j}\|_p^p)^{1/p} = (\sum_i \|A_{i\star}\|_p^p)^{1/p}$. For $x, x', x'' \in \mathbb{R}^m$, it can be shown using Hölder's inequality that $\|x - x'\|_p^p \le 2^{p-1}(\|x - x''\|_p^p + \|x'' - x'\|_p^p)$.

Two crucial ingredients in our proofs are ε-nets and tail inequalities. A subset $\mathcal{N}(D)$ of a set D equipped with a metric $\|\cdot\|$ is called an *ε-net* in D for some $\varepsilon > 0$ if for every $x \in D$ there is a $y \in \mathcal{N}(D)$ with $\|x - y\| \le \varepsilon$. In order to construct an ε-net for D it is enough to choose $\mathcal{N}(D)$ to be the maximal set of points that are pairwise ε apart. It is well known that the unit ball of a d-dimensional space has an ε-net of size at most $(3/\varepsilon)^d$ [8]. We will use the following version of the Bernstein's inequality.

THEOREM 3 (see [20, 7]). *Let $\{X_i\}_{i=1}^n$ be independent random variables with $E[X_i^2] < \infty$ and $X_i \ge 0$. Set $Y = \sum_i X_i$, and let $\gamma > 0$. Then*

$$\Pr[Y \le E[Y] - \gamma] \le \exp\left(\frac{-\gamma^2}{2\sum_i E[X_i^2]}\right). \tag{2}$$

If $X_i - E[X_i] \le \Delta$ for all i, then with $\sigma_i^2 = E[X_i^2] - E[X_i]^2$ we have

$$\Pr[Y \ge E[Y] + \gamma] \le \exp\left(\frac{-\gamma^2}{2\sum_i \sigma_i^2 + 2\gamma\Delta/3}\right). \tag{3}$$

Finally, throughout this paper, we will use the following sampling matrix formalism to represent our sampling operations. Given a set of n probabilities, $p_i \in (0,1]$ for $i = 1,\dots,n$, let S be an $n \times n$ diagonal sampling matrix such that S_{ii} is set to $1/p_i^{1/p}$ with probability p_i and to zero otherwise. Clearly, premultiplying A or b by S determines whether the ith row of A and the corresponding element of b will be included in the sample, and the expected number of rows/elements selected is $r' = \sum_{i=1}^n p_i$.

(In what follows, we will abuse notation slightly by ignoring zeroed-out rows and regarding S as an $r' \times n$ matrix and thus SA as an $r' \times m$ matrix.) Thus, e.g., sampling constraints from (1) and solving the induced subproblem may be represented as solving

$$\hat{\mathcal{Z}} = \min_{\hat{x} \in \mathbb{R}^m} \|SA\hat{x} - Sb\|_p . \tag{4}$$

A vector $\hat{x}$ is said to be a κ-*approximation* to the ℓ_p regression problem of (1) for $\kappa \geq 1$ if $\|A\hat{x} - b\|_p \leq \kappa \mathcal{Z}$.

3. Main technical ingredients.

3.1. Well-conditioned bases. We introduce the following notion of a "well-conditioned" basis.

DEFINITION 4 (well-conditioned basis). *Let A be an $n \times m$ matrix of rank d, let $p \in [1, \infty)$, and let q be its dual norm. Then an $n \times d$ matrix U is an* (α, β, p)–well-conditioned basis *for the column space of A if the columns of U span the column space of A and* (1) $|||U|||_p \leq \alpha$, *and* (2) *for all $z \in \mathbb{R}^d$, $\|z\|_q \leq \beta \|Uz\|_p$. We will say that U is a* p–well-conditioned basis *for the column space of A if α and β are $d^{O(1)}$, independent of m and n.*

Recall that any orthonormal basis U for span(A) satisfies both $|||U|||_2 = \|U\|_F = \sqrt{d}$ and also $\|z\|_2 = \|Uz\|_2$ for all $z \in \mathbb{R}^d$ and thus is a $(\sqrt{d}, 1, 2)$–well-conditioned basis. Thus, Definition 4 generalizes to an arbitrary p-norm for $p \in [1, \infty)$, the notion that an orthogonal matrix is well conditioned with respect to the 2-norm. Observe that the conditions are slightly different from those of the standard definition of a low-distortion embedding for the following reason. If U is a low distortion embedding, that is, if $\|z\|_p / C \leq \|Uz\|_p \leq \|z\|_p$ for some C, then we can easily see that U is a well-conditioned basis according to the above definition with α and β being $Cd^{O(1)}$. The reverse, however, does not hold. The well-conditioned basis definition above is intended to capture the essence of what is required of a basis for our subspace-sampling strategy to hold. Note also that duality is incorporated into Definition 4 since it relates the q-norm of the vector $z \in \mathbb{R}^d$ to the p-norm of the vector $Uz \in \mathbb{R}^n$, where p and q are dual[4] (i.e., $\frac{1}{q} + \frac{1}{p} = 1$).

The existence and efficient construction of these bases are given by the following.

THEOREM 5. *Let A be an $n \times m$ matrix of rank d, let $p \in [1, \infty)$, and let q be its dual norm. Then there exists an* (α, β, p)–well-conditioned basis U *for the column space of A such that if $p < 2$, then $\alpha = d^{\frac{1}{p} + \frac{1}{2}}$ and $\beta = 1$; if $p = 2$, then $\alpha = d^{\frac{1}{2}}$ and $\beta = 1$; and if $p > 2$, then $\alpha = d^{\frac{1}{p} + \frac{1}{2}}$ and $\beta = d^{\frac{1}{q} - \frac{1}{2}}$. Moreover, U can be computed in $O(nmd + nd^5 \log n)$ time (or in just $O(nmd)$ time if $p = 2$).*

Proof. Let $A = QR$, where Q is any $n \times d$ matrix that is an orthonormal basis for span(A) and R is a $d \times m$ matrix. If $p = 2$, then Q is the desired basis U; from the discussion following Definition 4, $\alpha = \sqrt{d}$ and $\beta = 1$, and computing the matrix U requires $O(nmd)$ time [15]. Otherwise, fix Q and p, and define the norm

$$\|z\|_{Q,p} \triangleq \|Qz\|_p .$$

[4] For $p = 2$, Drineas, Mahoney, and Muthukrishnan used this basis, i.e., an orthonormal matrix, to construct probabilities to sample the original matrix. For $p = 1$, Clarkson used a procedure similar to the one we describe in the proof of Theorem 5 to preprocess A such that the 1-norm of z is a $d\sqrt{d}$ factor away from the 1-norm of Az.

A quick check shows that $\|\cdot\|_{Q,p}$ is indeed a norm. ($\|z\|_{Q,p} = 0$ if and only if $z = 0$ since Q has full column rank; $\|\gamma z\|_{Q,p} = \|\gamma Q z\|_p = |\gamma| \|Qz\|_p = |\gamma| \|z\|_{Q,p}$; and $\|z + z'\|_{Q,p} = \|Q(z + z')\|_p \le \|Qz\|_p + \|Qz'\|_p = \|z\|_{Q,p} + \|z'\|_{Q,p}$.)

Consider the set $C = \{z \in \mathbb{R}^d : \|z\|_{Q,p} \le 1\}$, which is the unit ball of the norm $\|\cdot\|_{Q,p}$. In addition, define the $d \times d$ matrix F such that $\mathcal{E}_{\text{LJ}} = \{z \in \mathbb{R}^d : z^T F z \le 1\}$ is the Löwner–John ellipsoid of C. Since C is symmetric about the origin, $(1/\sqrt{d})\mathcal{E}_{\text{LJ}} \subseteq C \subseteq \mathcal{E}_{\text{LJ}}$; thus, for all $z \in \mathbb{R}^d$,

$$\|z\|_{\text{LJ}} \le \|z\|_{Q,p} \le \sqrt{d}\, \|z\|_{\text{LJ}}, \tag{5}$$

where $\|z\|_{\text{LJ}}^2 = z^T F z$ (see, e.g., [9, pp. 413–414]). Since the matrix F is symmetric positive definite, we can express it as $F = G^T G$, where G is full rank and upper triangular. Since Q is an orthogonal basis for span(A) and G is a $d \times d$ matrix of full rank, it follows that $U = QG^{-1}$ is an $n \times d$ matrix that spans the column space of A. Note that

$$A = QR = QG^{-1}GR = U\tau,$$

where $\tau = GR$. We claim that $U = QG^{-1}$ is the desired p–well-conditioned basis.

To establish this claim, let $z' = Gz$. Thus, $\|z\|_{\text{LJ}}^2 = z^T F z = z^T G^T G z = (Gz)^T Gz = z'^T z' = \|z'\|_2^2$. Furthermore, since G is invertible, $z = G^{-1}z'$, and thus $\|z\|_{Q,p} = \|Qz\|_p = \left\|QG^{-1}z'\right\|_p = \|Uz'\|_p$. By combining these expressions with (5), it follows that for all $z' \in \mathbb{R}^d$,

$$\|z'\|_2 \le \|Uz'\|_p \le \sqrt{d}\, \|z'\|_2. \tag{6}$$

Since $\|U\|_p^p = \sum_j \|U_{\star j}\|_p^p = \sum_j \|Ue_j\|_p^p \le \sum_j d^{\frac{p}{2}} \|e_j\|_2^p = d^{\frac{p}{2}+1}$, where the inequality follows from the upper bound in (6), it follows that $\alpha = d^{\frac{1}{p}+\frac{1}{2}}$. If $p < 2$, then $q > 2$ and $\|z\|_q \le \|z\|_2$ for all $z \in \mathbb{R}^d$; by combining this with (6), it follows that $\beta = 1$. On the other hand, if $p > 2$, then $q < 2$ and $\|z\|_q \le d^{\frac{1}{q}-\frac{1}{2}} \|z\|_2$; by combining this with (6), it follows that $\beta = d^{\frac{1}{q}-\frac{1}{2}}$.

In order to construct U, we need to compute Q and G and then invert G. Our matrix A can be decomposed into QR using the compact QR decomposition in $O(nmd)$ time [15]. The matrix F describing the Löwner–John ellipsoid of the unit ball of $\|\cdot\|_{Q,p}$ can be computed in $O(nd^5 \log n)$ time [19]. Finally, computing G from F takes $O(d^3)$ time, and inverting G takes $O(d^3)$ time. □

It is an open question whether the discontinuity at $p = 2$ in Theorem 5 is inherent in the structure of dual norms, or whether it is due to our inability to compute a better set of well-conditioned bases.

Connection to barycentric spanners. A point set $K = \{K_1, \ldots, K_d\} \subseteq D \subseteq \mathbb{R}^d$ is a *barycentric spanner* for the set D if every $z \in D$ may be expressed as a linear combination of elements of K using coefficients in $[-C, C]$ for $C = 1$. When $C > 1$, K is called a C-approximate barycentric spanner. Barycentric spanners were introduced by Awerbuch and R. Kleinberg in [5]. They showed that if a set is compact, then it has a barycentric spanner. Our proof shows that if A is an $n \times d$ matrix, then $B = \tau^{-1}/\sqrt{d} = R^{-1}G^{-1}/\sqrt{d} \in \mathbb{R}^{d \times d}$ is a $\sqrt{d}$-approximate barycentric spanner for $D = \{z \in \mathbb{R}^d : \|Az\|_p \le 1\}$. To see this, first note that each $B_{\star j}$ belongs to D since $\|AB_{\star j}\|_p = \frac{1}{\sqrt{d}} \|Ue_j\|_p \le \|e_j\|_2 = 1$, where the inequality is obtained

from (6). Moreover, since B spans $\mathbb{R}^d$, we can write any $z \in D$ as $z = B\nu$. Thus, $\nu = B^{-1}z = \sqrt{d}\tau z$. Hence,

$$\|\nu\|_\infty \leq \|\nu\|_2 \leq \|U\nu\|_p = \left\|\sqrt{d}U\tau z\right\|_p = \sqrt{d}\,\|Az\|_p \leq \sqrt{d},$$

where the second inequality is also obtained from (6). This shows that our basis has the added property that every element $z \in D$ can be expressed as a linear combination of elements (or columns) of B using coefficients whose ℓ_2-norm is bounded by $\sqrt{d}$.

Connection to Auerbach bases. An *Auerbach basis* $U = \{U_{\star j}\}_{j=1}^d$ for a d-dimensional normed space $\mathcal{A}$ is a basis such that $\|U_{\star j}\|_p = 1$ for all j and such that whenever $y = \sum_j \nu_j U_{\star j}$ is in the unit ball of $\mathcal{A}$, then $|\nu_j| \leq 1$. The existence of such a basis for every finite dimensional normed space was first proved by Auerbach [4] (see also [12, 27]). It can easily be shown that an Auerbach basis is an (α, β, p)–well-conditioned basis, with $\alpha = d$ and $\beta = 1$ for all p. Further, suppose U is an Auerbach basis for span(A), where A is an $n \times d$ matrix of rank d. Writing $A = U\tau$, it follows that τ^{-1} is an *exact* barycentric spanner for $D = \{z \in \mathbb{R}^d : \|Az\|_p \leq 1\}$. Specifically, each $\tau_{\star j}^{-1} \in D$ since $\|A\tau_{\star j}^{-1}\|_p = \|U_{\star j}\|_p = 1$. Now write $z \in D$ as $z = \tau^{-1}\nu$. Since the vector $y = Az = U\nu$ is in the unit ball of span(A), we have $|\nu_j| \leq 1$ for all $1 \leq j \leq d$. Therefore, computing a barycentric spanner for the compact set D—which is the preimage of the unit ball of span(A)—is equivalent (up to polynomial factors) to computing an Auerbach basis for span(A).

3.2. Subspace-preserving sampling. In the previous subsection (and in the notation of the proof of Theorem 5), we saw that given $p \in [1, \infty)$, any $n \times m$ matrix A of rank d can be decomposed as

$$A = QR = QG^{-1}GR = U\tau,$$

where $U = QG^{-1}$ is a p–well-conditioned basis for span(A) and $\tau = GR$. The significance of a p–well-conditioned basis is that we are able to minimize the variance in our sampling process by randomly sampling *rows* of the matrix A and elements of the vector b according to a probability distribution that depends on norms of the *rows* of the matrix U. This will allow us to preserve the subspace structure of span(A) and thus to achieve relative-error approximation guarantees.

More precisely, given $p \in [1, \infty)$ and any $n \times m$ matrix A of rank d decomposed as $A = U\tau$, where U is an (α, β, p)–well-conditioned basis for span(A), consider any set of sampling probabilities p_i for $i = 1, \ldots, n$ that satisfy

$$p_i \geq \min\left\{1, \frac{\|U_{i\star}\|_p^p}{|||U|||_p^p} r\right\}, \tag{7}$$

where $r = r(\alpha, \beta, p, d, \epsilon)$ to be determined below. Let us randomly sample the ith row of A with probability p_i for all $i = 1, \ldots, n$. Recall that we can construct a diagonal sampling matrix S, where each $S_{ii} = 1/p_i^{1/p}$ with probability p_i and 0 otherwise, in which case we can represent the sampling operation as SA.

The following theorem is our main result regarding this subspace-preserving sampling procedure.

THEOREM 6. *Let A be an $n \times m$ matrix of rank d, $\epsilon \leq 1/7$, and let $p \in [1, \infty)$. Let U be an (α, β, p)–well-conditioned basis for* span(A)*, and let us randomly sample rows of A according to the procedure described above using the probability distribution given*

by (7)*, where* $r \geq 16(2^p+2)(\alpha\beta)^p(d\ln(\frac{12}{\epsilon})+\ln(\frac{2}{\delta}))/(p^2\epsilon^2)$*. Then, with probability* $1-\delta$*, the following holds for all* $x \in \mathbb{R}^m$*:*

$$\left| \|SAx\|_p - \|Ax\|_p \right| \leq \epsilon \|Ax\|_p .$$

Proof. For simplicity of presentation, in this proof we will generally drop the subscript from our matrix and vector p-norms; i.e., unsubscripted norms will be p-norms. Note that it suffices to prove that, for all $x \in \mathbb{R}^m$,

$$(1-\epsilon)^p \|Ax\|^p \leq \|SAx\|^p \leq (1+\epsilon)^p \|Ax\|^p \tag{8}$$

with probability $1-\delta$. To this end, fix a vector $x \in \mathbb{R}^m$, define the random variable $X_i = (S_{ii}|A_{i\star}x|)^p$, and recall that $A_{i\star} = U_{i\star}\tau$ since $A = U\tau$. Clearly, $\sum_{i=1}^n X_i = \|SAx\|^p$. In addition, since $E[X_i] = |A_{i\star}x|^p$, it follows that $\sum_{i=1}^n E[X_i] = \|Ax\|^p$. To bound (8), first note that

$$\sum_{i=1}^{n} (X_i - E[X_i]) = \sum_{i:p_i<1} (X_i - E[X_i]) . \tag{9}$$

Equation (9) follows since, according to the definition of p_i in (7), p_i may equal 1 for some rows, and since these rows are always included in the random sample, $X_i = E[X_i]$ for these rows. To bound the right-hand side of (9), note that for all i such that $p_i < 1$,

$$\begin{aligned} |A_{i\star}x|^p / p_i &\leq \|U_{i\star}\|_p^p \|\tau x\|_q^p / p_i && \text{(by Hölder's inequality)} \\ &\leq |||U|||_p^p \|\tau x\|_q^p / r && \text{(by (7))} \\ &\leq (\alpha\beta)^p \|Ax\|^p / r && \text{(by Definition 4 and Theorem 5).} \end{aligned} \tag{10}$$

From (10) it follows that for each i such that $p_i < 1$,

$$X_i - E[X_i] \leq X_i \leq |A_{i\star}x|^p / p_i \leq (\alpha\beta)^p \|Ax\|^p / r.$$

Thus, we may define $\Delta = (\alpha\beta)^p \|Ax\|^p / r$. In addition, it also follows from (10) that

$$\begin{aligned} \sum_{i:p_i<1} E\left[X_i^2\right] &= \sum_{i:p_i<1} |A_{i\star}x|^p \frac{|A_{i\star}x|^p}{p_i} \\ &\leq \frac{(\alpha\beta)^p \|Ax\|^p}{r} \sum_{i:p_i<1} |A_{i\star}x|^p && \text{(by (10))} \\ &\leq (\alpha\beta)^p \|Ax\|^{2p} / r, \end{aligned}$$

from which it follows that $\sum_{i:p_i<1} \sigma_i^2 = \sum_{i:p_i<1} E\left[X_i^2\right] - (E[X_i])^2 \leq \sum_{i:p_i<1} E\left[X_i^2\right] \leq (\alpha\beta)^p \|Ax\|^{2p} / r$.

To apply the upper tail bound in Theorem 3, define $\gamma = ((1+\epsilon/4)^p - 1)\|Ax\|^p$. It follows that $\gamma^2 \geq (p\epsilon/4)^2 \|Ax\|^{2p}$ and also that

$$\begin{aligned} 2\sum_{i:p_i<1} \sigma_i^2 + 2\gamma\Delta/3 &\leq 2(\alpha\beta)^p \|Ax\|^{2p} / r + 2((1+\epsilon/4)^p - 1)(\alpha\beta)^p \|Ax\|^{2p} / 3r \\ &\leq \left(\frac{2}{3}\left(\frac{5}{4}\right)^p + \frac{4}{3}\right)(\alpha\beta)^p \|Ax\|^{2p} / r \\ &\leq (2^p+2)(\alpha\beta)^p \|Ax\|^{2p} / r, \end{aligned}$$

where the second inequality follows by standard manipulations since $\epsilon \leq 1$ and since $p \geq 1$. Thus, by (3) of Theorem 3, it follows that

$$\begin{aligned}\Pr\left[\|SAx\|^p > \|Ax\|^p + \gamma\right] &= \Pr\left[\sum_{i:p_i<1} X_i > E\left[\sum_{i:p_i<1} X_i\right] + \gamma\right] \\ &\leq \exp\left(\frac{-\gamma^2}{2\sum_{i:p_i<1}\sigma_i^2 + 2\gamma\Delta/3}\right) \\ &\leq \exp\left(\frac{-\epsilon^2 p^2 r}{16(2^p+2)(\alpha\beta)^p}\right).\end{aligned}$$

Similarly, to apply the lower tail bound of (2) of Theorem 3, define $\gamma = (1-(1-\epsilon/4)^p)\|Ax\|^p$. Since $\gamma \geq \epsilon\|Ax\|^p/4$, we can follow a similar line of reasoning to show that

$$\begin{aligned}\Pr\left[\|SAx\|^p < \|Ax\|^p - \gamma\right] &\leq \exp\left(\frac{-\gamma^2}{2\sum_{i:p_i<1}\sigma_i^2}\right) \\ &\leq \exp\left(\frac{-\epsilon^2 r}{32(\alpha\beta)^p}\right).\end{aligned}$$

Choosing $r \geq 16(2^p+2)(\alpha\beta)^p(d\ln(\frac{12}{\epsilon}) + \ln(\frac{2}{\delta}))/(p^2\epsilon^2)$, we get that for every fixed x, the following is true with probability at least $1 - \left(\frac{\epsilon}{12}\right)^d \delta$:

$$(1-\epsilon/4)^p\|Ax\|^p \leq \|SAx\|^p \leq (1+\epsilon/4)^p\|Ax\|^p.$$

Now, consider the ball $B = \{y \in \mathbb{R}^n : y = Ax, \|y\| \leq 1\}$, and consider an ε-net for B, with $\varepsilon = \epsilon/4$. The number of points in the ε-net is $\left(\frac{12}{\epsilon}\right)^d$. Thus, by the union bound, with probability $1-\delta$, (8) holds for all points in the ε-net. Now, to show that with the same probability (8) holds for all points $y \in B$, let $y^* \in B$ be such that $|\|Sy\| - \|y\||$ is maximized, and let $\eta = \sup\{|\|Sy\| - \|y\|| : y \in B\}$. Also, let $y_\varepsilon^* \in B$ be the point in the ε-net that is closest to y^*. By the triangle inequality,

$$\begin{aligned}\eta = |\|Sy^*\| - \|y^*\|| &= |\|Sy_\varepsilon^* + S(y^*-y_\varepsilon^*)\| - \|y_\varepsilon^* + (y^*-y_\varepsilon^*)\|| \\ &\leq |\|Sy_\varepsilon^*\| + \|S(y^*-y_\varepsilon^*)\| - \|y_\varepsilon^*\| + 2\|y^*-y_\varepsilon^*\| - \|y^*-y_\varepsilon^*\|| \\ &\leq |\|Sy_\varepsilon^*\| - \|y_\varepsilon^*\|| + |\|S(y^*-y_\varepsilon^*)\| - \|y^*-y_\varepsilon^*\|| + 2\|y^*-y_\varepsilon^*\| \\ &\leq \epsilon/4\|y_\varepsilon^*\| + \epsilon\eta/4 + \epsilon/2,\end{aligned}$$

where the last inequality follows since $\|y^*-y_\varepsilon^*\| \leq \varepsilon$, $(y^*-y_\varepsilon^*)/\varepsilon \in B$, and

$$|\|S(y^*-y_\varepsilon^*)/\varepsilon\| - \|(y^*-y_\varepsilon^*)/\varepsilon\|| \leq \eta.$$

Therefore, $\eta \leq \epsilon$ since $\|y_\varepsilon^*\| \leq 1$ and since we assume $\epsilon \leq 1/7$. Thus, (8) holds for all points $y \in B$, with probability at least $1-\delta$. Similarly, it holds for any $y \in \mathbb{R}^n$ such that $y = Ax$, since $y/\|y\| \in B$ and since $\|S(y/\|y\|) - y/\|y\|\| \leq \epsilon$ implies that $\|Sy - y\| \leq \epsilon\|y\|$, which completes the proof of the theorem. ∎

Several things should be noted about this result. First, it implies that $\text{rank}(SA) = \text{rank}(A)$, since otherwise we could choose a vector $x \in \text{null}(SA)$ and violate the theorem. In this sense, this theorem generalizes the subspace-preservation result of Lemma 4.1 of [13] to all $p \in [1,\infty)$. Second, regarding sampling complexity: if $p < 2$ the sampling complexity is $O(d^{\frac{p}{2}+2})$, if $p = 2$ it is $O(d^2)$, and if $p > 2$ it is $O(d(d^{\frac{1}{p}+\frac{1}{2}}d^{\frac{1}{q}-\frac{1}{2}})^p) = O(d^{p+1})$. Finally, note that this theorem is analogous to the main result of Schechtman [23], which uses the notion of Auerbach bases.

4. The sampling algorithm.

4.1. Statement of our main algorithm and theorem. Our main sampling algorithm for approximating the solution to the ℓ_p regression problem is presented in Figure 1. The algorithm takes as input an $n \times m$ matrix A of rank d, a vector $b \in \mathbb{R}^n$, and a number $p \in [1, \infty)$. It is a two-stage algorithm that returns as output a vector $\hat{x}_{\text{OPT}} \in \mathbb{R}^m$ (or a vector $\hat{x}_c \in \mathbb{R}^m$ if only the first stage is run). In either case, the output is the solution to the induced ℓ_p regression subproblem constructed on the randomly sampled constraints. Note that the set of constraints r_2 extracted by the second stage of the algorithm is a coreset for the ℓ_p regression problem.

Input: An $n \times m$ matrix A of rank d, a vector $b \in \mathbb{R}^n$, and $p \in [1, \infty)$.

Let $0 < \epsilon < 1/7$, and define $k = \max\{p/2 + 1, p\}$.

- Find a p–well-conditioned basis $U \in \mathbb{R}^{n \times d}$ for span(A) (as in the proof of Theorem 5).
- **Stage 1:** Define $p_i = \min\{1, \frac{\|U_{i\star}\|_p^p}{|||U|||_p^p} r_1\}$, where $r_1 = 16(2^p + 2)d^k \left(d \ln(8 \cdot 12) + \ln(200)\right)$.
  - Generate (implicitly) S where $S_{ii} = 1/p_i^{1/p}$ with probability p_i and 0 otherwise.
  - Let $\hat{x}_c$ be the solution to $\min_{x \in \mathbb{R}^m} \|S(Ax - b)\|_p$.
- **Stage 2:** Let $\hat{\rho} = A\hat{x}_c - b$, and unless $\hat{\rho} = 0$, define $q_i = \min\{1, \max\{p_i, \frac{|\hat{\rho}_i|^p}{\|\hat{\rho}\|_p^p} r_2\}\}$ with $r_2 = \frac{150 \cdot 24^p d^k}{\epsilon^2} \left(d \ln(\frac{280}{\epsilon}) + \ln(200)\right)$.
  - Generate (implicitly, a new) T where $T_{ii} = 1/q_i^{1/p}$ with probability q_i and 0 otherwise.
  - Let $\hat{x}_{\text{OPT}}$ be the solution to $\min_{x \in \mathbb{R}^m} \|T(Ax - b)\|_p$.

Output: $\hat{x}_{\text{OPT}}$ (or $\hat{x}_c$ if only the first stage is run).

FIG. 1. *Sampling algorithm for ℓ_p regression.*

The algorithm first computes a p–well-conditioned basis U for span(A), as described in the proof of Theorem 5. Then, in the first stage, the algorithm uses information from the norms of the rows of U to sample constraints from the input ℓ_p regression problem. In particular, roughly $O(d^{p+1})$ rows of A, and the corresponding elements of b, are randomly sampled according to the probability distribution given by

$$p_i = \min\left\{1, \frac{\|U_{i\star}\|_p^p}{|||U|||_p^p} r_1\right\}, \text{ where } r_1 = 16(2^p + 2)d^k \left(d \ln(8 \cdot 12) + \ln(200)\right), \tag{11}$$

implicitly represented by a diagonal sampling matrix S, where each $S_{ii} = 1/p_i^{1/p}$. For the remainder of the paper, we will use S to denote the sampling matrix for the first-stage sampling probabilities. The algorithm then solves, using any ℓ_p solver of one's choice, the smaller subproblem. If the solution to the induced subproblem is denoted $\hat{x}_c$, then, as we will see in Theorem 7, this is an 8-approximation to the original problem.[5]

In the second stage, the algorithm uses information from the residual of the 8-approximation computed in the first stage to refine the sampling probabilities. Define

[5]For $p = 2$, Drineas, Mahoney, and Muthukrishnan show that this first stage actually leads to a $(1+\epsilon)$-approximation. For $p = 1$, Clarkson develops a subgradient-based algorithm and runs it, after preprocessing the input, on all the input constraints to obtain a constant factor approximation in a stage analogous to our first stage. Here, however, we solve an ℓ_p regression problem on a small subset of the constraints to obtain the constant factor approximation. Moreover, our procedure works for all $p \in [1, \infty)$.

the residual $\hat{\rho} = A\hat{x}_c - b$ (and note that $\|\hat{\rho}\|_p \leq 8\,\mathcal{Z}$). Then, roughly $O(d^{p+1}/\epsilon^2)$ rows of A, and the corresponding elements of b, are randomly sampled according to the probability distribution

(12)
$$q_i = \min\left\{1, \max\left\{p_i, \frac{|\hat{\rho}_i|^p}{\|\hat{\rho}\|_p^p} r_2\right\}\right\}, \text{where } r_2 = \frac{150 \cdot 24^p d^k}{\epsilon^2}\left(d\ln\left(\frac{280}{\epsilon}\right) + \ln(200)\right).$$

As before, this can be represented as a diagonal sampling matrix T, where each $T_{ii} = 1/q_i^{1/p}$ with probability q_i and 0 otherwise. For the remainder of the paper, we will use T to denote the sampling matrix for the second-stage sampling probabilities. Again, the algorithm solves, using any ℓ_p solver of one's choice, the smaller subproblem. If the solution to the induced subproblem at the second stage is denoted $\hat{x}_{\text{OPT}}$, then, as we will see in Theorem 7, this is a $(1+\epsilon)$-approximation to the original problem.[6]

The following is our main theorem for the ℓ_p regression algorithm presented in Figure 1 showing that coresets exist for the ℓ_p regression problem and can be efficiently constructed.

THEOREM 7. *Let A be an $n \times m$ matrix of rank d, let $b \in \mathbb{R}^n$, let $p \in [1, \infty)$, and let $k = \max\{p/2+1, p\}$. Recall that $\epsilon \leq 1/7$, $r_1 = 16(2^p+2)d^k\,(d\ln(8\cdot 12) + \ln(200))$, and $r_2 = \frac{150\cdot 24^p d^k}{\epsilon^2}\left(d\ln(\frac{280}{\epsilon}) + \ln(200)\right)$. Then the following hold.*

- Constant factor approximation. *If only the first stage of the algorithm in Figure 1 is run, then with probability at least 0.6 the solution $\hat{x}_c$ to the sampled problem based on the p_i's of (7) is an 8-approximation to the ℓ_p regression problem.*
- Relative-error approximation. *If both stages of the algorithm are run, then with probability at least 0.5 the solution $\hat{x}_{\text{OPT}}$ to the sampled problem based on the q_i's of (12) is a $(1+\epsilon)$-approximation to the ℓ_p regression problem.*
- Running time. *The ith stage of the algorithm runs in time $O(nmd + nd^5 \log n + \phi(20ir_i, m))$, where $\phi(s,t)$ is the time taken to solve the regression problem $\min_{x\in\mathbb{R}^t} \|A'x - b'\|_p$, where $A' \in \mathbb{R}^{s\times t}$ is of rank d and $b' \in \mathbb{R}^s$.*

Note that since the algorithm of Figure 1 constructs the (α, β, p)–well-conditioned basis U using the procedure in the proof of Theorem 5, our sampling complexity depends on α and β. In particular, it will be $O(d(\alpha\beta)^p)$. Thus, if $p < 2$, our sampling complexity is $O(d \cdot d^{\frac{p}{2}+1}) = O(d^{\frac{p}{2}+2})$; if $p > 2$, it is $O(d(d^{\frac{1}{p}+\frac{1}{2}} d^{\frac{1}{q}-\frac{1}{2}})^p) = O(d^{p+1})$; and (although not explicitly stated, our proof will make it clear that) if $p = 2$, it is $O(d^2)$. Note also that we have stated the claims of the theorem as holding with constant probability, but they can be shown to hold with probability at least $1-\delta$ by using standard amplification techniques.

4.2. Proof for first-stage sampling: Constant factor approximation. To prove the claims of Theorem 7 having to do with the output of the algorithm after the first stage of sampling, we begin with two lemmas. First note that, because of our choice of r_1, we can use the subspace-preserving Theorem 6 with only a constant distortion $\epsilon = \frac{1}{8}$ and $\delta = \frac{1}{100}$; i.e., for all x, we have

(13)
$$\frac{7}{8}\|Ax\|_p \leq \|SAx\|_p \leq \frac{9}{8}\|Ax\|_p$$

[6] The subspace-based sampling probabilities (11) are similar to those used by Drineas, Mahoney, and Muthukrishnan [13], while the residual-based sampling probabilities (12) are similar to those used by Clarkson [11].

with probability at least 0.99. The first lemma below now states that the optimal solution to the original problem provides a small (constant factor) residual when evaluated in the sampled problem.

For simplicity of notation, we again drop the p-subscript from the norm notation, except where it might become confusing.

LEMMA 8. *$\|S(Ax_{\text{OPT}} - b)\| \leq 3\mathcal{Z}$, with probability at least $1 - 1/3^p$.*

Proof. Define $X_i = (S_{ii}|A_{i\star}x_{\text{OPT}} - b_i|)^p$. Thus, $\sum_i X_i = \|S(Ax_{\text{OPT}} - b)\|^p$, and the first moment is $E[\sum_i X_i] = \|Ax_{\text{OPT}} - b\|^p = \mathcal{Z}$. The lemma follows since, by Markov's inequality,

$$\Pr\left[\sum_i X_i > 3^p E\left[\sum_i X_i\right]\right] \leq \frac{1}{3^p};$$

i.e., $\|S(Ax_{\text{OPT}} - b)\|^p > 3^p \|Ax_{\text{OPT}} - b\|^p$ with probability no more than $1/3^p$. ∎

The next lemma states that if the solution to the sampled problem provides a constant factor approximation (when evaluated in the sampled problem), then when this solution is evaluated in the original regression problem we get a (slightly weaker) constant factor approximation.

LEMMA 9. *If $\|S(A\hat{x}_c - b)\| \leq 3\,\mathcal{Z}$, then with probability 0.99, $\|A\hat{x}_c - b\| \leq 8\,\mathcal{Z}$.*

Proof. We will prove the contrapositive: If $\|A\hat{x}_c - b\| > 8\,\mathcal{Z}$, then $\|S(A\hat{x}_c - b)\| > 3\,\mathcal{Z}$. To do so, note that, by Theorem 6 and the choice of r_1, we have that, with probability 0.99,

$$\frac{7}{8}\|Ax\|_p \leq \|SAx\|_p \leq \frac{9}{8}\|Ax\|_p.$$

Using this,

$$\begin{aligned}
\|S(A\hat{x}_c - b)\| &\geq \|SA(\hat{x}_c - x_{\text{OPT}})\| - \|S(Ax_{\text{OPT}} - b)\| && \text{(by the triangle inequality)}\\
&\geq \frac{7}{8}\|A\hat{x}_c - Ax_{\text{OPT}}\| - 3\,\mathcal{Z} && \text{(by Theorem 6 and Lemma 8)}\\
&\geq \frac{7}{8}(\|A\hat{x}_c - b\| - \|Ax_{\text{OPT}} - b\|) - 3\,\mathcal{Z} && \text{(by the triangle inequality)}\\
&> \frac{7}{8}(8\,\mathcal{Z} - \mathcal{Z}) - 3\,\mathcal{Z} && \text{(by the premise } \|A\hat{x}_c - b\| > 8\,\mathcal{Z})\\
&> 3\,\mathcal{Z},
\end{aligned}$$

which establishes the lemma. ∎

Clearly, $\|S(A\hat{x}_c - b)\| \leq \|S(Ax_{\text{OPT}} - b)\|$ (since $\hat{x}_c$ is an optimum for the sampled ℓ_p regression problem). Combining this with Lemmas 8 and 9, it follows that the solution $\hat{x}_c$ to the sampled problem based on the p_i's of (7) satisfies $\|A\hat{x}_c - b\| \leq 8\,\mathcal{Z}$; i.e., $\hat{x}_c$ is an 8-approximation to the original $\mathcal{Z}$.

To conclude the proof of the claims for the first stage of sampling, note that by our choice of r_1, Theorem 6 fails to hold for our first-stage sampling with probability no greater than $1/100$. In addition, the inequality in Lemma 8 fails to hold with probability no greater than $1/3^p$, which is no greater than $1/3$ for all $p \in [1, \infty)$. Finally, let $\widehat{r_1}$ be a random variable representing the number of rows chosen by our sampling scheme, and note that $E[\widehat{r_1}] \leq r_1$. By Markov's inequality, it follows that $\widehat{r_1} > 20r_1$ with probability less than $1/20$. Thus, the first stage of our algorithm fails to give an 8-approximation in the specified running time with a probability bounded by $1/3 + 1/20 + 1/100 < 2/5$.

4.3. Proof for second-stage sampling: Relative-error approximation. The proof of the claims of Theorem 7 having to do with the output of the algorithm after the second stage of sampling will parallel that for the first stage, but it will have several technical complexities that arise since the first triangle inequality approximation in the proof of Lemma 9 is too coarse for relative-error approximation. By our construction, since $q_i \geq p_i$, we have a finer result for subspace preservation. Thus, applying Theorem 6 with $\delta = \frac{1}{100}$, and a constant $\epsilon < \frac{1}{8}$, with probability 0.99, the following holds for all x:

$$(1-\epsilon)\left\|Ax\right\|_p \leq \left\|SAx\right\|_p \leq (1+\epsilon)\left\|Ax\right\|_p. \tag{14}$$

As before, we start with a lemma that states that the optimal solution to the original problem provides a small (now a relative-error) residual when evaluated in the sampled problem. This is the analogue of Lemma 8. An important difference is that the second-stage sampling probabilities significantly enhance the probability of success.

LEMMA 10. $\|T(Ax_{\text{OPT}} - b)\| \leq (1+\epsilon)\mathcal{Z}$ *with probability at least* 0.99.

Proof. Define the random variable $X_i = (T_{ii}|A_{i\star}x_{\text{OPT}} - b_i|)^p$, and recall that $A_{i\star} = U_{i\star}\tau$ since $A = U\tau$. Clearly, $\sum_{i=1}^n X_i = \|T(Ax_{\text{OPT}} - b)\|^p$. In addition, since $E[X_i] = |A_{i\star}x_{\text{OPT}} - b_i|^p$, it follows that $\sum_{i=1}^n E[X_i] = \|Ax_{\text{OPT}} - b\|^p$. We will use (3) of Theorem 3 to provide a bound for $\sum_i (X_i - E[X_i]) = \|T(Ax_{\text{OPT}} - b)\|^p - \|Ax_{\text{OPT}} - b\|^p$.

From the definition of q_i in (12), it follows that for some of the rows, q_i may equal 1 (just as in the proof of Theorem 6). Since $X_i = E[X_i]$ for these rows, $\sum_i (X_i - E[X_i]) = \sum_{i:q_i<1} (X_i - E[X_i])$, and thus we will bound this latter quantity with (3). To do so, we must first provide a bound for $X_i - E[X_i] \leq X_i$ and for $\sum_{i:q_i<1} \sigma_i^2 \leq \sum_i E\left[X_i^2\right]$. To that end, note that

$$\begin{aligned}
|A_{i\star}(x_{\text{OPT}} - \hat{x}_c)| &\leq \left\|U_{i\star}\right\|_p \left\|\tau(x_{\text{OPT}} - \hat{x}_c)\right\|_q \qquad \text{(by Hölders inequality)} \\
&\leq \left\|U_{i\star}\right\|_p \beta \left\|U\tau(x_{\text{OPT}} - \hat{x}_c)\right\|_p \qquad \text{(by Definition 4 and Theorem 5)} \\
&\leq \left\|U_{i\star}\right\|_p \beta \quad (\|Ax_{\text{OPT}} - b\| + \|A\hat{x}_c - b\|) \text{ (by the triangle inequality)} \\
&\leq \left\|U_{i\star}\right\|_p \beta 9\mathcal{Z}, \qquad (15)
\end{aligned}$$

where the final inequality follows from the definition of $\mathcal{Z}$ and the results from the first stage of sampling. Next, note that from the conditions on the probabilities q_i in (12), as well as by Definition 4 and the output of the first stage of sampling, it follows that

$$\frac{|\hat{\rho}_i|^p}{q_i} \leq \frac{\|\hat{\rho}\|^p}{r_2} \leq \frac{8^p\mathcal{Z}^p}{r_2} \quad \text{and} \quad \frac{\|U_{i\star}\|^p}{q_i} \leq \frac{\|\|U\|\|^p}{r_2} \leq \frac{\alpha^p}{r_2} \tag{16}$$

for all i such that $q_i < 1$.

Thus, since $X_i - E[X_i] \leq X_i \leq |A_{i\star}x_{\text{OPT}} - b_i|^p/q_i$, it follows that for all i such that $q_i < 1$,

$$\begin{aligned}
X_i - E[X_i] &\leq \frac{2^{p-1}}{q_i}\left(|A_{i\star}(x_{\text{OPT}} - \hat{x}_c)|^p + |\hat{\rho}_i|^p\right) \qquad (\text{since } \hat{\rho} = A\hat{x}_c - b\ ) \qquad (17) \\
&\leq 2^{p-1}\left(\frac{\|U_{i\star}\|_p^p \beta^p 9^p \mathcal{Z}^p}{q_i} + \frac{|\hat{\rho}_i|^p}{q_i}\right) \qquad \text{(by (15))} \\
&\leq 2^{p-1}\left(\alpha^p\beta^p 9^p\mathcal{Z}^p + 8^p\mathcal{Z}^p\right)/r_2 \qquad \text{(by (16))} \\
&\leq c_p(\alpha\beta)^p\mathcal{Z}^p/r_2, \qquad (18)
\end{aligned}$$

where we set $c_p = 2^{p-1}(9^p + 8^p) \le 18^p$. Thus, we may define $\Delta = c_p(\alpha\beta)^p \mathcal{Z}^p / r_2$. In addition, it follows that

$$\begin{aligned}\sum_{i:q_i<1} E\left[X_i^2\right] &= \sum_{i:q_i<1} |A_{i\star}x_{\text{OPT}} - b_i|^p \frac{|A_{i\star}x_{\text{OPT}} - b_i|^p}{q_i} \\ &\le \Delta \sum_i |A_{i\star}x_{\text{OPT}} - b_i|^p \qquad \text{(by (18))} \\ &\le c_p(\alpha\beta)^p \mathcal{Z}^{2p}/r_2. \qquad (19)\end{aligned}$$

To apply the upper tail bound of (3) of Theorem 3, define $\gamma = ((1+\epsilon)^p - 1)\mathcal{Z}^p$. We have $\gamma \ge p\epsilon\, \mathcal{Z}^p$, and since $\epsilon \le 1/7$, we also have $\gamma \le \left(\left(\frac{8}{7}\right)^p - 1\right) \mathcal{Z}^p$. Hence, by (3) of Theorem 3, it follows that

$$\begin{aligned}\ln \Pr\left[\|T(Ax_{\text{OPT}} - b)\|^p > \|Ax_{\text{OPT}} - b\|^p + \gamma\right] &\le \frac{-\gamma^2}{2\sum_{i:q_i<1} \sigma_i^2 + 2\gamma\Delta/3} \\ &\le \frac{-p^2\epsilon^2 r_2}{\left(2c_p + \frac{2c_p}{3}\left(\left(\frac{8}{7}\right)^p - 1\right)\right)(\alpha\beta)^p} \\ &\le \frac{-p^2\epsilon^2 r_2}{3 \cdot 18^p (\alpha\beta)^p}.\end{aligned}$$

Thus, $\Pr\left[\|T(Ax_{\text{OPT}} - b)\| > (1+\epsilon)\mathcal{Z}\right] \le \exp(\frac{-p^2\epsilon^2 r_2}{3\cdot 18^p(\alpha\beta)^p})$, from which the lemma follows by our choice of r_2. ∎

Next we show that if the solution to the sampled problem provides a relative-error approximation (when evaluated in the sampled problem), then when this solution is evaluated in the original regression problem we get a (slightly weaker) relative-error approximation. We first establish two technical lemmas.

The following lemma says that for all optimal solutions $\hat{x}_{\text{OPT}}$ to the second-stage sampled problem, $A\hat{x}_{\text{OPT}}$ is not too far from $A\hat{x}_c$, where $\hat{x}_c$ is the optimal solution from the first stage, in a p-norm sense. Hence, the lemma will allow us to restrict our calculations in Lemmas 12 and 13 to the ball of radius $12\,\mathcal{Z}$ centered at $A\hat{x}_c$.

LEMMA 11. $\|A\hat{x}_{\text{OPT}} - A\hat{x}_c\| \le 12\,\mathcal{Z}$ *with probability* 0.98.

Proof. With probability 0.98, both the inequalities in Lemma 9 and condition (14) hold true. By two applications of the triangle inequality, it follows that

$$\begin{aligned}\|A\hat{x}_{\text{OPT}} - A\hat{x}_c\| &\le \|A\hat{x}_{\text{OPT}} - Ax_{\text{OPT}}\| + \|Ax_{\text{OPT}} - b\| + \|A\hat{x}_c - b\| \\ &\le \|A\hat{x}_{\text{OPT}} - Ax_{\text{OPT}}\| + 9\mathcal{Z},\end{aligned}$$

where the second inequality follows since $\|A\hat{x}_c - b\| \le 8\,\mathcal{Z}$ from the first stage of sampling and since $\mathcal{Z} = \|Ax_{\text{OPT}} - b\|$. In addition, we have that

$$\begin{aligned}\|Ax_{\text{OPT}} - A\hat{x}_{\text{OPT}}\| &\le \frac{1}{(1-\epsilon)} \|T(A\hat{x}_{\text{OPT}} - Ax_{\text{OPT}})\| \qquad \text{(by Theorem 6)} \\ &\le (1+2\epsilon)\left(\|T(A\hat{x}_{\text{OPT}} - b)\| + \|T(Ax_{\text{OPT}} - b)\|\right) \\ &\qquad\qquad \text{(by the triangle inequality)} \\ &\le 2(1+2\epsilon)\|T(Ax_{\text{OPT}} - b)\| \\ &\le 2(1+2\epsilon)(1+\epsilon)\|Ax_{\text{OPT}} - b\| \qquad \text{(by Lemma 10)}\ ,\end{aligned}$$

where the third inequality follows since $\hat{x}_{\text{OPT}}$ is optimal for the sampled problem. The lemma follows since $\epsilon \le 1/8$. ∎

Thus, if we define the affine ball of radius $12\,\mathcal{Z}$ that is centered at $A\hat{x}_c$ and that lies in span(A),

$$B = \{y \in \mathbb{R}^n : y = Ax, x \in \mathbb{R}^m, \|A\hat{x}_c - y\| \leq 12\,\mathcal{Z}\}, \tag{20}$$

then Lemma 11 states that $A\hat{x}_{\text{OPT}} \in B$ for all optimal solutions $\hat{x}_{\text{OPT}}$ to the sampled problem. Let us consider an ε-net, and call it B_ε with $\varepsilon = \epsilon\,\mathcal{Z}$ for this ball B. Using arguments from [8], since B is a ball in a d-dimensional subspace, the size of the ε-net is $\left(\frac{3\cdot 12\,\mathcal{Z}}{\epsilon\,\mathcal{Z}}\right)^d = \left(\frac{36}{\epsilon}\right)^d$. The next lemma states that for all points in the ε-net, if that point provides a relative-error approximation (when evaluated in the sampled problem), then when this point is evaluated in the original regression problem we get a (slightly weaker) relative-error approximation.

LEMMA 12. *For all points Ax_ε in the ε-net, B_ε, if $\|T(Ax_\varepsilon - b)\| \leq (1+3\epsilon)\mathcal{Z}$, then $\|Ax_\varepsilon - b\| \leq (1+6\epsilon)\mathcal{Z}$ with probability* 0.99.

Proof. Fix a given point $y^*_\varepsilon = Ax^*_\varepsilon \in B_\varepsilon$. We will prove the contrapositive for this point; i.e., we will prove that if $\|Ax^*_\varepsilon - b\| > (1+6\epsilon)\mathcal{Z}$, then $\|T(Ax^*_\varepsilon - b)\| > (1+3\epsilon)\mathcal{Z}$ with probability at least $1 - \frac{1}{100}\left(\frac{\epsilon}{36}\right)^d$. The lemma will then follow from the union bound.

To this end, define the random variable $X_i = (T_{ii}|A_{i\star}x^*_\varepsilon - b_i|)^p$, and recall that $A_{i\star} = U_{i\star}\tau$ since $A = U\tau$. Clearly, $\sum_{i=1}^n X_i = \|T(Ax^*_\varepsilon - b)\|^p$. In addition, since $E[X_i] = |A_{i\star}x^*_\varepsilon - b_i|^p$, it follows that $\sum_{i=1}^n E[X_i] = \|Ax^*_\varepsilon - b\|^p$. We will use (2) of Theorem 3 to provide an upper bound for the probability of the event that $\|T(Ax^*_\varepsilon - b)\|^p \leq \|Ax^*_\varepsilon - b\|^p - \gamma$, where $\gamma = \|Ax^*_\varepsilon - b\|^p - (1+3\epsilon)^p\mathcal{Z}^p$, under the assumption that $\|Ax^*_\varepsilon - b\| > (1+6\epsilon)\mathcal{Z}$.

From the definition of q_i in (12), it follows that for some of the rows, q_i may equal 1 (just as in the proof of Theorem 6). Since $X_i = E[X_i]$ for these rows, $\sum_i (X_i - E[X_i]) = \sum_{i:p_i<1}(X_i - E[X_i])$, and thus we will bound this latter quantity with (2). To do so, we must first provide a bound for $\sum_{i:q_i<1} E\left[X_i^2\right]$. To that end, note that

$$\begin{aligned}
|A_{i\star}(x^*_\varepsilon - \hat{x}_c)| &= |U_{i\star}\tau(x^*_\varepsilon - \hat{x}_c)| \\
&\leq \|U_{i\star}\|_p \, \|\tau(x^*_\varepsilon - \hat{x}_c)\|_q && \text{(by Hölders inequality)} \\
&\leq \|U_{i\star}\|_p \, \beta \, \|U\tau(x^*_\varepsilon - \hat{x}_c)\|_p && \text{(by Definition 4 and Theorem 5)} \\
&\leq \|U_{i\star}\| \, \beta 12\mathcal{Z}, && (22)
\end{aligned} \tag{21}$$

where the final inequality follows from the radius of the high-dimensional ball in which the ε-net resides. From this, we can show that

$$\begin{aligned}
\frac{|A_{i\star}x^*_\varepsilon - b_i|}{q_i} &\leq \frac{2^{p-1}}{q_i}\left(|A_{i\star}x^*_\varepsilon - A_{i\star}\hat{x}_c|^p + |\hat{\rho}_i|^p\right) && \text{(since } \hat{\rho} = A\hat{x}_c - b \text{)} \\
&\leq 2^{p-1}\left(\frac{\|U_{i\star}\|^p \, 12^p\beta^p\mathcal{Z}^p}{q_i} + \frac{|\hat{\rho}_i|^p}{q_i}\right) && \text{(by (22))} \\
&\leq 2^{p-1}\left(\alpha^p 12^p\beta^p\mathcal{Z}^p + 8^p\mathcal{Z}^p\right)/r_2 && \text{(by (16))} \\
&\leq 24^p(\alpha\beta)^p\mathcal{Z}^p/r_2. && (23)
\end{aligned}$$

Therefore, we have that

$$\begin{aligned}\sum_{i:q_i<1} E\left[X_i^2\right] &= \sum_{i:q_i<1} |A_{i\star}x_\varepsilon^* - b_i|^p \frac{|A_{i\star}x_\varepsilon^* - b_i|^p}{q_i} \\ &\le \frac{24^p(\alpha\beta)^p \mathcal{Z}^p}{r_2} \sum_i |A_{i\star}x_\varepsilon^* - b_i|^p \qquad \text{(by (23))} \\ &\le 24^p(\alpha\beta)^p \left\|Ax_\varepsilon^* - b\right\|^{2p} / r_2. \qquad (24)\end{aligned}$$

To apply the lower tail bound of (2) of Theorem 3, define $\gamma = \|Ax_\varepsilon^* - b\|^p - (1+3\epsilon)^p\mathcal{Z}^p$. Thus, by (24) and by (2) of Theorem 3 it follows that

$$\begin{aligned}\ln[\|T(Ax_\varepsilon^* - b)\|^p &\le (1+3\epsilon)^p\mathcal{Z}^p] \\ &\le \frac{-r_2(\|Ax_\varepsilon^* - b\|^p - (1+3\epsilon)^p\mathcal{Z}^p)^2}{24^p(\alpha\beta)^p \|Ax_\varepsilon^* - b\|^{2p}} \\ &\le \frac{-r_2}{24^p(\alpha\beta)^p}\left(1 - \frac{(1+3\epsilon)^p\mathcal{Z}^p}{\|Ax_\varepsilon^* - b\|^p}\right)^2 \\ &< \frac{-r_2}{24^p(\alpha\beta)^p}\left(1 - \frac{(1+3\epsilon)^p\mathcal{Z}^p}{(1+6\epsilon)^p\mathcal{Z}^p}\right)^2 \qquad \text{(by the premise)} \\ &\le \frac{-r_2\epsilon^2}{24^p(\alpha\beta)^p}.\end{aligned}$$

The last line can be justified by the fact that $(1+3\epsilon)/(1+6\epsilon) \le 1-\epsilon$ since $\epsilon \le 1/3$, and that $(1-\epsilon)^p$ is maximized at $p=1$. Since $r_2 \ge 24^p(\alpha\beta)^p(d\ln(\frac{36}{\epsilon}) + \ln(200))/\epsilon^2$, it follows that $\|T(Ax_\varepsilon^* - b)\| \le (1+3\epsilon)\mathcal{Z}$ with probability no greater than $\frac{1}{200}\left(\frac{\epsilon}{36}\right)^d$. Since there are no more than $\left(\frac{36}{\epsilon}\right)^d$ such points in the ε-net, the lemma follows by the union bound. □

Finally, the next lemma states that if the solution to the sampled problem (in the second stage of sampling) provides a relative-error approximation (when evaluated in the sampled problem), then when this solution is evaluated in the original regression problem we get a (slightly weaker) relative-error approximation. This is the analogue of Lemma 9, and its proof will use Lemma 12.

LEMMA 13. *If* $\|T(A\hat{x}_{\text{OPT}} - b)\| \le (1+\epsilon)\mathcal{Z}$, *then* $\|A\hat{x}_{\text{OPT}} - b\| \le (1+7\epsilon)\mathcal{Z}$.

Proof. We will prove the contrapositive: If $\|A\hat{x}_{\text{OPT}} - b\| > (1+7\epsilon)\mathcal{Z}$, then $\|T(A\hat{x}_{\text{OPT}} - b)\| > (1+\epsilon)\mathcal{Z}$. Since $A\hat{x}_{\text{OPT}}$ lies in the ball B defined by (20) and since the ε-net is constructed in this ball, there exists a point $y_\varepsilon = Ax_\varepsilon$ (call it Ax_ε^*), such that $\|A\hat{x}_{\text{OPT}} - Ax_\varepsilon^*\| \le \epsilon\,\mathcal{Z}$. Thus,

$$\begin{aligned}\|Ax_\varepsilon^* - b\| &\ge \|A\hat{x}_{\text{OPT}} - b\| - \|Ax_\varepsilon^* - A\hat{x}_{\text{OPT}}\| \qquad \text{(by the triangle inequality)} \\ &\ge (1+7\epsilon)\mathcal{Z} - \epsilon\mathcal{Z} \qquad \text{(by assumption and the definition of } Ax_\varepsilon^*\text{)} \\ &= (1+6\epsilon)\mathcal{Z}.\end{aligned}$$

Next, since Lemma 12 holds for all points Ax_ε in the ε-net, it follows that

$$\|T(Ax_\varepsilon^* - b)\| > (1+3\epsilon)\mathcal{Z}. \qquad (25)$$

Finally, note that

$$\begin{aligned}
\|T(A\hat{x}_{\text{OPT}} - b)\| &\geq \|T(Ax^*_\varepsilon - b)\| - \|TA(x^*_\varepsilon - \hat{x}_{\text{OPT}})\| && \text{(by the triangle inequality)}\\
&> (1+3\epsilon)\mathcal{Z} - (1+\epsilon)\,\|A(x^*_\varepsilon - \hat{x}_{\text{OPT}})\| && \text{(by (25) and Theorem 6)}\\
&> (1+3\epsilon)\mathcal{Z} - (1+\epsilon)\epsilon\,\mathcal{Z} && \text{(by the definition of } A\hat{x}_\varepsilon)\\
&> (1+\epsilon)\mathcal{Z},
\end{aligned}$$

which establishes the lemma. □

Clearly, $\|T(A\hat{x}_{\text{OPT}} - b)\| \leq \|T(Ax_{\text{OPT}} - b)\|$, since $\hat{x}_{\text{OPT}}$ is an optimum for the sampled ℓ_p regression problem. Combining this with Lemmas 10 and 13, it follows that the solution $\hat{x}_{\text{OPT}}$ to the sampled problem based on the q_i's of (12) satisfies $\|A\hat{x}_{\text{OPT}} - b\| \leq (1+7\epsilon)\,\mathcal{Z}$; i.e., $\hat{x}_{\text{OPT}}$ is a $(1+7\epsilon)$-approximation to the original $\mathcal{Z}$.

To conclude the proof of the claims for the second stage of sampling, note that we can actually replace ϵ by $\epsilon/7$, thus getting the $(1+\epsilon)$-approximation with the corresponding bound on r_2 as in Theorem 7. To bound the failure probability, recall that the first stage failed with probability no greater than 2/5. Note also that by our choice of r_2, Theorem 6 fails to hold for our second-stage sampling with probability no greater than 1/100. In addition, Lemma 10 and Lemma 12 each fails to hold with probability no greater than 2/100 and 1/100, respectively. Finally, let $\widehat{r_2}$ be a random variable representing the number of rows actually chosen by our sampling scheme in the second stage, and note that $E[\widehat{r_2}] \leq 2r_2$. By Markov's inequality, it follows that $\widehat{r_2} > 40r_2$ with probability less than 1/20. Thus, the second stage of our algorithm fails with probability less than $1/20 + 1/100 + 2/100 + 1/100 < 1/10$. By combining both stages, our algorithm fails to give a $(1+\epsilon)$-approximation in the specified running time with a probability bounded from above by $2/5 + 1/10 = 1/2$.

Remark. It has been brought to our attention by an anonymous reviewer that one of the main results of this section can be obtained with a simpler analysis. Via an analysis similar to that of section 4.2, one can show that a relative factor (as opposed to a constant factor) approximation can be obtained in one stage by constructing the sampling probabilities using subspace information from both the data matrix A and the target vector b. In particular, we compute the sampling probabilities from a p–well-conditioned basis for the augmented matrix $[A\ \ b]$ as opposed to only from A. Although it simplifies the analysis, this scheme has the disadvantage that a p–well-conditioned basis needs to be constructed for each target vector b. Using our two-stage algorithm, one need only construct one such basis for A which can subsequently be used to compute probabilities for any target vector b (see, e.g., the extension to *generalized ℓ_p regression* in the next section).

5. Extensions. In this section we outline several immediate extensions of our main algorithmic result.

Constrained ℓ_p regression. Our sampling strategies are transparent to constraints placed on x. In particular, suppose we constrain the output of our algorithm to lie within a convex set $\mathcal{C} \subseteq \mathbb{R}^m$. If there is an algorithm to solve the constrained ℓ_p regression problem $\min_{z\in\mathcal{C}} \|A'x - b'\|$, where $A' \in \mathbb{R}^{s\times m}$ is of rank d and $b' \in \mathbb{R}^s$, in time $\phi(s,m)$, then by modifying our main algorithm in a straightforward manner, we can obtain an algorithm that gives a $(1+\epsilon)$-approximation to the constrained ℓ_p regression problem in time $O(nmd + nd^5 \log n + \phi(40r_2, m))$.

Generalized ℓ_p regression. Our sampling strategies extend to the case of generalized ℓ_p regression: given as input a matrix $A \in \mathbb{R}^{n\times m}$ of rank d, a target

matrix $B \in \mathbb{R}^{n\times p}$, and a real number $p \in [1,\infty)$, find a matrix $X \in \mathbb{R}^{m\times p}$ such that $|||AX - B|||_p$ is minimized. To do so, we generalize our sampling strategies in a straightforward manner. The probabilities p_i for the first stage of sampling are the same as before. Then, if $\hat{X}_c$ is the solution to the first-stage sampled problem, we can define the $n \times p$ matrix $\hat{\rho} = A\hat{X}_c - B$ and define the second-stage sampling probabilities to be $q_i = \min\left(1, \max\{p_i, r_2\|\hat{\rho}_{i\star}\|_p^p/|||\hat{\rho}|||_p^p\}\right)$. Then, we can show that the $\hat{X}_{\text{OPT}}$ computed from the second-stage sampled problem satisfies $|||A\hat{X}_{\text{OPT}} - B|||_p \le (1+\epsilon)\min_{X\in\mathbb{R}^{m\times p}} |||AX - B|||_p$ with probability at least 1/2.

Weighted ℓ_p regression. Our sampling strategies also generalize to the case of ℓ_p regression involving weighted p-norms: if $w_1,\dots,w_m$ are a set of nonnegative weights, then the weighted p-norm of a vector $x \in \mathbb{R}^m$ may be defined as $\|x\|_{p,w} = \left(\sum_{i=1}^m w_i|x_i|^p\right)^{1/p}$, and the weighted analogue of the matrix p-norm $|||\cdot|||_p$ may be defined as $|||U|||_{p,w} = \left(\sum_{j=1}^d \|U_{\star j}\|_{p,w}\right)^{1/p}$. Our sampling scheme proceeds as before. First, we compute a well-conditioned basis U for $\text{span}(A)$ with respect to this weighted p-norm. The sampling probabilities p_i for the first stage of the algorithm are then $p_i = \min(1, r_1 w_i \|U_{i\star}\|_p^p / |||U|||_{p,w}^p)$, and the sampling probabilities q_i for the second stage are $q_i = \min\left(1, \max\{p_i, r_2 w_i|\hat{\rho}_i|^p/\|\hat{\rho}\|_{p,w}^p\}\right)$, where $\hat{\rho}$ is the residual from the first stage.

General sampling probabilities. More generally, consider any sampling probabilities of the form $p_i \ge \min\{1, \max\{\frac{\|U_{i\star}\|_p^p}{|||U|||_p^p}, \frac{|(\rho_{\text{OPT}})_i|^p}{\mathcal{Z}^p}\}r\}$, where $\rho_{\text{OPT}} = Ax_{\text{OPT}} - b$ and $r \ge \frac{36^p d^k}{\epsilon^2}\left(d\ln(\frac{36}{\epsilon}) + \ln(200)\right)$ and where we adopt the convention that $\frac{0}{0} = 0$. Then, by an analysis similar to that presented for our two-stage algorithm, we can show that, by picking $O(36^p d^{p+1}/\epsilon^2)$ rows of A and the corresponding elements of b (in a single stage of sampling) according to these probabilities, the solution $\hat{x}_{\text{OPT}}$ to the sampled ℓ_p regression problem is a $(1+\epsilon)$-approximation to the original problem with probability at least 1/2. (Note that these sampling probabilities, if an equality is used in this expression, depend on the entries of the vector $\rho_{\text{OPT}} = Ax_{\text{OPT}} - b$; in particular, they require the solution of the original problem. This is reminiscent of the results of [13]. Our main two-stage algorithm shows that by solving a problem in the first stage based on coarse probabilities, we can refine our probabilities to approximate these probabilities and thus obtain an $(1+\epsilon)$-approximation to the ℓ_p regression problem more efficiently.)

Acknowledgment. We would like to thank Robert Kleinberg for pointing out several useful references.

REFERENCES

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, *Approximating extent measures of points*, J. ACM, 51 (2004), pp. 606–635.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, *Geometric approximation via coresets*, in Combinatorial and Computational Geometry, J. E. Goodman, J. Pach, and E. Welzl, eds., Math. Sci. Res. Inst. Publ. 52, Cambridge University Press, Cambridge, UK, 2005, pp. 1–30.

[3] N. Ailon and B. Chazelle, *Approximate nearest neighbors and the fast Johnson–Lindenstrauss transform*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, ACM, New York, 2006, pp. 557–563.

[4] H. Auerbach, *On the Area of Convex Curves with Conjugate Diameters*, Ph.D. thesis, University of Lwów, Lwów, Poland, 1930 (in Polish).

[5] B. Awerbuch and R. D. Kleinberg, *Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 45–53.
[6] M. Bădoiu and K. L. Clarkson, *Smaller core-sets for balls*, in Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2003, pp. 801–802.
[7] S. Bernstein, *Theory of Probability*, Moscow, 1927 (in Russian).
[8] J. Bourgain, J. Lindenstrauss, and V. Milman, *Approximation of zonoids by zonotopes*, Acta Math., 162 (1989), pp. 73–141.
[9] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, UK, 2004.
[10] S. Chatterjee, A. S. Hadi, and B. Price, *Regression Analysis by Example*, Wiley Series in Probability and Statistics, Wiley, New York, 2000.
[11] K. L. Clarkson, *Subgradient and sampling algorithms for ℓ_1 regression*, in Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2005, pp. 257–266.
[12] M. Day, *Polygons circumscribed about closed convex curves*, Trans. Amer. Math. Soc., 62 (1947), pp. 315–319.
[13] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, *Sampling algorithms for ℓ_2 regression and applications*, in Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2006, pp. 1127–1136.
[14] D. Feldman, A. Fiat, and M. Sharir, *Coresets for weighted facilities and their applications*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, D.C., 2006, pp. 315–324.
[15] G. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Studies in Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 1996.
[16] S. Har-Peled and S. Mazumdar, *On coresets for k-means and k-median clustering*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 291–300.
[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer-Verlag, New York, 2003.
[18] J. Kleinberg and M. Sandler, *Using mixture models for collaborative filtering*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 569–578.
[19] L. Lovasz, *An Algorithmic Theory of Numbers, Graphs, and Convexity*, CBMS-NSF Regoinal Conf. Ser. in Appl. Math. 50, SIAM, Philadelphia, 1986.
[20] A. Maurer, *A bound on the deviation probability for sums of non-negative random variables*, JIPAM. J. Inequal. Pure Appl. Math., 4 (2003).
[21] J. Matousek, *Lectures on Discrete Geometry*, Grad. Texts in Math., Springer-Verlag, New York, 2002.
[22] T. Sarlós, *Improved approximation algorithms for large matrices via random projections*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, D.C., 2006, pp. 143–152.
[23] G. Schechtman, *More on embedding subspaces of L_p in ℓ_r^n*, Compositio Math., 61 (1987), pp. 159–169.
[24] G. Schechtman and A. Zvavitch, *Embedding subspaces of L_p into ℓ_p^N, $0 < p < 1$*, Math. Nachr., 227 (2001), pp. 133–142.
[25] M. Talagrand, *Embedding subspaces of L_1 into ℓ_1^N*, Proc. Amer. Math. Soc., 108 (1990), pp. 363–369.
[26] M. Talagrand, *Embedding subspaces of L_p into ℓ_p^N*, Oper. Theory Adv. Appl., 77 (1995), pp. 311–325.
[27] A. Taylor, *A geometric theorem and its application to biorthogonal systems*, Bull. Amer. Math. Soc., 53 (1947), pp. 614–616.
[28] P. Wojtaszczyk, *Banach Spaces for Analysts*, Cambridge Stud. Adv. Math. 25, Cambridge University Press, Cambridge, UK, 1991.

SIAM J. COMPUT.
Vol. 38, No. 5, pp. 2079–2112

HIERARCHICAL UNAMBIGUITY*

HOLGER SPAKOWSKI† AND RAHUL TRIPATHI‡

Abstract. We develop techniques to investigate relativized hierarchical unambiguous computation. We apply our techniques to generalize known constructs involving relativized unambiguity based complexity classes (UP and Promise-UP) to new constructs involving arbitrary higher levels of the relativized unambiguous polynomial hierarchy (UPH). Our techniques are developed on constraints imposed by hierarchical arrangement of *unambiguous* nondeterministic polynomial-time Turing machines, and so they differ substantially, in applicability and in nature, from standard methods (such as the switching lemma [J. Håstad, *Computational Limitations of Small-Depth Circuits*, MIT Press, Cambridge, 1987]), which play roles in carrying out similar generalizations. Aside from achieving these generalizations, we resolve a question posed by Cai, Hemachandra, and Vyskoč in [*Complexity Theory*, Cambridge University Press, Cambridge, UK, 1993, pp. 101–146], on an issue related to nonadaptive Turing access to UP and adaptive smart Turing access to Promise-UP.

Key words. unambiguous computation, computational complexity, promise problems, relativization

AMS subject classifications. 68Q05, 68Q10, 68Q15, 03D15

DOI. 10.1137/07068196X

1. Introduction.

1.1. Background. Baker, Gill, and Solovay in their seminal paper [4] introduced the concept of relativization in complexity theory and showed that the bottom levels of the polynomial hierarchy P and NP separate in some relativized world. Baker and Selman [5] made progress in extending this relativized separation to the next levels of the polynomial hierarchy: They proved that there is a relativized world where $\Sigma_2^p \neq \Pi_2^p$. However, Baker and Selman [5] noted that their proof techniques do not apply at higher levels of the polynomial hierarchy because of certain constraints in their counting argument. Thus, it required the development of entirely different proof techniques for separating all the levels of the relativized polynomial hierarchy. The landmark paper by Furst, Saxe, and Sipser [23] established the connection between the relativization of the polynomial hierarchy and lower bounds for small depth circuits computing certain functions. Techniques for proving such lower bounds were developed in a series of papers [23, 45, 49, 29], which were motivated by questions about the relativized structure of the polynomial hierarchy. Yao [49] finally succeeded in separating the levels of the relativized polynomial hierarchy by applying these new techniques. Håstad [29] gave the most refined presentation of these techniques via the *switching lemma*. Even to date, Håstad's switching lemma [29] is used as an essential tool to separate relativized hierarchies, composed of classes stacked one on top of

*Received by the editors February 7, 2007; accepted for publication (in revised form) October 29, 2008; published electronically February 6, 2009. A preliminary version of this paper was presented at the MFCS '06 conference.

http://www.siam.org/journals/sicomp/38-5/68196.html

†Department of Mathematics and Applied Mathematics, University of Cape Town, Rondebosch 7701, South Africa (Holger.Spakowski@uct.ac.za). This work was done in part while this author was at Heinrich-Heine-Universität Düsseldorf. This author's research was supported in part by the DFG under grants RO 1202/9-1 and RO 1202/9-3.

‡Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 (tripathi@cse.usf.edu). This author's research was supported by the New Researcher Grant of the University of South Florida, Tampa.

another. (See, for instance, [29, 37, 9, 46], where the switching lemma is used as a strong tool for proving the feasibility of oracle constructions.)

A major contribution of our paper lies in demonstrating that known oracle constructions involving the initial levels of the unambiguous polynomial hierarchy (UPH) and the promise unambiguous polynomial hierarchy ($\mathcal{UPH}$), i.e., UP and $\mathrm{P}_s^{\text{Promise-UP}}$, respectively, can be extended to oracle constructions involving arbitrary higher levels of UPH by application only of pure counting arguments. Alongside this contribution, we resolve a question posed by Cai, Hemachandra, and Vyskoč [16] concerning the closure of UP under nonadaptive Turing reductions and the closure of Promise-UP under adaptive smart Turing reductions.

The class UP is the unambiguous version of NP. UP has proved to be useful, for instance, in studying worst-case one-to-one one-way functions [35, 26], obtaining potential counterexamples to the Berman–Hartmanis isomorphism conjecture [34], and studying the complexity of closure properties of #P [41]. Lange and Rossmanith [38] generalized the notion of unambiguity to higher levels of the polynomial hierarchy. They introduced the following unambiguity based hierarchies: AUPH, UPH, and $\mathcal{UPH}$. It is known that AUPH $\subseteq$ UPH $\subseteq \mathcal{UPH} \subseteq$ UAP [38, 18], where UAP (unambiguous alternating polynomial-time) is the analogue of UP for alternating polynomial-time Turing machines. These hierarchies received renewed interests in some recent papers (see, for instance, [1, 18, 46, 24]). Spakowski and Tripathi [46], further developing circuit complexity-theoretic proof techniques of Sheu and Long [44], and of Ko [37] obtained results on the relativized structure of these hierarchies. They proved that there is a relativized world where these hierarchies are infinite. They also proved that for each $k \geq 2$ there is a relativized world where these hierarchies collapse so that they have exactly k distinct levels and their kth levels collapse to PSPACE. The present paper supplements this investigation with a focus on the structure of the UPH.

1.2. Results. We prove a combinatorial lemma (Lemma 3.1) and demonstrate its usefulness in generalizing known relativization results involving classes such as UP and Promise-UP to new relativization results that involve arbitrary levels of the UPH.

In subsection 4.1, we use Lemma 3.1 to construct relativized worlds in which certain inclusion relationships between bounded ambiguity classes ($\mathrm{UP}_{O(1)}$ and FewP) and the levels of the UPH do not hold. Theorem 4.1 of this subsection subsumes an oracle result of Beigel [6] for any constant integer $k \geq 1$, and Corollary 4.5 generalizes a result of Cai, Hemachandra, and Vyskoč [16] from the case of $k = 2$ to the case of any arbitrary integer $k \geq 2$.

Subsection 4.2 studies the issue of simulating nonadaptive access to $\mathrm{U}\Sigma_h^p$, the hth level of the UPH, by adaptive access to $\mathrm{U}\Sigma_h^p$. Theorem 4.7 of this subsection generalizes a result of Cai, Hemachandra, and Vyskoč [15] from the case of $h = 1$ to the case of any arbitrary integer $h \geq 1$. Lemma 3.1 is used as a key tool for proving Theorem 4.7.

We improve upon Theorem 4.7 in subsection 4.3. There are compelling reasons for the transition from subsection 4.2 to subsection 4.3, which we discuss in subsection 4.3. Theorem 4.10 in that subsection not only resolves a question posed by Cai, Hemachandra, and Vyskoč [16] but also generalizes one of their results. In particular, Theorem 4.10 holds for any total, polynomial-time computable, and polynomially bounded function $k(\cdot)$ and arbitrary integer $h \geq 1$, while a similar result of Cai, Hemachandra, and Vyskoč [16] holds only for any arbitrary *constant* integer k and $h = 1$. Lemma 3.1 is one of the ingredients in the proof of this theorem.

Subsection 4.4 investigates the complimentary issue of simulating adaptive access to $\mathrm{U}\Sigma_h^p$ by nonadaptive access to $\mathrm{U}\Sigma_h^p$. Theorem 4.13 of this subsection generalizes a result of Cai, Hemachandra, and Vyskoč [16] from the case of $h = 1$ to the case of any arbitrary constant integer $h \geq 1$. Again, Lemma 3.1 is useful in making this generalization possible.

In subsection 4.5, we study the notion of one-sided helping introduced by Ko [36]. Theorem 4.17 of this subsection generalizes and improves one of the results of Cai, Hemachandra, and Vyskoč [16].

Finally, in section 5 we consider the possibility of imposing a more stringent restriction in the statement of Lemma 3.1. The investigation in this section leads to a generic oracle collapse of UPH to P under the assumption that P = NP in the unrelativized world. This extends a result of Blum and Impagliazzo [10], which showed a generic oracle collapse of UP (the first level of UPH) to P assuming that P = NP in the unrelativized world.

2. Preliminaries.

2.1. Notation. Let $\mathbb{N}^+$ denote the set of positive integers. Σ denotes the alphabet $\{0,1\}$. Let $[n] =_{df} \{1,2,\ldots,n\}$ for every $n \in \mathbb{N}^+$. NPTM stands for "nondeterministic polynomial-time Turing machine." For every oracle NPTM N, oracle A, and string $x \in \Sigma^*$, we use the shorthand $N^A(x)$ for "the computation tree of N with oracle A on input x." We fix a standard, polynomial-time computable and invertible, one-to-one, multiarity pairing function $\langle .,\ldots,. \rangle$ throughout the paper. Let $\circ$ denote the composition operator on functions. For any polynomial $p(.)$ and integer $i \geq 1$, let $(p\circ)^i(\cdot)$ denote

$$\underbrace{p \circ p \circ \cdots \circ p}_{i}(\cdot),$$

i.e., the polynomial obtained by i compositions of p. All polynomials $p(\cdot)$ appearing in this paper are without loss of generality nondecreasing and satisfy $p(n) \geq n$ for every $n \in \mathbb{N}^+$. Let σ be an equivalence relation on a set S. For each $x \in S$, the *equivalence class* $[x]$ of x determined by σ is $\{y \in S \mid x\sigma y\}$. The set S/σ of all equivalence classes determined by σ is called the *quotient set* determined by σ. For any set S, we use $\wp(S)$ to denote the *power set* of S, i.e., the set of all subsets of S. The *join* of two sets A and B over Σ is defined as $A \oplus B = \{0x \,\big|\, x \in A\} \cup \{1x \,\big|\, x \in B\}$.

We define the notion of a computation path of oracle machines independent of any concrete oracle. A *computation path* of an oracle NPTM N encodes a complete valid computation that N can have relative to some/any oracle; i.e., it contains the sequence of configurations including the query strings and the answers from the oracle. Hence two computation paths ρ_1 and ρ_2 of an oracle NPTM are equal if and only if the configuration sequences, oracle queries, and oracle answers are the same for the computation paths. For any computation path ρ, let $Q^+(\rho)$ denote the set of strings that are queried along ρ and answered positively, and let $Q^-(\rho)$ denote the set of strings that are queried along ρ and answered negatively. Let $Q(\rho) = Q^+(\rho) \cup Q^-(\rho)$. For any concrete oracle A and input x, a given path ρ may or may not appear in $N^A(x)$. For instance, if $\alpha \in Q^+(\rho)$, then ρ does not appear in $N^A(x)$ for any A with $\alpha \notin A$. In this case we also say "$N^A(x)$ does not have path ρ."

For any complexity class $\mathcal{C}$ and for any natural notion of polynomial-time reducibility r (e.g., $r \in \{m, dtt, tt, k\text{-}tt, T, k\text{-}T, b\}$), let $R_r^p(\mathcal{C})$ denote the closure of $\mathcal{C}$ under r. That is, $R_r^p(\mathcal{C}) =_{df} \{L \mid (\exists L' \in \mathcal{C})[L \leq_r^p L']\}$. We refer the reader to any

standard textbook in complexity theory (e.g., [11, 42, 31]) for complexity classes and reductions not defined in this paper.

Given a complexity class $\mathcal{C}$, the unique existential ($\exists!$), and the unique universal ($\forall!$) operators on $\mathcal{C}$ yield complexity classes. Formally, we have the following.

DEFINITION 2.1. *For any arbitrary complexity class $\mathcal{C}$, the following hold.*

1. *$\exists! \cdot \mathcal{C}$ is defined to be the class of all sets L for which there exist a polynomial $p(\cdot)$ and a set $L' \in \mathcal{C}$ such that for all $x \in \Sigma^*$,*

$$x \in L \Longrightarrow (\text{there exists a unique } y \in \Sigma^{p(|x|)})[\langle x, y\rangle \in L'], \text{ and}$$
$$x \notin L \Longrightarrow (\text{for all } y \in \Sigma^{p(|x|)})[\langle x, y\rangle \notin L'].$$

2. *$\forall! \cdot \mathcal{C}$ is defined to be the class of all sets L for which there exist a polynomial $p(\cdot)$ and a set $L' \in \mathcal{C}$ such that for all $x \in \Sigma^*$,*

$$x \in L \Longrightarrow (\text{for all } y \in \Sigma^{p(|x|)})[\langle x, y\rangle \in L'], \text{ and}$$
$$x \notin L \Longrightarrow (\text{there exists a unique } y \in \Sigma^{p(|x|)})[\langle x, y\rangle \notin L'].$$

We introduce the notion of a $\Sigma_k(A)$-system. This notion is useful for concisely representing the computation of a stack of oracle NPTMs.

DEFINITION 2.2.

1. *For any $k \in \mathbb{N}^+$ and $A \subseteq \Sigma^*$, we call a tuple $[A; N_1, N_2, \ldots, N_k]$, where A is an oracle and $N_1, N_2, \ldots, N_k$ are nondeterministic oracle Turing machines, a $\Sigma_k(A)$-system.* The language accepted by a $\Sigma_k(A)$-system*, denoted by $L[A; N_1, N_2, \ldots, N_k]$, is defined inductively as follows:*

$$L[A; N_1, N_2, \ldots, N_k] =_{df} \begin{cases} L(N_1^A) & \text{if } k = 1, \text{ and} \\ L(N_1^{L[A;N_2,N_3,\ldots,N_k]}) & \text{if } k > 1. \end{cases}$$

2. *The* computation of a $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ *on input x, denoted by $[A; N_1, N_2, \ldots, N_k](x)$, is defined as follows:*

$$[A; N_1, N_2, \ldots, N_k](x) =_{df} \begin{cases} N_1^A(x) & \text{if } k = 1, \text{ and} \\ N_1^{L[A;N_2,N_3,\ldots,N_k]}(x) & \text{if } k > 1. \end{cases}$$

We define the notion of unambiguity in $\Sigma_k(A)$-systems as follows.

DEFINITION 2.3.

1. *We say that a $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ is* unambiguous *if for every $1 \le i \le k$ and for every $x \in \Sigma^*$, $[A; N_i, N_{i+1}, \ldots, N_k](x)$ has at most one accepting path.*

2. *For any $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$, we define*

$$L_{\text{unambiguous}}[A; N_1, N_2, \ldots, N_k] = \begin{cases} L[A; N_1, N_2, \ldots, N_k] & \text{if } [A; N_1, N_2, \ldots, N_k] \text{ is} \\ & \text{unambiguous,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Roughly speaking, a property of an oracle machine is called *robust* if the machine retains that property with respect to every oracle. Below we define the property of *robust unambiguity* for a $\Sigma_k(A)$-system.

DEFINITION 2.4. *We say that a $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$ is* robustly unambiguous *if for every set B, the $\Sigma_k(B)$-system $[B; N_1, N_2, \ldots, N_k]$ is unambiguous.*

As we will see in section 5, the notion of a robustly unambiguous $\Sigma_k(\cdot)$-system is useful in extending a result of Blum and Impagliazzo [10].

2.2. Promise problems and smart reductions. Even, Selman, and Yacobi [19] introduced and studied the notion of promise problems. Promise problems are generalizations of decision problems in that the set of Yes-instances and the set of No-instances must partition the set of all instances in a decision problem, whereas this is not necessarily the case with promise problems. Thus, for a promise problem, a set of disallowed strings may be defined, which represent neither Yes-instances nor No-instances. Over the years, the notion of promise problems has proved to be useful, in several areas of computational complexity theory. (See [25] for a nice survey of some applications of promise problems in computational complexity theory.)

DEFINITION 2.5 (based on [25]; cf. [19]). *A promise problem* $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ *is defined in terms of disjoint sets* $\Pi_{\text{yes}}, \Pi_{\text{no}} \subseteq \Sigma^*$. *The set* Π_{yes} *is called the set of Yes-instances, the set* Π_{no} *is called the set of No-instances, and the set* $\Pi_{\text{yes}} \cup \Pi_{\text{no}}$ *is called the promise set.*

Some technicalities are involved when oracle access to a promise problem is defined. If a query to a promise problem falls inside the promise set, then the answer to the query is well defined (i.e., the answer is 1 if q is a Yes-instance and 0 if q is a No-instance). However, if a query falls outside the promise set, then it is not immediately clear how that query should be handled by the promise problem, i.e., the oracle. Several natural models of oracle access to a promise problem are definable. (See [26, 16] for a few possible approaches to defining oracle access to promise problems.)

Grollmann and Selman [26] proposed a model of oracle access to a promise problem that *prohibits* queries that fall outside the promise set. In this model, a querying machine always asks queries from the promise set; i.e., the queries asked by the querying machine always obey the underlying promise of the promise problem. For instance, let us define a promise problem $\Pi_{\text{unique}} = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ in terms of the acceptance mechanism of an NPTM N as follows: $\Pi_{\text{yes}} = \{x \in \Sigma^* \mid \#\text{acc}_N(x) = 1\}$ and $\Pi_{\text{no}} = \{x \in \Sigma^* \mid \#\text{acc}_N(x) = 0\}$. Then a Turing access to Π_{unique} in the model proposed by Grollmann and Selman [26] requires that for any query y asked by the querying machine on some input, the computation of N on y must be unambiguous; i.e., $\#\text{acc}_N(y)$ must be either 0 or 1. A Turing reduction that obeys the constraints of this model (i.e., any query ever asked belongs to the promise set) is called a *smart* Turing reduction [26]. The definition given below formally captures the notion of a smart Turing reduction from a decision problem to a promise problem.[1]

DEFINITION 2.6. *A set* L *polynomial-time* smart *Turing reduces to a promise problem* $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$, *denoted by* $L \leq^p_{s,T} \Pi$ *or* $L \in \text{P}^{\Pi}_s$, *if there is a deterministic polynomial-time oracle Turing machine* M *such that for all* $x \in \Sigma^*$,

1. $x \in L \iff M^{\Pi}(x)$ *accepts, and*
2. *if* $M^{\Pi}(x)$ *asks a query* y *to* Π, *then* $y \in \Pi_{\text{yes}} \cup \Pi_{\text{no}}$.

If on all inputs $x \in \Sigma^*$ *the querying machine* M *asks at most* k *queries for some integer constant* $k \geq 1$, *then we say that the set of* L *polynomial-time* smart k*-Turing reduces to* Π, *and we write* $L \leq^p_{s,k\text{-}T} \Pi$ *or* $L \in \text{P}^{\Pi[k]}_s$.

In the above definition, we followed Grollmann and Selman's notion of smart Turing reductions from *decision* problems to *promise* problems. We may extend this notion to define reductions that reduce *promise* problems to *promise* problems. (See, for instance, [25] for a generalization of smart Turing reductions to reductions among promise problems.) In this paper, we will consider only smart Turing reductions (i.e.,

[1] Cai, Hemachandra, and Vyskoč [16] referred to Grollmann and Selman's *smart* oracle access by the term *guarded* access.

reductions from decision problems to promise problems) as given by Grollmann and Selman.

The following two definitions are standard.

DEFINITION 2.7. *Let* Π *be any promise problem.* $R^p_{s,T}(\Pi)$ *is the class of all sets* L *such that* $L \leq^p_{s,T} \Pi$*; for all* $k \in \mathbb{N}^+$, $R^p_{s,k\text{-}T}(\Pi)$ *is the class of all sets* L *such that* $L \leq^p_{s,k\text{-}T} \Pi$*;* $R^p_{s,b}(\Pi)$ *is the class of all sets* L *for which there exists some* $k \in \mathbb{N}^+$ *such that* $L \leq^p_{s,k\text{-}T} \Pi$.

DEFINITION 2.8. *For any class of promise problems* $\mathcal{C}$ *and any reduction* $r \in \{T, k\text{-}T, b\}$*, we define* $R^p_{s,r}(\mathcal{C}) =_{df} \bigcup_{\Pi\in\mathcal{C}} R^p_{s,r}(\Pi)$.

We will study the computational power of smart Turing reductions to a particular class of promise problems, namely, the class Promise-UP, which is defined as follows.

DEFINITION 2.9. Promise-UP *is the class of all promise problems* $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ *for which there exists an NPTM* N *such that for all* $x \in \Sigma^*$,

$$x \in \Pi_{\text{yes}} \Longrightarrow \#\text{acc}_N(x) = 1, \text{ and}$$
$$x \in \Pi_{\text{no}} \Longrightarrow \#\text{acc}_N(x) = 0.$$

The class $\text{P}^{\text{Promise-UP}}_s$ of sets that polynomial-time smart Turing reduce to Promise-UP is a prominent class that behaves remarkably differently than the related class P^{UP}. While $\text{P}^{\text{Promise-UP}}_s$ is known to contain the class FewP [16] and the graph isomorphism problem [2], similar results for the case of P^{UP} are unknown.[2]

2.3. Unambiguity based hierarchies. Niedermeier and Rossmanith [40] observed that the notion of unambiguity in NPTMs can be generalized in three ways, each of which defines an unambiguity based hierarchy.

DEFINITION 2.10 (unambiguity based hierarchies [38, 40]).

1. *The* alternating unambiguous polynomial hierarchy *is defined as follows:*

$$\text{AUPH} =_{df} \bigcup_{k\geq 0} \text{AU}\Sigma^p_k = \bigcup_{k\geq 0} \text{AU}\Pi^p_k,$$

where

$$\text{AU}\Sigma^p_k = \begin{cases} \text{P} & \text{if } k = 0, \\ \exists! \cdot \text{AU}\Pi^p_{k-1} & \text{if } k \geq 1, \end{cases} \quad \text{and} \quad \text{AU}\Pi^p_k = \begin{cases} \text{P} & \text{if } k = 0, \\ \forall! \cdot \text{AU}\Sigma^p_{k-1} & \text{if } k \geq 1. \end{cases}$$

2. *The* unambiguous polynomial hierarchy *is defined as follows:*

$$\text{UPH} =_{df} \bigcup_{k\geq 0} \text{U}\Sigma^p_k = \bigcup_{k\geq 0} \text{U}\Pi^p_k,$$

where

$$\text{U}\Sigma^p_k = \begin{cases} \text{P} & \text{if } k = 0, \\ \text{UP}^{\text{U}\Sigma^p_{k-1}} & \text{if } k \geq 1, \end{cases} \quad \text{and} \quad \text{U}\Pi^p_k = \begin{cases} \text{P} & \text{if } k = 0, \\ \text{coUP}^{\text{U}\Sigma^p_{k-1}} & \text{if } k \geq 1. \end{cases}$$

[2] Arvind and Kurur [2] showed that the graph isomorphism problem (GI) belongs to SPP, a class introduced in [27, 41, 20]. Subsequently, Crâsmaru et al. [18] observed that the proof of classifying GI into SPP, as given by Arvind and Kurur [2], actually yields a somewhat improved classification for GI: GI belongs to $R^p_{s,T}$(Promise-UP), a subclass of SPP [18].

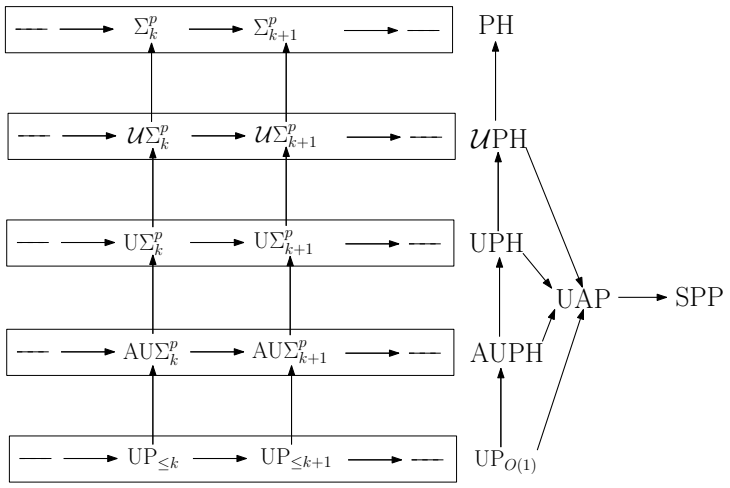


FIG. 1. *Known inclusion structure of unambiguity based classes and other central classes. The arrows point from subclasses to superclasses.*

3. *The* promise unambiguous polynomial hierarchy *is defined as follows:*

$$\mathcal{UPH} =_{df} \bigcup_{k\geq 0} \mathcal{U}\Sigma_k^p = \bigcup_{k\geq 0} \mathcal{U}\Pi_k^p,$$

where $\mathcal{U}\Sigma_0^p =_{df} \mathrm{P}$, $\mathcal{U}\Sigma_1^p =_{df} \mathrm{UP}$, *and for every* $k \geq 2$, $\mathcal{U}\Sigma_k^p$ *is the class of all sets* $L \in \Sigma_k^p$ *such that for some oracle* NPTM*s* N_1, N_2, ..., N_k, $L = L[\emptyset; N_1, N_2, \ldots, N_k]$, *and for every* $x \in \Sigma^*$ (i) $[\emptyset; N_1, N_2, \ldots, N_k](x)$ *has at most one accepting path, and* (ii) *for every* $1 \leq i \leq k-1$, *if* N_i *asks a query* w *to its oracle* $L[\emptyset; N_{i+1}, N_{i+2}, \ldots, N_k]$ *during the computation of* $[\emptyset; N_1, N_2, \ldots, N_k](x)$, *then* $[\emptyset; N_{i+1}, N_{i+2}, \ldots, N_k](w)$ *has at most one accepting path. For each* $k \geq 0$, *the class* $\mathcal{U}\Pi_k^p$ *is defined by* $\mathcal{U}\Pi_k^p =_{df} \mathrm{co}\mathcal{U}\Sigma_k^p$.

Notice that $\mathrm{U}\Sigma_k^p$ is also the class of all sets accepted by unambiguous $\Sigma_k(\emptyset)$-systems. Also notice that for any set $L = L[\emptyset; N_1, N_2, \ldots, N_k] \in \mathcal{U}\Sigma_k^p$, the $\Sigma_k(\emptyset)$-system $[\emptyset; N_1, N_2, \ldots, N_k]$ does not need to be unambiguous. In other words, for any string w that never appears as a query in the computation of $[\emptyset; N_1, N_2, \ldots, N_k]$ it may happen for some $1 \leq i \leq k$ that $[\emptyset; N_i, \ldots, N_k](w)$ has more than one accepting path.

The following inclusion relationships between unambiguity based classes and other central classes are known (see also Figure 1).

THEOREM 2.11.

1. *For all* $k \geq 0$, $\mathrm{AU}\Sigma_k^p \subseteq \mathrm{U}\Sigma_k^p \subseteq \mathcal{U}\Sigma_k^p \subseteq \Sigma_k^p$ [38].
2. *For all* $k \geq 1$, $\mathrm{UP}_{\leq k} \subseteq \mathrm{AU}\Sigma_k^p \subseteq \mathrm{U}\Sigma_k^p \subseteq \mathcal{U}\Sigma_k^p \subseteq \mathrm{UAP} \subseteq \mathrm{SPP}$ ([38] + [40] + [18]).
3. $\mathrm{P}^{\mathrm{FewP}} \subseteq \mathrm{P}_s^{\text{Promise-UP}}$ [16].

Despite the attention these hierarchies deserve, much less is known about the structure of these hierarchies since Lange and Rossmanith [38] first posed questions on their structure—such as, whether these hierarchies intertwine, or whether some unambiguity based hierarchy is contained in a fixed level of some other hierarchy, or whether some/all of these hierarchies collapse to a fixed level. On the positive side, there have been some advances in understanding the structure of these hierarchies. Hemaspaandra and Rothe [33] related the structure of these hierarchies to the exis-

tence of sparse Turing complete sets for UP. The structure of these hierarchies received renewed interest in some recent work (see [1, 18, 24, 46]). For instance, Spakowski and Tripathi [46] investigated the relativized structure of these hierarchies. They proved that the unambiguity based hierarchies AUPH, UPH, and $\mathcal{UPH}$ are infinite in some relativized world. They also proved a contrasting result on their relativized structure: For each $k \geq 2$, there is a relativized world where these hierarchies collapse so that they have exactly k distinct levels and their kth levels coincide with PSPACE.

3. Proof technique.

3.1. Main lemma. Our main lemma is Lemma 3.1, which we will use throughout this paper for generalizing known oracle constructions involving unambiguity based classes such as UP and Promise-UP to new oracle constructions involving arbitrary levels of the UPH. Roughly, Lemma 3.1 states computational limitations of a $\Sigma_k(\mathcal{O})$-system for any arbitrary $k \geq 1$, under certain weak conditions.

LEMMA 3.1. *Fix a $\Sigma_k(\mathcal{O})$-system $[\mathcal{O}; N_1, N_2, \ldots, N_k]$, a string $x \in \Sigma^*$, and a set $U \subseteq \Sigma^*$ such that $\mathcal{O} \cap U = \emptyset$. Let $r(.)$ be a polynomial that bounds the running time of each of the machines $N_1, N_2, \ldots, N_k$ (on any input and with any oracle). Then the following hold.*

(I) *Suppose $[\mathcal{O}; N_1, N_2, \ldots, N_k](x)$ accepts and for every $A \subseteq U$ with $||A|| \leq k$, $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$ is unambiguous. Let*

$$C = \{\alpha \in U \mid [\mathcal{O} \cup \{\alpha\}; N_1, N_2, \ldots, N_k](x) \text{ rejects}\}.$$

Then $||C|| \leq 5^k \cdot \prod_{i=1}^{k} (r\circ)^i(|x|)$.

(II) *Suppose $[\mathcal{O}; N_1, N_2, \ldots, N_k](x)$ rejects and for every $A \subseteq U$ with $||A|| \leq k+1$, $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$ is unambiguous. Let*

$$C = \{\alpha \in U \mid [\mathcal{O} \cup \{\alpha\}; N_1, N_2, \ldots, N_k](x) \text{ accepts}\}.$$

Then $||C|| \leq 5^k \cdot \prod_{i=1}^{k} (r\circ)^i(|x|)$.

Intuitively, this lemma says the following. If a $\Sigma_k(\mathcal{O})$-system is unambiguous when any subset A, which contains at most $k+1$ elements, of some fixed set U is added to the oracle $\mathcal{O}$, then this $\Sigma_k(\mathcal{O})$-system behaves similarly to a deterministic polynomial-time oracle Turing machine upon addition of single strings from U to $\mathcal{O}$. That is, if we want to change the acceptance behavior of the $\Sigma_k(\mathcal{O})$-system on some input x by adding a single string $\alpha \in U$ to the oracle $\mathcal{O}$, then we need to take α from a specific small (polynomial-size) subset C (which depends on x) of U.

Proof of Lemma 3.1. By induction on k, we prove the claim "For any oracle $\mathcal{O}$, any $\Sigma_k(\mathcal{O})$-system $[\mathcal{O}; N_1, N_2, \ldots, N_k]$, any $x \in \Sigma^*$, and any set $U \subseteq \Sigma^*$ such that $\mathcal{O} \cap U = \emptyset$, statements (I) and (II) hold."

For the base case $k = 1$ of (I), we have $[\mathcal{O}; N_1](x)$ accepts. Also, since $[\mathcal{O}; N_1]$ is unambiguous by the assumption made in (I), there is a unique accepting path in $[\mathcal{O}; N_1](x)$. Let C' be the set of all queries $w \in U$ along this unique accepting path. Then clearly $||C|| \leq ||C'|| \leq r(|x|)$. Thus (I) holds for the base case.

For the base case $k = 1$ of (II), we have $[\mathcal{O}; N_1](x)$ rejects. Suppose that $||C|| > 5 \cdot r(|x|)$. Note that for every $\alpha \in C$, $[\mathcal{O} \cup \{\alpha\}; N_1](x)$ accepts. Thus, for every $\alpha \in C$, let $\lambda(\alpha)$ be the accepting path in $[\mathcal{O} \cup \{\alpha\}; N_1](x)$. It is easy to show that $\lambda(\alpha_1) \neq \lambda(\alpha_2)$ for any distinct $\alpha_1, \alpha_2 \in C$. To see this, let $\rho = \lambda(\alpha)$ for some $\alpha \in C$. Then, by the definition of $\lambda(\alpha)$ and the assumption that $[\mathcal{O}; N_1](x)$ rejects, we notice that path ρ appears in $N_1^{\mathcal{O} \cup \{\alpha\}}(x)$ but does not appear in $N_1^{\mathcal{O}}(x)$. From this, it follows

that α must be answered positively along ρ, i.e., $\alpha \in Q^+(\rho)$, since otherwise ρ would also appear in $N_1^{\mathcal{O}}(x)$. Therefore, for any oracle $\mathcal{B}$ with $\alpha \notin \mathcal{B}$, ρ cannot appear in $N_1^{\mathcal{B}}(x)$. In particular, ρ cannot appear in $N_1^{\mathcal{O}\cup\{\alpha'\}}(x)$ for any α' with $\alpha' \neq \alpha$. Hence, we get that $\lambda(\alpha_1) \neq \lambda(\alpha_2)$ for any distinct $\alpha_1, \alpha_2 \in C$.

We define, for any $\alpha \in C$,

$$\text{conflicting}(\alpha) = \{\beta \in C \mid \lambda(\alpha) \text{ is not an accepting path in } [\mathcal{O} \cup \{\alpha, \beta\}; N_1](x)\}.$$

Since $N_1(x)$ with any oracle asks at most $r(|x|)$ queries, there can be at most $r(|x|)$ strings $\beta \in C$ that can cause $\lambda(\alpha)$ not to appear in $[\mathcal{O} \cup \{\alpha, \beta\}; N_1](x)$. In other words, for any $\alpha \in C$, it holds that $||\text{conflicting}(\alpha)|| \leq r(|x|)$. Thus, it follows by an easy counting argument and the assumption that $||C|| > 5 \cdot r(|x|)$ that there exist distinct $\alpha_1, \alpha_2 \in C$ such that $\alpha_1 \notin \text{conflicting}(\alpha_2)$ and $\alpha_2 \notin \text{conflicting}(\alpha_1)$. We have already shown that $\lambda(\alpha_1)$ and $\lambda(\alpha_2)$ are distinct paths for distinct $\alpha_1, \alpha_2 \in C$. Therefore, $[\mathcal{O} \cup \{\alpha_1, \alpha_2\}; N_1](x)$ has two distinct accepting paths, namely, $\lambda(\alpha_1)$ and $\lambda(\alpha_2)$. This contradicts our assumption that $[\mathcal{O} \cup A; N_1]$ is unambiguous for every $A \subseteq U$ with $||A|| \leq 2$. Thus (II) also holds for the base case.

Induction hypothesis. For $k \geq 1$, the following claim is true: For any oracle $\mathcal{O}$, any $\Sigma_k(\mathcal{O})$-system $[\mathcal{O}; N_1, N_2, \ldots, N_k]$, any $x \in \Sigma^*$, and any set $U \subseteq \Sigma^*$ such that $\mathcal{O} \cap U = \emptyset$, statements (I) and (II) hold.

Inductive step. Let $[\mathcal{O}; N_1, N_2, \ldots, N_{k+1}]$ be a $\Sigma_{k+1}(\mathcal{O})$-system, let $r(.)$ be a polynomial that bounds the running time of each of $N_1, N_2, \ldots, N_k, N_{k+1}$ (for any oracle), let $U \subseteq \Sigma^*$ such that $\mathcal{O} \cap U = \emptyset$, and let $x \in \Sigma^*$.

We first prove (I). Suppose $[\mathcal{O}; N_1, N_2, \ldots, N_{k+1}](x)$ accepts and $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_{k+1}]$ is unambiguous for every $A \subseteq U$ with $||A|| \leq k+1$. Let λ denote the unique accepting path of $[\mathcal{O}; N_1, N_2, \ldots, N_{k+1}](x)$. For every query w along λ, the $\Sigma_k(\mathcal{O})$-system $[\mathcal{O}; N_2, N_3, \ldots, N_{k+1}]$ computes $[\mathcal{O}; N_2, N_3, \ldots, N_{k+1}](w)$. By Definition 2.3(1), for every $A \subseteq U$ with $||A|| \leq k+1$, the $\Sigma_k(\mathcal{O} \cup A)$-system $[\mathcal{O} \cup A; N_2, N_3, \ldots, N_{k+1}]$ is unambiguous. Thus, it follows by the induction hypothesis that for every query w along λ and for all but at most $5^k \cdot \prod_{i=1}^{k}(r\circ)^i(|w|) \leq 5^k \cdot \prod_{i=2}^{k+1}(r\circ)^i(|x|)$ strings $\alpha \in U$, adding $\alpha \in U$ to $\mathcal{O}$ does not change the decision (i.e., acceptance or rejection) of $[\mathcal{O}; N_2, N_3, \ldots, N_{k+1}](w)$. Since the number of such queries w along λ is at most $r(|x|)$, it follows that λ is an accepting path in $[\mathcal{O} \cup \{\alpha\}; N_1, N_2, \ldots, N_{k+1}](x)$ for all but at most $r(|x|) \cdot 5^k \cdot \prod_{i=2}^{k+1}(r\circ)^i(|x|) < 5^{k+1} \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$ strings $\alpha \in U$. This proves (I), the first part of the inductive step.

We now prove (II). Suppose that $||C|| > 5^{k+1} \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$. For any $\alpha \in C$, let $\lambda(\alpha)$ denote the unique accepting path in $[\mathcal{O} \cup \{\alpha\}; N_1, N_2, \ldots, N_{k+1}](x)$. We define an equivalence relation σ on C as follows: For every $\alpha_1, \alpha_2 \in C$,

$$\alpha_1 \sigma \alpha_2 \iff \lambda(\alpha_1) = \lambda(\alpha_2).$$

The following cases are exhaustive.

Case 1. There is an equivalence class of σ of size $> 5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$. Let $[\alpha]$ be such an equivalence class. Then we have the following situation: The accepting path $\lambda(\alpha)$ does not appear in $[\mathcal{O}; N_1, N_2, \ldots, N_{k+1}](x)$, but for every $\beta \in [\alpha]$, path $\lambda(\alpha)$ appears in $[\mathcal{O} \cup \{\beta\}; N_1, N_2, \ldots, N_{k+1}](x)$. Hence for every query $w \in \Sigma^*$ along $\lambda(\alpha)$, it holds that for every $\beta, \beta' \in [\alpha]$, $[\mathcal{O} \cup \{\beta\}; N_2, N_3, \ldots, N_{k+1}](w)$ and $[\mathcal{O} \cup \{\beta'\}; N_2, N_3, \ldots, N_{k+1}](w)$ have the same acceptance behavior; i.e, $[\mathcal{O} \cup$

$\{\beta\}; N_2, N_3, \ldots, N_{k+1}](w)$ accepts if and only if $[\mathcal{O} \cup \{\beta'\}; N_2, N_3, \ldots, N_{k+1}](w)$ accepts. There must be at least one query $w' \in \Sigma^*$ along $\lambda(\alpha)$ such that for some $\beta \in [\alpha]$ (and hence by the previous sentence for *every* $\beta \in [\alpha]$), adding β to $\mathcal{O}$ changes the answer of $[\mathcal{O}; N_2, N_3, \ldots, N_{k+1}](w')$, since otherwise $\lambda(\alpha)$ would also appear in $[\mathcal{O}; N_1, N_2, \ldots, N_{k+1}](x)$. Also note that for every $A \subseteq U$ with $||A|| \leq k+1$, $[\mathcal{O} \cup A; N_2, N_3, \ldots, N_{k+1}]$ is unambiguous. Thus we get a contradiction with the induction hypothesis, since $|||[\alpha]||| > 5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|) \geq 5^k \cdot \prod_{i=2}^{k+1}(r\circ)^i(|x|) \geq 5^k \cdot \prod_{i=1}^{k}(r\circ)^i(|w'|)$.

Case 2. Every equivalence class of σ is of size $\leq 5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$. We need the following claim, which demonstrates that if σ is an equivalence relation over some set C consisting only of small equivalence classes, then C can be partitioned into two sufficiently large disjoint sets C_1 and C_2 such that every equivalence class is contained in either C_1 or C_2.

CLAIM 1. *Let σ be any equivalence relation over some set C, and let $||C|| > 5s$. If $|||[\alpha]||| \leq s$ for every $\alpha \in C$, then there is a partition (C_1, C_2) of C such that $||C_1||$, $||C_2|| > 2s$, and for every $\alpha \in C$, $[\alpha] \subseteq C_1$ or $[\alpha] \subseteq C_2$.*

Proof of Claim 1. Let D be any subset of C such that (i) for every $\alpha \in C$, $[\alpha] \subseteq D$ or $[\alpha] \cap D = \emptyset$, (ii) $||D|| \leq 2s$, and (iii) $||D \cup [\beta]|| > 2s$ for some $\beta \in C$. Such a set D exists because $||C|| > 5s$ and $|||[\alpha]||| \leq s$ for every $\alpha \in C$.

Let $C_1 = D \cup [\beta]$ for some $\beta \in C$ such that $||C_1|| > 2s$. Because $||\beta|| \leq s$, we have that $||C_1|| \leq ||D|| + |||[\beta]||| \leq 2s + s = 3s$. Now let $C_2 = C - C_1$. We see that $||C_2|| = ||C|| - ||C_1|| > 5s - 3s = 2s$. Moreover, every equivalence class is contained in either C_1 or C_2. This completes the proof of Claim 1. □

It is easy to see that our equivalence class σ over C satisfies the preconditions of Claim 1 for $s = 5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$. Let (C_1, C_2) be the partition of C given by Claim 1. Then both $||C_1||$ and $||C_2||$ are greater than $2 \cdot 5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$. For every $\alpha_1 \in C_1$ and $\alpha_2 \in C_2$, we define

$$\text{conflicting}(\alpha_1) = \{\beta_2 \in C_2 \mid \lambda(\alpha_1) \text{ is not an accepting path in } [\mathcal{O} \cup \{\alpha_1, \beta_2\}; N_1, N_2, \ldots, N_{k+1}](x)\},$$

and

$$\text{conflicting}(\alpha_2) = \{\beta_1 \in C_1 \mid \lambda(\alpha_2) \text{ is not an accepting path in } [\mathcal{O} \cup \{\alpha_2, \beta_1\}; N_1, N_2, \ldots, N_{k+1}](x)\}.$$

We claim that both $||\text{conflicting}(\alpha_1)||$ and $||\text{conflicting}(\alpha_2)||$ are no more than $5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$. To prove this, first note that $N_1(x)$ with any oracle can ask at most $r(|x|)$ queries along a computation path. Second, for each $\gamma \in \{\alpha_1, \alpha_2\}$, we have that for any $A \subseteq U$ with $||A|| \leq k+1$, $[\mathcal{O} \cup \{\gamma\} \cup A; N_1, N_2, \ldots, N_{k+1}]$ is unambiguous. Therefore, it follows by the induction hypothesis (where now $\mathcal{O}$ is $\mathcal{O} \cup \{\gamma\}$ and $N_1, N_2, \ldots, N_k$ are $N_2, N_3, \ldots, N_{k+1}$) that for every query w made by $N_1(x)$ along $\lambda(\gamma)$ to the oracle $L[\mathcal{O} \cup \{\gamma\}; N_2, \ldots, N_{k+1}]$, there can be at most $5^k \cdot \prod_{i=1}^{k}(r\circ)^i(|w|) \leq 5^k \cdot \prod_{i=2}^{k+1}(r\circ)^i(|x|)$ strings $\beta \in U$ such that adding β to $\mathcal{O} \cup \{\gamma\}$ changes the decision (acceptance or rejection) of $[\mathcal{O} \cup \{\gamma\}; N_2, N_3, \ldots, N_{k+1}](w)$. These two facts imply the stated bound on $||\text{conflicting}(\alpha_1)||$ and $||\text{conflicting}(\alpha_2)||$.

With the bounds $||C_1||, ||C_2|| > 2 \cdot 5^k \cdot \prod_{i=1}^{k+1}(r\circ)^i(|x|)$, a simple counting argument now shows that there exist $\alpha_1 \in C_1$ and $\alpha_2 \in C_2$ such that $\alpha_1 \notin \text{conflicting}(\alpha_2)$ and $\alpha_2 \notin \text{conflicting}(\alpha_1)$. As a consequence, $[\mathcal{O} \cup \{\alpha_1, \alpha_2\}; N_1, N_2, \ldots, N_{k+1}](x)$

accepts along two distinct paths, namely, $\lambda(\alpha_1)$ and $\lambda(\alpha_2)$, which are indeed distinct since $[\alpha_1] \neq [\alpha_2]$. This gives a contradiction to the assumption that $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_{k+1}]$ is unambiguous for every $A \subseteq U$ with $||A|| \leq k+1$. This completes the proof of Lemma 3.1. □

3.2. The notion of (h,t)-ambiguity for functions on $\wp(\Sigma^*)$. Any oracle machine can be interpreted as a function mapping a set of strings to another set of strings as follows: A machine N maps any set $\mathcal{O}$ to the set $L(N^{\mathcal{O}})$. Therefore, it makes sense to consider the (possibly partial) function $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ defined by a $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$. (That is, define $\mathcal{L}$ so that for every $\mathcal{O} \subseteq \Sigma^*$, $\mathcal{L}(\mathcal{O}) =_{df} L_{\text{unambiguous}}[\mathcal{O}; N_1, N_2, \ldots, N_k]$.) We introduce a convenient notion called "(h,t)-ambiguity" for (partial or total) functions, which we will later apply to functions defined by $\Sigma_k(\cdot)$-systems.

DEFINITION 3.2. *For any $h \in \mathbb{N}^+$ and polynomial $t(\cdot)$, we call a partial or total function $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$* (h,t)-ambiguous *if, for every $\mathcal{O}, U \subseteq \Sigma^*$ with $\mathcal{O} \cap U = \emptyset$, one of the following is true:*

1. *For some $A \subseteq U$ with $||A|| \leq h$, $\mathcal{L}(\mathcal{O} \cup A)$ is undefined, or*
2. *for every $w \in \Sigma^*$,*

$$||\{\alpha \in U \mid w \in \mathcal{L}(\mathcal{O} \cup \{\alpha\}) \iff w \notin \mathcal{L}(\mathcal{O})\}|| \leq t(|w|).$$

Many of the proofs in this paper apply Proposition 3.3 below. The proof of Proposition 3.3 is based on the definition of $L_{\text{unambiguous}}$ given in Definition 2.3 and on Lemma 3.1.

PROPOSITION 3.3. *For any $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$, the function $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ defined for any $\mathcal{B} \subseteq \Sigma^*$ by*

$$\mathcal{L}(\mathcal{B}) = L_{\text{unambiguous}}[\mathcal{B}; N_1, N_2, \ldots, N_k]$$

is $(k+1,t)$-ambiguous, where $t(\cdot) = 5^k \cdot \prod_{i=1}^{k} (p\circ)^i(\cdot)$ and $p(\cdot)$ is a polynomial that bounds the running time of the machines $N_1, N_2, \ldots, N_k$.

Proof. For every $\mathcal{O}, U \subseteq \Sigma^*$ with $\mathcal{O} \cap U = \emptyset$, we have the following two cases: either there is a set $A \subseteq U$ with $||A|| \leq k+1$ such that $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$ is ambiguous or there is no such set A. In the first case, $\mathcal{L}(\mathcal{O} \cup A)$ is undefined by the definition of $\mathcal{L}$ in the proposition and by Definition 2.3(2). In the second case, for every $A \subseteq U$ with $||A|| \leq k+1$, $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$ is unambiguous, and furthermore $\mathcal{L}(\mathcal{O} \cup A) = L[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$. Thus, in this case, it follows by Lemma 3.1 that for every $w \in \Sigma^*$, there can be at most $t(|w|)$ strings $\alpha \in U$ such that adding α to $\mathcal{O}$ changes the membership of w in $\mathcal{L}(\mathcal{O})$. Clearly, in both cases we get that $\mathcal{L}$ is $(k+1,t)$-ambiguous. □

Some words are in order on the proof technique used in this paper. The machine N_1 in a $\Sigma_k(\mathcal{O})$-system $[\mathcal{O}; N_1, N_2, \ldots, N_k]$ has oracle access to the set $L[\mathcal{O}; N_2, N_3, \ldots, N_k]$. In many of our oracle constructions, we use the following central idea: At any stage, either there is a finite extension B of the current partial oracle $\mathcal{A}$ such that the $\Sigma_{k-1}(\mathcal{A} \cup B)$-system $[\mathcal{A} \cup B; N_2, N_3, \ldots, N_k]$ is ambiguous and simultaneously the constraints imposed by the construction at that stage are satisfied, or there is no such extension. In the first case, we extend $\mathcal{A}$ to $\mathcal{A} \cup B$, ensure that the $\Sigma_{k-1}(\mathcal{A}')$-system $[\mathcal{A}'; N_2, N_3, \ldots, N_k]$ remains ambiguous for the partial oracle $\mathcal{A}'$ at any later stage, and move to the next stage. In the second case, all finite extensions B of $\mathcal{A}$ that satisfy the imposed constraints lead to an unambiguous $\Sigma_{k-1}(\mathcal{A} \cup B)$-system $[\mathcal{A} \cup B; N_2, N_3, \ldots, N_k]$. So for these extensions B we have that $L_{\text{unambiguous}}[\mathcal{A} \cup$

$B; N_2, N_3, \ldots, N_k]$ equals $L[\mathcal{A} \cup B; N_2, N_3, \ldots, N_k]$. We define a function $\mathcal{L}'$ (as in Proposition 3.3) for any $\mathcal{B} \subseteq \Sigma^*$ by $\mathcal{L}'(\mathcal{B}) = L_{\text{unambiguous}}[\mathcal{B}; N_2, \ldots, N_k]$. Notice that by Proposition 3.3, $\mathcal{L}'$ is (k,t)-ambiguous. Moreover, for all finite extensions B of $\mathcal{A}$ mentioned above, $\mathcal{L}'(\mathcal{A} \cup B) = L[\mathcal{A} \cup B; N_2, N_3, \ldots, N_k]$.

It turns out that the (k,t)-ambiguity property of the function $\mathcal{L}'(\cdot)$, defined above for the $\Sigma_{k-1}(\cdot)$-subsystem $[\cdot; N_2, N_3, \ldots, N_k]$, is the only property of $\mathcal{L}'$ that is needed in our proofs. Therefore, we can assume without losing anything that in the $\Sigma_k(\mathcal{A})$-system $[\mathcal{A}; N_1, N_2, \ldots, N_k]$, the NPTM N_1 has oracle access to a set $\mathcal{L}(\mathcal{A})$, where $\mathcal{L}$ is an arbitrary (k,t)-ambiguous function, rather than to the set $\mathcal{L}'(\mathcal{A})$. This approach of using $\mathcal{L}(\mathcal{A})$ has its own advantages: It greatly simplifies our proof arguments and makes expressions compact, since we no longer need to deal with stacks of oracle NPTMs.

4. Applications. In this section, we demonstrate applications of our proof techniques. We show that our counting techniques are useful in generalizing certain known relativization results involving bounded ambiguity classes such as UP and Promise-UP to new results involving arbitrary levels of the UPH. This stands in contrast to generalizations achieved for relativization results involving levels of the PH. For instance, extending the relativized separation of initial levels of the PH [4, 5] to relativized separations of arbitrary levels of the PH [49] required applications of sophisticated circuit-theoretic techniques. For the case of the UPH, however, we show (via Theorem 4.1) that a relativized separation of its levels can be achieved by counting techniques alone, i.e., without resorting to circuit-theoretic tools and techniques.

4.1. Comparing bounded ambiguity classes with the levels of UPH. We compare classes defined by NPTMs having restrictions on the number of accepting paths ($\mathrm{UP}_{O(1)}$ and FewP) with levels of the UPH. It is known that $\mathrm{UP}_{\leq k} \subseteq \mathrm{U}\Sigma_k^p$ in all relativized worlds. Theorem 4.1 shows the optimality of this inclusion with respect to relativizable proof techniques. Beigel [6] constructed an oracle relative to which $\mathrm{UP}_{k(n)+1} \not\subseteq \mathrm{UP}_{k(n)}$ for every polynomial $k(n) \geq 2$. Theorem 4.1 subsumes this oracle result of Beigel [6] for any constant k.

By a slight modification of the oracle construction in Theorem 4.1, we can show that the second level $\mathcal{U}\Sigma_2^p$ of the promise unambiguous hierarchy $\mathcal{UPH}$ is not contained in the unambiguous polynomial hierarchy UPH. Results on relativized separations of levels of some unambiguity based hierarchy from another hierarchy have been investigated earlier. Rossmanith (see [40]) gave a relativized separation of $\mathrm{AU}\Sigma_k^p$ from $\mathrm{U}\Sigma_k^p$ for any $k \geq 2$. Spakowski and Tripathi [46] constructed an oracle relative to which $\mathrm{AU}\Sigma_k^p \not\subseteq \Pi_k^p$, for any $k \geq 1$. Our relativized separation of $\mathcal{U}\Sigma_2^p$ from UPH does not seem to be implied from these previous results in any obvious way.

THEOREM 4.1. *For any integer $k \geq 1$, there exists an oracle $\mathcal{A}$ such that* $\mathrm{UP}_{\leq k+1}^{\mathcal{A}} \not\subseteq \mathrm{U}\Sigma_k^{p,\mathcal{A}}$.

Proof. Our test language is $L(\mathcal{A}) = \{0^n \mid \mathcal{A}^{=n} \neq \emptyset\}$. We will create an oracle $\mathcal{A}$ that, for any length $n \in \mathbb{N}^+$, satisfies $||\mathcal{A}^{=n}|| \leq k+1$. Let $(N_{i,1}, N_{i,2}, \ldots, N_{i,k})$ be an enumeration of tuples, where $N_{i,\star}$ is a nondeterministic polynomial-time oracle Turing machine. Initially, $\mathcal{A} := \emptyset$.

Stage i. Let $p(\cdot)$ be a polynomial that bounds the running time of $N_{i,\star}$. Choose a large integer n such that (a) $2^n > 5^k \cdot \prod_{j=1}^{k} (p\circ)^j(n)$, (b) n satisfies the promises made in the previous stages, (c) no string of length $\geq n$ is ever queried in any of the previous stages, and (d) n is larger than the value in the previous stage. Because 2^n grows faster than any polynomial, it is easy to see that there always exists such an n.

If there exists a set $B \subseteq \Sigma^n$ such that $||B|| \leq k+1$ and $[\mathcal{A} \cup B; N_{i,1}, N_{i,2}, \ldots, N_{i,k}]$ is ambiguous, then set $\mathcal{A} := \mathcal{A} \cup B$. Promise to choose the value of n in the next stage to be larger than $(p\circ)^k(|w|)$, where w is an arbitrary string witnessing that $[\mathcal{A} \cup B; N_{i,1}, N_{i,2}, \ldots, N_{i,k}]$ is ambiguous, and then move to the next stage.

Otherwise, choose a string $\alpha \in \Sigma^n$ (as guaranteed by Lemma 3.1 and by our choice of n) such that

$$[\mathcal{A}; N_{i,1}, N_{i,2}, \ldots, N_{i,k}](0^n) \text{ accepts } \iff [\mathcal{A} \cup \{\alpha\}; N_{i,1}, N_{i,2}, \ldots, N_{i,k}](0^n) \text{ accepts.}$$

If $[\mathcal{A}; N_{i,1}, N_{i,2}, \ldots, N_{i,k}](0^n)$ accepts, then move to the next stage. Otherwise, if $[\mathcal{A}; N_{i,1}, N_{i,2}, \ldots, N_{i,k}](0^n)$ rejects, then set $\mathcal{A} := \mathcal{A} \cup \{\alpha\}$ and move to the next stage. *End of stage.*

Clearly the construction guarantees that $L(\mathcal{A}) \in \mathrm{UP}^{\mathcal{A}}_{\leq k+1} - \mathrm{U}\Sigma^{p,\mathcal{A}}_k$. □

A straightforward adaptation of the proof of Theorem 4.1 allows us to separate the second level, $\mathcal{U}\Sigma^p_2$, of the promise unambiguous polynomial hierarchy, $\mathcal{UPH}$, from the unambiguous polynomial hierarchy, UPH, in some relativized world. We obtain this relativized separation via Theorem 4.2, where the subclass $\mathrm{FewP}^{\mathcal{A}}$ of $\mathcal{U}\Sigma^{p,\mathcal{A}}_2$ (see Theorem 2.11(3)) is separated from $\mathrm{UPH}^{\mathcal{A}}$.

THEOREM 4.2. *There exists an oracle $\mathcal{A}$ such that* $\mathrm{FewP}^{\mathcal{A}} \not\subseteq \mathrm{UPH}^{\mathcal{A}}$.

Proof sketch. Take the test language $L(\mathcal{A})$ used in the proof of Theorem 4.1. Maintain the stipulation that the oracle $\mathcal{A}$ satisfies, for all $n \in \mathbb{N}^+$, $||\mathcal{A}^{=n}|| \leq n$; this ensures that $L(\mathcal{A}) \in \mathrm{FewP}^{\mathcal{A}}$. Finally, for all $k \in \mathbb{N}^+$, diagonalize against all tuples $(N_{i,1}, N_{i,2}, \ldots, N_{i,k})$ as in the proof of Theorem 4.1. □

COROLLARY 4.3. *There is a relativized world where $\mathcal{U}\Sigma^p_2$ is not contained in* UPH.

Cai, Hemachandra, and Vyskoč [16] proved that smart 2-Turing access to Promise-UP cannot be subsumed by $\mathrm{coNP}^{\mathrm{UP}} \cup \mathrm{NP}^{\mathrm{UP}}$ in some relativized world. As a consequence, they showed that there is a relativized world where smart bounded adaptive reductions to Promise-UP and smart bounded nonadaptive reductions to Promise-UP are nonequivalent, a characteristic that stands in contrast to the cases of UP and NP. (Both UP and NP are known to have equivalence between bounded adaptive reductions and bounded nonadaptive reductions in all relativized worlds (see [16, 48]).) We obtain a generalization of their result as a corollary of Theorem 4.4: There is a relativized world where smart k-Turing access to Promise-UP is not contained in $\mathrm{coNP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{k-1}} \cup \mathrm{NP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{k-1}}$ for any $k \geq 2$. The proof of Theorem 4.4 gives a first example of the role of (h,t)-ambiguity in derivations of our results.

THEOREM 4.4. *For any integer $k \geq 2$, there exists an oracle $\mathcal{A}$ such that*

$$\mathrm{UP}^{\mathcal{A}}_{\leq k} \not\subseteq \mathrm{coNP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{k-1}}.$$

Proof. The test language $L(\mathcal{A})$ is the same as that in the proof of Theorem 4.1. We maintain the stipulation that for every $n \in \mathbb{N}^+$, $||\mathcal{A}^{=n}|| \leq k$; then, clearly $L(\mathcal{A}) \in \mathrm{UP}^{\mathcal{A}}_{\leq k}$. We will show that $L(\mathcal{A}) \notin \mathrm{coNP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{k-1}}$.

Let $(N_{i,1}, N_{i,2}, \ldots, N_{i,k})$ be an enumeration of tuples, where $N_{i,\star}$ is a nondeterministic polynomial-time oracle Turing machine. Initially, $\mathcal{A} := \emptyset$.

Stage i. Let $p(\cdot)$ be a polynomial that bounds the running time of $N_{i,\star}$. Choose a large integer n such that (a) $2^n > 5^{k-1} \cdot \prod_{j=1}^{k} (p\circ)^j(n)$, (b) n satisfies any promises made in the previous stages, (c) n is larger than the value of n in the previous stages, and (d) no queries of length $\geq n$ are made in the previous stages.

If there exists a set $B \subseteq \Sigma^n$ such that $||B|| \leq k$ and $[\mathcal{A} \cup B; N_{i,2}, N_{i,3}, \ldots, N_{i,k}]$ is ambiguous, then set $\mathcal{A} := \mathcal{A} \cup B$. Promise to choose the value of n in the next stage to be larger than $(p\circ)^{k-1}(|w|)$, where w is any arbitrary string witnessing that $[\mathcal{A} \cup B; N_{i,2}, N_{i,3}, \ldots, N_{i,k}]$ is ambiguous, and then move to the next stage.

Otherwise proceed as follows. Let $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ be the function defined so that for every $\mathcal{O} \subseteq \Sigma^*$,

$$\mathcal{L}(\mathcal{O}) =_{df} L_{\text{unambiguous}}[\mathcal{O}; N_{i,2}, N_{i,3}, \ldots, N_{i,k}].$$

It follows from Proposition 3.3 that $\mathcal{L}$ is (k,t)-ambiguous for polynomial $t(\cdot) =_{df} 5^{k-1} \cdot \prod_{j=1}^{k-1}(p\circ)^j(\cdot)$. Notice that for every $B \subseteq \Sigma^n$ satisfying $||B|| \leq k$, $\mathcal{L}(\mathcal{A} \cup B)$ is defined and moreover $\mathcal{L}(\mathcal{A} \cup B) = L[\mathcal{A} \cup B; N_{i,2}, N_{i,3}, \ldots, N_{i,k}]$.

If $N_{i,1}^{\mathcal{L}(\mathcal{A})}(0^n)$ rejects, then move to the next stage. Otherwise, fix an accepting path ρ in $N_{i,1}^{\mathcal{L}(\mathcal{A})}(0^n)$. For each query $w \in Q(\rho)$, let $C(w) =_{df} \{\alpha \in \Sigma^n \mid w \in \mathcal{L}(\mathcal{A} \cup \{\alpha\}) \iff w \notin \mathcal{L}(\mathcal{A})\}$. By the definition of (k,t)-ambiguity, for each $w \in Q(\rho)$, we have $||C(w)|| \leq t(|w|) \leq t(p(n))$. Because n is large enough, we can choose some $\alpha \in \Sigma^n$ such that $\alpha \notin \bigcup_{w \in Q(\rho)} C(w)$. Set $\mathcal{A} := \mathcal{A} \cup \{\alpha\}$ and move to the next stage. *End of stage.*

Clearly the construction guarantees that $L(\mathcal{A}) \notin \text{coNP}^{\text{U}\Sigma_{k-1}^{p,A}}$.

This completes the proof of Theorem 4.4. □

COROLLARY 4.5. *For any integer $k \geq 2$, there exists an oracle $\mathcal{A}$ such that*

$$R^p_{s,k\text{-}T}(\text{Promise-UP}^{\mathcal{A}}) \not\subseteq \text{coNP}^{\text{U}\Sigma_{k-1}^{p,A}} \cup \text{NP}^{\text{U}\Sigma_{k-1}^{p,A}}.$$

Proof. This follows from Theorem 4.4 because for all oracles $\mathcal{A}$, $\text{UP}^{\mathcal{A}}_{\leq k} \subseteq R^p_{s,k\text{-}T}(\text{Promise-UP}^{\mathcal{A}})$ and $R^p_{s,k\text{-}T}(\text{Promise-UP}^{\mathcal{A}})$ is closed under complementation. □

4.2. Simulating nonadaptive access by adaptive access (non–promise case). It is known that adaptive Turing access to NP is exponentially more powerful compared to nonadaptive Turing access to NP. That is, $R^p_{(2^k-1)\text{-}tt}(\text{NP}) \subseteq R^p_{k\text{-}T}(\text{NP})$ [7], and this inclusion relativizes. However, for the case of unambiguous nondeterministic computation, such a relationship between nonadaptive access and adaptive access is not known. Cai, Hemachandra, and Vyskoč [15] showed that even proving the superiority of adaptive Turing access over nonadaptive Turing access with one single query more might be nontrivial for unambiguous nondeterministic computation.

THEOREM 4.6 ([15]). *For any total, polynomial-time computable, and polynomially bounded function $k(\cdot)$, there exists an oracle $\mathcal{A}$ such that*

$$R^p_{(k(n)+1)\text{-}tt}(\text{UP}^{\mathcal{A}}) \not\subseteq R^{p,\mathcal{A}}_{k(n)\text{-}T}(\text{UP}^{\mathcal{A}}).$$

In the next theorem, we generalize this result to the higher levels of the unambiguous polynomial hierarchy UPH.

THEOREM 4.7. *For any total, polynomial-time computable, and polynomially bounded function $k(\cdot)$ and integer $h \geq 1$, there exists an oracle $\mathcal{A}$ such that*

$$\text{R}^p_{(k(n)+1)\text{-}dtt}(\text{UP}^{\mathcal{A}}_{\leq h}) \not\subseteq \text{R}^{p,\mathcal{A}}_{k(n)\text{-}T}(\text{U}\Sigma^{p,\mathcal{A}}_h),$$

and hence $\text{R}^p_{(k(n)+1)\text{-}dtt}(\text{U}\Sigma^{p,\mathcal{A}}_h) \not\subseteq \text{R}^{p,\mathcal{A}}_{k(n)\text{-}T}(\text{U}\Sigma^{p,\mathcal{A}}_h)$.

For the proof of Theorem 4.7, we need the following lemma.

LEMMA 4.8. *Fix an oracle* NPTM N *with running time bounded by some polynomial* $p(\cdot)$*, string* $x \in \Sigma^*$*, and sets* $\mathcal{O}, U_1, U_2 \subseteq \Sigma^*$ *such that* $\mathcal{O} \cap U_1 = \mathcal{O} \cap U_2 = U_1 \cap U_2 = \emptyset$*. Let* $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ *be an* (h,t)*-ambiguous function such that* $\mathcal{L}(\mathcal{O} \cup A_1 \cup A_2)$ *is defined for every* $A_1 \subseteq U_1$ *and* $A_2 \subseteq U_2$ *with* $||A_1|| \leq h$ *and* $||A_2|| \leq h$*. Let*

$$C_1 = \{\alpha \in U_1 \mid N^{\mathcal{L}(\mathcal{O})}(x) \text{ accepts} \iff N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x) \text{ rejects}\} \text{ and}$$
$$C_2 = \{\alpha \in U_2 \mid N^{\mathcal{L}(\mathcal{O})}(x) \text{ accepts} \iff N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x) \text{ rejects}\}.$$

If $N^{\mathcal{L}(\mathcal{O}\cup A_1\cup A_2)}(x)$ *is unambiguous for every* $A_1 \subseteq U_1$ *and* $A_2 \subseteq U_2$ *with* $||A_1|| \leq 1$ *and* $||A_2|| \leq 1$*, then* $\min\{||C_1||, ||C_2||\} \leq 2 \cdot p(|x|) \cdot t(p(|x|)) \cdot (p(|x|) \cdot t(p(|x|)) + 1)$.

Proof. We start with the easier case that $N^{\mathcal{L}(\mathcal{O})}(x)$ accepts. Let ρ be the (unique) accepting computation path in $N^{\mathcal{L}(\mathcal{O})}(x)$. Then $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x)$ accepts unless for some query $w \in Q(\rho)$, it is the case that $w \in \mathcal{L}(\mathcal{O} \cup \{\alpha\}) \iff w \notin \mathcal{L}(\mathcal{O})$. Since $N^{\mathcal{L}(\mathcal{O})}(x)$ queries at most $p(|x|)$ queries along every path, since each query $w \in Q(\rho)$ is of length $\leq p(|x|)$, since $\mathcal{L}$ is (h,t)-ambiguous, and since $\mathcal{L}(\mathcal{O} \cup A_1 \cup A_2)$ is defined for every $A_1 \subseteq U_1$ and $A_2 \subseteq U_2$ with $||A_1||, ||A_2|| \leq h$, there cannot be more than $p(|x|) \cdot t(p(|x|))$ strings $\alpha \in U_1$ (or $\alpha \in U_2$) making $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x)$ reject. Hence $||C_1|| \leq p(|x|) \cdot t(p(|x|))$ and $||C_2|| \leq p(|x|) \cdot t(p(|x|))$.

For the other case, suppose that $N^{\mathcal{L}(\mathcal{O})}(x)$ rejects. To get a contradiction, assume that $\min\{||C_1||, ||C_2||\} > 2 \cdot p(|x|) \cdot t(p(|x|)) \cdot (p(|x|) \cdot t(p(|x|)) + 1)$. For every $\alpha \in C_1 \cup C_2$, denote by $s(\alpha)$ the unique accepting computation path in $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x)$. Define an equivalence relation σ_1 on C_1 as follows: For all $\alpha_1, \alpha_2 \in C_1$,

$$\alpha_1 \sigma_1 \alpha_2 \iff s(\alpha_1) = s(\alpha_2).$$

Let C_1/σ_1 be the quotient set of C_1 determined by σ_1. We first prove that

$$||C_1/\sigma_1|| \geq \frac{||C_1||}{p(|x|) \cdot t(p(|x|))}. \tag{4.1}$$

To this end, consider any $\alpha \in C_1$. Note that for every query $w \in Q(s(\alpha))$, there cannot be more than $t(|w|)$ different strings $\alpha' \in C_1$ such that

$$w \in \mathcal{L}(\mathcal{O} \cup \{\alpha'\}) \iff w \notin \mathcal{L}(\mathcal{O}).$$

This holds because $\mathcal{L}$ is (h,t)-ambiguous and because $\mathcal{L}(\mathcal{O} \cup A)$ is defined for every $A \subseteq U_1$ with $||A|| \leq h$. Also since there are at most $p(|x|)$ queries $w \in Q(s(\alpha))$ and each such query is of length $\leq p(|x|)$, there cannot be more than $p(|x|) \cdot t(p(|x|))$ strings $\alpha' \in C_1$ such that for some query $w \in Q(s(\alpha))$, it is the case that $w \in \mathcal{L}(\mathcal{O} \cup \{\alpha'\})$ if and only if $w \notin \mathcal{L}(\mathcal{O})$. In other words, for all but $p(|x|) \cdot t(p(|x|))$ strings $\alpha' \in C_1$, the membership in $\mathcal{L}(\mathcal{O})$ of every query $w \in Q(s(\alpha))$ remains unchanged on inclusion of α' to $\mathcal{O}$. Since, by assumption, $N^{\mathcal{L}(\mathcal{O})}(x)$ rejects, it follows that for no more than $p(|x|) \cdot t(p(|x|))$ strings $\alpha' \in C_1$, (accepting) path $s(\alpha)$ appears in $N^{\mathcal{L}(\mathcal{O}\cup\alpha')}(x)$. Hence there cannot be more than $p(|x|) \cdot t(p(|x|))$ different strings $\alpha' \in C_1$ such that $s(\alpha') = s(\alpha)$. Thus we have proved statement (4.1).

Analogously, the same can be proved for C_2 with appropriately defined equivalence class σ_2.

Define $\widetilde{C}_1$ to be a maximal subset of C_1 such that $s(\alpha_1) \neq s(\alpha_2)$ for every $\alpha_1, \alpha_2 \in \widetilde{C}_1$ with $\alpha_1 \neq \alpha_2$. Analogously, define $\widetilde{C}_2$. Clearly,

$$||\widetilde{C}_1|| \geq \frac{||C_1||}{p(|x|) \cdot t(p(|x|))} > 2 \cdot (p(|x|) \cdot t(p(|x|)) + 1), \tag{4.2}$$

and

$$||\widetilde{C}_2|| \geq \frac{||C_2||}{p(|x|) \cdot t(p(|x|))} > 2 \cdot (p(|x|) \cdot t(p(|x|)) + 1). \tag{4.3}$$

For every $\alpha_1 \in \widetilde{C}_1$, let

$$\begin{aligned}\text{conflicting}(\alpha_1) = \{\beta_2 \in \widetilde{C}_2 \,\big|\, & s(\alpha_1) \text{ does not appear in } N^{\mathcal{L}(\mathcal{O}\cup\{\alpha_1,\beta_2\})}(x) \\ & \text{or } s(\alpha_1) = s(\beta_2)\},\end{aligned}$$

and for every $\alpha_2 \in \widetilde{C}_2$, let

$$\begin{aligned}\text{conflicting}(\alpha_2) = \{\beta_1 \in \widetilde{C}_1 \,\big|\, & s(\alpha_2) \text{ does not appear in } N^{\mathcal{L}(\mathcal{O}\cup\{\alpha_2,\beta_1\})}(x) \\ & \text{or } s(\alpha_2) = s(\beta_1)\}.\end{aligned}$$

Fix an arbitrary $\alpha \in \widetilde{C}_1$. Since $\mathcal{L}$ is (h,t)-ambiguous and $\mathcal{L}(\mathcal{O} \cup A_1 \cup A_2)$ is defined for every $||A_1|| \leq 1$ and $||A_2|| \leq h$, it holds that for every query $w \in Q(s(\alpha))$ in $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x)$, there cannot be more than $t(|w|)$ different strings $\beta \in \widetilde{C}_2$ such that

$$w \in \mathcal{L}(\mathcal{O} \cup \{\alpha, \beta\}) \iff w \notin \mathcal{L}(\mathcal{O} \cup \{\alpha\}).$$

Also, since no more than $p(|x|)$ strings are queried on each path in $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha\})}(x)$, since each queried string is of length $\leq p(|x|)$, and since, for distinct α_1 and α_2, $s(\alpha_1) \neq s(\alpha_2)$ whenever $\{\alpha_1, \alpha_2\} \subseteq \widetilde{C}_2$, we have that for any $\alpha \in \widetilde{C}_1$,

$$||\text{conflicting}(\alpha)|| \leq p(|x|) \cdot t(p(|x|)) + 1.$$

Analogously, we can prove that for any $\alpha' \in \widetilde{C}_2$,

$$||\text{conflicting}(\alpha')|| \leq p(|x|) \cdot t(p(|x|)) + 1.$$

With the lower bounds on $||\widetilde{C}_1||$ and $||\widetilde{C}_2||$ given by (4.2) and (4.3), it now follows by a simple counting argument that there is a pair $(\alpha_1, \alpha_2) \in \widetilde{C}_1 \times \widetilde{C}_2$ such that $\alpha_2 \notin \text{conflicting}(\alpha_1)$ and $\alpha_1 \notin \text{conflicting}(\alpha_2)$. Take two such strings α_1 and α_2. It is easy to see that both $s(\alpha_1)$ and $s(\alpha_2)$ appear in $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha_1,\alpha_2\})}(x)$. Furthermore, $s(\alpha_1) \neq s(\alpha_2)$. Hence, $N^{\mathcal{L}(\mathcal{O}\cup\{\alpha_1,\alpha_2\})}(x)$ has two different accepting computation paths: $s(\alpha_1)$ and $s(\alpha_2)$. This gives a contradiction to the assumption that $N^{\mathcal{L}(\mathcal{O}\cup A_1\cup A_2)}$ is unambiguous whenever $A_1 \subseteq U_1$ and $A_2 \subseteq U_2$ with $||A_1|| \leq 1$ and $||A_2|| \leq 1$. This completes the proof of Lemma 4.8. □

Proof of Theorem 4.7. The test language $L(\mathcal{A})$ for our oracle construction is inspired by the one in [16, Theorem 3.1]. For length n, we will reserve the following segment of $k(n)+1$ regions $S_{n,f} = 1^n 01^f 0\Sigma^n$, where $f \in [k(n)+1]$. For $n \geq 1$, define $S_n = \bigcup_{f=1}^{k(n)+1} S_{n,f}$. For all $n \geq 1$ and $f \in [k(n)+1]$, we stipulate that $||\mathcal{A} \cap S_{n,f}|| \leq h$. Let

$$L(\mathcal{A}) = \{0^n \,\big|\, ||\mathcal{A} \cap S_n|| \geq 1\}.$$

Clearly, as long as the oracle set $\mathcal{A}$ maintains the stipulation that $||\mathcal{A} \cap S_{n,f}|| \leq h$, we have $L(\mathcal{A}) \in \mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h})$ and hence $L(\mathcal{A}) \in \mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h)$. We construct an oracle $\mathcal{A}$ such that $L(\mathcal{A}) \notin \mathrm{R}^p_{k(n)\text{-}T}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h)$.[3]

[3]The construction can be easily modified to prove the stronger result that $L(\mathcal{A}) \notin \mathrm{R}^{p,\mathcal{A}}_{k(n)-T}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h)$.

We give a brief informal outline of how the diagonalization for $L(\mathcal{A}) \notin \mathrm{R}^p_{k(n)\text{-}T}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h)$ is carried out. Fix some input 0^n. The crucial fact is that we have $k(n)+1$ regions but only $k(n)$ adaptive Turing queries. There are two cases. The easier case is when we can destroy the unambiguity of one of the machines defining the $\mathrm{U}\Sigma^{p,\mathcal{A}}_h$ set by adding some strings to the current segment (but, of course, without violating the above stipulation, i.e., the stipulation that $||\mathcal{A} \cap S_{n,f}|| \leq h$ for every $f \in [k(n)+1]$). In this case we can simply add these strings to the oracle and move to the next stage. Otherwise, we can use Lemma 4.8 to show that each Turing query is *insensitive* to all but at most one of the $k(n)+1$ regions. A Turing query β is *insensitive* to a region if adding a single string α to that region does not change the answer to β, unless the string α comes from a very small (i.e., polynomially bounded) number of exceptions. But we have only $k(n)$ Turing queries. That is why there must be a region $S_{n,f}$ to which all Turing queries are insensitive. Since $S_{n,f}$ has exponentially many strings but only polynomially many exceptions, there must be at least one string $\alpha^* \in S_{n,f}$ that we can add to the current segment without changing the answers to any of the Turing queries, and hence without changing the decision (i.e, acceptance or rejection) of the deterministic machine M making the Turing reduction. We add this string α^* to the oracle (thereby changing the membership of 0^n in the test language) if and only if M rejects. This completes the construction in the current stage, and so we move to the next stage.

Now we come to the formal description of the diagonalization. Let $(N_{i,1}, N_{i,2}, \ldots, N_{i,h}, M_j)$ be an enumeration of tuples where $N_{i,*}$ is a nondeterministic polynomial-time oracle Turing machine, and M_j is a deterministic polynomial-time oracle Turing machine making, for any set $\mathcal{A}$ and any input of length n, at most $k(n)$ queries to $L[\mathcal{A}; N_{i,1}, N_{i,2}, \ldots, N_{i,h}]$. Initially, let $\mathcal{A} := \emptyset$.

Stage $\langle i, j\rangle$. Let $p(\cdot)$ be a polynomial that bounds the running time of both $N_{i,*}$ and M_j. Choose an integer n satisfying the following requirements: (a) $2^n > 2 \cdot k(n) \cdot p(p(n)) \cdot t(p(p(n))) \cdot (p(p(n)) \cdot t(p(p(n))) + 1)$, where $t(\cdot)$ is a polynomial defined later in this proof, (b) n is large enough so that n satisfies any promises made in the previous stages and no string of length greater than or equal to n is queried in any of the previous stages, and (c) n is larger than the value of n in the previous stage.

If there exists a set $B \subseteq S_n$ satisfying $||B \cap S_{n,f}|| \leq h$ for every $f \in [k(n)+1]$ such that $[\mathcal{A} \cup B; N_{i,1}, N_{i,2}, \ldots, N_{i,h}]$ is ambiguous, then set $\mathcal{A} := \mathcal{A} \cup B$. Promise to choose the value of n in the next stage to be larger than $(p\circ)^h(|w|)$, where w is an arbitrary input string witnessing that $[\mathcal{A} \cup B; N_{i,1}, N_{i,2}, \ldots, N_{i,h}]$ is ambiguous, and then move to the next stage.

Otherwise proceed as follows. Let $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ be the function defined so that for every $\mathcal{O} \subseteq \Sigma^*$,

$$\mathcal{L}(\mathcal{O}) =_{df} \begin{cases} \mathcal{O} & \text{if } h = 1, \text{ and} \\ L_{\text{unambiguous}}[\mathcal{O}; N_{i,2}, N_{i,3}, \ldots, N_{i,h}] & \text{if } h \geq 2. \end{cases}$$

It is easy to see that for the case $h = 1$, $\mathcal{L}$ is (h,t)-ambiguous by Definition 3.2, where $t(\cdot) = 1$ is a constant polynomial, and for the case $h \geq 2$, $\mathcal{L}$ is (h,t)-ambiguous by Proposition 3.3, where $t(\cdot) = 5^{h-1} \cdot \prod_{\ell=1}^{h-1} (p\circ)^\ell(\cdot)$. Also, note that for all integers $h \geq 1$, $\mathcal{L}(\mathcal{A} \cup B)$ is defined for every $B \subseteq S_n$ satisfying $||B \cap S_{n,f}|| \leq h$ for every $f \in [k(n)+1]$.

If $M_j(0^n)$ with oracle $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$ is accepting, then move on to the next stage. If $M_j(0^n)$ with oracle $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$ is rejecting, then look for a string $\alpha \in S_n$ that can be

added to $\mathcal{A}$ without changing the decision (i.e., acceptance or rejection) of $M_j(0^n)$ with oracle $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$. Set $\mathcal{A} := \mathcal{A} \cup \{\alpha\}$ and move to the next stage. *End of stage.*

It remains to show that such a string α always exists. Consider $M_j(0^n)$ with oracle $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$. Let $\beta_1, \beta_2, \ldots, \beta_{k(n)}$ be the sequence of queries made by $M_j(0^n)$ to the oracle $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$. The following claim states that, for any $\beta \in \Sigma^*$, there is one special region $S_{n,\text{sensitive}(\beta)}$ such that, for all regions $S_{n,f}$ different from $S_{n,\text{sensitive}(\beta)}$, and for all but polynomially many $\alpha \in S_{n,f}$, the decision (i.e., acceptance or rejection) of $N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta)$ remains unchanged on the addition of α to $\mathcal{A}$.

CLAIM 2. *For each $\beta \in \Sigma^*$, there is an integer* $\text{sensitive}(\beta) \in [k(n)+1]$ *such that the following is true.*

For every $f \in [k(n)+1] - \{\text{sensitive}(\beta)\}$, there is a set $C_f(\beta) \subseteq S_{n,f}$ with $||C_f(\beta)|| \le 2 \cdot p(|\beta|) \cdot t(p(|\beta|)) \cdot (p(|\beta|) \cdot t(p(|\beta|)) + 1)$ such that, for every $\alpha \in S_{n,f} - C_f(\beta)$,

$$\beta \in L(N_{i,1}^{\mathcal{L}(\mathcal{A})}) \iff \beta \in L(N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha\})}).$$

Let us assume that the claim is true. There are $k(n)+1$ regions in S_n but only $k(n)$ queries $\beta_1, \beta_2, \ldots, \beta_{k(n)}$ made by $M_j(0^n)$ to $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$. Let $\ell \in [k(n)+1]$ such that $\ell \neq \text{sensitive}(\beta_e)$ for every $e \in [k(n)]$. Let $C = C_\ell(\beta_1) \cup C_\ell(\beta_2) \cup \cdots \cup C_\ell(\beta_{k(n)})$. Note that, for each $e \in [k(n)]$, $|\beta_e| \le p(n)$ and so $||C_\ell(\beta_e)|| \le 2 \cdot p(p(n)) \cdot t(p(p(n))) \cdot (p(p(n)) \cdot t(p(p(n))) + 1)$. Hence, by the assumption on our choice of n, $||C|| \le k(n) \cdot \max_{\ell \in [k(n)]}\{||C_\ell(\beta_e)||\} < 2^n$.

Let α be any string in $S_{n,\ell} - C$. Such a string exists because $||C|| < 2^n = ||S_{n,\ell}||$. It is easy to see by the above definition of the set C that we can add α to $\mathcal{A}$ without changing the decision of $N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_e)$ for any $e \in [k(n)]$. Clearly, $M_j(0^n)$ with its $k(n)$ queries to $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$ accepts if and only if $M_j(0^n)$ with its $k(n)$ queries to $L(N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha\})})$ accepts. This completes the proof of Theorem 4.7. □

Proof of Claim 2. Suppose the claim were false. Then there exists $\beta \in \Sigma^*$ and $f_1, f_2 \in [k(n)+1]$ with $f_1 \neq f_2$ such that

- $C_{f_1}(\beta) =_{df} ||\{\alpha \in S_{n,f_1} \,|\, \beta \in L(N_{i,1}^{\mathcal{L}(\mathcal{A})}) \iff \beta \notin L(N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha\})})\}|| > 2 \cdot p(|\beta|) \cdot t(p(|\beta|)) \cdot (p(|\beta|) \cdot t(p(|\beta|)) + 1)$, and
- $C_{f_2}(\beta) =_{df} ||\{\alpha \in S_{n,f_2} \,|\, \beta \in L(N_{i,1}^{\mathcal{L}(\mathcal{A})}) \iff \beta \notin L(N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha\})})\}|| > 2 \cdot p(|\beta|) \cdot t(p(|\beta|)) \cdot (p(|\beta|) \cdot t(p(|\beta|)) + 1)$.

Apply Lemma 4.8 with $N := N_{i,1}$, $x := \beta$, $\mathcal{O} := \mathcal{A}$, $U_1 := S_{n,f_1}$, and $U_2 := S_{n,f_2}$. We obtain $\min\{||C_{f_1}(\beta)||, ||C_{f_2}(\beta)||\} \le 2 \cdot p(|\beta|) \cdot t(p(|\beta|)) \cdot (p(|\beta|) \cdot t(p(|\beta|)) + 1)$, which is a contradiction. □

4.3. Simulating nonadaptive access by adaptive access (promise case). Cai, Hemachandra, and Vyskoč [16] proved the following partial improvement of their Theorem 4.6.

THEOREM 4.9 ([16]). *For any constant k, there exists an oracle $\mathcal{A}$ such that*

$$R^p_{(k+1)\text{-}tt}(\text{UP}^{\mathcal{A}}) \not\subseteq R^{p,\mathcal{A}}_{s,k\text{-}T}(\text{Promise-UP}^{\mathcal{A}}).$$

Note that we have replaced "UP" by "Promise-UP" on the right-hand side of the noninclusion relation of Theorem 4.6. This is a significant improvement for the following reason. The computational powers of $R^p_b(\text{UP})$ and $R^p_{s,b}(\text{Promise-UP})$ (the bounded Turing closure of UP and the bounded smart Turing closure of Promise-UP, respectively) are known to be remarkably different in certain relativized worlds. While

it is easy to show that $\mathrm{UP}_{\leq k}$ is robustly (i.e., for every oracle) contained in $\mathrm{P}^{\text{Promise-UP}}_{s,k\text{-}T}$ for any $k \geq 1$, we have shown in the proof of Theorem 4.4 that for no $k \geq 2$ is $\mathrm{UP}_{\leq k}$ robustly contained in P^{UP}. Therefore, it is not immediately clear whether this improvement is impossible, i.e., whether $R^p_{(k+1)\text{-}tt}(\mathrm{UP}) \subseteq R^p_{s,k\text{-}T}(\text{Promise-UP})$ holds relative to all oracles.

However, Cai, Hemachandra, and Vyskoč [16, Theorem 3.1] could achieve this improvement only by paying a heavy price. In their own words:

> In our earlier version dealing with UP^A, the constant k can be replaced by any arbitrary polynomial-time computable function $f(n)$ with polynomially bounded value. It remains open whether the claim of the current strong version of Theorem 3.1 can be similarly generalized to nonconstant access.

We resolve this open question. We show that Theorem 4.9 holds with constant k replaced by any total, polynomial-time computable, and polynomially bounded function $k(\cdot)$. This result is subsumed as the special case $h = 1$ of our main result, Theorem 4.10, of this subsection.

THEOREM 4.10. *For any total, polynomial-time computable, and polynomially bounded function $k(\cdot)$, and integer $h \geq 1$, there exists an oracle $\mathcal{A}$ such that*

$$\mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \not\subseteq \mathrm{R}^{p,\mathcal{A}}_{s,k(n)\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}}),$$

and hence $\mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h) \not\subseteq \mathrm{R}^{p,\mathcal{A}}_{s,k(n)\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$.

Theorem 4.10 is furthermore a generalization of Theorem 4.9 to higher levels of the unambiguous polynomial hierarchy.

The proof of Theorem 4.10 is much more challenging than the proof of Theorem 4.7 because we now require diagonalizing against $\mathrm{R}^{p,\mathcal{A}}_{s,k(n)\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$ as opposed to diagonalizing against $R^{p,\mathcal{A}}_{k(n)\text{-}T}(\mathrm{UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$. To diagonalize against $R^{p,\mathcal{A}}_{k(n)\text{-}T}(\mathrm{UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$ as in the proof of Theorem 4.7, it was sufficient at any stage to extend the current oracle $\mathcal{A}$ so that the $\Sigma_h(\mathcal{A})$-system (corresponding to the stage) becomes ambiguous on some input string, even if the input string witnessing the ambiguity of the $\Sigma_h(\mathcal{A})$-system would never arise in a valid computation of the deterministic querying machine (corresponding to the same stage). This is, however, not sufficient when we diagonalize against $\mathrm{R}^{p,\mathcal{A}}_{s,k(n)\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$. Any input string witnessing the ambiguity of the $\Sigma_h(\mathcal{A})$-system must now have its origin from a valid computation of the deterministic querying machine.

To prove Theorem 4.9, Cai, Hemachandra, and Vyskoč [16] presented the following combinatorial lemma.

LEMMA 4.11 (the gaming lemma [16]). *For $1 \leq i \leq m$, let $\mathcal{S}_i$ be a collection of nonempty subsets of $[n]$ with the following properties:*

1. $(\forall j \in [n])(\exists \ell \in [m])[\{j\} \in \mathcal{S}_\ell]$, *and*
2. $(\forall j \in [m])\,[(A, B \in S_j \text{ and } A \neq B) \Longrightarrow (\exists \ell \in [j-1])(\exists C \in \mathcal{S}_\ell)[C \subseteq A \cup B]]$.

Then $m \geq n$.

Cai, Hemachandra, and Vyskoč [16] gave the following informal interpretation of this lemma. Suppose a combinatorial game is to be played given the set $[n]$ at hand. The game is played in steps by a single player. At each step i, the player can generate a collection $\mathcal{S}_i$ of nonempty subsets of $[n]$ with a restriction: If sets $A, B \subseteq [n]$ are generated at some step $i > 1$, then there must be a previous step $j < i$ and a set C generated at that step such that $C \subseteq A \cup B$. The game ends as soon as all the

singletons $\{k\} \subseteq [n]$ are eventually produced. The gaming lemma states that this combinatorial game requires at least n steps.

Our proof of Theorem 4.10 also makes use of the gaming lemma. However, the actual diagonalization steps are considerably different from those in [16]. The most tricky part is the proof of Lemma 4.12. It demonstrates the existence of especially *nice* strings $\alpha_1, \alpha_2, \ldots, \alpha_r$ satisfying certain useful properties. Each string α_i serves as a representative for one region $S_{n,i}$ of the oracle. These strings satisfy a kind of *independence* property in the following sense. Let A and B be two different minimal subsets of $\{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ that are minimal in the sense that adding A or B to an oracle makes an oracle NPTM N accept, but adding any proper subset of A or B to the oracle makes N reject. Then the independence property implies that adding all the strings in $A \cup B$ to the oracle will make N have at least two accepting paths.

In each stage of the diagonalization, there are two cases. The easier case is when we can destroy the unambiguity of the $\mathrm{U}\Sigma_{h-1}^{p,\mathcal{A}}$ oracle in the $\mathrm{NP}^{\mathrm{U}\Sigma_{h-1}^{p,\mathcal{A}}}$-machine, the machine against which we are diagonalizing, by adding some strings (but, of course, without violating certain requirements of the stage) to the oracle $\mathcal{A}$. In that case, we can simply add these strings to the oracle $\mathcal{A}$ and move to the next stage. Otherwise, we apply the gaming lemma (Lemma 4.11) to show that by adding only the *nice* strings α_i to the oracle $\mathcal{A}$, the desired diagonalization step for the current stage can be achieved. The existence of these *nice* strings α_i was a key idea that led us to the resolution of the question by Cai, Hemachandra, and Vyskoč [16] and in generalizing their result.

LEMMA 4.12. *Fix an oracle* NPTM N *with running time bounded by some polynomial* $p(\cdot)$*, a set* $\mathcal{O} \subseteq \Sigma^*$*, and sets* $X = \{x_1, x_2, \ldots, x_d\} \subseteq (\Sigma^*)^{\leq m}$ *and* $Y = \{y_1, y_2, \ldots, y_{d'}\} \subseteq (\Sigma^*)^{\leq m}$*. Let* $U_1, U_2, \ldots, U_r \subseteq \Sigma^*$ *be such that for each distinct* U_ℓ*,* $U_{\ell'}$*, it holds that* $\mathcal{O} \cap U_\ell = \mathcal{O} \cap U_{\ell'} = U_\ell \cap U_{\ell'} = \emptyset$ *and* $||U_\ell|| = ||U_{\ell'}|| \geq u$*. Let* $\mathcal{L}$ *be an* (h,t)*-ambiguous function such that* $\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in [r]} A_\ell))$ *is defined for every* $A_\ell \subseteq U_\ell$ *satisfying* $||A_\ell|| \leq h$*. If* $u > (3 \cdot d + d') \cdot r \cdot 2^{2r} \cdot p(m) \cdot t(p(m))$*, then there exist strings* $\alpha_1 \in U_1$*,* $\alpha_2 \in U_2, \ldots, \alpha_r \in U_r$ *such that the following properties hold.*

(A) *For every* $x \in X$ *and for every pair of distinct, nonempty sets* S_1*,* $S_2 \in \wp(\{1, 2, \ldots, r\})$*, if the following conditions are satisfied, then there are at least two accepting paths in* $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S_1 \cup S_2} \{\alpha_\ell\}))}(x)$*:*

(A.1) $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S_1} \{\alpha_\ell\}))}(x)$ *accepts and for all* $S_1' \subset S_1$*,* $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S_1'} \{\alpha_\ell\}))}(x)$ *rejects, and*

(A.2) $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S_2} \{\alpha_\ell\}))}(x)$ *accepts and for all* $S_2' \subset S_2$*,* $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S_2'} \{\alpha_\ell\}))}(x)$ *rejects.*

(B) *For every* $y \in Y$ *and for every nonempty set* $S \in \wp(\{1, 2, \ldots, r\})$*, the following is true:*

if $N^{\mathcal{L}(\mathcal{O})}(y)$ *accepts, then* $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S} \{\alpha_\ell\}))}(y)$ *also accepts.*

Proof. We prove the lemma using the probabilistic method. Choose $\alpha_1 \in U_1$, $\alpha_2 \in U_2, \ldots, \alpha_r \in U_r$ uniformly and independently at random. Let E_1 be the event that there exist an $x \in X$ and distinct, nonempty sets $S_1, S_2 \in \wp(\{1, 2, \ldots, r\})$ satisfying the conditions (A.1) and (A.2) given in the lemma, but $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S_1 \cup S_2} \{\alpha_\ell\}))}(x)$ has at most one accepting path. Let E_2 be the event that there exist a $y \in Y$ and a nonempty set $S \in \wp(\{1, 2, \ldots, r\})$ such that $N^{\mathcal{L}(\mathcal{O})}(y)$ accepts, but $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S} \{\alpha_\ell\}))}(y)$ rejects. We will prove that $\mathrm{Prob}(E_1) + \mathrm{Prob}(E_2) < 1$, thus completing the proof of the lemma.

We first make the following claim.

CLAIM 3. *Fix a string* $z \in \Sigma^*$*. Let* $\mathcal{O}$*,* $U_1, U_2, \ldots, U_r$ *be sets defined in the*

statement of Lemma 4.12. *Let $\mathcal{L}$ be an (h,t)-ambiguous function such that $\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell\in[r]} A_\ell))$ is defined for every $A_\ell \subseteq U_\ell$ satisfying $||A_\ell|| \leq h$. Let $T \subseteq [r]$. Fix $\alpha_i \in U_i$ for each $i \in T$ arbitrarily. If we choose $\alpha_j \in U_j$ for each $j \in [r]-T$, uniformly and independently at random, then for any T_1, $T_2 \subseteq [r]$ satisfying $(T_1 \Delta T_2) \cap T = \emptyset$,*

$$z \in \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_1}\{\alpha_\ell\}\right)\right) \Longleftrightarrow z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_2}\{\alpha_\ell\}\right)\right)$$

is true with probability $\leq r \cdot t(|z|)/u$.

Proof of Claim 3. Let $V = T_1 \cap T_2$. Because $\mathcal{L}$ is (h,t)-ambiguous, for any $j \in T_1 \Delta T_2$, the probability over uniform random choice of $\alpha_j \in U_j$ that

$$z \in \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in V}\{\alpha_\ell\}\right)\right) \Longleftrightarrow z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in V}\{\alpha_\ell\}\right) \cup \{\alpha_j\}\right)$$

is at most $t(|z|)/u$. Successively choose $\alpha_j \in U_j$ uniformly at random, where $j \in T_1 \Delta T_2$, and add j to V until V equals T_1. The probability that

$$z \in \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_1}\{\alpha_\ell\}\right)\right) \Longleftrightarrow z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_1\cap T_2}\{\alpha_\ell\}\right)\right)$$

is, therefore, at most $||T_1 - T_2|| \cdot t(|z|)/u$. Likewise, the probability that

$$z \in \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_2}\{\alpha_\ell\}\right)\right) \Longleftrightarrow z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_1\cap T_2}\{\alpha_\ell\}\right)\right)$$

is at most $||T_2 - T_1|| \cdot t(|z|)/u$. Hence the probability that

$$z \in \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_1}\{\alpha_\ell\}\right)\right) \Longleftrightarrow z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell\in T_2}\{\alpha_\ell\}\right)\right)$$

is at most $(||T_1-T_2||+||T_2-T_1||)\cdot t(|z|)/u \leq r\cdot t(|z|)/u$. Thus Claim 3 is proved. □

Using Claim 3, we obtain an upper bound on $\mathrm{Prob}(E_1)$ in Claim 4 and an upper bound on $\mathrm{Prob}(E_2)$ in Claim 5.

Claim 4. $\mathrm{Prob}(E_1) \leq 3 \cdot d \cdot r \cdot 2^{2r} \cdot p(m) \cdot t(p(m))/u$.

Proof of Claim 4. Let C_{x,S_1,S_2} stand for the condition "input x and the pair S_1, S_2 satisfy the conditions (A.1) and (A.2) given in the lemma." Fix an $x \in X$ and distinct, nonempty sets S_1 and S_2. Let E_{x,S_1,S_2} denote the event that C_{x,S_1,S_2} is satisfied but $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_1\cup S_2}\{\alpha_\ell\}))}(x)$ has at most one accepting path. Clearly, $\mathrm{Prob}(E_1) \leq \sum_{x,S_1,S_2} \mathrm{Prob}(E_{x,S_1,S_2})$. If condition C_{x,S_1,S_2} is satisfied, then for each $i \in \{1,2\}$, we can fix an accepting path $\rho(S_i)$ in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_i}\{\alpha_\ell\}))}(x)$. For definiteness, let $\rho(S_i)$ be the lexicographically first accepting path in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_i}\{\alpha_\ell\}))}(x)$.

If E_{x,S_1,S_2} occurs, then at least one of the events J, $H(S_1)$, or $H(S_2)$ occurs, where

- J is the event that condition C_{x,S_1,S_2} is satisfied, and $\rho(S_1)$ and $\rho(S_2)$ are equal,
- $H(S_1)$ is the event that condition C_{x,S_1,S_2} is satisfied, and $\rho(S_1)$ does not appear in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_1\cup S_2}\{\alpha_\ell\}))}(x)$, and

- $H(S_2)$ is the event that condition C_{x,S_1,S_2} is satisfied, and $\rho(S_2)$ does not appear in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_1\cup S_2}\{\alpha_\ell\}))}(x)$.

We first determine an upper bound on $\mathrm{Prob}(J)$. To this end, we determine an upper bound on $\mathrm{Prob}(J \mid \alpha_i = \beta_i \text{ for all } i \in S_1)$ for arbitrary fixed strings $\beta_i \in U_i$ for each $i \in S_1$. Hence, suppose henceforth that $\alpha_i = \beta_i$ for all $i \in S_1$. Because $S_1 \neq S_2$ and because we can assume that S_1 satisfies the condition (A.1) given in the lemma, the (accepting) path $\rho(S_1)$ in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_1}\{\alpha_\ell\}))}(x)$ does not appear in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_1\cap S_2}\{\alpha_\ell\}))}(x)$. Hence, for at least one string $z \in Q(\rho(S_1))$ queried along $\rho(S_1)$, we have

$$z \in \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_1\cap S_2}\{\alpha_\ell\}\right)\right) \iff z \notin \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_1}\{\alpha_\ell\}\right)\right). \tag{4.4}$$

Note that this condition depends only on strings α_i with $i \in S_1$, which are fixed by $\alpha_i = \beta_i$. Fix one string z satisfying statement (4.4). Applying Claim 3 with $T := S_1$, $T_1 := S_1 \cap S_2$, and $T_2 := S_2$, we get that

$$z \in \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_1\cap S_2}\{\alpha_\ell\}\right)\right) \iff z \in \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_2}\{\alpha_\ell\}\right)\right) \tag{4.5}$$

holds with probability $\geq 1 - r\cdot t(|z|)/u \geq 1 - r\cdot t(p(m))/u$. Statements (4.4) and (4.5) together imply that with probability $\geq 1 - r\cdot t(p(m))/u$,

$$z \in \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_1}\{\alpha_\ell\}\right)\right) \iff z \notin \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_2}\{\alpha_\ell\}\right)\right). \tag{4.6}$$

Hence with probability $\geq 1 - r\cdot t(p(m))/u$, path $\rho(S_1)$ does not appear in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_2}\{\alpha_\ell\}))}(x)$, and therefore $\rho(S_1) \neq \rho(S_2)$. We have proven that for arbitrary fixed strings $\{\beta_i \in U_i \,|\, i \in S_1\}$, it holds that $\mathrm{Prob}(J \mid \alpha_i = \beta_i \text{ for all } i \in S_1) \leq r\cdot t(p(m))/u$. By the law of total probability, also $\mathrm{Prob}(J) \leq r\cdot t(p(m))/u$.

To determine an upper bound on $\mathrm{Prob}(H(S_1))$, we determine an upper bound on $\mathrm{Prob}(H(S_1) \mid \alpha_i = \beta_i \text{ for all } i \in S_1)$ for arbitrary fixed strings $\beta_i \in U_i$ for each $i \in S_1$. Hence, suppose henceforth that $\alpha_i = \beta_i$ for all $i \in S_1$. Clearly, $\rho(S_1)$ depends only on strings α_i with $i \in S_1$, which we have fixed by $\alpha_i = \beta_i$. Then the event $H(S_1)$ occurs only if it holds that $\rho(S_1)$ queries some string z with

$$z \in \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_1}\{\alpha_\ell\}\right)\right) \iff z \notin \mathcal{L}\left(\mathcal{O}\cup\left(\bigcup_{\ell\in S_1\cup S_2}\{\alpha_\ell\}\right)\right).$$

Note that path $\rho(S_1)$ in $N^{\mathcal{L}(\mathcal{O}\cup(\bigcup_{\ell\in S_1}\{\alpha_\ell\}))}(x)$ queries at most $p(m)$ strings, each of which is of length $\leq p(m)$. Applying Claim 3 with $T := S_1$, $T_1 := S_1$, and $T_2 := S_1 \cup S_2$, we get that $\mathrm{Prob}(H(S_1) \mid \alpha_i = \beta_i \text{ for all } i \in S_1) \leq p(m)\cdot r\cdot t(p(m))/u$. We have thus proven that for arbitrary fixed strings $\{\beta_i \in U_i \,|\, i \in S_1\}$, it holds that $\mathrm{Prob}(H(S_1) \mid \alpha_i = \beta_i \text{ for all } i \in S_1) \leq p(m)\cdot r\cdot t(p(m))/u$. Therefore, also $\mathrm{Prob}(H(S_1)) \leq p(m)\cdot r\cdot t(p(m))/u$.

Analogously, we can prove that $\mathrm{Prob}(H(S_2)) \leq p(m)\cdot r\cdot t(p(m))/u$.

Thus $\mathrm{Prob}(E_{x,S_1,S_2}) \leq \mathrm{Prob}(J \cup H(S_1) \cup H(S_2)) \leq \mathrm{Prob}(J) + \mathrm{Prob}(H(S_1)) + \mathrm{Prob}(H(S_2)) \leq 3\cdot p(m)\cdot r\cdot t(p(m))/u$. It follows that $\mathrm{Prob}(E_1) \leq 3\cdot d\cdot r\cdot 2^{2r}\cdot p(m)\cdot t(p(m))/u$. This completes the proof of Claim 4. □

CLAIM 5. $\mathrm{Prob}(E_2) \le d' \cdot r \cdot 2^r \cdot p(m) \cdot t(p(m))/u$.

Proof of Claim 5. Fix a $y \in Y$ and a nonempty set $S \in \wp(\{1, 2, \ldots, r\})$. Let $E_{y,S}$ denote the event that $N^{\mathcal{L}(\mathcal{O})}(y)$ accepts but $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S} \{\alpha_\ell\}))}(y)$ rejects. Clearly, $\mathrm{Prob}(E_2) \le \sum_{y,S} \mathrm{Prob}(E_{y,S})$. If $N^{\mathcal{L}(\mathcal{O})}(y)$ accepts, then we can fix an accepting path ρ in $N^{\mathcal{L}(\mathcal{O})}(y)$. For definiteness, let ρ be the lexicographically first accepting path in $N^{\mathcal{L}(\mathcal{O})}(y)$.

Suppose $E_{y,S}$ occurs. Since $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S} \{\alpha_\ell\}))}(y)$ rejects, the (accepting) path ρ in $N^{\mathcal{L}(\mathcal{O})}(y)$ does not appear in $N^{\mathcal{L}(\mathcal{O} \cup (\bigcup_{\ell \in S} \{\alpha_\ell\}))}(y)$. Hence for at least one string $z \in Q(\rho)$ queried along ρ, we have

$$z \in \mathcal{L}(\mathcal{O}) \iff z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell \in S} \{\alpha_\ell\}\right)\right).$$

Applying Claim 3 with $T := \emptyset$, $T_1 := \emptyset$, and $T_2 := S$, we get that for each $z \in Q(\rho)$,

$$z \in \mathcal{L}(\mathcal{O}) \iff z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell \in S} \{\alpha_\ell\}\right)\right)$$

holds with probability $\le r \cdot t(|z|)/u$. Since $||Q(\rho)|| \le p(m)$ and since the length of each query $z \in Q(\rho)$ is at most $p(m)$, with probability $\le p(m) \cdot r \cdot t(p(m))/u$ we have, for some $z \in Q(\rho)$,

$$z \in \mathcal{L}(\mathcal{O}) \iff z \notin \mathcal{L}\left(\mathcal{O} \cup \left(\bigcup_{\ell \in S} \{\alpha_\ell\}\right)\right).$$

Thus we have shown that $\mathrm{Prob}(E_{y,S}) \le p(m) \cdot r \cdot t(p(m))/u$. It follows that $\mathrm{Prob}(E_2) \le d' \cdot r \cdot 2^r \cdot p(m) \cdot t(p(m))/u$. Thus Claim 5 is proved. □

From Claim 4 and Claim 5, we get $\mathrm{Prob}(E_1) + \mathrm{Prob}(E_2) \le (3 \cdot d + d') \cdot r \cdot 2^{2r} \cdot p(m) \cdot t(p(m))/u < 1$, by our choice of u. This completes the proof of Lemma 4.12. □

Now it is relatively easy to prove the main result of this subsection.

Proof of Theorem 4.10. For each length n, we will reserve the following segment of $k(n) + 1$ regions $S_{n,f} = 1^n 01^f 0 \Sigma^{3k(n)+n}$, where $f \in [k(n) + 1]$. For $n \ge 1$, define $S_n = \bigcup_{f=1}^{k(n)+1} S_{n,f}$. We take the test language $L(\mathcal{A})$ used in the proof of Theorem 4.7. The oracle $\mathcal{A}$ is constructed in stages. Let $(N_{i,1}, N_{i,2}, \ldots, N_{i,h}, M_j)$ be an enumeration of tuples where $N_{i,*}$ is a nondeterministic polynomial-time oracle Turing machine, and M_j is a deterministic polynomial-time oracle Turing machine making, for any set $\mathcal{A}$ and any input of length n, at most $k(n)$ queries to $L[\mathcal{A}; N_{i,1}, N_{i,2}, \ldots, N_{i,h}]$ and at most polynomially many queries to $\mathcal{A}$. Initially $\mathcal{A} := \emptyset$.

Stage $\langle i, j \rangle$. Let $p(\cdot)$ be a polynomial that bounds the running time of both $N_{i,\star}$ and M_j. Choose a very large integer n such that (a) $2^{3k(n)+n} - p(n) > 4 \cdot k(n) \cdot (k(n) + 1) \cdot 2^{2(k(n)+1)} \cdot p(p(n)) \cdot t(p(p(n)))$, where $t(\cdot)$ is a polynomial defined later in this proof, (b) no string of length n or more is queried in any of the previous stages, (c) n is larger than the value in the previous stage, and (d) n satisfies any promises made in the previous stages.

The easy case. (i) For $h = 1$, if there exists a set $B \subseteq S_n$, satisfying $||B \cap S_{n,f}|| \le h$ for every $f \in [k(n)+1]$, such that the requirement R1 is satisfied, where R1 is defined as

- R1: $M_j(0^n)$ makes a query β to $L[\mathcal{A} \cup B; N_{i,1}]$ and $N_{i,1}^{\mathcal{A} \cup B}(\beta)$ has at least two accepting paths,

then set $\mathcal{A} := \mathcal{A} \cup B$. (ii) For $h \geq 2$, if there exists a set $B \subseteq S_n$, satisfying $||B \cap S_{n,f}|| \leq h$ for every $f \in [k(n)+1]$, such that at least one of the requirements R2 and R3 is satisfied, where R2 and R3 are defined as

- R2: $[\mathcal{A} \cup B; N_{i,2}, N_{i,3}, \ldots, N_{i,h}]$ is ambiguous,
- R3: $M_j(0^n)$ makes a query β to $L[\mathcal{A} \cup B; N_{i,1}, N_{i,2}, \ldots, N_{i,h}]$ and $N_{i,1}^{L[\mathcal{A}\cup B; N_{i,2}, N_{i,3}, \ldots, N_{i,h}]}(\beta)$ has at least two accepting paths,

then set $\mathcal{A} := \mathcal{A} \cup B$.

For both (i) and (ii), promise to choose the value of n in the next stage to be sufficiently large so that the requirements satisfied in this stage cannot become invalid in the next stage.

The hard case. This means that there is no set $B \subseteq S_n$ as described in (i) and (ii) above. We proceed as follows. Define function $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ such that for every $\mathcal{O} \subseteq \Sigma^*$,

$$\mathcal{L}(\mathcal{O}) =_{df} \begin{cases} \mathcal{O} & \text{if } h = 1, \text{ and} \\ L_{\text{unambiguous}}[\mathcal{O}; N_{i,2}, N_{i,3}, \ldots, N_{i,h}] & \text{if } h \geq 2. \end{cases}$$

It is easy to see that for the case $h = 1$, $\mathcal{L}$ is (h,t)-ambiguous by Definition 3.2, where $t(\cdot) = 1$ is a constant polynomial, and for the case $h \geq 2$, $\mathcal{L}$ is (h,t)-ambiguous by Proposition 3.3, where $t(\cdot) = 5^{h-1} \cdot \prod_{\ell=1}^{h-1} (p\circ)^\ell(\cdot)$. Also, note that (since we are in the hard case) for all integers $h \geq 1$, $\mathcal{L}(\mathcal{A} \cup B)$ is defined for every $B \subseteq S_n$ that satisfies $||B \cap S_{n,f}|| \leq h$ for every $f \in [k(n)+1]$. We next use Claim 6 to successfully finish this stage.

Claim 6. *There is a string $\alpha \in S_n$ such that $M_j(0^n)$ with oracle $\mathcal{A} \oplus L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$ is identical to $M_j(0^n)$ with oracle $(\mathcal{A} \cup \{\alpha\}) \oplus L(N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha\})})$.*

That is, if $M_j(0^n)$ with oracle $\mathcal{A} \oplus L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$ rejects, then we set $\mathcal{A} := \mathcal{A} \cup \{\alpha\}$; otherwise, if $M_j(0^n)$ with oracle $\mathcal{A} \oplus L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$ accepts, then we leave $\mathcal{A}$ unchanged. Finally, we move to the next stage. *End of stage.*

This completes the proof of Theorem 4.10. □

Proof of Claim 6. Let $\beta_1, \beta_2, \ldots, \beta_{k(n)}$ be the sequence of queries made by $M_j(0^n)$ to the oracle $L(N_{i,1}^{\mathcal{L}(\mathcal{A})})$. Let $I = \{\ell \mid N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_\ell) \text{ accepts}\}$. Let $\widetilde{Q}$ be the set of strings that are queried by $M_j(0^n)$ to oracle $\mathcal{A}$. Clearly, $||\widetilde{Q}|| \leq p(n)$.

Apply Lemma 4.12 with $N := N_{i,1}$, $\mathcal{O} := \mathcal{A}$, $d := k(n)$, $d' := k(n)$, $X = \{\beta_1, \beta_2, \ldots, \beta_{k(n)}\}$, $Y = \{\beta_1, \beta_2, \ldots, \beta_{k(n)}\}$, $m := p(n)$, $r := k(n)+1$, $U_f := S_{n,f} - \widetilde{Q}$ for each $f \in [k(n)+1]$, and $u := 2^{3k(n)+n} - p(n)$. We obtain strings $\alpha_1 \in S_{n,1} - \widetilde{Q}$, $\alpha_2 \in S_{n,2} - \widetilde{Q}, \ldots, \alpha_{k(n)+1} \in S_{n,k(n)+1} - \widetilde{Q}$, which satisfy the properties (A) and (B) given in Lemma 4.12. Now assign to each query β_ℓ with $\ell \in [k(n)] - I$ a collection $\mathcal{S}_\ell \subseteq \wp([k(n)+1])$ in the following way: $\{f_1, f_2, \ldots, f_s\} \in \mathcal{S}_\ell$ if and only if

(a) adding $\{\alpha_{f_1}, \alpha_{f_2}, \ldots, \alpha_{f_s}\}$ to $\mathcal{A}$ makes $N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_\ell)$ change from rejection to acceptance, i.e., $N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_\ell)$ rejects but $N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha_{f_1}, \alpha_{f_2}, \ldots, \alpha_{f_s}\})}(\beta_\ell)$ accepts, and

(b) no set $T \subset \{f_1, f_2, \ldots, f_s\}$ satisfies (a), i.e., for no such set T does it hold that $N_{i,1}^{\mathcal{L}(\mathcal{A}\cup(\bigcup_{j\in T}\{\alpha_j\}))}(\beta_\ell)$ accepts.

Note that no collection $\mathcal{S}_\ell$ contains the empty set. However, some of these collections may be empty.

Suppose that Claim 6 is not true. Then, for every $e \in [k(n)+1]$, there is an $\ell \in [k(n)]$ such that

$$N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_\ell) \text{ rejects } \iff N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha_e\})}(\beta_\ell) \text{ accepts.} \tag{4.7}$$

Because $\alpha_1, \alpha_2, \ldots, \alpha_{k(n)+1}$ satisfy property (B) of Lemma 4.12, statement (4.7) can only be true for $\ell \notin I$. This implies that for every $e \in [k(n)+1]$, there is an $\ell \in [k(n)] - I$ such that

$$N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_\ell) \text{ rejects and } N_{i,1}^{\mathcal{L}(\mathcal{A}\cup\{\alpha_e\})}(\beta_\ell) \text{ accepts.}$$

It follows from the definition of the collections $\mathcal{S}_\ell$ that for every $e \in [k(n)+1]$, there is a collection $\mathcal{S}_\ell$ such that the singleton $\{\alpha_e\}$ is contained in $\mathcal{S}_\ell$. Thus we have proven condition 1 of Lemma 4.11 (the gaming lemma).

Now take two distinct sets $A, B \in \mathcal{S}_\ell$ for some $\ell \in [k(n)]$. Then by the definition of the collections $\mathcal{S}_\ell$ together with property (A) of Lemma 4.12, $N_{i,1}^{\mathcal{L}(\mathcal{A}\cup(\bigcup_{e\in A\cup B}\{\alpha_e\}))}(\beta_\ell)$ has at least two accepting paths. Notice that for any valid query β by $M_j(0^n)$ to $L(N_{i,1}^{\mathcal{L}(\mathcal{A}\cup(\bigcup_{e\in A\cup B}\{\alpha_e\}))})$, there can be at most one accepting path in $N_{i,1}^{\mathcal{L}(\mathcal{A}\cup(\bigcup_{e\in A\cup B}\{\alpha_e\}))}(\beta)$ (since we are in the *hard* case). Therefore, we can be sure that adding $\bigcup_{e\in A\cup B}\{\alpha_e\}$ to $\mathcal{A}$ changes the decision (i.e., acceptance or rejection) of $N_{i,1}^{\mathcal{L}(\mathcal{A})}$ for a previous query $\beta_{\ell'}$ with $\ell' < \ell$. The decision of $N_{i,1}^{\mathcal{L}(\mathcal{A})}(\beta_{\ell'})$ on the addition of $\bigcup_{e\in A\cup B}\{\alpha_e\}$ to $\mathcal{A}$ must change from rejection to acceptance, and not from acceptance to rejection, because $\alpha_1, \alpha_2, \ldots, \alpha_{k(n)+1}$ satisfy property (B) of Lemma 4.12. Hence there is a set $C \in \mathcal{S}_{\ell'}$ such that $C \subseteq A \cup B$. This proves condition 2 of Lemma 4.11. Lemma 4.11 implies that the number of queries $k(n)$ is greater than or equal to the number of regions $k(n)+1$, which is a contradiction. This completes the proof of Claim 6. □

4.4. Simulating adaptive access by nonadaptive access. Sections 4.2 and 4.3 studied the limitations of simulating nonadaptive queries to $\mathrm{UP}_{\leq h}$ by adaptive queries to $\mathrm{U}\Sigma_h^p$ in relativized settings. This section complements these investigations. In particular, Corollary 4.14 of this section shows that in a certain relativized world, it is impossible to simulate adaptive k-Turing access to $\mathrm{UP}_{\leq h}$ by nonadaptive (2^k-2)-tt access to $\mathrm{U}\Sigma_h^p$. This also implies optimality of robustly (i.e., for every oracle) simulating adaptive k-Turing accesses by nonadaptive (2^k-1)-tt accesses to classes such as $\mathrm{UP}_{\leq h}$ and $\mathrm{U}\Sigma_h^p$, since for any class $\mathcal{C}$ we can easily, via a brute-force method, simulate adaptive k-Turing reduction to $\mathcal{C}$ by nonadaptive (2^k-1)-tt reduction to $\mathcal{C}$.

The proof of Theorem 4.13 employs a technique of Buhrman, Spaan, and Torenvliet [12], which Cai, Hemachandra, and Vyskoč [16] referred to as the "force your way through the tree" technique. Buhrman, Spaan, and Torenvliet [12] used their technique to prove that NEXP has a set that is complete for k-Turing reductions but not complete for (2^k-2)-tt reductions. Cai, Hemachandra, and Vyskoč [16] used this technique to prove Theorem 4.13 for the case of $h = 1$. We use the same approach to generalize the result of Cai, Hemachandra, and Vyskoč [16] from the case of $h = 1$ to the case of arbitrary integer $h \geq 1$.

THEOREM 4.13. *For any integers $k, h \geq 1$, there exists an oracle $\mathcal{A}$ such that*

$$R_{k\text{-}T}^p(\mathrm{UP}_{\leq h}^{\mathcal{A}}) \not\subseteq R_{(2^k-2)\text{-}tt}^{p,\mathcal{A}}(\mathrm{NP}^{\mathrm{U}\Sigma_{h-1}^{p,\mathcal{A}}}).$$

Proof. The theorem clearly holds for $k = 1$ since there is an oracle that separates UP from P. Henceforth, we assume that $k \geq 2$. For each length n, we will reserve the following segment of $2^k - 1$ regions: $S_{n,f} = 1^n 0 f 0 \Sigma^n$, where $f \in (\Sigma^*)^{\leq k-1}$. For $n \geq 1$, define $S_n = \bigcup_{f\in(\Sigma^*)^{\leq k-1}} S_{n,f}$. For each length n and set $A \subseteq \Sigma^*$, we also

define a sequence $b^A_{n,1}b^A_{n,2}\dots b^A_{n,k}$ of bits as follows:

$$(\star) \qquad b^A_{n,1} = \begin{cases} 1 & \text{if } S_{n,\epsilon} \bigcap A \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

and for each ℓ with $2 \leq \ell \leq k$,

$$b^A_{n,\ell} = \begin{cases} 1 & \text{if } S_{n,b^A_{n,1}b^A_{n,2}\dots b^A_{n,\ell-1}} \bigcap A \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Our test language is $L(\mathcal{A}) = \{0^n \mid b^{\mathcal{A}}_{n,k} = 1\}$. Note that our test language is the same as that in [16, Theorem 3.3]. We stipulate that for all $n \geq 1$ and for all $f \in (\Sigma^*)^{\leq k-1}$, $||S_{n,f}|| \leq h$. Clearly, if the oracle set $\mathcal{A}$ maintains this stipulation, then we have $L(\mathcal{A}) \in R^p_{k\text{-}T}(\mathrm{UP}^{\mathcal{A}}_{\leq h})$. We construct an oracle $\mathcal{A}$ such that $L(\mathcal{A}) \notin R^p_{(2^k-2)\text{-}tt}(\mathrm{NP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$. The construction can be easily modified to prove the stronger result that $L(\mathcal{A}) \notin R^{p,\mathcal{A}}_{(2^k-2)\text{-}tt}(\mathrm{NP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$.

Let $(N_{i,1}, N_{i,2}, \dots, N_{i,h}, M_j)$ be an enumeration of tuples, where $N_{i,*}$ is a nondeterministic polynomial-time oracle Turing machine, and M_j is a deterministic polynomial-time oracle Turing machine making, for any set $\mathcal{A}$ and for any input, at most $2^k - 2$ nonadaptive queries to $L[\mathcal{A}; N_{i,1}, N_{i,2}, \dots, N_{i,h}]$. Initially, let $\mathcal{A} := \emptyset$.

Stage $\langle i, j\rangle$. Let $p(\cdot)$ be a polynomial that bounds the running time of both $N_{i,\star}$ and M_j. Choose a very large integer n such that (a) $2^n > 2^{2k} \cdot p(p(n)) \cdot t(p(p(n)))$, where $t(\cdot)$ is a polynomial defined later in this proof, (b) no string of length n or more is queried in any of the previous stages, (c) n is larger than the value in the previous stage, and (d) n satisfies any promises made in the previous stages.

For the case $h \geq 2$, if there exists a set $B \subseteq S_n$ satisfying $||B \cap S_{n,f}|| \leq h$ for every $f \in (\Sigma^*)^{\leq k-1}$ such that $[\mathcal{A} \cup B; N_{i,2}, \dots, N_{i,h}]$ is not unambiguous, then set $\mathcal{A} := \mathcal{A} \cup B$. Promise to choose the value of n in the next stage to be larger than $(p\circ)^{h-1}(|w|)$, where w is an arbitrary string witnessing that $[\mathcal{A} \cup B; N_{i,2}, \dots, N_{i,h}]$ is not unambiguous, and then move to the next stage.

If no such set B exists for $h \geq 2$ or if $h = 1$, then we define a function $\mathcal{L} : \wp(\Sigma^*) \to \wp(\Sigma^*)$ as follows: For every $\mathcal{O} \subseteq \Sigma^*$, let

$$\mathcal{L}(\mathcal{O}) =_{df} \begin{cases} \mathcal{O} & \text{if } h = 1, \text{ and} \\ L_{\text{unambiguous}}[\mathcal{O}; N_{i,2}, N_{i,3}, \dots, N_{i,h}] & \text{if } h \geq 2. \end{cases}$$

It is easy to see that for the case $h = 1$, $\mathcal{L}$ is (h,t)-ambiguous by Definition 3.2, where $t(\cdot) = 1$ is a constant polynomial, and for the case $h \geq 2$, $\mathcal{L}$ is (h,t)-ambiguous by Proposition 3.3, where $t(\cdot) = 5^{h-1} \cdot \prod_{\ell=1}^{h-1}(p\circ)^\ell(\cdot)$. Also, note that for all integers $h \geq 1$, $\mathcal{L}(\mathcal{A} \cup B)$ is defined for every $B \subseteq S_n$ that satisfies $||B \cap S_{n,f}|| \leq h$ for every $f \in (\Sigma^*)^{\leq k-1}$.

Let $\beta_1, \beta_2, \dots, \beta_{2^k-2}$ be the sequence of nonadaptive queries made by $M_j(0^n)$ to the oracle $L(N^{\mathcal{L}(\mathcal{A})}_{i,1})$. Consider the following procedure *Diagonalize*.

Procedure Diagonalize($\{\alpha_f \in S_{n,f} \mid f \in (\Sigma^*)^{\leq k-1}\}$)
1. $\mathcal{A}_1 := \mathcal{A}$;
2. For $t := 1$ to $2^k - 1$ do
3. Let $b_{n,1}^{\mathcal{A}_t} b_{n,2}^{\mathcal{A}_t} \dots b_{n,k}^{\mathcal{A}_t}$ be the bit sequence given by ($\star$);
4. If ($b_{n,k}^{\mathcal{A}_t} = 0 \iff M_j^{L(N_{i,1}^{\mathcal{L}(\mathcal{A}_t)})}(0^n)$ accepts) is true, then
5. output $\mathcal{A}_t$ and terminate;
6. Else /* That is, $b_{n,k}^{\mathcal{A}_t} = 0 \iff M_j^{L(N_{i,1}^{\mathcal{L}(\mathcal{A}_t)})}(0^n)$ rejects */
(6.1) Let $s := \max\{\ell \in [k] \mid b_{n,\ell}^{\mathcal{A}_t} = 0\}$;
(6.2) $\mathcal{A}_{t+1} := \mathcal{A}_t \cup \{\alpha_{b_{n,1}^{\mathcal{A}_t} b_{n,2}^{\mathcal{A}_t} \cdots b_{n,s-1}^{\mathcal{A}_t}}\}$; /* That is, flip $b_{n,k}^{\mathcal{A}_t}$. */
(6.3) If for every query β_ℓ, where $\ell \in [2^k - 2]$, the following holds:
"if $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$ rejects, then $N_{i,1}^{\mathcal{L}(\mathcal{A}_{t+1})}(\beta_\ell)$ also rejects," then
output $\mathcal{A}_{t+1}$ and terminate.
(6.4) Else /* there is a query β_ℓ such that $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$ rejects, but $N_{i,1}^{\mathcal{L}(\mathcal{A}_{t+1})}(\beta_\ell)$ accepts. */
Return to the For loop;
End of Procedure

CLAIM 7. *For each $f \in (\Sigma^*)^{\leq k-1}$, there exists $\widehat{\alpha_f} \in S_{n,f}$ such that for each $t \in [2^k - 1]$ and for each $\ell \in [2^k - 2]$, the following holds in the execution of Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$):*

$$\text{if } N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell) \text{ accepts, then } N_{i,1}^{\mathcal{L}(\mathcal{A}_{t+1})}(\beta_\ell) \text{ also accepts.}$$

Let us assume that Claim 7 is true. Then there exist strings $\widehat{\alpha_f} \in S_{n,f}$ for each $f \in (\Sigma^*)^{\leq k-1}$, satisfying the property stated in the claim. Set $\mathcal{A} :=$ Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) and move to the next stage. *End of stage.*

For each $f \in (\Sigma^*)^{\leq k-1}$, let $\widehat{\alpha_f} \in S_{n,f}$ be the strings promised in that claim. Notice that the procedure Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) never adds more than one string in any region $S_{n,f}$. This follows because each region $S_{n,f}$ is associated with exactly one string $\widehat{\alpha_f} \in S_{n,f}$, and only these associated strings are ever considered for inclusion in the oracle. Also note that the effect of step (6.2) in the procedure is to increment the binary number $b_{n,1}^{\mathcal{A}_t} b_{n,2}^{\mathcal{A}_t} \dots b_{n,k}^{\mathcal{A}_t}$ by 1. That is, we have, for each $t \in [2^k - 1]$ considered until the termination of the For loop, $b_{n,1}^{\mathcal{A}_{t+1}} b_{n,2}^{\mathcal{A}_{t+1}} \dots b_{n,k}^{\mathcal{A}_{t+1}} := b_{n,1}^{\mathcal{A}_t} b_{n,2}^{\mathcal{A}_t} \dots b_{n,k}^{\mathcal{A}_t} + 1$. This implies that after the execution of step (6.2), the bit $b_{n,k}^{\mathcal{A}_t}$ is flipped; i.e., $b_{n,k}^{\mathcal{A}_{t+1}} = 1 - b_{n,k}^{\mathcal{A}_t}$.

If Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) terminates at step 5, then clearly $0^n \in L(\mathcal{A}) \iff 0^n \notin L(M_j^{L(N_{i,1}^{\mathcal{L}(\mathcal{A})})})$, and so we successfully finish the stage. Otherwise, Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) terminates at the execution of step (6.3) or it terminates because the For loop had finished iterating over the range of values of t.

If Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) terminates at the execution of step (6.3), then we have the following situation:

- $b_{n,k}^{\mathcal{A}_t} = 0 \iff M_j^{L(N_{i,1}^{\mathcal{L}(\mathcal{A}_t)})}(0^n)$ rejects.
- $b_{n,k}^{\mathcal{A}_{t+1}} = 1 - b_{n,k}^{\mathcal{A}_t}$.
- For every query β_ℓ, where $\ell \in [2^k - 2]$, if $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$ rejects, then $N_{i,1}^{\mathcal{L}(\mathcal{A}_{t+1})}(\beta_\ell)$ also rejects.
- For every query β_ℓ, where $\ell \in [2^k - 2]$, if $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$ accepts, then $N_{i,1}^{\mathcal{L}(\mathcal{A}_{t+1})}(\beta_\ell)$ also accepts, by Claim 7.

It follows that $b_{n,k}^{\mathcal{A}_{t+1}} = 1$ if and only if $M_j^{L(N_{i,1}^{\mathcal{L}(\mathcal{A}_{t+1})})}(0^n)$ rejects. Hence, we successfully finish the stage. We next claim that if Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) does not terminate at step 5, then it must terminate at the execution of step (6.3).

To this end, for each $t \in [2^k]$, let us define $Q_{\mathrm{acc}}(t)$ to be the set of queries β_ℓ on which $N_{i,1}$ with oracle $\mathcal{L}(\mathcal{A}_t)$ accepts. Formally, for any $t \in [2^k]$, let $Q_{\mathrm{acc}}(t) =_{df} \{\beta_\ell \mid \ell \in [2^k-2]$ and $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$ accepts$\}$. By Claim 7, once a query β_ℓ becomes a member of Q_t, the query β_ℓ remains accepted by $N_{i,1}^{\mathcal{L}(\mathcal{A}_{t'})}$ for any $t \leq t' \in [2^k]$. That is, $Q_{\mathrm{acc}}(t) \subseteq Q_{\mathrm{acc}}(t+1)$ for all $t \in [2^k-1]$. By the definition of the sets $Q_{\mathrm{acc}}(t)$, it follows that if the condition in step (6.4) is true at some iteration t of the For loop, then there exist queries $\beta_\ell \in Q_{\mathrm{acc}}(t+1) - Q_{\mathrm{acc}}(t)$; i.e., we have $||Q_{\mathrm{acc}}(t+1)|| > ||Q_{\mathrm{acc}}(t)||$ at these iterations t. Thus, there will be an iteration at which the condition in step (6.4) will not be true. (This follows because the number of iterations, (2^k-1), of the For loop is greater than the maximum possible size, 2^k-2, of $Q_{\mathrm{acc}}(t)$.) Therefore, at that iteration, the condition in step (6.3) will be true. Hence, Diagonalize($\{\widehat{\alpha_f} \mid f \in (\Sigma^*)^{\leq k-1}\}$) will terminate at the execution of step (6.3). This completes the proof of Theorem 4.13. □

Proof of Claim 7. Let f be arbitrary in $(\Sigma^*)^{\leq k-1}$. We will prove that there is a small set $\widetilde{Q}(f)$ such that for each $t \in [2^k-1]$ and for each $\ell \in [2^k-2]$, the following holds for all $\alpha_f \in S_{n,f} - \widetilde{Q}(f)$:

$$\text{if } N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell) \text{ accepts, then } N_{i,1}^{\mathcal{L}(\mathcal{A}_t \cup \{\alpha_f\})}(\beta_\ell) \text{ also accepts.} \tag{4.8}$$

To this end, fix $t \in [2^k-1]$ and $\ell \in [2^k-2]$ and assume that $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$ accepts. Let ρ be an arbitrary accepting path in $N_{i,1}^{\mathcal{L}(\mathcal{A}_t)}(\beta_\ell)$. Because $\mathcal{L}$ is (h,t)-ambiguous, for each $z \in \Sigma^*$ there can be at most $t(|z|)$ strings $\alpha_f \in S_{n,f}$ such that

$$z \in \mathcal{L}(\mathcal{A}_t) \iff z \notin \mathcal{L}(\mathcal{A}_t \cup \{\alpha_f\}).$$

Since there are at most $p(p(n))$ strings queried on ρ, there is a set $\widetilde{Q}(f,t,\ell)$ with $||\widetilde{Q}(f,t,\ell)|| \leq p(p(n)) \cdot t(p(p(n)))$ ensuring that $N_{i,1}^{\mathcal{L}(\mathcal{A}_t \cup \{\alpha_f\})}(\beta_\ell)$ accepts for every $\alpha_f \in S_{n,f} - \widetilde{Q}(f,t,\ell)$.

It is easy to see that statement (4.8) is satisfied with $\widetilde{Q}(f) = \bigcup_{t,\ell} \widetilde{Q}(f,t,\ell)$. Clearly, $||\widetilde{Q}(f)|| \leq (2^k-1)\cdot(2^k-2)\cdot p(p(n))\cdot t(p(p(n))) < 2^n$ by our choice of n. Since $||S_{n,f}|| = 2^n$, there exists a string $\widehat{\alpha_f} \in S_{n,f}$ witnessing the correctness of Claim 7. Thus Claim 7 is proved. □

COROLLARY 4.14. *For any integers $k, h \geq 1$, there exists an oracle $\mathcal{A}$ such that*

$$R_{k\text{-}T}^{p}(\mathrm{UP}_{\leq h}^{\mathcal{A}}) \nsubseteq R_{(2^k-2)\text{-}tt}^{p,\mathcal{A}}(\mathrm{U}\Sigma_h^{p,\mathcal{A}}).$$

4.5. Fault-tolerant access. Ko [36] introduced the notion of *one-sided helping* by a set A in the computation of a set B. A set A is said to provide *one-sided help* to a set B if there is a deterministic oracle Turing machine M computing B and a polynomial $p(\cdot)$ such that (a) on any input $x \in B$, $M^A(x)$ accepts in time $p(|x|)$, and (b) for all inputs y and for all oracles C, $M^C(y)$ accepts (though perhaps $M^C(y)$ may take a longer time than $p(|y|)$) if and only if $y \in B$. Since the machine M, accepting the set B, is capable of answering correctly on faulty oracles, i.e., oracles C different from the oracle A that provides one-sided help to B, the oracle access mechanism is

termed fault-tolerant (see [16]). Ko [36] defined $\mathrm{P}_{\text{1-help}}(A)$ to be the class of all sets B that can be one-sided helped by A.

It is known that sets that can be one-sided helped (by any arbitrary helper) are precisely those in NP [36]. Therefore, the notion of one-sided helping provides an avenue for understanding the structure of NP. For instance, given any class $\mathcal{C} \subseteq \mathrm{NP}$, what class of sets in NP can help the computation of sets in $\mathcal{C}$? Given any class $\mathcal{C}'$ of helpers, what class $\mathcal{C} \subseteq \mathrm{NP}$ can be helped by sets in $\mathcal{C}'$? It is worth studying the relationships between helpers and help-receivers to gain more insights into the notion of one-sided helping.

A restriction of the notion of one-sided helping, called the concept of *helping*, was earlier introduced and studied by Schöning [43]. A set A is said to *help* a set B if B is computed by a deterministic oracle Turing machine M such that on any input $x \in \Sigma^*$, (a) $M^A(x)$ halts in polynomial time, and (b) for all oracles C, $M^C(x)$ accepts (though perhaps $M^C(x)$ may take a longer time than $M^A(x)$ to terminate for $C \neq A$) if and only if $x \in B$.

Definition 4.15 ([43, 36]).

1. *A deterministic oracle Turing machine M is* robust *if for all oracles A, M^A halts on each input and $L(M^A) = L(M^\emptyset)$.*

2. *A set L is in the class $\mathrm{P}_{\text{1-help}}(A)$ if there exist a robust deterministic oracle Turing machine M and a polynomial $p(\cdot)$ such that $L = L(M^\emptyset)$ and for all $x \in L$, $M^A(x)$ halts in $p(|x|)$ steps. If $\mathcal{C}$ is a complexity class, then $\mathrm{P}_{\text{1-help}}(\mathcal{C}) = \bigcup_{A \in \mathcal{C}} \mathrm{P}_{\text{1-help}}(A)$.*

3. *A set L is in the class $\mathrm{P}_{\text{help}}(A)$ if there exist a robust deterministic oracle Turing machine M and a polynomial $p(\cdot)$ such that $L = L(M^\emptyset)$ and for all $x \in \Sigma^*$, $M^A(x)$ halts in $p(|x|)$ steps. If $\mathcal{C}$ is a complexity class, then $\mathrm{P}_{\text{help}}(\mathcal{C}) = \bigcup_{A \in \mathcal{C}} \mathrm{P}_{\text{help}}(A)$.*

The following observation is a direct consequence of Definition 4.15.

Observation 4.16. *For any complexity $\mathcal{C}$, $\mathrm{P}_{\text{help}}(\mathcal{C}) \subseteq \mathrm{P}_{\text{1-help}}(\mathcal{C})$.*

There has been much investigation of the complexity of sets that can be one-sided helped by sets belonging to particular complexity classes. For instance, Ko [36] proved that $\mathrm{NP} = \mathrm{P}_{\text{1-help}}(\mathrm{NP})$, $\mathrm{UP} \subseteq \mathrm{P}_{\text{1-help}}(\mathrm{UP})$, and $\mathrm{P}_{\text{1-help}}(\mathrm{BPP}) \subseteq \mathrm{RP}$. Ko [36] posed the question of whether $\mathrm{P}_{\text{1-help}}(\mathrm{UP})$ is exactly the same as UP. Cai, Hemachandra, and Vyskoč [16] proved that relativizable proof techniques cannot resolve this question: There is a relativized world, where $\mathrm{P}_{\text{1-help}}(\mathrm{UP})$ is not contained in UP [16]. Cintioli and Silvestri [17] strengthened this result. They exhibited an oracle A such that $\mathrm{P}^A_{\text{help}}(\mathrm{UP}^A) \not\subseteq \mathrm{Few}^A$ and an oracle B such that $\mathrm{P}^B_{\text{help}}(\mathrm{UP}^B) \not\subseteq R^{p,B}_b(\mathrm{FewP}(n^{\mathrm{polylog}(n)})^B)$, where $\mathrm{FewP}(n^{\mathrm{polylog}(n)})$ is the class of all sets accepted by NPTMs with at most $n^{\mathrm{polylog}(n)}$ accepting paths on inputs of length n. Despite these negative (oracle) results on the provability of containment of $\mathrm{P}_{\text{help}}(\mathrm{UP})$ in classes such as UP, FewP, and Few, Cai, Hemachandra, and Vyskoč [16] were successful in obtaining an exact characterization of $\mathrm{P}_{\text{1-help}}(\mathrm{UP})$. They proved that $\mathrm{P}_{\text{1-help}}(\mathrm{UP})$ is the closure of UP under $\leq^p_{lpos}$ reductions, where $\leq^p_{lpos}$ is the polynomial-time *locally positive* reduction introduced by Hemachandra and Jain [30].

In Theorem 4.17 below, we generalize and improve the relativized separation of $\mathrm{P}_{\text{1-help}}(\mathrm{UP})$ from UP by Cai, Hemachandra, and Vyskoč [16]. In the case of $h = 1$, Theorem 4.17 gives a relativized separation of $\mathrm{P}_{\text{1-help}}(\mathrm{UP})$ from $R^p_{s,b}(\text{Promise-UP})$, which is a potentially larger class than UP.[4]

[4]There is an oracle relative to which $R^p_{s,b}(\text{Promise-UP}) \not\subseteq \mathrm{UP}$, which follows from Beigel's oracle relative to which $\mathrm{UP}_{\leq 2} \not\subseteq \mathrm{UP}$ [6] and the fact that $\mathrm{UP}_{\leq 2} \subseteq R^p_{s,b}(\text{Promise-UP})$ relative to all oracles.

It remains an open question whether any of the oracle results by Cintioli and Silvestri [17], i.e., existence of oracles A and B such that $\mathrm{P}^A_{\mathrm{help}}(\mathrm{UP}^A) \not\subseteq \mathrm{Few}^A$ and $\mathrm{P}^B_{\mathrm{help}}(\mathrm{UP}^B) \not\subseteq R^{p,B}_b(\mathrm{FewP}(n^{\mathrm{polylog}(n)})^B)$, can be easily extended to our result in Theorem 4.17 for the case of $h = 1$, or vice versa. It also remains an open question whether the oracle results by Cintioli and Silvestri [17] can be generalized so that they hold for $\mathrm{P}_{\mathrm{help}}(\mathrm{UP}_{\leq h})$, for any $h \geq 1$.

THEOREM 4.17. *For every integer $h \geq 1$, there exists an oracle $\mathcal{A}$ such that*

$$\mathrm{P}_{\text{1-help}}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \not\subseteq R^{p,\mathcal{A}}_{s,b}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}}).$$

Proof. The proof relies on Ko's result that for any oracle A, $R^p_{dtt}(\mathrm{UP}^A) \subseteq \mathrm{P}_{\text{1-help}}(\mathrm{UP}^A)$ [36]. Cai, Hemachandra, and Vyskoč [16] used this result to construct an oracle $\mathcal{A}$ such that $\mathrm{P}_{\text{1-help}}(\mathrm{UP}^{\mathcal{A}}) \not\subseteq \mathrm{UP}^{\mathcal{A}}$. We note that Ko's proof extends easily to the case of $\mathrm{UP}_{\leq h}$. That is, for any oracle A, it holds that $R^p_{dtt}(\mathrm{UP}^A_{\leq h}) \subseteq \mathrm{P}_{\text{1-help}}(\mathrm{UP}^A_{\leq h})$.

It thus suffices to show that there is an oracle A such that $R^p_{dtt}(\mathrm{UP}^A_{\leq h})$ is not contained in $R^{p,A}_{s,b}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,A}_{h-1}})$. We claim that we can take the oracle $\mathcal{A}$ from Theorem 4.10 for $k(n) = n$. To see this, suppose that $R^p_{dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \subseteq R^{p,\mathcal{A}}_{s,b}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$. By definition, we know that $R^p_{(n+1)\text{-}dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \subseteq R^p_{dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h})$. Further, we have $R^{p,\mathcal{A}}_{s,b}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}}) \subseteq R^{p,\mathcal{A}}_{s,n\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$ since an n-Turing reduction can make more queries than any bounded Turing reduction except for finitely many n, which can be handled by a look-up table.

The last three inclusions together imply that $R^p_{(n+1)\text{-}dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \subseteq R^{p,\mathcal{A}}_{s,n\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$, contradicting Theorem 4.10. □

5. Robust unambiguity. So far we have looked at several applications of Lemma 3.1 in constructing relativized worlds involving arbitrary levels of the unambiguous polynomial hierarchy. Lemma 3.1, in essence, shows computational limitations of $\Sigma_k(A)$-systems under certain weak restrictions. What happens if we impose a more stringent restriction on a $\Sigma_k(A)$-system? This question is relevant to the following investigation.

We study the power of robustly unambiguous $\Sigma_k(\cdot)$-systems in Theorem 5.1. (Recall from Definition 2.4 that a $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$ is robustly unambiguous if for every oracle B, $[B; N_1, N_2, \ldots, N_k]$ is unambiguous.) Theorem 5.1 illustrates the following fact: A robustly unambiguous $\Sigma_k(\cdot)$-system is so weak that given any oracle set B and input x, the hierarchical nondeterministic polynomial-time oracle access to B in $[B; N_1, N_2, \ldots, N_k](x)$ can be stripped down and turned into a deterministic polynomial-time oracle access (to B) without changing the decision (i.e., acceptance or rejection) of the $\Sigma_k(B)$-system on input x. As a corollary, we obtain a generic oracle collapse of UPH to P assuming P = NP in the unrelativized case. (See, e.g., [10, 21] for generic oracles and concepts related to them.)

THEOREM 5.1. *For every $k \geq 1$, if the $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$ is robustly unambiguous, then there is a deterministic polynomial-time oracle Turing machine M such that for every $B \subseteq \Sigma^*$,*

$$L[B; N_1, N_2, \ldots, N_k] = L(M^{\Sigma^p_k \oplus B}) \in \mathrm{P}^{\Sigma^p_k \oplus B}.$$

Proof. We prove by induction on k the following statement: For each robustly unambiguous $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$, there is a deterministic polynomial-

time oracle Turing machine M such that for every $B \subseteq \Sigma^*$, $L[B; N_1, N_2, \ldots, N_k] = L(M^{\Sigma_k^p \oplus B}) \in \mathrm{P}^{\Sigma_k^p \oplus B}$.

The base case $k = 1$ follows by relativization of [28, Theorem 2.1]. Hence suppose that $k > 1$. For every $B \subseteq \Sigma^*$, let $L(B)$ denote $L[B; N_2, N_3, \ldots, N_k]$. Notice the following easily verifiable facts: For all $B \subseteq \Sigma^*$, we have (a) $L[B; N_1, N_2, \ldots, N_k] = L[L(B); N_1]$, (b) $[L(B); N_1]$ is unambiguous, and (c) $[\cdot; N_2, N_3, \ldots, N_k]$ is robustly unambiguous. Thus, by induction hypothesis, we have that there is a deterministic polynomial-time oracle Turing machine M such that for all $B \subseteq \Sigma^*$, $L(B) = L(M^{\Sigma_{k-1}^p \oplus B}) \in \mathrm{P}^{\Sigma_{k-1}^p \oplus B}$. Let D be any Σ_{k-1}^p-complete set. We can now construct a nondeterministic polynomial-time oracle Turing machine N_1' such that for all $B \subseteq \Sigma^*$

- $L[L(B); N_1] = L[D \oplus B; N_1']$, and
- $[D \oplus B; N_1']$ is unambiguous.

We let machine N_1' on any input x simulate $N_1(x)$, with only one change in the simulation behavior: Whenever $N_1(x)$ makes a query q to $L(B)$, machine N_1' executes $M^{D \oplus B}(q)$ and uses the output returned by this deterministic execution as the answer to q. Thus, N_1' is a nondeterministic polynomial-time oracle Turing machine that accesses oracle $D \oplus B$. Since, for every $B \subseteq \Sigma^*$, $L(B) = L(M^{\Sigma_{k-1}^p \oplus B})$ and $[L(B); N_1]$ is unambiguous, we have that for every $B \subseteq \Sigma^*$, $L[L(B); N_1] = L[D \oplus B; N_1']$ and $[D \oplus B; N_1']$ is unambiguous. The latter means that for every $B \subseteq \Sigma^*$, ${N_1'}^{D \oplus B}$ is unambiguous. By relativization of [28, Theorem 2.1], it follows that there is a deterministic polynomial-time oracle Turing machine $\widehat{M}$ such that for every $B \subseteq \Sigma^*$, $L[D \oplus B; N_1'] = L(\widehat{M}^{\mathrm{NP}^D \oplus B})$. Since $D \in \Sigma_{k-1}^p$, and for every $B \subseteq \Sigma^*$, $L[D \oplus B; N_1'] = L[B; N_1, N_2, \ldots, N_k]$, we also have for every $B \subseteq \Sigma^*$

$$L[B; N_1, N_2, \ldots, N_k] = L(\widehat{M}^{\Sigma_k^p \oplus B}) \in \mathrm{P}^{\Sigma_k^p \oplus B}.$$

This completes the inductive step. □

Theorem 5.1 leads to the following corollary, which can be easily proved using the same techniques found in, e.g., [10, 21].

COROLLARY 5.2. *If* P = NP*, then, relative to a (Cohen) generic* G*,* P = UPH.

This corollary generalizes the following result of Blum and Impagliazzo: If P = NP, then, relative to a (Cohen) generic G, $\mathrm{P}^G = \mathrm{UP}^G$ [10]. Fortnow and Yamakami [22] demonstrated that similar collapses relative to any (Cohen) generic G do not occur at higher levels of the polynomial hierarchy. They proved that for each $k \geq 2$, there exists a tally set in $\mathrm{UP}^{\Sigma_{k-1}^{p,G},G} \cap \Pi_k^{p,G}$ but not in $\mathrm{P}^{\Sigma_{k-1}^{p,G},G}$. Thus Corollary 5.2 contrasts with this generic separation by Fortnow and Yamakami.

6. Conclusion and open problems. We presented a counting technique to investigate the structure of relativized hierarchical unambiguous computation. However, two interesting problems have remained open, for whose resolutions the technique presented in this paper could be useful. These problems are as follows.

1. *Simultaneous immunity and simplicity in the relativized unambiguous polynomial hierarchy.* Complexity class separations can be evaluated in terms of their quality. A separation of a complexity class $\mathcal{C}_1$ from another class $\mathcal{C}_2$ by an *immune* set requires the existence of an infinite set L in $\mathcal{C}_1$ such that no infinite set in $\mathcal{C}_2$ can be a subset of L; the set L is called $\mathcal{C}_2$-*immune* or *immune* to $\mathcal{C}_2$. A separation of a complexity class $\mathcal{C}_1$ from a class $\mathcal{C}_2$ by a *simple* set requires the existence of a coinfinite set (i.e., a set whose complement is infinite) L in $\mathcal{C}_1$ such that L is not in $\mathcal{C}_2$ and $\overline{L}$ is immune to $\mathcal{C}_1$; the set L is called $\mathcal{C}_1$-*simple* or *simple* for $\mathcal{C}_1$. Finally, a separation of a

complexity class $\mathcal{C}_1$ from a class $\mathcal{C}_2$ by a set that is both *simple and immune* requires the existence of a set L in $\mathcal{C}_1$ such that L is $\mathcal{C}_1$-simple and $\mathcal{C}_2$-immune.

An oracle separation of a complexity class from another class by a set that is both simple and immune is considered a much more difficult problem than the oracle separations of the same classes by simple sets or by immune sets alone. This point has been discussed in [13], which we explain in our own words as follows: Intuitively, if a set L is $\mathcal{C}$-immune, then the set L must have low density since no infinite set in $\mathcal{C}$ can be a subset of L. Similarly, if a set L is $\mathcal{C}$-simple, then the set L must have high density since $\overline{L}$ is $\mathcal{C}$-immune. Consequently, separation of a complexity class $\mathcal{C}_1$ from another class $\mathcal{C}_2$ by a set $L \in \mathcal{C}_1$ that is both $\mathcal{C}_1$-simple and $\mathcal{C}_2$-immune requires the set L to have conflicting requirements: L must be dense enough so as to be $\mathcal{C}_1$-simple and must be thin enough so as to be $\mathcal{C}_2$-immune.

Buhrman and Torenvliet [13] showed that, relative to an oracle, the first level and the second level of the polynomial hierarchy separate by sets that are both simple and immune. Using Kolmogorov complexity for oracle constructions, they proved that, relative to an oracle A, NP has a set that is both NP-simple and (NP$\cap$coNP)-immune, and, relative to an oracle B, Π_2^p has a set that is both Π_2^p-simple and $(\Sigma_2^p \cap \Pi_2^p)$-immune. However, it is currently open whether there is an oracle relative to which the third level or any higher level of the polynomial hierarchy separates by a set that is both simple and immune. In fact, it is also open whether there is an oracle relative to which NP has a set that is both NP-simple and coNP-immune.

We expect that the situation for the UPH is quite different from the one for the polynomial hierarchy. We hope that it might be possible that an application of our proof technique in conjunction with the Kolmogorov arguments of Buhrman and Torenvliet [13] lead to a construction of an oracle relative to which all the levels of the UPH separate by sets that are both simple and immune.

2. *Random oracle separation of the relativized unambiguous polynomial hierarchy.* There has been an abundance of complexity theoretic results that hold with probability one relative to a random oracle. Some prominent random oracle results are (1) probability one separation of NP from P with bi-immunity, and of NP from coNP [8], (2) probability one separation of NP from P/poly [39], (3) probability one separation of coNP from NP with immunity [47], and (4) probability one separation of PSPACE from PH [14, 3]. Despite so many random oracle results, the probability one separation of the levels of the polynomial hierarchy relative to a random oracle is still an open problem. (See [32] for an extensive discussion of this problem.) Currently, only the circuit complexity-theoretic approach is known for separating the higher levels (levels beyond three) of the polynomial hierarchy, but the circuit approach has so far not been successful in resolving this longstanding open problem.

We believe that the case of the unambiguous polynomial hierarchy is easier. In Theorem 4.1, we have used our counting technique to show that for all $k \geq 1$, there is an oracle A such that $\mathrm{UP}^A_{\leq k+1}$ is not contained in $\mathrm{U}\Sigma_k^{p,A}$. Thus, unlike the case of the polynomial hierarchy for which only the circuit approach is known for the relativized separation of all its levels, the levels of the relativized UPH are separable by counting arguments alone, and thus by completely avoiding the machinery of circuit complexity. This raises our hope that a probability one separation of the levels of the UPH might be easier to achieve than its counterpart, i.e., a probability one separation of the levels of the PH relative to a random oracle.

Acknowledgments. We thank Lane Hemaspaandra and Jörg Rothe for their constant encouragement and support. We also thank the anonymous referees for their helpful comments.

REFERENCES

[1] S. Aida, M. Crâsmaru, K. Regan, and O. Watanabe, *Games with uniqueness properties*, Theory Comput. Syst., 37 (2004), pp. 29–47.

[2] V. Arvind and P. Kurur, *Graph isomorphism is in SPP*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 743–750.

[3] L. Babai, *Random oracles separate* PSPACE *from the polynomial-time hierarchy*, Inform. Process. Lett., 26 (1987), pp. 51–53.

[4] T. Baker, J. Gill, and R. Solovay, *Relativizations of the* P=?NP *question*, SIAM J. Comput., 4 (1975), pp. 431–442.

[5] T. Baker and A. Selman, *A second step toward the polynomial hierarchy*, Theoret. Comput. Sci., 8 (1979), pp. 177–187.

[6] R. Beigel, *On the relativized power of additional accepting paths*, in Proceedings of the 4th Structure in Complexity Theory Conference, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 216–224.

[7] R. Beigel, *Bounded queries to* SAT *and the boolean hierarchy*, Theoret. Comput. Sci., 84 (1991), pp. 199–223.

[8] C. H. Bennett and J. Gill, *Relative to a random oracle A,* $\mathrm{P}^A \neq \mathrm{NP}^A \neq \text{co-NP}^A$ *with probability* 1, SIAM J. Comput., 10 (1981), pp. 96–113.

[9] C. Berg and S. Ulfberg, *A lower bound for perceptrons and an oracle separation of the* $\mathrm{PP}^{\mathrm{PH}}$ *hierarchy*, J. Comput. System Sci., 56 (1998), pp. 263–271.

[10] M. Blum and R. Impagliazzo, *Generic oracles and oracle classes*, in Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 118–126.

[11] D. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice–Hall, New York, 1994.

[12] H. Buhrman, E. Spaan, and L. Torenvliet, *Bounded reductions*, in Complexity Theory, K. Ambos-Spies, S. Homer, and U. Schöning, eds., Cambridge University Press, Cambridge, UK, 1993, pp. 83–99.

[13] H. Buhrman and L. Torenvliet, *Complicated complementations*, in Proceedings of the 14th Annual IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 227–236.

[14] J. Cai, *With probability one, a random oracle separates* PSPACE *from the polynomial-time hierarchy*, J. Comput. System Sci., 38 (1989), pp. 68–85.

[15] J. Cai, L. Hemachandra, and J. Vyskoč, *Promise problems and access to unambiguous computation*, in Proceedings of the 17th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 629, Springer-Verlag, New York, 1992, pp. 162–171.

[16] J. Cai, L. Hemachandra, and J. Vyskoč, *Promises and fault-tolerant database access*, in Complexity Theory, K. Ambos-Spies, S. Homer, and U. Schöning, eds., Cambridge University Press, Cambridge, UK, 1993, pp. 101–146.

[17] P. Cintioli and R. Silvestri, *Helping by unambiguous computation and probabilistic computation*, Theory Comput. Systems, 30 (1997), pp. 165–180.

[18] M. Crâsmaru, C. Glasser, K. Regan, and S. Sengupta, *A protocol for serializing unique strategies*, in Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 3153, Springer-Verlag, New York, 2004, pp. 660–672.

[19] S. Even, A. Selman, and Y. Yacobi, *The complexity of promise problems with applications to public-key cryptography*, Inform. and Control, 61 (1984), pp. 159–173.

[20] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, J. Comput. System Sci., 48 (1994), pp. 116–148.

[21] S. Fenner, L. Fortnow, S. Kurtz, and L. Li, *An oracle builder's toolkit*, Inform. and Comput., 182 (2003), pp. 95–136.

[22] L. Fortnow and T. Yamakami, *Generic separations*, J. Comput. System Sci., 52 (1996), pp. 191–197.

[23] M. Furst, J. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[24] C. Glasser and S. Travers, *Machines That Can Output Empty Words*, Technical report TR05–147, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[25] O. Goldreich, *On Promise Problems*, Technical report TR05–018, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[26] J. Grollmann and A. L. Selman, *Complexity measures for public-key cryptosystems*, SIAM J. Comput., 17 (1988), pp. 309–335.
[27] S. Gupta, *On the closure of certain function classes under integer division by polynomially bounded functions*, Inform. Process. Lett., 44 (1992), pp. 205–210.
[28] J. Hartmanis and L. Hemachandra, *Robust machines accept easy sets*, Theoret. Comput. Sci., 74 (1990), pp. 217–225.
[29] J. Håstad, *Computational Limitations of Small-Depth Circuits*, MIT Press, Cambridge, MA, 1987.
[30] L. Hemachandra and S. Jain, *On the limitations of locally robust positive reductions*, Internat. J. Found. Comput. Sci., 2 (1991), pp. 237–255.
[31] L. Hemaspaandra and M. Ogihara, *The Complexity Theory Companion*, Springer-Verlag, New York, 2002.
[32] L. Hemaspaandra, A. Ramachandran, and M. Zimand, *Worlds to die for*, SIGACT News, 26 (1995), pp. 5–15.
[33] L. A. Hemaspaandra and J. Rothe, *Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets*, SIAM J. Comput., 26 (1997), pp. 634–653.
[34] D. Joseph and P. Young, *Some remarks on witness functions for non-polynomial and non-complete sets in* NP, Theoret. Comput. Sci., 39 (1985), pp. 225–237.
[35] K. Ko, *On some natural complete operators*, Theoret. Comput. Sci., 37 (1985), pp. 1–30.
[36] K. Ko, *On helping by robust oracle machines*, Theoret. Comput. Sci., 52 (1987), pp. 15–36.
[37] K. Ko, *Relativized polynomial time hierarchies having exactly k levels*, SIAM J. Comput., 18 (1989), pp. 392–408.
[38] K.-J. Lange and P. Rossmanith, *Unambiguous polynomial hierarchies and exponential size*, in Proceedings of the 9th Structure in Complexity Theory Conference, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 106–115.
[39] J. Lutz and W. Schmidt, *Circuit size relative to pseudorandom oracles*, Theoret. Comput. Sci., 107 (1993), pp. 95–120.
[40] R. Niedermeier and P. Rossmanith, *Unambiguous computations and locally definable acceptance types*, Theoret. Comput. Sci., 194 (1998), pp. 137–161.
[41] M. Ogiwara and L. Hemachandra, *A complexity theory for feasible closure properties*, J. Comput. System Sci., 46 (1993), pp. 295–325.
[42] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[43] U. Schöning, *Robust algorithms: A different approach to oracles*, Theoret. Comput. Sci., 40 (1985), pp. 57–66.
[44] M. Sheu and T. Long, UP *and the low and high hierarchies: A relativized separation*, Math. Systems Theory, 29 (1996), pp. 423–449.
[45] M. Sipser, *Borel sets and circuit complexity*, in Proceedings of the 15th ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 61–69.
[46] H. Spakowski and R. Tripathi, *On the power of unambiguity in alternating machines*, Theory Comput. Syst., 41 (2007), pp. 291–326.
[47] N. Vereshchagin, *Relationships between* NP*-sets,* co-NP*-sets, and* P*-sets relative to random oracles*, in Proceedings of the 8th Structure in Complexity Theory Conference, IEEE Computer Society, Los Alamitos, CA, 1993, pp. 132–138.
[48] K. W. Wagner, *Bounded query classes*, SIAM J. Comput., 19 (1990), pp. 833–846.
[49] A. Yao, *Separating the polynomial-time hierarchy by oracles*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1985, pp. 1–10.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2113–2129

SEPARATING AC^0 FROM DEPTH-2 MAJORITY CIRCUITS*

ALEXANDER A. SHERSTOV†

Abstract. We construct a function in AC^0 that cannot be computed by a depth-2 majority circuit of size less than $\exp(\Theta(n^{1/5}))$. This solves an open problem due to Krause and Pudlák [*Theoret. Comput. Sci.*, 174 (1997), pp. 137–156] and matches Allender's classic result [A note on the power of threshold circuits, in *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Research Triangle Park, NC, 1989, pp. 580–584] that AC^0 can be efficiently simulated by depth-3 majority circuits. To obtain our result, we develop a novel technique for proving lower bounds on communication complexity. This technique, the *Degree/Discrepancy Theorem*, is of independent interest. It translates lower bounds on the threshold degree of any Boolean function into upper bounds on the discrepancy of a related function. Upper bounds on the discrepancy, in turn, immediately imply lower bounds on communication and circuit size. In particular, we exhibit the first known function in AC^0 with exponentially small discrepancy, $\exp(-\Omega(n^{1/5}))$, thereby establishing the separations $\Sigma_2^{cc} \not\subseteq \mathsf{PP}^{cc}$ and $\Pi_2^{cc} \not\subseteq \mathsf{PP}^{cc}$ in communication complexity.

Key words. majority circuits, constant-depth AND/OR/NOT circuits, communication complexity, discrepancy, threshold degree of Boolean functions, Degree/Discrepancy Theorem

AMS subject classifications. 03D15, 68Q15, 68Q17

DOI. 10.1137/08071421X

1. Introduction.

A natural and important computational model is that of a polynomial-size circuit of majority gates. This model has been extensively studied for the past two decades [14, 15, 26, 27, 38, 43, 44, 45]. Research has shown that majority circuits of depth 2 and 3 already possess surprising computational power. Indeed, it is a long-standing open problem [21] to exhibit a Boolean function that *cannot* be computed by a depth-3 majority circuit of polynomial size. To illustrate, the arithmetic operations of powering, multiplication, and division on n-bit integer arguments can all be computed by depth-3 majority circuits of polynomial size [45]. An even more striking example is the addition of n n-bit integers, which is computable by a depth-2 majority circuit of polynomial size [45]. Depth-2 majority circuits of polynomial size can also compute every symmetric function (such as PARITY) and every disjunctive normal form (DNF) and conjunctive normal form (CNF) formula of polynomial size.

The chief goal of this paper is to relate the computational power of majority circuits to that of AC^0, another extensively studied class, which consists of polynomial-size constant-depth circuits of AND, OR, NOT gates. A well-known result due to Allender [1] states that every function in AC^0 can be computed by a depth-3 majority circuit of quasi-polynomial size. For over ten years, it has been an open problem to determine whether Allender's simulation is optimal. Specifically, Krause and Pudlák [21, sect. 6] ask whether every function in AC^0 can be computed by a depth-2 majority circuit of quasi-polynomial size. We solve this open problem completely, even in the

*Received by the editors January 25, 2008; accepted for publication (in revised form) September 16, 2008; published electronically February 11, 2009. This paper appeared in preliminary form in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing* (STOC), 2007, pp. 294–301.

http://www.siam.org/journals/sicomp/38-6/71421.html

†Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712 (sherstov@cs.utexas.edu).

more general setting of majority-of-threshold circuits (i.e., depth-2 circuits in which a majority gate receives inputs from arbitrary linear threshold gates).

THEOREM 1.1 (main result). *There is a function $f : \{-1,1\}^n \to \{-1,1\}$, explicitly given and computable by an AC^0 circuit of depth 3, whose computation requires a majority vote of $\exp(\Omega(n^{1/5}))$ linear threshold gates.*

In other words, Allender's simulation is optimal in a strong sense. The lower bound in Theorem 1.1 is an exponential improvement over previous work. The best previous lower bound [15, 27] was quasi polynomial and followed trivially from the observation that AC^0 can compute INNER PRODUCT MODULO 2 on $\log^c n$ variables, for any constant $c > 1$.

1.1. A communication-complexity perspective. A different and perhaps more revealing view of this work is in terms of communication complexity [23]. The communication complexity of Boolean functions in different models has long been an active area of research, due to its inherent appeal as a complexity subject as well as its numerous applications in theoretical computer science. Our work contributes a novel and powerful technique for communication lower bounds, which is based on the representation of a Boolean function as the sign of a real-valued polynomial. Specifically, fix a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$. Its *threshold degree*, $\deg_\pm(f)$, is defined as the minimum degree of a polynomial $p(x_1, x_2, \ldots, x_n)$ that represents f in sign: $f(x) \equiv \operatorname{sign}(p(x))$. This concept has played a prominent role in the study of circuit complexity [2, 21, 22, 26] and has yielded valuable insights in other areas, including computational learning theory [17, 20]. In many cases [26], it is straightforward to obtain strong lower bounds on the threshold degree. Since the threshold degree is a measure of the complexity of a given Boolean function, it is natural to wonder whether it can yield lower bounds on communication in a suitable setting. Our work confirms this intuition for every f.

More precisely, fix a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ with threshold degree d. Let N be a given integer, $N \geqslant n$. We introduce and study the two-party communication problem of computing,

$$f(x|_S),$$

where the Boolean string $x \in \{-1,1\}^N$ is Alice's input and the set $S \subset \{1,2,\ldots,N\}$ of size $|S| = n$ is Bob's input. The symbol $x|_S$ stands for the projection of x onto the indices in S, in other words, $x|_S = (x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \in \{-1,1\}^n$, where $i_1 < i_2 < \cdots < i_n$ are the elements of S. Intuitively, this problem models a situation when Alice and Bob's joint computation depends on only n of the inputs $x_1, x_2, \ldots, x_N$. Alice knows the values of all the inputs $x_1, x_2, \ldots, x_N$ but does not know which n of them are relevant. Bob, on the other hand, knows which n inputs are relevant but does not know their values. As one would hope, we prove that d gives a lower bound on the communication requirements of this problem. We phrase our result in terms of *discrepancy*, a central quantity in communication complexity that immediately yields lower bounds on communication in a variety of models (see section 2.1).

THEOREM 1.2 (Degree/Discrepancy Theorem). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be given with threshold degree $d \geqslant 1$. Then for $N \geqslant n$, the matrix $M = [f(x|_S)]_{x,S}$ has discrepancy*

$$\operatorname{disc}(M) \leqslant \left(\frac{4\mathrm{e}n^2}{Nd}\right)^{d/2},$$

where $\mathrm{e} = 2.718\ldots$.

Theorem 1.2, which we call the *Degree/Discrepancy Theorem* for obvious reasons, is a separate contribution of our work. Given a function f with threshold degree d, it generates a communication problem with discrepancy at most 2^{-d} (by setting $N \geqslant 16en^2/d$). This exponentially small upper bound on the discrepancy immediately translates into an $\Omega(d)$ lower bound on communication in a variety of models (deterministic, nondeterministic, randomized, quantum with and without entanglement). Moreover, the resulting lower bounds on communication remain valid when Alice and Bob merely seek to predict the answer with nonnegligible advantage, a critical aspect for lower bounds against threshold circuits. This contrasts with other communication-complexity methods [32, 34], which only apply to high-accuracy computation. Finally, discrepancy arises prominently in contexts beyond communication complexity, such as estimation of margin complexity [25, 40] and approximate rank [19]. For all these reasons, we believe that the Degree/Discrepancy Theorem is of considerable interest in its own right. We prove it by a novel application of Gordan's transposition theorem [37, sect. 7.8], which is a classical result from the theory of linear inequalities.

We are now in a position to outline the proof of our main result, Theorem 1.1. We start with a well-known DNF formula Φ, constructed by Minsky and Papert [26], that has high threshold degree. An application of Theorem 1.2 to Φ yields a communication problem with low discrepancy. By design, this communication problem can be viewed as an AC^0 circuit of depth 3. Recalling that its discrepancy is exponentially small, we immediately conclude that it cannot be computed by a depth-2 majority circuit of subexponential size. This completes the proof.

1.2. On the discrepancy of AC^0 circuits. Recall that a key component of our proof is the construction of an AC^0 circuit with exponentially small discrepancy. Prior to this work, it was not known whether such a circuit existed. In particular, all previously known functions with exponentially small discrepancy (see, e.g., [14, 27]) contain PARITY or MAJORITY as a subfunction and therefore cannot be computed in AC^0. In view of the intrinsic value of discrepancy as a complexity measure, we state this result on its own.

THEOREM 1.3 (discrepancy of AC^0 circuits). *There is a function $f : \{-1,1\}^n \times \{-1,1\}^n \to \{-1,1\}$, explicitly given and computable by an AC^0 circuit of depth 3, that has discrepancy $\exp(-\Omega(n^{1/5}))$ with respect to an explicitly given distribution.*

Theorem 1.3 is best possible in that every AC^0 circuit of depth 1 or 2 has discrepancy at least $n^{-O(1)}$ with respect to all distributions and all partitions of the variables (see section 5). As a direct corollary to Theorem 1.3, we separate communication classes Σ_2^{cc} and Π_2^{cc} from PP^{cc}.

COROLLARY 1.4. $\Sigma_2^{cc} \not\subseteq \mathsf{PP}^{cc}$, $\Pi_2^{cc} \not\subseteq \mathsf{PP}^{cc}$.

Here Σ_2^{cc} and Π_2^{cc} are the second level of the polynomial hierarchy in communication complexity, whereas PP^{cc} is the class of all communication matrices with nonnegligible discrepancy. Prior to this work, it was entirely conceivable that PP^{cc} contained both Σ_2^{cc} and Π_2^{cc} and the rest of the polynomial hierarchy PH^{cc}. See section 6 for further details.

Another AC^0 circuit of depth 3 with exponentially small discrepancy was constructed independently by Buhrman, Vereshchagin, and de Wolf [5, sect. 3]. Their proof uses quite different techniques (approximation theory and quantum communication complexity). Their circuit has discrepancy $\exp(-\Omega(n^{1/3}))$, which is a stronger bound than Theorem 1.3. An advantage of the construction in this paper is that it is self-contained and derived from first principles, whereas the work of Buhrman, Vereshchagin, and de Wolf builds on a subtle result of Razborov [35]. In addition,

our method is not restricted to AC^0 but, rather, applies to *any* function with high threshold degree.

1.3. Recent progress. We are pleased to report that the Degree/Discrepancy Theorem has inspired important progress in communication complexity by several researchers, which we briefly survey. The first of these new results is concerned with bounded-error communication. By combining the method of Theorem 1.2 with techniques from matrix analysis and approximation theory, the author [41] obtained strong lower bounds on bounded-error communication for a broad new class of problems. These lower bounds remain valid in the quantum model (regardless of prior entanglement) and subsume Razborov's breakthrough lower bounds for symmetric functions [35].

In another development [42], we used the method of Theorem 1.2 as a starting point to derive essentially optimal lower bounds on the *unbounded-error* communication complexity of every symmetric function. The unbounded-error model is more powerful than all the familiar models of communication (both classical and quantum), and proving lower bounds in it is a substantial challenge. The only previous nontrivial lower bounds for this model appeared in the groundbreaking work of Forster [12] and its extensions.

Razborov and Sherstov [36] recently obtained the first exponential lower bound on the *sign-rank* of AC^0, thereby solving a long-standing open problem due to Babai, Frankl, and Simon [3]. The results in [36] additionally give strong lower bounds for the probably approximately correct (PAC) learning of polynomial-size DNF and CNF formulas (see Remark 8.2). The method of the Degree/Discrepancy Theorem is one of the starting points in that work.

The Degree/Discrepancy Theorem has also found applications to multiparty communication complexity. The first of these is the work by Chattopadhyay [6], who observed that the method of Theorem 1.2 adapts in a straightforward manner to the multiparty model. Analogous to this adaptation, Lee and Shraibman [24] and Chattopadhyay and Ada [7] adapted to the multiparty model the author's recent work [41] on two-party bounded-error communication. They thereby obtained improved lower bounds on the bounded-error communication complexity of DISJOINTNESS for up to $\log\log n$ players. David and Pitassi [9] combined this line of work with the probabilistic method, establishing a separation of the communication classes NP^{cc}_k and BPP^{cc}_k for up to $k = (1-\epsilon)\log n$ players. Their construction was derandomized in a follow-up paper by David, Pitassi, and Viola [10], resulting in an explicit separation. See the survey article [39] for a unified guide to these results, complete with all the key proofs.

1.4. Organization. The remainder of this paper is organized as follows. Section 2 provides relevant background on communication complexity and threshold functions. Section 3 is devoted to the proof of the Degree/Discrepancy Theorem, our main technical tool. Section 4 studies a particular AC^0 circuit Φ with high threshold degree. Section 5 applies the Degree/Discrepancy Theorem to Φ, yielding an explicit AC^0 circuit f of depth 3 with exponentially small discrepancy. Section 6 uses this discrepancy result to separate the classes Σ_2^{cc} and Π_2^{cc} from PP^{cc} in communication complexity. In section 7, we derive our main result, an exponential lower bound on the size of depth-2 majority circuits that compute AC^0. Section 8 concludes with an application of our results to computational learning theory.

2. Preliminaries. Throughout this work, we identify -1 and 1 with "true" and "false," respectively. We view Boolean functions as mappings $X \to \{-1, 1\}$, where X

is a finite set such as $X = \{-1,1\}^n$. The symbol $[n]$ stands for the set $\{1,2,\ldots,n\}$. For integers N, n with $N \geqslant n$, the symbol $\binom{[N]}{n}$ denotes the family of all size-n subsets of $\{1,2,\ldots,N\}$. For a string $x \in \{-1,1\}^N$ and a set $S \in \binom{[N]}{n}$, we define $x|_S = (x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \in \{-1,1\}^n$, where $i_1 < i_2 < \cdots < i_n$ are the elements of S. The notation $\mathbb{R}^{m\times n}$ refers to the family of all $m \times n$ matrices with real entries. We specify matrices by their generic entry, e.g., $A = [F(i,j)]_{i,j}$. As usual, we denote the base of the natural logarithm by $\mathrm{e} = 2.718\ldots$.

Recall that AC^0 is the family of polynomial-size unbounded-fanin constant-depth circuits with AND, OR, NOT gates. We adopt the following standard definition of the sign function:

$$\operatorname{sign}(t) = \begin{cases} 1 & \text{if } t > 0, \\ 0 & \text{if } t = 0, \\ -1 & \text{if } t < 0. \end{cases}$$

A *linear threshold gate* is a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ of the form $f(x) \equiv \operatorname{sign}(\sum_{i=1}^n a_i x_i - \theta)$ for some fixed reals $a_1, \ldots, a_n, \theta$. Observe that a linear threshold gate generalizes the familiar majority gate.

2.1. Communication complexity. We consider Boolean functions $f : X \times Y \to \{-1,1\}$. Typically $X = Y = \{-1,1\}^n$, but we also allow X and Y to be arbitrary finite sets. We identify a function f with its *communication matrix* $M = [f(x,y)]_{x\in X,\, y\in Y}$. In particular, we use the terms "communication complexity of f" and "communication complexity of M" interchangeably (and likewise for other complexity measures, such as discrepancy). The two communication models of interest to us are the randomized model and the deterministic model. The *randomized complexity* $\mathrm{R}_{1/2-\gamma/2}(f)$ of f is the minimum cost of a randomized protocol for f that computes $f(x,y)$ correctly with probability at least $\frac{1}{2} + \frac{1}{2}\gamma$ (equivalently, with *advantage* γ) on every input (x,y). The *public-coin randomized complexity* $\mathrm{R}^{\mathrm{pub}}_{1/2-\gamma/2}(f)$ is defined analogously, with the only difference that the communicating parties now have a source of shared random bits; i.e., they can observe tosses of a common coin without communicating. The *distributional complexity* $\mathrm{D}^{\mu}_{1/2-\gamma/2}(f)$ is the minimum cost of a deterministic protocol for f that has error at most $\frac{1}{2} - \frac{1}{2}\gamma$ (equivalently, *advantage* γ) with respect to the distribution μ over the inputs.

A *rectangle* of $X \times Y$ is any set $R = A \times B$ with $A \subseteq X$ and $B \subseteq Y$. For a fixed distribution μ over $X \times Y$, the *discrepancy* of f is defined as

$$\operatorname{disc}_\mu(f) = \max_R \left| \sum_{(x,y)\in R} \mu(x,y) f(x,y) \right|,$$

where the maximum is taken over all rectangles R. We define $\operatorname{disc}(f) = \min_\mu \operatorname{disc}_\mu(f)$. The *discrepancy method* is a fundamental technique that places a lower bound on the randomized and distributional complexity in terms of the discrepancy.

PROPOSITION 2.1 (Kushilevitz and Nisan [23, pp. 36–38]). *For every Boolean function $f : X \times Y \to \{-1,1\}$, every distribution μ on $X \times Y$, and every $\gamma > 0$,*

$$\mathrm{R}_{1/2-\gamma/2}(f) \;\geqslant\; \mathrm{R}^{\mathrm{pub}}_{1/2-\gamma/2}(f) \;\geqslant\; \mathrm{D}^{\mu}_{1/2-\gamma/2}(f) \;\geqslant\; \log_2 \frac{\gamma}{\operatorname{disc}_\mu(f)}.$$

The above definition of discrepancy is not convenient to work with. The following well-known lemma bounds the discrepancy in terms of a more analytically pleasing quantity. For completeness, we include a proof.

LEMMA 2.2 (discrepancy bound; cf. [4, 8, 11, 33]). *Let $f : X \times Y \to \{-1,1\}$ be given, and let μ be a probability distribution over $X \times Y$. Then*

$$\operatorname{disc}_\mu(f)^2 \leqslant |X| \sum_{y,y' \in Y} \left| \sum_{x \in X} \mu(x,y)\mu(x,y')f(x,y)f(x,y') \right|.$$

Proof (adapted from Raz [33]). Let $R = A \times B$ be a rectangle over which the discrepancy is achieved. Define $\alpha_x = 1$ for all $x \in A$, and likewise $\beta_y = 1$ for all $y \in B$. For all remaining x and y, let α_x and β_y be independent random variables distributed uniformly over $\{-1,1\}$. Passing to expectations,

$$\begin{aligned}
&\left| \mathbf{E}\left[\sum_{x,y} \alpha_x \beta_y \mu(x,y) f(x,y) \right] \right| \\
&\qquad = \left| \sum_{(x,y) \in R} \underbrace{\mathbf{E}[\alpha_x \beta_y]}_{=1} \mu(x,y) f(x,y) + \sum_{(x,y) \notin R} \underbrace{\mathbf{E}[\alpha_x \beta_y]}_{=0} \mu(x,y) f(x,y) \right| \\
&\qquad = \operatorname{disc}_\mu(M).
\end{aligned}$$

In particular, there exists a fixed assignment $\alpha_x, \beta_y \in \{-1,1\}$ for all x, y such that

$$\operatorname{disc}_\mu(f) \leqslant \left| \sum_{x,y} \alpha_x \beta_y \mu(x,y) f(x,y) \right|.$$

Squaring both sides and applying the Cauchy–Schwarz inequality,

$$\begin{aligned}
\operatorname{disc}_\mu(f)^2 &\leqslant |X| \sum_x \left(\alpha_x \sum_y \beta_y \mu(x,y) f(x,y) \right)^2 \\
&= |X| \sum_{y,y'} \beta_y \beta_{y'} \sum_x \mu(x,y)\mu(x,y') f(x,y) f(x,y') \\
&\leqslant |X| \sum_{y,y'} \left| \sum_x \mu(x,y)\mu(x,y') f(x,y) f(x,y') \right|,
\end{aligned}$$

as desired. □

A definitive resource for further details is the book of Kushilevitz and Nisan [23].

2.2. Threshold degree. Let $f : \{-1,1\}^n \to \{-1,1\}$ be a given Boolean function. The *threshold degree* of f, denoted $\deg_\pm(f)$, is the least degree of a polynomial $p(x_1, x_2, \ldots, x_n)$ such that $f(x) \equiv \operatorname{sign}(p(x))$. In view of the domain of f, any such polynomial p can be assumed to be multilinear. Note that any function f that depends on k or fewer of the n variables has threshold degree at most k. For a set $S \subseteq [n]$, we write $\chi_S = \prod_{i \in S} x_i$. In this notation, the threshold degree of f is the smallest d such that $f(x) \equiv \operatorname{sign}(\sum_{|S| \leqslant d} a_S \chi_S(x))$ for some fixed reals a_S. Threshold degree is also known in the literature as "strong degree" [2], "voting polynomial degree" [21], "polynomial threshold function (PTF) degree" [30], and "sign degree" [5].

Crucial to understanding the threshold degree is the following result from the theory of linear inequalities, which follows in a straightforward manner from linear-programming duality.

THEOREM 2.3 (Gordan's transposition theorem [37, sect. 7.8]). *Let $A \in \mathbb{R}^{m\times n}$. Then exactly one of the following statements holds:*

(i) *$u^{\mathsf{T}}A > 0$ for some vector u.*

(ii) *$Av = 0$ for some nonzero vector $v \geqslant 0$.*

The vector notation $u^{\mathsf{T}}A > 0$ and $v \geqslant 0$ above is to be understood entrywise, as usual. A consequence of Gordan's transposition theorem is the following well-known result regarding threshold representations.

THEOREM 2.4 (see [2, 30]). *Let $\phi_1, \phi_2, \ldots, \phi_k : \{-1,1\}^n \to \mathbb{R}$ be arbitrary real functions, and let $f : \{-1,1\}^n \to \{-1,1\}$ be a given Boolean function. Then exactly one of the following statements holds:*

(i) *$f(x) \equiv \operatorname{sign}(\sum_{i=1}^k a_i\phi_i(x))$ for some reals $a_1, a_2, \ldots, a_k$.*

(ii) *There is a distribution μ over $\{-1,1\}^n$ such that*

$$\mathop{\mathbf{E}}_{x\sim\mu}[f(x)\phi_i(x)] = 0, \qquad i = 1, 2, \ldots, k.$$

Proof. Consider the $k \times 2^n$ matrix $A = [f(x)\phi_i(x)]_{i,x}$. The claim follows from Theorem 2.3, with u playing the role of a set of coefficients $(a_1, a_2, \ldots, a_k) \in \mathbb{R}^k$, and v playing the role of a probability distribution. □

COROLLARY 2.5. *Let $f : \{-1,1\}^n \to \{-1,1\}$ be arbitrary and d be a nonnegative integer. Then exactly one of the following holds:*

(i) *$\deg_{\pm}(f) \leqslant d$.*

(ii) *There is a distribution μ over $\{-1,1\}^n$ such that*

$$\mathop{\mathbf{E}}_{x\sim\mu}[f(x)\chi_S(x)] = 0, \qquad |S| = 0, 1, \ldots, d.$$

3. The Degree/Discrepancy Theorem. This section marks the beginning of our proof. Its purpose is to establish Theorem 1.2 (the Degree/Discrepancy Theorem), which plays a central role in the development to follow.

THEOREM 1.2 (restated from section 1.1). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be given with threshold degree $d \geqslant 1$. Let N be a given integer, $N \geqslant n$. Define $M = [f(x|_S)]_{x,S}$, where the indices range as follows: $x \in \{-1,1\}^N$, $S \in \binom{[N]}{n}$. Then*

$$\operatorname{disc}(M) \leqslant \left(\frac{4en^2}{Nd}\right)^{d/2}. \tag{3.1}$$

Proof. Let μ be a probability distribution over $\{-1,1\}^n$ with respect to which $\mathbf{E}_{z\sim\mu}[f(z)p(z)] = 0$ for every real-valued function p of $d-1$ or fewer of the variables $z_1, \ldots, z_n$. The existence of μ is assured by Corollary 2.5. Throughout this proof, the symbol $\mathcal{U}$ shall stand for the uniform distribution over the relevant domain. We will analyze the discrepancy of M with respect to the distribution

$$\lambda(x, S) = 2^{-N+n}\binom{N}{n}^{-1}\mu(x|_S).$$

By Lemma 2.2,

$$\operatorname{disc}_\lambda(M)^2 \leqslant 4^n \mathop{\mathbf{E}}_{(S,T)\sim\mathcal{U}}|\Gamma(S,T)|, \tag{3.2}$$

where we let

$$\Gamma(S,T) = \mathop{\mathbf{E}}_{x\sim\mathcal{U}}\left[\mu(x|_S)\mu(x|_T)f(x|_S)f(x|_T)\right].$$

To analyze this expression, we prove two key claims.

CLAIM 3.1. *Assume that* $|S \cap T| \leqslant d-1$. *Then* $\Gamma(S,T) = 0$.

Proof. For notational convenience, assume that $S = \{1,2,\ldots,n\}$. Then

$$\begin{aligned}\Gamma(S,T) &= \mathop{\mathbf{E}}_{x\sim\mathcal{U}}\Big[\mu(x_1,\ldots,x_n)\mu(x|_T)f(x_1,\ldots,x_n)f(x|_T)\Big] \\ &= \frac{1}{2^N}\sum_{x_1,\ldots,x_n}\mu(x_1,\ldots,x_n)f(x_1,\ldots,x_n)\sum_{x_{n+1},\ldots,x_N}\mu(x|_T)f(x|_T) \\ &= \frac{1}{2^N}\mathop{\mathbf{E}}_{(x_1,\ldots,x_n)\sim\mu}\left[f(x_1,\ldots,x_n)\cdot\underbrace{\left(\sum_{x_{n+1},\ldots,x_N}\mu(x|_T)f(x|_T)\right)}_{*}\right].\end{aligned}$$

Since $|S\cap T| \leqslant d-1$, the starred expression is a real-valued function of at most $d-1$ variables. The claim follows by the definition of μ. □

CLAIM 3.2. *Assume that* $|S \cap T| = k$. *Then* $|\Gamma(S,T)| \leqslant 2^{k-2n}$.

Proof. For notational convenience, assume that

$$\begin{aligned}S &= \{1,2,\ldots,n\}, \\ T &= \{1,2,\ldots,k\} \cup \{n+1, n+2, \ldots, n+(n-k)\}.\end{aligned}$$

We have

$$\begin{aligned}|\Gamma(S,T)| &\leqslant \mathop{\mathbf{E}}_{x\sim\mathcal{U}}\Big|\mu(x|_S)\mu(x|_T)f(x|_S)f(x|_T)\Big| \\ &= \mathop{\mathbf{E}}_{x_1,\ldots,x_{2n-k}}[\mu(x_1,\ldots,x_n)\mu(x_1,\ldots,x_k,x_{n+1},\ldots,x_{2n-k})] \\ &\leqslant \underbrace{\mathop{\mathbf{E}}_{x_1,\ldots,x_n}[\mu(x_1,\ldots,x_n)]}_{=2^{-n}}\cdot\max_{x_1,\ldots,x_k}\underbrace{\mathop{\mathbf{E}}_{x_{n+1},\ldots,x_{2n-k}}[\mu(x_1,\ldots,x_k,x_{n+1},\ldots,x_{2n-k})]}_{\leqslant 2^{-(n-k)}}.\end{aligned}$$

The bounds 2^{-n} and $2^{-(n-k)}$ follow because μ is a probability distribution. □

In view of Claims 3.1 and 3.2, inequality (3.2) simplifies to

$$\operatorname{disc}_\lambda(M)^2 \leqslant \sum_{k=d}^{n} 2^k\,\mathbf{P}[|S\cap T| = k].$$

Since

$$\mathbf{P}[|S\cap T| = k] = \binom{n}{k}\binom{N-n}{n-k}\binom{N}{n}^{-1} \leqslant \binom{n}{k}\left(\frac{n}{N}\right)^k \leqslant \left(\frac{\mathrm{e}n^2}{Nk}\right)^k,$$

and since the discrepancy cannot exceed 1, we conclude that

$$\operatorname{disc}_\lambda(M)^2 \leqslant \min\left\{1,\ \sum_{k=d}^{n}\left(\frac{2\mathrm{e}n^2}{Nk}\right)^k\right\} \leqslant \left(\frac{4\mathrm{e}n^2}{Nd}\right)^d.$$

This completes the proof of Theorem 1.2. □

Remark 3.3. The proof above analyzes the discrepancy of $M = [f(x|_S)]_{x,S}$ with respect to a certain distribution λ, derived directly from a distribution μ under which

f is uncorrelated with every function of fewer than $\deg_\pm(f)$ variables. Therefore, the discrepancy bound (3.1) is achieved with respect to an explicitly given distribution whenever μ is explicitly given.

Remark 3.4. The discrepancy bound (3.1) holds not only for M but also for any sign matrix that contains M. This observation is immediate from the definition of discrepancy. It allows a considerable degree of flexibility in applying Theorem 1.2, as will become apparent in section 5.

Remark 3.5. The discrepancy bound in Theorem 1.2 is not tight. In subsequent work [41, Thm. 7.3], the author strengthened it to $\mathrm{disc}(M) \leqslant (4n/N)^{d/2}$. Moreover, this new bound continues to hold when Bob's input S is restricted to have a particularly simple form. This stronger Degree/Discrepancy Theorem leads to quantitative improvements on this paper's Theorems 1.1 and 1.3; see [41, sect. 7] for details. These improvements are built around a matrix-analytic approach to estimating the discrepancy, as opposed to the combinatorial derivation above. However, the proof of the Degree/Discrepancy Theorem in this paper has the advantage that it easily adapts to the multiparty model and is the foundation of the recent multiparty results [6, 7, 9, 10, 24]. The matrix-analytic approach does not seem to extend to three or more communicating parties.

4. A function with high threshold degree. Consider the Boolean function MP_m on $n = 4m^3$ variables, given by

$$\mathrm{MP}_m(x) = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{4m^2} x_{i,j}.$$

A moment's reflection shows that the threshold degree of MP_m is at most m. Indeed,

$$\mathrm{MP}_m(x) = \operatorname{sign}\left\{ -\frac{1}{2} + \prod_{i=1}^{m} (4m^2 + x_{i,1} + x_{i,2} + \cdots + x_{i,4m^2}) \right\}.$$

(Recall that $x_{i,j} \in \{-1, 1\}$, where -1 corresponds to "true.") Minsky and Papert [26], who originally defined this function, proved that this upper bound is tight.

THEOREM 4.1 (Minsky and Papert [26]). *MP_m has threshold degree m.*

Minsky and Papert's proof, while short and elegant, does not yield an explicit distribution over $\{-1, 1\}^{4m^3}$ with respect to which MP_m is orthogonal to all functions of fewer than m variables. The existence of such a distribution is assured by Corollary 2.5. The purpose of this section is to construct it. While this construction is not needed for our circuit lower bound (Theorem 1.1), it yields additional insight into the discrepancy of AC^0 (Theorem 1.3).

We shall construct the desired distribution by extending an earlier argument, due to O'Donnell and Servedio [29], that makes the crux of the Minsky–Papert construction explicit. A starting point in our discussion is the following fact.

PROPOSITION 4.2 (O'Donnell and Servedio [29]). *Let ν be the binomial distribution over $\{0, 1, \ldots, 2m\}$, i.e., $\nu(t) = 2^{-2m}\binom{2m}{t}$. Then for every polynomial p of degree at most $2m - 1$,*

$$\mathop{\mathbf{E}}_{t \sim \nu}[(-1)^t p(t)] = 0.$$

Proof. We present the proof from [29]. The claim holds for the monomials

$p = 1, t, t^2, \ldots, t^{2m-1}$ in view of the combinatorial identity

$$\sum_{t=0}^{2m} \binom{2m}{t} (-1)^t t^d = 0, \qquad d = 0, 1, \ldots, 2m-1.$$

By linearity of expectation, this completes the proof. □

O'Donnell and Servedio used Proposition 4.2 to obtain an explicit distribution over $\{0, 1, \ldots, 2m\}$ under which every low-degree symmetric polynomial has zero correlation with MP_m. However, what we seek is an explicit distribution over $\{-1, 1\}^{4m^3}$. To this end, we take the argument of O'Donnell and Servedio a step further. The technical exposition follows.

For $t = 0, 1, \ldots, 2m$, define

$$X_t = \left\{ x : \sum_{j=1}^{4m^2} \frac{1 - x_{i,j}}{2} = 4m^2 - (t - (2i-1))^2 \quad \text{for } i = 1, 2, \ldots, m \right\}. \tag{4.1}$$

Thus, $X_0, X_1, \ldots, X_{2m}$ are disjoint sets of inputs. The same sets of inputs figure in previous analyses [26, 29]. It is easy to verify that for $t = 0, 1, \ldots, 2m$,

$$x \in X_t \qquad \Longrightarrow \qquad \mathrm{MP}_m(x) = (-1)^t. \tag{4.2}$$

Let ν be the distribution from Proposition 4.2. We will work with the following distribution μ over $\{-1, 1\}^{4m^3}$:

$$\mu(x) = \begin{cases} \nu(0)/|X_0| & \text{if } x \in X_0, \\ \nu(1)/|X_1| & \text{if } x \in X_1, \\ \quad \vdots & \\ \nu(2m)/|X_{2m}| & \text{if } x \in X_{2m}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

We are now in a position to prove the main result of this section.

THEOREM 4.3 (explicit distribution for MP_m). *Let μ be given by* (4.3). *Then*

$$\mathop{\mathbf{E}}_{\mu}[\mathrm{MP}_m \cdot \chi_S] = 0, \qquad |S| = 0, 1, \ldots, m-1.$$

Proof. Let χ_S be arbitrary with $|S| \leqslant m-1$. Call the variables $x_{i,1}, x_{i,2}, \ldots, x_{i,4m^2}$ the *ith block* of x. Let $\sigma_1, \sigma_2, \ldots, \sigma_m$ be fixed permutations for blocks $1, 2, \ldots, m$, respectively. The theorem follows immediately from the following two claims.

CLAIM 4.4. $\mathbf{E}_\mu[\mathrm{MP}_m \cdot (\chi_S \circ (\sigma_1, \ldots, \sigma_m))] = \mathbf{E}_\mu[\mathrm{MP}_m \cdot \chi_S]$ *for all* $\sigma_1, \ldots, \sigma_m$.

CLAIM 4.5. $\sum_{\sigma_1, \ldots, \sigma_m} \mathbf{E}_\mu[\mathrm{MP}_m \cdot (\chi_S \circ (\sigma_1, \ldots, \sigma_m))] = 0$.

We prove these claims below. This completes the proof of the theorem. □

Proof of Claim 4.4. The functions $\mathrm{MP}_m(x)$ and $\mu(x)$ depend only on the sum of the bits in each block. Formally, $\mathrm{MP}_m \equiv \mathrm{MP}_m \circ (\sigma_1, \ldots, \sigma_m)$ and $\mu \equiv \mu \circ (\sigma_1, \ldots, \sigma_m)$. The claim follows. □

Proof of Claim 4.5. Write $\chi_S = \chi_{S_1} \chi_{S_2} \cdots \chi_{S_m}$, where

$$S_i = S \cap \{(i,1), \ldots, (i, 4m^2)\}, \qquad i = 1, 2, \ldots, m.$$

Then,

$$\begin{aligned}\sum_{\sigma_1,\ldots,\sigma_m} \mathop{\mathbf{E}}_{\mu}[\mathrm{MP}_m \cdot (\chi_S \circ (\sigma_1,\ldots,\sigma_m))] &= \sum_{\sigma_1,\ldots,\sigma_m} \mathop{\mathbf{E}}_{\mu}\left[\mathrm{MP}_m \cdot \prod_{i=1}^{m}(\chi_{S_i}\circ\sigma_i)\right] \\ &= \mathop{\mathbf{E}}_{\mu}\left[\mathrm{MP}_m \cdot \prod_{i=1}^{m}\left(\sum_{\sigma_i}\chi_{S_i}\circ\sigma_i\right)\right] \\ &= \mathop{\mathbf{E}}_{\mu}\left[\mathrm{MP}_m \cdot \prod_{i=1}^{m} p_i(x_{i,1}+x_{i,2}+\cdots+x_{i,4m^2})\right],\end{aligned}$$

where $p_1, p_2, \ldots, p_m$ are polynomials of degree at most $|S_1|, |S_2|, \ldots, |S_m|$, respectively. We now use the definition of μ as follows to simplify the last equation:

$$\begin{aligned}&\mathop{\mathbf{E}}_{\mu}\left[\mathrm{MP}_m \cdot \prod_{i=1}^{m} p_i(x_{i,1}+x_{i,2}+\cdots+x_{i,4m^2})\right] \\ &\quad= \sum_{x}\mu(x)\mathrm{MP}_m(x)\prod_{i=1}^{m} p_i(x_{i,1}+x_{i,2}+\cdots+x_{i,4m^2}) \\ &\quad= \sum_{t=0}^{2m}\sum_{x\in X_t}\frac{\nu(t)}{|X_t|}\mathrm{MP}_m(x)\prod_{i=1}^{m} p_i(x_{i,1}+x_{i,2}+\cdots+x_{i,4m^2}) \\ &\quad= \sum_{t=0}^{2m}\sum_{x\in X_t}\frac{\nu(t)}{|X_t|}(-1)^t \underbrace{\prod_{i=1}^{m} p_i(2[t-(2i-1)]^2-4m^2)}_{\text{call this } p(t)} \qquad \text{by (4.1), (4.2)} \\ &\quad= \sum_{t=0}^{2m}\nu(t)(-1)^t p(t) \\ &\quad= 0,\end{aligned}$$

where the last equality follows by Proposition 4.2 since $p(t)$ has degree at most $2\sum_i |S_i| = 2|S| \leqslant 2m-2$. □

5. Discrepancy of AC^0 circuits. This section proves an exponentially small upper bound on the discrepancy of an explicit function in AC^0.

THEOREM 1.3 (rephrased from section 1.2). *There exists a function $f : \{-1,1\}^N \times \{-1,1\}^N \to \{-1,1\}$, explicitly given and computable by an AC^0 circuit of depth 3, that has discrepancy $\exp(-\Omega(N^{1/5}))$ with respect to an explicitly given distribution.*

Proof. Consider the function MP_m on $n = 4m^3$ variables. Theorem 4.1 states that $\deg_{\pm}(\mathrm{MP}_m) = m$. Put $N = \lceil 16en^2/m\rceil = \lceil 256em^5\rceil$ and define the matrix $M = [\mathrm{MP}_m(x|_S)]_{x,S}$, where $x \in \{-1,1\}^N$ and $S \in \binom{[N]}{n}$. By Theorem 1.2,

$$\mathrm{disc}_\lambda(M) \leqslant 2^{-m} = \mathrm{e}^{-\Theta(N^{1/5})}$$

for a certain distribution λ. By Remark 3.3 and Theorem 4.3, the distribution λ is given explicitly in terms of (4.3).

Represent a set $S \subset [N]$ with elements $i_1 < i_2 < \cdots < i_n$ by the Boolean string $(y^{(1)}, y^{(2)}, \ldots, y^{(n)}) \in (\{-1,1\}^{\log N})^n$, where $y^{(k)}$ is the binary encoding of the integer i_k. We define $F : \{-1,1\}^N \times (\{-1,1\}^{\log N})^n \to \{-1,1\}$ by

$$F(x, y^{(1)}, y^{(2)}, \ldots, y^{(n)}) = \mathrm{MP}_m(x|_S),$$

where S is the set corresponding to $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$. In the event that the strings $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ do not specify a legal set S (e.g., they are not all distinct or ordered), the value of F is irrelevant. By construction,

$$\operatorname{disc}_\lambda(F) = \operatorname{disc}_\lambda(M) \leqslant \mathrm{e}^{-\Theta(N^{1/5})}.$$

It remains to show that F is computable by an AC^0 circuit of depth 3. For this, observe that

$$F(x, y) = \mathrm{MP}_m(\phi(x, y^{(1)}), \ldots, \phi(x, y^{(n)})),$$

where $\phi(x, y^{(i)})$ computes $x_{\mathrm{decimal}(y^{(i)})}$, i.e., computes x_a with a being the decimal integer whose binary representation is $y^{(i)}$. Each $\phi(x, y^{(i)})$ is clearly computable by a CNF formula of size $O(N)$. Hence, F is computable by an AC^0 circuit of depth 3 (by collapsing the two middle layers of AND gates). □

Remark 5.1. The function F in Theorem 1.3 can be viewed as a communication problem in which Alice is given an input $x \in \{-1, 1\}^N$, Bob is given a polynomial-size DNF formula $f : \{-1, 1\}^N \to \{-1, 1\}$ (from a restricted set), and their objective is to evaluate $f(x)$. The proof of Theorem 1.3 shows that the communication matrix of this problem has discrepancy $\exp(-\Omega(N^{1/5}))$. We will revisit this observation in section 8.

Theorem 1.3 exhibits an AC^0 circuit of depth 3 with exponentially small discrepancy. At the same time, the discrepancy of every AC^0 circuit of depth 2 is at least $n^{-O(1)}$. To our knowledge, this fact has not been noted in the literature, and we present its proof below.

PROPOSITION 5.2. *Let $f : \{-1, 1\}^n \times \{-1, 1\}^n \to \{-1, 1\}$ be an AC^0 circuit of depth 1 or 2. Then $\operatorname{disc}_\mu(f) \geqslant n^{-O(1)}$ for every distribution μ.*

Proof. By assumption, f is a polynomial-size DNF or CNF formula. Without loss of generality, assume the former, i.e., $f = T_1 \vee T_2 \vee \cdots \vee T_s$, where $s = n^{O(1)}$ and each of $T_1, T_2, \ldots, T_s$ is a conjunction of literals. Observe that

$$f = \textsc{majority}(T_1, \ldots, T_s, T_{s+1}, \ldots, T_{2s-1}),$$

where we define $T_{s+1} = T_{s+2} = \cdots = T_{2s-1} = -1$ (identically true). Consider the public-coin randomized protocol in which the parties pick $i \in \{1, 2, \ldots, 2s-1\}$ uniformly at random, evaluate T_i using constant communication, and output the result. This protocol evaluates f correctly with probability at least $\frac{1}{2} + \Omega\left(\frac{1}{s}\right)$. Thus,

$$\mathrm{R}^{\mathrm{pub}}_{1/2-\Omega(1/s)}(f) = O(1).$$

Proposition 2.1 now implies that $\operatorname{disc}_\mu(f) \geqslant \Omega(1/s) \geqslant n^{-O(1)}$ for all μ. □

6. Discrepancy and the polynomial hierarchy. In this section, we will briefly digress from the main development and explore the consequences of Theorem 1.3 in the study of communication complexity classes $\mathsf{PP}^{cc}, \Sigma_2^{cc}, \Pi_2^{cc}$.

Throughout this section, the symbol $\{f_n\}$ shall stand for a family of functions $f_1, f_2, \ldots, f_n, \ldots$, where $f_n : \{-1, 1\}^n \times \{-1, 1\}^n \to \{-1, 1\}$.

Babai, Frankl, and Simon [3] originally defined the class PP^{cc} as the class of communication problems that have an efficient protocol with nonnegligible bias. For our purposes, it will be more convenient to use an equivalent characterization of PP^{cc} in terms of discrepancy, obtained by Klauck [16].

THEOREM 6.1 (Klauck [16]). *A family* $\{f_n\}$ *is in* PP^{cc} *if and only if for some constant* $c > 1$ *and all* n,

$$\operatorname{disc}(f_n) > 2^{-\log^c n}.$$

We now define classes Σ_2^{cc} and Π_2^{cc}, which represent the second level of the polynomial hierarchy in communication complexity. A function $f_n : \{-1,1\}^n \times \{-1,1\}^n \to \{-1,1\}$ is called a *rectangle* if there exist subsets $A, B \subseteq \{-1,1\}^n$ such that

$$f_n(x,y) = -1 \quad \Leftrightarrow \quad x \in A,\ y \in B.$$

We call f_n the *complement of a rectangle* if the negated function $\neg f_n = -f_n$ is a rectangle.

DEFINITION 6.2 (Babai, Frankl, and Simon [3, sect. 4]).

(1) *A family* $\{f_n\}$ *is in* Π_0^{cc} *if each* f_n *is a rectangle. A family* $\{f_n\}$ *is in* Σ_0^{cc} *if* $\{\neg f_n\}$ *is in* Π_0^{cc}.

(2) *Fix an integer* $k = 1, 2, \ldots$. *A family* $\{f_n\}$ *is in* Σ_k^{cc} *if for some constant* $c > 1$ *and all* n,

$$f_n = \bigvee_{i_1=1}^{2^{\log^c n}} \bigwedge_{i_2=1}^{2^{\log^c n}} \bigvee_{i_3=1}^{2^{\log^c n}} \cdots \bigodot_{i_k=1}^{2^{\log^c n}} g_n^{i_1,i_2,\ldots,i_k},$$

where $\bigodot = \bigvee$ *(resp.,* $\bigodot = \bigwedge$*) for* k *odd (resp., even), and each* $g_n^{i_1,i_2,\ldots,i_k}$ *is a rectangle (resp., the complement of a rectangle) for* k *odd (resp., even). A family* $\{f_n\}$ *is in* Π_k^{cc} *if* $\{\neg f_n\}$ *is in* Σ_k^{cc}.

(3) *The polynomial hierarchy is given by* $\mathsf{PH}^{cc} = \bigcup_k \Sigma_k^{cc} = \bigcup_k \Pi_k^{cc}$, *where* $k = 0, 1, 2, 3, \ldots$ *ranges over all constants.*

Thus, the zeroth level (Σ_0^{cc} and Π_0^{cc}) of the polynomial hierarchy consists of rectangles and complements of rectangles, the simplest functions in communication complexity. The first level is easily seen to correspond to functions with efficient nondeterministic or co-nondeterministic protocols: $\Sigma_1^{cc} = \mathsf{NP}^{cc}$ and $\Pi_1^{cc} = \mathsf{coNP}^{cc}$.

The circuit class AC^0 is related to the polynomial hierarchy PH^{cc} in communication complexity in the obvious way. Specifically, if $f_n : \{-1,1\}^n \times \{-1,1\}^n \to \{-1,1\}$, $n = 1, 2, 3, 4, \ldots$, is an AC^0 circuit family of depth k with an OR gate at the top (resp., AND gate), then $\{f_n\} \in \Sigma_{k-1}^{cc}$ (resp., $\{f_n\} \in \Pi_{k-1}^{cc}$). In particular, the depth-3 circuit family $\{f_n\}$ in Theorem 1.3 is in Σ_2^{cc}, whereas $\{\neg f_n\}$ is in Π_2^{cc}. In this light, Theorems 1.3 and 6.1 have the following corollary.

COROLLARY 1.4 (restated from section 1.2). $\Sigma_2^{cc} \not\subseteq \mathsf{PP}^{cc}$, $\Pi_2^{cc} \not\subseteq \mathsf{PP}^{cc}$.

Observe that the separations in Corollary 1.4 are achieved for explicit functions, constructed in Theorem 1.3. Corollary 1.4 is tight in that PP^{cc} trivially contains $\Sigma_0^{cc}, \Sigma_1^{cc}, \Pi_0^{cc}, \Pi_1^{cc}$.

7. Lower bounds for majority-of-threshold circuits. At last, we are in a position to prove the main result of this paper. We will follow an established argument, due to Nisan [27], that relates discrepancy to the size of majority-of-threshold circuits. The key piece of the argument is the following statement.

THEOREM 7.1 (Nisan [27]). *Let* $f : \{-1,1\}^n \to \{-1,1\}$ *be a linear threshold function. Then* $\mathrm{R}_\epsilon^{\mathrm{pub}}(f) = O(\log n + \log \frac{1}{\epsilon})$ *for any partition of the variables and any* $\epsilon = \epsilon(n)$.

We have the following theorem.

THEOREM 1.1 (rephrased from section 1). *There exists a function* $f : \{-1,1\}^N \times \{-1,1\}^N \to \{-1,1\}$*, explicitly given and computable by an* AC^0 *circuit of depth* 3*, whose computation requires a majority vote of* $\exp(\Omega(N^{1/5}))$ *linear threshold gates.*

Proof. Nisan [27, Thm. 4] proved an analogous statement for the function INNER PRODUCT MODULO 2, and we merely adapt his argument to our setting. Let F be the function in the statement of Theorem 1.3, with $\operatorname{disc}(F) = \exp(-\Omega(N^{1/5}))$. Proposition 2.1 implies that for any $\gamma > 0$,

$$\mathrm{R}^{\mathrm{pub}}_{1/2-\gamma/2}(F) = \Omega(N^{1/5}) - \log\frac{1}{\gamma}. \tag{7.1}$$

On the other hand, suppose that $F = \textsc{majority}(h_1, h_2, \ldots, h_s)$, where each h_i is a linear threshold function. Then the parties can randomly pick $i \in \{1,2,\ldots,s\}$, evaluate h_i correctly with probability $1 - 1/(4s)$ using Theorem 7.1, and output the result. This protocol would have communication cost $O(\log N + \log s)$ and would predict F correctly with probability at least $(\frac{1}{2} + \frac{1}{2s}) - \frac{1}{4s} = \frac{1}{2} + \frac{1}{4s}$ on every input. Thus,

$$\mathrm{R}^{\mathrm{pub}}_{1/2-1/4s}(F) = O(\log N + \log s). \tag{7.2}$$

Comparing (7.1) and (7.2), we see that $s = \exp(\Omega(N^{1/5}))$. □

8. An application to learning DNF formulas. We conclude with an application of our results to computational learning theory. Let $\mathcal{C}$ be an arbitrary set of Boolean functions $\{-1,1\}^n \to \{-1,1\}$. Suppose it is possible to fix polynomial-time computable Boolean functions $h_1, \ldots, h_d : \{-1,1\}^n \to \{-1,1\}$ such that every function $f \in \mathcal{C}$ can be represented as

$$f(x) \equiv \operatorname{sign}\left(\sum_{i=1}^{d} a_i h_i(x)\right)$$

for some integers $a_1, \ldots, a_d$ with $|a_1| + \cdots + |a_d| \leqslant W$. The obvious complexity measures of this representation are d and W. If d and W are polynomial in n, simple and efficient algorithms exist for learning $\mathcal{C}$ from random examples under every distribution, e.g., the classic perceptron algorithm [26, 28]. Such classes $\mathcal{C}$ admit learning with *large margin* and therefore possess a variety of desirable characteristics [18].

Given $\mathcal{C}$, it is thus natural to ask whether it is possible to choose $h_1, \ldots, h_d$ such that $d = \mathsf{poly}(n)$ and $W = \mathsf{poly}(n)$. The question is particularly intriguing for polynomial-size DNF and CNF formulas, a concept class that has eluded every attempt at an efficient, distribution-free learning algorithm. Our machinery yields a strong negative answer to this question. We restrict our attention to DNF formulas, as the CNF case is closely analogous.

THEOREM 8.1. *Let* $\mathcal{C}$ *denote the concept class of polynomial-size DNF formulas. Let* $h_1, \ldots, h_d : \{-1,1\}^n \to \{-1,1\}$ *be arbitrary Boolean functions such that every* $f \in \mathcal{C}$ *can be expressed as* $f(x) \equiv \operatorname{sign}(\sum_{i=1}^{d} a_i h_i(x))$ *for some integers* $a_1, \ldots, a_d$ *with* $|a_1| + \cdots + |a_d| \leqslant W$*. Then*

$$dW \geqslant \mathrm{e}^{\Omega(n^{1/5})}.$$

Proof. Consider the communication problem F in which Alice is given an input $x \in \{-1,1\}^n$, Bob is given a function $f \in \mathcal{C}$, and their objective is to compute $f(x)$.

By Remark 5.1, the communication matrix of this problem has discrepancy

$$\operatorname{disc}(F) \leqslant \mathrm{e}^{-\Omega(n^{1/5})}.$$

We will construct a cost-2 public-coin randomized protocol for the problem, with advantage $1/(dW)$ on every input. Proposition 2.1 will then imply that

$$\frac{1}{dW} \leqslant 4 \operatorname{disc}(F),$$

and the proof will be complete.

The idea behind the protocol is not original; see [13, 14, 25, 31] for similar work. First, the parties pick an index $i \in \{1, \ldots, d\}$ uniformly at random. Then Alice sends $h_i(x)$ to Bob. Bob retrieves the representation of f as $f(x) \equiv \operatorname{sign}(\sum_{i=1}^{d} a_i h_i(x))$ for some integers $a_1, \ldots, a_d$. With probability $\frac{1}{2} + \frac{1}{2} \cdot \frac{|a_i|}{|a_1|+\cdots+|a_d|}$, Bob announces $h_i(x) \cdot \operatorname{sign}(a_i)$ as the output. With the remaining probability, he announces $-h_i(x) \cdot \operatorname{sign}(a_i)$. Thus, Bob's expected output is $\frac{a_i h_i(x)}{|a_1|+\cdots+|a_d|}$. As a result, the protocol achieves the following desired advantage:

$$f(x) \cdot \sum_{i=1}^{d} \frac{1}{d} \cdot \frac{a_i h_i(x)}{|a_1| + \cdots + |a_d|} = \frac{1}{d} \cdot \frac{|a_1 h_1(x) + \cdots + a_d h_d(x)|}{|a_1| + \cdots + |a_d|} \geqslant \frac{1}{dW}. \qquad \square$$

Remark 8.2. Using subtle techniques, Razborov and Sherstov [36] have recently proved the following substantially stronger result. Let $h_1, \ldots, h_d : \{-1, 1\}^n \to \mathbb{R}$ be arbitrary real functions such that every DNF formula f of linear size is representable as $f(x) \equiv \operatorname{sign}(\sum_{i=1}^{d} a_i h_i(x))$ for some reals $a_1, \ldots, a_d$. Then $d \geqslant \exp(\Omega(n^{1/3}))$. This lower bound on d is essentially optimal [17] and rules out the possibility of PAC learning DNF formulas in the important *dimension complexity* framework; see [36] for details.

Acknowledgments. I would like to thank Anna Gál, Adam Klivans, and Sasha Razborov for helpful discussions and feedback on an earlier version of this manuscript. I am thankful to Adam Klivans for pointing me to the problem of separating AC^0 from depth-2 majority circuits.

REFERENCES

[1] E. Allender, *A note on the power of threshold circuits*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Research Triangle Park, NC, 1989, pp. 580–584.

[2] J. Aspnes, R. Beigel, M. L. Furst, and S. Rudich, *The expressive power of voting polynomials*, Combinatorica, 14 (1994), pp. 135–148.

[3] L. Babai, P. Frankl, and J. Simon, *Complexity classes in communication complexity theory*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Toronto, 1986, pp. 337–347.

[4] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs*, J. Comput. System Sci., 45 (1992), pp. 204–232.

[5] H. Buhrman, N. K. Vereshchagin, and R. de Wolf, *On computation and communication with small bias*, in Proceedings of the 22nd Annual IEEE Conference on Computational Complexity (CCC), San Diego, CA, 2007, pp. 24–32.

[6] A. Chattopadhyay, *Discrepancy and the power of bottom fan-in in depth-three circuits*, in Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Providence, RI, 2007, pp. 449–458.

[7] A. Chattopadhyay and A. Ada, *Multiparty communication complexity of disjointness*, in Electronic Colloquium on Computational Complexity (ECCC), 2008, Report TR08-002 (electronic).

[8] F. R. K. Chung and P. Tetali, *Communication complexity and quasi randomness*, SIAM J. Discrete Math., 6 (1993), pp. 110–123.

[9] M. David and T. Pitassi, *Separating NOF communication complexity classes* RP *and* NP, in Electronic Colloquium on Computational Complexity (ECCC), 2008, Report TR08-014 (electronic).

[10] M. David, T. Pitassi, and E. Viola, *Improved separations between nondeterministic and randomized multiparty communication*, in Proceedings of the 12th International Workshop on Randomization and Computation (RANDOM), 2008, pp. 371–384.

[11] J. Ford and A. Gál, *Hadamard tensors and lower bounds on multiparty communication complexity*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), Springer, Berlin, 2005, pp. 1163–1175.

[12] J. Forster, *A linear lower bound on the unbounded error probabilistic communication complexity*, J. Comput. System Sci., 65 (2002), pp. 612–625.

[13] J. Forster, M. Krause, S. V. Lokam, R. Mubarakzjanov, N. Schmitt, and H.-U. Simon, *Relations between communication complexity, linear arrangements, and computational complexity*, in Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS), Springer, Berlin, 2001, pp. 171–182.

[14] M. Goldmann, J. Håstad, and A. A. Razborov, *Majority gates vs. general weighted threshold gates*, Comput. Complex., 2 (1992), pp. 277–300.

[15] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, J. Comput. System Sci., 46 (1993), pp. 129–154.

[16] H. Klauck, *Lower bounds for quantum communication complexity*, SIAM J. Comput., 37 (2007), pp. 20–46.

[17] A. R. Klivans and R. A. Servedio, *Learning DNF in time* $2^{\tilde{O}(n^{1/3})}$, J. Comput. System Sci., 68 (2004), pp. 303–318.

[18] A. R. Klivans and R. A. Servedio, *Learning intersections of halfspaces with a margin*, in Proceedings of the 17th Conference on Learning Theory (COLT), Springer, Berlin, 2004, pp. 348–362.

[19] A. R. Klivans and A. A. Sherstov, *A lower bound for agnostically learning disjunctions*, in Proceedings of the 20th Conference on Learning Theory (COLT), Springer, Berlin, 2007, pp. 409–423.

[20] A. R. Klivans and A. A. Sherstov, *Unconditional lower bounds for learning intersections of halfspaces*, Mach. Learn., 69 (2007), pp. 97–114.

[21] M. Krause and P. Pudlák, *On the computational power of depth-2 circuits with threshold and modulo gates*, Theoret. Comput. Sci., 174 (1997), pp. 137–156.

[22] M. Krause and P. Pudlák, *Computing Boolean functions by polynomials and threshold circuits*, Comput. Complexity, 7 (1998), pp. 346–370.

[23] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[24] T. Lee and A. Shraibman, *Disjointness is hard in the multi-party number-on-the-forehead model*, in Proceedings of the 23rd IEEE Conference on Computational Complexity (CCC), College Park, MD, 2008, pp. 81–91.

[25] N. Linial and A. Shraibman, *Learning complexity vs. communication complexity*, in Proceedings of the 23rd IEEE Conference on Computational Complexity (CCC), College Park, MD, 2008, pp. 53–63.

[26] M. L. Minsky and S. A. Papert, *Perceptrons: Expanded Edition*, MIT Press, Cambridge, MA, 1988.

[27] N. Nisan, *The communication complexity of threshold gates*, in Combinatorics, Paul Erdős Is Eighty, Vol. 1, János Bolyai Math. Soc., Budapest, 1993, pp. 301–315.

[28] A. B. J. Novikoff, *On convergence proofs for perceptrons*, in Proceedings of the Symposium on the Mathematical Theory of Automata, Vol. XII, Polytechnic Press, Brooklyn, NY, 1962, pp. 615–622.

[29] R. O'Donnell and R. A. Servedio, *New degree bounds for polynomial threshold functions*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA, 2003, pp. 325–334.

[30] R. O'Donnell and R. A. Servedio, *Extremal properties of polynomial threshold functions*, J. Comput. System Sci., 74 (2008), pp. 298–312.

[31] R. Paturi and J. Simon, *Probabilistic communication complexity*, J. Comput. System Sci., 33 (1986), pp. 106–123.

[32] R. Raz, *Fourier analysis for probabilistic communication complexity*, Comput. Complex., 5 (1995), pp. 205–221.
[33] R. Raz, *The BNS-Chung criterion for multi-party communication complexity*, Comput. Complex., 9 (2000), pp. 113–122.
[34] A. A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 106 (1992), pp. 385–390.
[35] A. A. Razborov, *Quantum communication complexity of symmetric predicates*, Izv. Math., 67 (2003), pp. 145–159.
[36] A. A. Razborov and A. A. Sherstov, *The sign-rank of* AC^0, in Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Philadelphia, 2008, pp. 57–66.
[37] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, New York, 1998.
[38] A. A. Sherstov, *Powering requires threshold depth* 3, Inform. Process. Lett., 102 (2007), pp. 104–107.
[39] A. A. Sherstov, *Communication lower bounds using dual polynomials*, Bull. EATCS, 95 (2008), pp. 59–93.
[40] A. A. Sherstov, *Halfspace matrices*, Comput. Complex., 17 (2008), pp. 149–178.
[41] A. A. Sherstov, *The pattern matrix method for lower bounds on quantum communication*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC), Victoria, British Columbia, 2008, pp. 85–94.
[42] A. A. Sherstov, *The unbounded-error communication complexity of symmetric functions*, in Proceedings of the 49th Symposium on Foundations of Computer Science (FOCS), 2008, pp. 384–393.
[43] K.-Y. Siu and J. Bruck, *On the power of threshold circuits with small weights*, SIAM J. Discrete Math., 4 (1991), pp. 423–435.
[44] K.-Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister, *Depth efficient neural networks for division and related problems*, IEEE Trans. Inform. Theory, 39 (1993), pp. 946–956.
[45] K.-Y. Siu and V. P. Roychowdhury, *On optimal depth threshold circuits for multiplication and related problems*, SIAM J. Discrete Math., 7 (1994), pp. 284–292.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2130–2161

INTERPOLATION OF DEPTH-3 ARITHMETIC CIRCUITS WITH TWO MULTIPLICATION GATES*

AMIR SHPILKA†

Abstract. In this paper we consider the problem of constructing a small arithmetic circuit for a polynomial for which we have oracle access. Our focus is on n-variate polynomials, over a finite field $\mathbb{F}$, that have depth-3 arithmetic circuits (with an addition gate at the top) with two multiplication gates of degree at most d. We obtain the following results: 1. *Multilinear case.* When the circuit is multilinear (multiplication gates compute multilinear polynomials) we give an algorithm that outputs, with probability $1-o(1)$, *all* the depth-3 circuits with two multiplication gates computing the polynomial. The running time of the algorithm is $\mathrm{poly}(n,|\mathbb{F}|)$. 2. *General case.* When the circuit is not multilinear we give a quasi-polynomial (in $n,d,|\mathbb{F}|$) time algorithm that outputs, with probability $1-o(1)$, a succinct representation of the polynomial. In particular, if the depth-3 circuit for the polynomial is not of small *depth*-3 *rank* (namely, after removing the g.c.d. (greatest common divisor) of the two multiplication gates, the remaining linear functions span a not too small linear space), then we output the depth-3 circuit itself. In the case that the rank is small we output a depth-3 circuit with a quasi-polynomial number of multiplication gates. ⋄ Prior to our work there have been several interpolation algorithms for restricted models. However, all the techniques used there completely fail when dealing with depth-3 circuits with even just two multiplication gates. Our proof technique is new and relies on the factorization algorithm for multivariate black-box polynomials, on lower bounds on the length of linear locally decodable codes with two queries, and on a theorem regarding the structure of identically zero depth-3 circuits with four multiplication gates.

Key words. arithmetic circuits, exact learning, interpolation, reconstruction, depth-3

AMS subject classification. 68Q25

DOI. 10.1137/070694879

1. Introduction.

In this work we consider the problem of constructing a small arithmetic circuit for a polynomial for which we have oracle access. That is, there is a black box holding a polynomial f, and we would like to find a small arithmetic circuit that computes f. We are allowed to pick inputs (adaptively) and query the black box for the value of f on those inputs. The focus of this work is on n-variate polynomials that have small depth-3 arithmetic circuits[1] over some finite field $\mathbb{F}$. We consider the *simplest* such circuits, that is, those with only two multiplication gates (also known as $\Sigma\Pi\Sigma(2)$ circuits). We obtain the following results. Let f be a polynomial, for which we have oracle access, that is computed by a $\Sigma\Pi\Sigma(2)$ circuit. When f is computed by a *multilinear* $\Sigma\Pi\Sigma(2)$ circuit we give a polynomial time algorithm that outputs, with high probability, *all* the multilinear $\Sigma\Pi\Sigma(2)$ circuits that compute f. When f does not have a multilinear $\Sigma\Pi\Sigma(2)$ circuit, we output in quasi-polynomial time, with high probability (w.h.p.), a short description for f (depending on a technical condition, we output either a $\Sigma\Pi\Sigma(2)$ circuit for it or a depth-3 circuit of quasi-polynomial size).

*Received by the editors June 20, 2007; accepted for publication (in revised form) September 22, 2008; published electronically February 11, 2009. A preliminary version of this paper appeared in [Shp07]. This research was supported by Israel Science Foundation grant 439/06.
http://www.siam.org/journals/sicomp/38-6/69487.html
†Faculty of Computer Science, Technion, Haifa 32000, Israel (shpilka@cs.technion.ac.il).
[1]In this work, whenever we say *depth*-3 *circuit* we mean a circuit with an addition gate at the top—the reason being that, by factoring the circuit and then applying known interpolation algorithms for depth-2 circuits, it is easy to reconstruct depth-3 circuits that have a multiplication gate at the top.

The problem of reconstructing a small arithmetic circuit for a polynomial using queries is a basic problem in algebraic complexity and is closely related to problems in learning theory. We now give some background that explains why studying depth-3 circuits is the next natural step given our current state of knowledge.

1.1. Computational learning theory. This paper considers the task of exact learning of algebraic functions. The analogous problem for Boolean functions is well studied, and we first summarize the state of knowledge there. The question of whether we can compute a small description for a Boolean function, for which we have oracle access, is a fundamental problem in learning theory. The problem, also known as the exact learning problem using membership queries, attracted a lot of research, and both positive and negative results were proved. On the negative side it was shown that if a class of Boolean circuits $\mathcal{C}$ contains *trapdoor functions* or *pseudorandom functions*, then there are no efficient learning algorithms for it [OGM86, KV94, Kha95]. In particular, there are no efficient interpolation algorithms for the class TC^0_4 (the class of depth-4 threshold circuits), under a widely believed cryptographic assumption [KL01, NR04]. Moreover, in [RR97] it was proved that if we consider a class of circuits $\mathcal{C}$ that can compute pseudorandom functions efficiently, then it is hard to determine, in exponential time, whether a function given by its truth table can be computed efficiently by a circuit from $\mathcal{C}$. In other words, even if the algorithm is given the whole truth table as input, it cannot determine whether f has a polynomial size circuit in $\mathcal{C}$ or not, in exponential time (i.e., in time polynomial in the size of the truth table).

On the positive side, there are many works showing that in some restricted models of computation, e.g., when f has a small circuit from a restricted class of circuits, exact learning from membership queries is possible (see, e.g., [SS96, BBV96, BBTV97, BBB$^+$00]). However, no exact learning algorithms are known for the class of bounded depth Boolean circuits. Moreover, even if we allow the algorithm to run in exponential time and have access to the truth table of the function, it is still not known how to compute a small bounded depth circuit for it.

To conclude, exact learning is known only for very restricted classes of circuits, and we cannot hope to learn the class of depth-4 threshold circuits if certain cryptographic assumptions hold.

1.2. Interpolation of arithmetic circuits. As mentioned above, we consider the algebraic analogue of the exact learning problem. Let $\mathcal{A}$ be a class of arithmetic circuits over a field $\mathbb{F}$. We are given oracle access to a polynomial $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ that can be computed by an arithmetic circuit from $\mathcal{A}$. We are allowed to ask for the value of the polynomial at points of our choice, and we would like to output a succinct representation for it.[2] Ideally we would like to output an arithmetic circuit from $\mathcal{A}$ that computes the polynomial. This problem is also known as the polynomial interpolation problem.

Unlike the exact learning scenario, there are almost no results that show the impossibility of interpolating arithmetic circuits. To the best of our knowledge, the only hardness result was obtained by Fortnow and Klivans [FK06], who proved that polynomial time interpolation of arithmetic circuits implies a lower bound for the class $\mathrm{ZPEXP}^{\mathrm{RP}}$. A probable explanation for the lack of stronger hardness results is that no reasonable notion of *pseudorandom polynomials* is known in the algebraic domain. However, it is widely believed that, analogously to the exact learning case, it is impossible to efficiently interpolate arithmetic circuits of a certain constant depth. The

[2] In the case of finite fields we may ask for the value over an algebraic extension field of $\mathbb{F}$.

reason for this belief is that depth-3 arithmetic circuits can compute the arithmetic analogue of threshold functions (see, e.g., [SW01]), and as efficient exact learning of TC^0_4 is believed to be impossible, we also expect efficient interpolation of bounded depth arithmetic circuits to be impossible. It is natural to ask then, what is the maximal depth for which efficient interpolation is possible.

Similarly to the exact learning version, a lot of effort was invested in trying to interpolate restricted classes of arithmetic circuits. In particular, the class of depth-2 arithmetic circuits[3] received a great deal of attention, and several interpolation algorithms were devised for it [BOT88, GKS94, KS96, Man95, SS96, KS01]. Many works also focused on circuits that can be represented by small *multiplicity automata* [BBV96, BBTV97, BBB+00, KS06] and on the class of read-once arithmetic formulae [HH91, BHH95, BC98, BB98, SV08]. One unifying feature of all these classes is that they all compute polynomials whose partial derivatives span a low dimensional space (see, e.g., [KS06], where depth-3 circuits of a very special form are discussed). In contrast, it is easy to give an example of a multilinear $\Sigma\Pi\Sigma(2)$ circuit that computes a polynomial whose partial derivatives span a high dimensional space. Thus, known techniques cannot give efficient algorithms for interpolating polynomials computed by multilinear $\Sigma\Pi\Sigma(2)$ circuits. This highlights the gap in our understanding of depth-2 circuits and depth-3 circuits (even those with only two multiplication gates).

Thus, current techniques are incapable of interpolating depth-3 circuits, even those with two multiplication gates, and it is believed that above some constant depth efficient interpolation is impossible.

In this work we introduce new techniques that enable us to give interpolation algorithms to the class of depth-3 circuits with two multiplication gates. Before presenting our results we need to give several definitions.

1.3. Some definitions and statement of our results. Let f be a polynomial computed by a $\Sigma\Pi\Sigma(2)$ circuit. Then f has the following form:

$$f(\bar{x}) = \prod_{i=1}^{d_1} L_i^{(1)}(\bar{x}) + \prod_{i=1}^{d_2} L_i^{(2)}(\bar{x}), \tag{1}$$

where the $L_i^{(j)}$'s are linear functions in the variables $\bar{x} = (x_1, \ldots, x_n)$, over the field $\mathbb{F}$:

$$L_i^{(j)}(\bar{x}) = \sum_{k=1}^{n} \alpha_{i,j,k} x_k + \alpha_{i,j,0}$$

for $\alpha_{i,j,k} \in \mathbb{F}$. Let M_1 and M_2 be the multiplication gates of the circuit. That is,

$$M_1 = \prod_{i=1}^{d_1} L_i^{(1)}(\bar{x}) \text{ and } M_2 = \prod_{i=1}^{d_2} L_i^{(2)}(\bar{x}).$$

For a $\Sigma\Pi\Sigma(2)$ circuit C we denote by $\deg(C)$ the maximal degree of its multiplication gates. For example, if C is given by (1), then $\deg(C) = \max(d_1, d_2)$. Our first result deals with the case of multilinear $\Sigma\Pi\Sigma(2)$ circuits. A multilinear circuit is a circuit in which every multiplication gate computes a multilinear polynomial. In particular, the degree of a multilinear circuit is bounded by n.

[3]Polynomials computed by small depth-2 circuits are also known as sparse polynomials, i.e., polynomials with a small number of monomials.

THEOREM 1. *Let f be a multilinear polynomial in n variables that is computed by a degree d multilinear $\Sigma\Pi\Sigma(2)$ circuit, over a field $\mathbb{F}$. Then there is a randomized interpolation algorithm that, given black-box access to f and the parameters d and n, runs in* $\mathrm{poly}(n, |\mathbb{F}|)$*-time and with probability* $1 - o(1)$ *outputs* all *the $\Sigma\Pi\Sigma(2)$ circuits computing f. When $|\mathbb{F}| < n^5$ the algorithm is allowed to make queries to f from a polynomial size algebraic extension field of $\mathbb{F}$.*

Notice that we output all the multilinear $\Sigma\Pi\Sigma(2)$ circuits computing f. While this was not our purpose when first studying the problem, it is a nice consequence of our techniques (and in particular implies that there are only polynomially many such circuits).

In order to state our main theorem we need some more definitions. Let $\mathcal{C}$ be as in (1). Define

$$\gcd(C) \stackrel{\triangle}{=} \text{g.c.d.}(M_1, M_2)$$

as the greatest common divisor of the multiplication gates. It is clear that we can write $\gcd(C) = \prod_{i=1}^{k} L_i(\bar{x})$ for some set of linear functions. Following the notations of [DS06] we define the simplification of C, $\mathrm{sim}(C)$, to be the circuit

$$\mathrm{sim}(C) \stackrel{\triangle}{=} C/\gcd(C).$$

From the definition of $\mathrm{sim}(C)$ it is clear that there exists a subset[4] $I_1 \subseteq [d_1]$ and a subset $I_2 \subseteq [d_2]$, such that $|I_1| = d_1 - k$, $|I_2| = d_2 - k$, and

$$\mathrm{sim}(C) = \prod_{i \in I_1} L_i^{(1)}(\bar{x}) + \prod_{i \in I_2} L_i^{(2)}(\bar{x}). \tag{2}$$

We shall also need the notions of the rank of a $\Sigma\Pi\Sigma(2)$ circuit, which we denote by $\mathrm{rank}(C)$, and of *depth-3 rank* of f, which we denote by $\mathrm{rank}(f)$. Given a $\Sigma\Pi\Sigma(2)$ arithmetic circuit C, let $\mathrm{sim}(C), I_1$, and I_2 be as in (2). We define

$$\mathrm{rank}(C) \stackrel{\triangle}{=} \dim\left(\mathrm{span}\left\{L_i^{(1)}, L_j^{(2)} \ : \ i \in I_1,\ j \in I_2\right\}\right).$$

In other words, the rank of C is defined to be the dimension of the space spanned by the linear functions in $\mathrm{sim}(C)$. Let $\mathrm{rank}_d(f)$ be the minimum, over all $\Sigma\Pi\Sigma(2)$ circuits C of degree $\leq d$ that compute f, of $\mathrm{rank}(C)$. The motivation for the definition will become clearer in the analysis of our algorithm. When the degree d is clear from the context, we drop the subscript d and simply write $\mathrm{rank}(f)$ instead of $\mathrm{rank}_d(f)$.

We can now state our second result that deals with general $\Sigma\Pi\Sigma(2)$ circuits.

THEOREM 2. *Let f be an n-variate polynomial computed by a $\Sigma\Pi\Sigma(2)$ circuit of degree d, over a field $\mathbb{F}$. Then there is a randomized interpolation algorithm that, given black-box access to f and the parameters d and n, runs in quasi-polynomial time (in n, d, $|\mathbb{F}|$) and has the following properties:*

- *If* $\mathrm{rank}(f) = \Omega\left(\log^2(d)\right)$*, then with probability* $1 - o(1)$ *the algorithm outputs the (unique) $\Sigma\Pi\Sigma(2)$ circuit for f.*
- *If* $\mathrm{rank}(f) = O\left(\log^2(d)\right)$*, then the algorithm outputs, with probability* $1 - o(1)$*, a polynomial* $\mathrm{Lin}(f)$*, a polynomial $Q(y_1, \ldots, y_k)$, and k linear functions $L_1, \ldots, L_k$, where $k \leq \mathrm{rank}(f)$, such that* $\mathrm{Lin}(f)$ *is the product of all the linear factors of f and* $\mathrm{Lin}(f) \cdot Q(L_1, \ldots, L_k) = f$.

[4] As usual, $[k]$ stands for the set $\{1, \ldots, k\}$.

When $|\mathbb{F}| < \max(d^5, n^5)$, the algorithm is allowed to make queries to f from a polynomial size extension field of $\mathbb{F}$.

We note that in the case when $\mathrm{rank}(f) = O\left(\log^2(d)\right)$, the polynomial $Q(L_1, \ldots, L_k)$ can be easily represented as a $\Sigma\Pi\Sigma$ circuit with quasi-polynomially many multiplication gates (as Q has at most a quasi-polynomial number of monomials).

1.4. Our techniques. Our algorithms are based on the following scheme. We first restrict the inputs to the unknown polynomial to a low dimensional random subspace of $\mathbb{F}^n$ (although in the multilinear case the subspace is not completely random, the intuition is the same). We then interpolate the polynomial on this subspace. The next step is to lift the representation that we found to the whole space. While this is the general scheme, it has different realizations in the multilinear case and the general case, and even within each case we have to deal differently with the case of high rank and the case of low rank. However, similar problems lie at the core of the different cases. The following questions give a good intuition to the difficulties that the algorithm has to overcome:

1. Let $V_1, \ldots, V_k$ be subspaces of codimension 1 inside a linear space V. Given circuits $C_1, \ldots, C_k$ such that C_i computes $f|_{V_i}$ (the restriction of f to V_i), how can we construct from them a single circuit C for $f|_V$?
2. Given linear spaces $V \subseteq U$ such that V is of codimension 1 in U, and a circuit C computing $f|_V$, how many circuits C' are there that compute $f|_U$, and whose restriction $C'|_V$ is equal to C?

The first question arises in the case of general $\Sigma\Pi\Sigma(2)$ circuits of high rank when we interpolate the restriction of f to a random subspace V. Our algorithm first interpolates the restrictions of one of the multiplication gates to codimensional 1 subspaces of V and then combines the different results to get a representation of that gate over V (then by using the factoring algorithms of [Kal85, KT90, Kal95] we are able to interpolate $f|_V$).

To deal with the problem, we consider linear functions in the different C_i's that look similar to each other, and we try to glue them together to get a new function. This process may fail if for any linear function in, say, C_1 there are many other linear functions in C_1 that are at Hamming distance 1 from it. In such a situation it is hard for us to tell what is the "true" image of that linear function in the other C_i's. On the other hand, if this is indeed the case, then the linear functions in C_1 generate a *locally decodable code* and, using the results of [GKST06, DS06] on the length of such codes, we can prove that such an anomaly cannot occur. Therefore we can always find a linear function to learn, until eventually we find the whole multiplication gate.

The motivation for the second question is that when we lift the representation that we found over V to U there may be many different circuits that are possible lifts, and we somehow have to pick the right one with which to continue the lifting process. To deal with this problem we note that if a polynomial has two different lifts, then the difference of the lifts is the zero polynomial. By a result of [DS06] regarding the structure of identically zero depth-3 circuits, we get that the different lifts must be of small rank. This enables us to solve the problem for the high rank case (as in this case the lift is unique). The low rank case is indeed more problematic, and this is why we need to output a circuit with quasi-polynomially many multiplication gates when f has low rank (although in the multilinear case, we manage to overcome this difficulty by proving that the total number of possible lifts is polynomial).

1.5. Recent progress. In a recent joint work with Karnin [KS08] we managed to extend this result to the case of $\Sigma\Pi\Sigma(k)$ circuits. Moreover, we managed to get a

deterministic reconstruction algorithm (the algorithms in this paper are all randomized). The ideas in the current paper serve as a basic building block to the ideas of [KS08]; however, some new ideas were required for the more general case.

1.6. Organization. The paper is organized as follows. In section 2 we give some definitions and describe the results of [DS06] regarding identically zero depth-3 circuits. The algorithm and proof for the multilinear case are given in section 3. In section 4 we give the algorithm (and proof) for the general case.

2. Preliminaries. For a natural number n we denote $[n] = \{1, \ldots, n\}$. For a set $S \subset [n]$ we denote by $\bar{S}$ the complement of S.

Let $\mathbb{F}$ be a field. We denote by $\mathbb{F}^n$ the n-dimensional vector space over $\mathbb{F}$. We shall use the notation $\bar{x} = (x_1, \ldots, x_n)$ to denote the vector of n indeterminates. For $v \in \mathbb{F}^n$ we denote by $\mathrm{wt}(v)$ the weight of v, i.e., the number of nonzero coordinates in v. For two nonzero linear functions L_1, L_2 we will write $L_1 \sim L_2$ whenever L_1 and L_2 are linearly dependent. Let $V = V_0 + v_0 \subseteq \mathbb{F}^n$ be an affine subspace, where $v_0 \in \mathbb{F}^n$, and where $V_0 \subseteq \mathbb{F}^n$ is a linear subspace. Let $L(\bar{x})$ be a linear function. We denote by $L|_V$ the restriction of L to V. We say that a set of linear functions $\{L_1, \ldots, L_k\}$ is linearly independent over V if the only linear combination of the L_i's whose restriction to V is identically zero is the all zero combination. We now introduce coordinates to the space V. Let $v_1, \ldots, v_s$ be a basis of V_0. Let $L(\bar{x})$ be a linear function. We consider the restriction of L to V with respect to the basis $\{v_i\}$. Then $L|_V$ can be written as a function in s variables. In particular if $v = \alpha_1 v_1 + \cdots + \alpha_s v_s + v_0$, then we define $L|_V(\alpha_1, \ldots, \alpha_s) = L(v) = L(\alpha_1 v_1 + \cdots + \alpha_s v_s) + L(v_0)$.

For a polynomial f we denote by $\mathrm{Lin}(f)$ the product of all the linear factors of f (with the appropriate multiplicities). We also define $\mathrm{sim}(f) = f/\mathrm{Lin}(f)$ to be the simplification of f. Clearly $\mathrm{sim}(f)$ does not have any linear factors.

2.1. Identically zero depth-3 circuits. In this section we state some results of [DS06] regarding identically zero depth-3 circuits. We start by giving some necessary definitions. A $\Sigma\Pi\Sigma(k)$ circuit (that is, a depth-3 circuit with k multiplication gates) is identically zero if it computes the zero polynomial. Notice that this is a syntactic definition; we are thinking of the circuit as computing a polynomial and not as computing a function over the field.[5] Let $C = M_1 + \cdots + M_k$ be an identically zero $\Sigma\Pi\Sigma(k)$ circuit, where the M_i's are the multiplication gates. We say that C is minimal if there is no $\emptyset \neq I \subsetneq [k]$ such that $\sum_{i \in I} M_i \equiv 0$. C is simple if the g.c.d. of its multiplication gates is 1. The following theorem gives a bound on the degree of multilinear $\Sigma\Pi\Sigma(k)$ circuits that are identically zero.

THEOREM 3 (Corollary 6.9 of [DS06]). *There exists an integer function $D(k) = 2^{O(k^2)}$ such that every simple, minimal, identically zero multilinear $\Sigma\Pi\Sigma(k)$ circuit is of degree $d \leq D(k)$.*

In other words, if C is an identically zero $\Sigma\Pi\Sigma(k)$ multilinear circuit that is simple and minimal, then the degree of C is bounded by a constant depending on k. We will need to use the result only for $\Sigma\Pi\Sigma(4)$ circuits. We state this as a corollary and introduce the constant D_4 that will play a part in our interpolation algorithms.

COROLLARY 4. *There exists a constant D_4, such that every identically zero multilinear $\Sigma\Pi\Sigma(4)$ circuit that is simple and minimal is of degree $\leq D_4$.*

The next theorem deals with a general $\Sigma\Pi\Sigma(k)$ circuit.

[5] In our case the field size is much larger than the degree of the polynomial and so the syntactic definition and the semantic one are the same. However, we mention this distinction as it is sometimes a cause of confusion.

THEOREM 5 (Lemma 5.2 of [DS06]). *Let $k \geq 3$, $d \geq 2$ be integers and $\mathcal{C} \equiv 0$ be a simple and minimal $\Sigma\Pi\Sigma(k)$ circuit. Then* $\text{rank}(\mathcal{C}) \leq 2^{O(k^2)} \log^{k-2}(d)$.

As before, we shall need the following corollary and the constant R_4.

COROLLARY 6. *There exists a constant R_4 such that every identically zero $\Sigma\Pi\Sigma(4)$ circuit, of degree d, that is simple and minimal is of rank at most $R_4 \cdot \log^2(d)$.*

The following corollary shows how to use Corollaries 4 and 6 to guarantee uniqueness of a $\Sigma\Pi\Sigma(2)$ circuit.

COROLLARY 7. *Let d be some integer, and let D_4 and R_4 be as in Corollaries* 4 and 6*, respectively. If a polynomial f has a degree d (multilinear), $\Sigma\Pi\Sigma(2)$ circuit and* $\text{rank}(f) > R_4 \cdot \log^2(d)$ $(\deg(f) > D_4)$*, then f has a unique (multilinear) $\Sigma\Pi\Sigma(2)$ circuit of degree d.*

Proof. We prove the claim only for general $\Sigma\Pi\Sigma(2)$ circuits. The proof for the multilinear case is identical. Let $C_1 = A_1 + A_2$ be a $\Sigma\Pi\Sigma(2)$ circuit for f, where A_1, A_2 are its multiplication gates. By our assumption on $\text{rank}(f)$ we know that

$$\text{rank}\left(C_1/\text{g.c.d.}(A_1, A_2)\right) > R_4 \cdot \log^2(d).$$

Assume that $C_2 = B_1 + B_2$ is another $\Sigma\Pi\Sigma(2)$ circuit for f, where B_1, B_2 are its multiplication gates. Consider the circuit $C = A_1 + A_2 - B_1 - B_2$; then C is a $\Sigma\Pi\Sigma(4)$ circuit, and $C \equiv 0$. By the assumption on $\text{rank}(C_1)$ we get that $\text{rank}(C) > R_4 \cdot \log^2(d)$. By Corollary 6 (for the case of multilinear circuits, we use Corollary 4) this implies that C is either not simple or not minimal. Note that by the assumption on $\text{rank}\left(C_1/\text{g.c.d.}(A_1, A_2)\right)$ we get that, even if we remove the g.c.d. of C, the remaining circuit still has rank $> R_4 \cdot \log^2(d)$. From this we conclude that C is not minimal. As $C_1 \not\equiv 0$, we get that $A_1 - B_1 \equiv 0$ or $A_1 - B_2 \equiv 0$. □

3. Multilinear circuits. In this section we prove Theorem 1. For ease of reading we repeat it here.

THEOREM 1. *Let f be a multilinear polynomial in n variables that is computed by a degree d multilinear $\Sigma\Pi\Sigma(2)$ circuit, over a field $\mathbb{F}$. Then there is a randomized interpolation algorithm that, given black-box access to f and the parameters d and n, runs in* $\text{poly}(n, |\mathbb{F}|)$*-time and with probability* $1 - o(1)$ *outputs* all *the $\Sigma\Pi\Sigma(2)$ circuits computing f. When $|\mathbb{F}| < n^5$ the algorithm is allowed to make queries to f from a polynomial size algebraic extension field of $\mathbb{F}$.*

Before sketching the proof we repeat some of the notations that we will use. We shall have the following representation of f in mind: $f = M_1 + M_2$, where M_1 and M_2 are the multiplication gates of a multilinear $\Sigma\Pi\Sigma(2)$ circuit for f that are given by the equations

$$M_1 = \prod_{i=1}^{d} L_i(S_i) \quad \text{and} \quad M_2 = \prod_{j=1}^{d'} L'_j(S'_j), \tag{3}$$

where the L_i's (resp., L'_j's) are linear functions, and the sets $\{S_i\}_{i\in[d]}$ $(\{S'_j\}_{j\in[d']})$ form a partition of the set of variables (recall that M_1 and M_2 compute multilinear polynomials). In particular,

$$f(\bar{x}) = \prod_{i=1}^{d} L_i(S_i) + \prod_{j=1}^{d'} L'_j(S'_j). \tag{4}$$

To prove the theorem we will give an algorithm for reconstructing all the $\Sigma\Pi\Sigma(2)$ circuits for f and prove its correctness. The algorithm basically follows the scheme

that was described in section 1.4. We have two cases, the low rank case and the high rank case (in fact we look at low degree and high degree, but for multilinear circuits these correspond to low rank and high rank). More precisely, in the multilinear setting low rank means constant rank (where the constant depends on D_4 as defined in Corollary 4). This makes life easier compared to the general case that we will study in section 4 in which low rank can be as large as $O(\log^2(d))$. We now give a slightly more detailed sketch of the proof as follows:

- *Preprocessing*: The first step in the proof is a preprocessing step in which we find all the linear factors of f. We then show that in the multilinear case each linear factor of f is also a linear factor of both multiplication gates in every $\Sigma\Pi\Sigma(2)$ circuit for f. This enables us to reduce the problem of finding all the circuits for f to finding the circuits for $\mathrm{sim}(f)$. After this step the algorithm treats separately low degree circuits and high degree circuits.
- *Low degree circuits*: The idea is to consider all the restrictions of $\mathrm{sim}(f)$ to small subsets of the variables. Namely, for a set S the restriction of $\mathrm{sim}(f)$ to S is done by setting the variables not in S to zero. We denote the restriction of $\mathrm{sim}(f)$ to S with $\mathrm{sim}(f)(S)$. We will consider only sets S of small size, and so we can find (in a brute force manner) all the simple multilinear $\Sigma\Pi\Sigma(2)$ circuits for $\mathrm{sim}(f)(S)$. Then we will prove that for every multilinear $\Sigma\Pi\Sigma(2)$ circuit C for $\mathrm{sim}(f)$ there exists a set S and a simple multilinear circuit A on the variables in S such that A is the restriction of C to S (which is defined in a similar manner to $\mathrm{sim}(f)(S)$) and that there is a relatively easy way of reconstructing C from A by "revealing" each variable separately.
- *High degree circuits*: The algorithm for the high degree case is similar in spirit to the low degree case with a few exceptions. The first difference is that we will not restrict the variables not in S to zero but rather substitute random values from $\mathbb{F}$ to each of them. The second difference is that in the low degree case the degree of the circuit A (that was described in the previous item) is the same as the degree of C, whereas in the high degree case after restricting the circuit to a small set the degree must drop (it can be at most $|S|$ as the circuit is multilinear).

3.1. Preprocessing step. Before describing the algorithms for the low rank and high rank cases, we have a preprocessing step in which we find all the linear factors of f. We will show that in the case of multilinear $\Sigma\Pi\Sigma(2)$ circuits we actually get that the linear factors of f are also linear factors of each multiplication gate separately. This allows us to reduce the case of reconstructing multilinear $\Sigma\Pi\Sigma(2)$ circuits to the problem of interpolating multilinear $\Sigma\Pi\Sigma(2)$ circuits that do not have linear factors. We start by showing that we can find the linear factors efficiently. The following theorem is an immediate corollary of the results of [Kal85, KT90, Kal95]. The theorem requires that the field that we are working with is not too small, so from now on we shall assume that $|\mathbb{F}| \geq n^5$ (we can make this assumption as we are allowed to query f on inputs from an extension field).

THEOREM 8. *Let d, n be integers. Let $\mathbb{F}$ be a finite field. Then there is a randomized algorithm A that gets as input a black-box access to f and the parameters n and d, and outputs, in $\mathrm{poly}(n, d, \log|\mathbb{F}|)$ time, with probability $1 - \exp(-n)$, all the linear factors, with their multiplicities, of f.*

Let $\mathrm{Lin}(f)$ be the product of all the linear factors of f. The following lemma shows that if f is not a product of linear functions, then every linear function in its factorization also divides the multiplication gates M_1 and M_2 (note that this is not

the case for nonmultilinear $\Sigma\Pi\Sigma(2)$ circuits).[6]

LEMMA 9. *Let f be a polynomial that is computable by a multilinear $\Sigma\Pi\Sigma(2)$ circuit (as in (4)). Assume that f cannot be represented as a product of linear functions. Then if a linear function L divides f, then L must also divide both multiplication gates, in any multilinear $\Sigma\Pi\Sigma(2)$ circuit for f.*

Proof. Consider a multilinear $\Sigma\Pi\Sigma(2)$ circuit for f. Assume w.l.o.g. that it is given by (4). As we assume that f cannot be represented as a product of linear functions, we can also assume w.l.o.g. that g.c.d.$(M_1, M_2) = 1$. Assume for a contradiction that L does not divide M_1 (and therefore does not divide M_2). Consider the (affine) subspace on which $L = 0$. We must have that $f|_{L=0} = 0$. This implies that $M_1|_{L=0} = -M_2|_{L=0}$. As M_1 and M_2 are both products of linear functions, we get that they share the same linear factors modulo L. In particular we can assume w.l.o.g. that $d' = d$ and $L_i|_{L=0} \sim L'_i|_{L=0}$.[7] By examination we get that the support of L (that is, the set of nonzero coordinates of L) is contained in the union of the supports of L_i and L'_i, for every $1 \leq i \leq d$ (recall that we assume that g.c.d.$(M_1, M_2) = 1$). If $d > 2$, then this is not possible, as the circuit for f is multilinear and the S_i's are disjoint (and so are the S'_j's). Thus we get that $d = 2$ (recall that we assumed that we removed the g.c.d.). However, if $d = 2$ and L divides f, then there is another linear function L' such that $f = L \cdot L'$ in contradiction to the assumption that f is not a product of linear functions. □

Thus, contrary to the general case (i.e., nonmultilinear circuits), every linear factor of f is also a linear factor of both multiplication gates in every multilinear $\Sigma\Pi\Sigma(2)$ circuit for f (unless f itself is a product of linear functions). Recall that $\mathrm{sim}(f) = f/\mathrm{Lin}(f)$. Thus, if we have $\mathrm{Lin}(f)$ at hand, then it is easy to simulate oracle access to $\mathrm{sim}(f)$ by making queries to f.[8] Moreover, as we assume that we are given the degree d of a multilinear $\Sigma\Pi\Sigma(2)$ circuit for f, then it is easy to compute $D \stackrel{\Delta}{=} d - \deg(\mathrm{Lin}(f))$. Hence, in the rest of the section we will reconstruct all the simple multilinear $\Sigma\Pi\Sigma(2)$ circuits for $\mathrm{sim}(f)$ of degree D, and from them we will immediately get all the multilinear $\Sigma\Pi\Sigma(2)$ circuits for f.

As described in the sketch we have two cases, the case that $D \leq D_4$ (low degree case) and the case that $D > D_4$ (high degree case).[9] We begin with the low degree case.

3.2. Multilinear circuits: Low degree case. In this section we give an algorithm that finds all the multilinear $\Sigma\Pi\Sigma(2)$ circuits of degree $D \leq D_4$ that compute $\mathrm{sim}(f)$. We assume w.l.o.g. that the number of variables in $\mathrm{sim}(f)$ is more than $10D_4$, as otherwise we can find all the circuits for $\mathrm{sim}(f)$ by a brute force search (this assumption is used in Lemma 11).

We start with a sketch of the algorithm. We first compute the sum-product representation of f. Note that as the degree of $\mathrm{sim}(f)$ is at most D_4, we can interpolate $\mathrm{sim}(f)$ in polynomial time. This will be helpful in future steps where we have to verify that a given circuit computes a certain restriction of $\mathrm{sim}(f)$. In the next step, for each

[6] E.g., consider the circuit $(x + w)(x + y)(x + z) - wyz$. It is not hard to see that it is divisible by x but cannot be represented as a product of linear functions.

[7] We can also have $d = d' \pm 1$ but the analysis remains the same.

[8] Given $\mathrm{Lin}(f)$, an oracle access to f, and a point $\alpha \in \mathbb{F}^n$, we would like to output $\mathrm{sim}(f) = f(\alpha)/\mathrm{Lin}(f)(\alpha)$. There may be a problem when $\mathrm{Lin}(f)(\alpha) = 0$; however, this can be easily taken care of by passing a line through α that contains enough (e.g., more than d) points on which $\mathrm{Lin}(f)$ does not vanish. This is a routine procedure and so we omit its details here.

[9] Recall that D_4 is defined in Corollary 4.

subset S of the variables of size $|S| = 10D_4$, we consider the restriction of $\mathrm{sim}(f)$ in which every variable not in S is set to zero. We denote this polynomial by $\mathrm{sim}(f)(S)$. We then find, in a brute force manner, all the degree $\leq D$ multilinear $\Sigma\Pi\Sigma(2)$ circuits that compute $\mathrm{sim}(f)(S)$. We now wish to construct all the circuits computing $\mathrm{sim}(f)$ from the circuits for the different $\mathrm{sim}(f)(S)$'s. To do so, we consider for every such set S all the sets of the form $S \cup \{i\}$ for $i \notin S$. For each such new set we again compute all the circuits for $\mathrm{sim}(f)(S \cup \{i\})$. Now we try to combine circuits for $\mathrm{sim}(f)(S \cup \{1\}), \ldots, \mathrm{sim}(f)(S \cup \{n\})$ into a single circuit for $\mathrm{sim}(f)$. At first sight it may not be clear how to do this combination; however, we prove that for every circuit for $\mathrm{sim}(f)$ there is a set S for which such a combination will be easy to find. We now give a more formal description of the algorithm (Algorithm 1).

Algorithm 1 shows how to find all the multilinear $\Sigma\Pi\Sigma(2)$ circuits that compute $\mathrm{sim}(f)$, when its degree D is at most D_4. As we are looking for multilinear circuits and we have $\mathrm{Lin}(f)$, we shall only consider variables that do not belong to $\mathrm{Lin}(f)$. In order to not add further notations we shall assume that the variables appearing in $\mathrm{sim}(f)$ are $\{x_1, \ldots, x_n\}$.

ALGORITHM 1 (multilinear circuits of low degree).

1. Interpolate the polynomial $\mathrm{sim}(f)$ to get an explicit representation of it as a sum of monomials.
2. $\forall S \subseteq [n]$ of size $|S| = 10D_4$ find all the degree D *simple* multilinear circuits in the variables of S that compute $\mathrm{sim}(f)(S)$ (recall that $\mathrm{sim}(f)(S)$ is the polynomial obtained after setting the variables not in S to zero).
3. For each such set S and circuit A computing $\mathrm{sim}(f)(S)$ do the following: for $i \notin S$ set $S_i = S \cup \{i\}$. Repeat the previous steps and find all the multilinear $\Sigma\Pi\Sigma(2)$ circuits A_i that compute $\mathrm{sim}(f)(S_i)$ and such that $A_i|_{x_i=0} \equiv A$ (i.e., after substituting $x_i = 0$ the circuits are identical). If for some i there is no such circuit, then move to the next circuit A for $\mathrm{sim}(f)(S)$.
4. For each S and A for which we found $\{A_i\}_{i \in \bar{S}}$ combine, if possible, the different circuits into one multilinear $\Sigma\Pi\Sigma(2)$ circuit: for every linear function $L \in A$ let $L_i \in A_i$ be such that $L_i|_{x_i=0} = L$. Denote $L_i = L + \alpha_i \cdot x_i$. Replace L with $\tilde{L} = L + \sum_{i \notin S} \alpha_i \cdot x_i$.
5. For each circuit found in the previous step verify that it computes $\mathrm{sim}(f)$.

Before giving the analysis of the algorithm we explain the way in which it follows the scheme described in section 1.4. Recall that according to the sketch in section 1.4 we first have to restrict $\mathrm{sim}(f)$ to a random subspace of the inputs. As restriction to a subspace does not preserve multilinearity (imagine the polynomial xy restricted to the subspace $x = y$), we only consider subspaces in which some of the variables are restricted to zero. Clearly a restriction to such a subspace preserves multilinearity. Indeed, in step 2 we basically interpolate all the restrictions of $\mathrm{sim}(f)$ to subspaces of this form that have a low dimension. The lifting is preformed in step 4.

The following theorem summarizes the required properties of Algorithm 1.

THEOREM 10. *Let f be a multilinear polynomial that can be computed by a multilinear $\Sigma\Pi\Sigma(2)$ circuit. Assume that $\deg(\mathrm{sim}(f)) > 3$. Then Algorithm 1, when given oracle access to $\mathrm{sim}(f)$, outputs all the multilinear $\Sigma\Pi\Sigma(2)$ circuits of degree D computing $\mathrm{sim}(f)$. The running time of the algorithm is polynomial in n and $|\mathbb{F}|$.*

Note that the theorem only discusses polynomials f such that $\mathrm{sim}(f)$ has degree larger than 3. This is because when the degree of $\mathrm{sim}(f)$ is too small, there may be more possible circuits computing it. We deal with this case in section 3.2.1.

Proof. Before proving the correctness of the algorithm, we first analyze its running time. The first step in the algorithm is a simple interpolation of a multilinear polynomial of degree D in (at most) n variables. This step runs in time polynomial in n^D (we simply query the polynomial on all inputs in $\{0,1\}^n$ of weight at most D, and solve a system of linear equations to find the coefficients, remembering to look only at coefficients of monomials of degree at most D). Hence we can assume that we have an explicit representation of $\mathrm{sim}(f)$ as sum of monomials. It is clear that in the second step we get a polynomial number of circuits. Indeed, there are at most n^{10D_4} many sets S, and for each set S there are at most $|\mathbb{F}|^{2D\cdot(10D_4+1)}$ multilinear $\Sigma\Pi\Sigma(2)$ circuits in the variables of S (we just count the number of sets of at most $2D$ linear functions, in $10D_4$ variables). In the same way we see that computing the different A_i's also requires polynomial time. As we have a description of $\mathrm{sim}(f)$ at our disposal, it is easy to verify whether a given circuit computes $\mathrm{sim}(f)(S)$. Thus, steps 1–3 can be computed in polynomial time (in $|\mathbb{F}|$ and n). It is also clear that step 4 requires a polynomial time, as the number of circuits that we computed in the previous steps is also polynomial. The last step (verification step) also requires a polynomial running time, and hence the total running time is polynomial in n and $|\mathbb{F}|$.

The correctness of the algorithm follows from the next two lemmas. The first lemma (Lemma 11) shows that there exists a set S such that $\mathrm{sim}(f)(S)$ does not have linear factors, and hence step 2 can be implemented for $\mathrm{sim}(f)$. The second lemma (Lemma 12) shows that if S satisfies a certain property and A computes $\mathrm{sim}(f)(S)$, then for every $i \notin S$ there is a unique circuit A_i that computes $\mathrm{sim}(f)(S \cup \{i\})$ and that equals A after setting $x_i = 0$.

LEMMA 11. *Let $g(x_1,\ldots,x_n)$ be a polynomial on $n > 2d+1$ variables (recall our assumption at the beginning of section* 3.2*) that has no linear factors and that can be computed by a $\Sigma\Pi\Sigma(2)$ circuit of degree d. Then there exists a variable x_k such that $g|_{x_k=0}$ has no linear factors.*

Proof. Let $C = \prod_{i=1}^{d_1} L_i + \prod_{i=1}^{d_2} L'_i$ be some $\Sigma\Pi\Sigma(2)$ circuit computing g, for some $d_1, d_2 \le d$. It is clear that $\dim(\mathrm{span}(\{1, L_1,\ldots,L_{d_1}, L'_1,\ldots,L'_{d_2}\})) \le 2d+1$. In particular there is $1 \le i \le n$ such that x_i is not spanned by the linear functions in the circuit (and the constant function). Assume w.l.o.g. that $i = n$. It is now clear that if we set x_n to zero, then the resulting circuit C' will be "isomorphic" to C in the following sense: there exists an invertible linear transformation $\phi : \mathbb{F}^n \to \mathbb{F}^n$ such that for every $(a_1,\ldots,a_n) \in \mathbb{F}^n$ we have that $C(a_1,\ldots,a_n) = C'(\phi(a_1,\ldots,a_n))$ (we think of C' as a circuit in n variables although x_n does not appear in it). In particular if C' has a linear factor, then so does C (as C is equal to the composition of C' with a linear transformation). As C has no linear factors (by our assumption on g) we get that $C' = C|_{x_n=0}$ has no linear factors as well. ∎

Note that the lemma works for any $\Sigma\Pi\Sigma(2)$ circuit and not just multilinear circuits. We also note that it is not difficult to change the proof of the lemma to hold also for the case that $n = 2d+1$ (we just need to consider the homogeneous part of each L_i and L'_j and not consider the constant function 1). In addition, it is also easy to see that we cannot have $n = 2d$ because of the following example: $f(x_1,\ldots,x_{2d}) = \prod_{i=1}^{d} x_i + \prod_{i=d+1}^{2d} x_i$. The next lemma shows that if $D > 3$, then for every S_i and circuit A that was computed in step 2 there is at most one multilinear $\Sigma\Pi\Sigma(2)$ circuit A_i with $A_i = \mathrm{sim}(f)(S_i)$ and $A_i|_{x_i=0} \equiv A$. The proof is by a simple manipulation of polynomials. Note that this implies that if there exists a circuit C for f such that A is its restriction to the variables S (that is, A is the resulting circuit after setting all the variables not in S to zero), then step 4 will return the circuit C

(because of the uniqueness, we are assured that we combine the linear functions in the different A_i's in the only possible way).

LEMMA 12. *Let $g(x_1,\ldots,x_t)$ be a polynomial of degree $D>3$ that does not have any linear factors. Assume that we have a multilinear $\Sigma\Pi\Sigma(2)$ circuit computing $g|_{x_t=0}$. In other words, we have*

$$g(x_1,\ldots,x_t)|_{x_t=0}=\prod_{i=1}^{D}L_i+\prod_{j=1}^{D'}L'_j.$$

Then there is at most one multilinear circuit of the form

$$\prod_{i=1}^{D}(L_i+\alpha_i x_t)+\prod_{j=1}^{D'}(L'_j+\alpha'_j x_t)$$

that computes g (note that such a circuit may not exist).

Proof. Assume for a contradiction that there are two different multilinear circuits of that form that compute g. Assume w.l.o.g. that the first circuit is

$$g=C_1=(L_1+\alpha x_t)\cdot\prod_{i=2}^{D}L_i+(L'_1+\alpha' x_t)\cdot\prod_{j=2}^{D'}L'_j.$$

There are three canonical options for the second circuit:

$$C_2=L_1\cdot(L_2+\beta x_t)\cdot\prod_{i=3}^{D}L_i+L'_1\cdot(L'_2+\beta' x_t)\cdot\prod_{j=3}^{D'}L'_j, \tag{5}$$

$$C_2=(L_1+\beta x_t)\cdot L_2\cdot\prod_{i=3}^{D}L_i+L'_1\cdot(L'_2+\beta' x_t)\cdot\prod_{j=3}^{D'}L'_j, \tag{6}$$

$$C_2=(L_1+\beta x_t)\cdot L_2\cdot\prod_{i=3}^{D}L_i+(L'_1+\beta' x_t)\cdot L'_2\cdot\prod_{j=3}^{D'}L'_j. \tag{7}$$

We shall only analyze the first case (given in (5)), as it is the more interesting one. The analysis of the other cases is similar (and somewhat simpler).

So let us assume that C_2 is given by (5). As $C_1=C_2$ we get, by exchanging sides, that

$$x_t\cdot(\alpha L_2-\beta L_1)\cdot\prod_{i=3}^{D}L_i=x_t\cdot(\beta' L'_1-\alpha' L'_2)\cdot\prod_{j=3}^{D'}L'_j.$$

Since $D>3$, there is $3\le i\le D$ such that $L_i\not\mid(\beta' L'_1-\alpha' L'_2)$, and so there must be $3\le j\le D$ such that $L_i\sim L'_j$. This implies that L_i divides g. However, we assumed that g does not have linear factors, so we have a contradiction. ∎

The proof of Theorem 10 now follows easily. Indeed, by Lemma 11 we know that for every circuit C for $\mathrm{sim}(f)$ there exists a set S of size $|S|=10D_4$ and a simple multilinear circuit A that compute $\mathrm{sim}(f)(S)$. Lemma 12 and the discussion proceeding it show that C will be one of the circuits computed in step 4. It is clear that the last step will keep only the correct circuits. ∎

We now handle the cases of $D=3$ and $D=2$ (note that Lemma 12 says nothing for such D's and indeed, the claim is not true in this case).

3.2.1. Case $D \leq 3$. Algorithm 1 in its current form cannot compute all the circuits of degree ≤ 3 for $\mathrm{sim}(f)$. However, the required change is minor and so we describe it here and do not repeat the algorithm itself. We also give a sketch of a proof, as the proof itself contains easy manipulations similar in spirit to those that were already performed for the case $D > 3$. The basic difference is in steps 3 and 4. Instead of computing a circuit A_i for $\mathrm{sim}(f)(S \cup \{i\})$ we need to compute a circuit $A_{i,j}$ for $\mathrm{sim}(f)(S \cup \{i,j\})$. Another change is that it may be the case that for some i the circuit A_i is not the unique lift (in contrast to what Lemma 12 guarantees for higher degrees). However, in such a case we prove that $A_{i,j}$ is unique for every $j \notin S \cup \{i\}$. The rest of the proof should now be clear (given that we prove the above claim regarding $A_{i,j}$). In fact, this discussion is true only for $D = 3$ and for $D = 2$, yet another slight modification is required, but it is similar in spirit and so we omit it.

We first analyze the case $D = 3$. The only possible difference from the case $D > 3$ is when there is some index i such that there are two (or more) different ways of extending the circuit A to a circuit A_i. From the proof of Lemma 12 we get that w.l.o.g. A has the following form:

$$A = L_1 \cdot L_2 \cdot L_3 + L'_1 \cdot L'_2 \cdot L'_3,$$

and the two different lifts are[10]

$$A_i = (L_1 + \alpha_2 x_i) \cdot L_2 \cdot L_3 + (L'_1 - \beta_2 x_i) \cdot L'_2 \cdot L'_3$$

and

$$A'_i = L_1 \cdot (L_2 - \alpha_1 x_i) \cdot L_3 + L'_1 \cdot (L'_2 + \beta_1 x_i) \cdot L'_3$$

for constant $\alpha_1, \alpha_2, \beta_1$, and β_2. In particular we must have that $L_3 = c \cdot (\beta_1 L'_1 + c_2 \cdot \beta_2 L'_2)$ and $L'_3 = c \cdot (\alpha_1 L_1 + \alpha_2 L_2)$ for some $c \neq 0$. Using the same reasoning as in the proof of Lemma 12, it is easy to show that for any variable $j \notin S \cup \{i\}$ there is at most one circuit $A_{i,j}(S \cup \{i,j\})$ satisfying

$$A_{i,j} = g(S \cup \{i,j\})$$

and

$$A_{i,j}|_{x_j=0} \equiv A_i.$$

In other words, for the case $D = 3$ we have to perform steps 3 and 4 of Algorithm 1 for the circuit $A_i(S \cup \{i\})$ and its possible lifts $A_{i,j}(S \cup \{i,j\})$.

To conclude, if we have an index i with two different lifts A_i and A'_i (as described above), then, as in the case of $D > 3$, we get that there are at most two circuits $\hat{A}$ and $\hat{A}'$ that for every $j \notin S \cup \{i\}$ satisfy

$$\hat{A}(S \cup \{i,j\}) \equiv A_{i,j}$$

and

$$\hat{A}'(S \cup \{i,j\}) \equiv A'_{i,j}.$$

[10]There are two more cases to consider (corresponding to (6) and (7)) but the analysis follows from similar arguments.

Therefore we get, again, that at most polynomially many circuits are found by the algorithm, that those polynomials are computed in polynomial time, and that they contain all the possible representations for $\mathrm{sim}(f)$ (we can find those circuits by repeating step 4 of Algorithm 1 for each of the A_i's and the corresponding set $\{A_{i,j}\}$). Hence we can find in polynomial time all the different multilinear $\Sigma\Pi\Sigma(2)$ circuits for f.

The case $D = 2$ follows by a similar case analysis. We omit the proof, as this case (as well as the case $D = 3$) is not very interesting and the proofs are simple and resemble the proofs seen so far.

3.3. Multilinear circuits: High degree case. We now turn to the high degree case. As described above, the algorithm for this case is similar to Algorithm 1 with the exception that we do not substitute zeroes for the variables outside S and that "gluing" the different circuits for $\mathrm{sim}(f)(S \cup \{i\})$ is slightly more complicated. Algorithm 2 shows how to interpolate the circuit in the case when $\deg(\mathrm{sim}(f)) > D_4$.

ALGORITHM 2 (multilinear circuits of high degree).

$\forall S \subseteq [n]$ *such that* $|S| = 2D_4$do the following:

1. $\forall i \notin S$ pick a random assignment to x_i from $\mathbb{F}$. Denote the final assignment with ρ. Let $\mathrm{sim}(f)|_\rho$ be the polynomial $\mathrm{sim}(f)$ after substituting the partial assignment ρ.
2. For every simple multilinear $\Sigma\Pi\Sigma(2)$ circuit in the variables $\{x_i\}_{i\in S}$, over $\mathbb{F}$, of *degree greater than* D_4 (and at most $2D_4$ of course), check whether the circuit computes $\mathrm{sim}(f)|_\rho$. If no such circuit computing $\mathrm{sim}(f)|_\rho$ is found, then move to the next S.
3. For every $i \neq j$ such that $i, j \notin S$, set $S_{i,j} = S \cup \{i, j\}$. Find a simple multilinear $\Sigma\Pi\Sigma(2)$ circuit $A_{i,j}$ that computes $\mathrm{sim}(f)|_{\rho_{i,j}}$, where $\rho_{i,j}$ is the assignment that equals ρ on all the coordinates outside $S_{i,j}$ (namely, we "forget" the assignments to x_i and x_j).
4. Glue the different circuits for the $S_{i,j}$'s to a single multilinear $\Sigma\Pi\Sigma(2)$ circuit. That is, find the unique circuit C whose restriction to $\rho_{i,j}$ equals $A_{i,j}$ (the exact way of gluing will be explained in the analysis of the algorithm).

If no circuit is found, then output "fail."

The basic intuition behind the algorithm is the following. As we will see in Lemma 15 there is a unique circuit for $\mathrm{sim}(f)$ of degree greater than D_4 (given that one exists). Moreover, we will see that w.h.p. even when we substitute random field elements for most of the variables, there is still a unique circuit for the restricted polynomial. Thus, in order to find the circuit for $\mathrm{sim}(f)$ we first find the circuit for the restricted polynomial $\mathrm{sim}(f)|_\rho$ (steps 1 and 2) and then find the unique way of recovering the other variables. The main difference from the low degree case is that by exposing the other variables, new linear functions may appear (as the degree of the restricted polynomial is at most $|S|$), and in order to find the partition of the other variables to the new linear functions we have to consider pairs of variables (i.e., find the circuit for $\mathrm{sim}(f)(S \cup \{i, j\})$ instead of just finding a circuit for $\mathrm{sim}(f)(S \cup \{i\})$). This is done in steps 3 and 4. The following theorem shows that Algorithm 2 is indeed correct and analyzes its running time.

THEOREM 13. *Let f be a multilinear polynomial such that* $\mathrm{sim}(f)$ *can be computed by a multilinear* $\Sigma\Pi\Sigma(2)$ *circuit of degree $D > D_4$. Then Algorithm* 2*, when given oracle access to* $\mathrm{sim}(f)$*, outputs with probability* $\geq 1 - (n+1)^2/|F|$ *the unique multilinear* $\Sigma\Pi\Sigma(2)$ *circuits of degree D computing* $\mathrm{sim}(f)$*. The running time of the algorithm is polynomial in n and $|\mathbb{F}|$.*

Note that in particular the theorem implies that there is a unique degree D $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)$. This is, however, a simple fact given Corollary 7 (and in Lemma 15 we prove a slightly more general version of this fact).

Proof. The idea of the proof is the following. We first show (Lemma 14) that for every set S, w.h.p. over the choice of ρ, there is no linear factor dividing $\mathrm{sim}(f)|_\rho$ (i.e., no new linear factor was introduced). We then show (Lemma 15) that this implies that there is a unique $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)|_\rho$ of degree larger than D_4, if such a circuit exists. Combining the two lemmas, we see that there is a set S and an assignment ρ for which there is a unique $\Sigma\Pi\Sigma(2)$ circuit of degree larger than D_4 for $\mathrm{sim}(f)|_\rho$ (the set S is picked so that the degree of the circuit for $\mathrm{sim}(f)_\rho$ is larger than D_4 w.h.p. over the choice of ρ). At this point we will have at hand the unique circuit for $\mathrm{sim}(f)|_\rho$ and we would like to add to it the variables not in S to get the circuit for $\mathrm{sim}(f)$. To do so we find (again the unique) $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)|_{\rho_{i,j}}$. From that circuit we can immediately find whether x_i and x_j belong to the same linear function (for each of the multiplication gates) and what is the ratio between their coefficients. Given this information for all pairs of variables it is easy to construct the two multiplication gates. We now give the formal proof.

Let $S \subset [n]$ be a set of size $2D_4$ and $C = M_1 + M_2$ be a multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)$ of degree higher than D_4. We say that the assignment ρ is *good* for C if no new linear factors were introduced (note that the degree of $\mathrm{sim}(f)|_\rho$ may be smaller though). In other words, ρ is good if $\mathrm{Lin}(\mathrm{sim}(f)|_\rho) = 1$.

LEMMA 14. *Let $S \subset [n]$ be a set of size $2D_4$ and $C = M_1 + M_2$ be a multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)$ of degree $D > D_4$. The probability that an assignment ρ, to the variables not in S, is not good for C is smaller than $(n+1)^2/|\mathbb{F}|$.*

The proof of the lemma is a simple union bound over the event that two linearly independent linear functions in the circuit for $\mathrm{sim}(f)$ become linearly dependent nonconstant linear functions after substituting ρ.

Proof. We first bound the probability that M_1 or M_2 were set to zero. This may happen only if a linear function from any of them was set to zero, and this happens with probability $1/|\mathbb{F}|$. As there are $2D$ linear functions, we get an upper bound on the probability of $2D/|\mathbb{F}|$ (recall that we work only with the variables that do not belong to $\mathrm{Lin}(f)$). To get a bound on the probability that a new linear factor was introduced, we note that this is possible only if there is a linear function L_1 dividing M_1 and a linear function L_2 dividing M_2 such that $L_1|_\rho \sim L_2|_\rho$, and they were not set to constants. As there is only one possible coefficient c for which $L_1|_\rho = c \cdot L_2|_\rho$ (i.e., c depends only on S, L_1, and L_2 and not on ρ), we see that the probability that this happened is equal to the probability that the part in $L_1 - cL_2$ that depends on the variables outside S was set to zero. As before, this happens with probability $1/|\mathbb{F}|$. As there are at most D^2 such pairs we get an upper bound of $D^2/|\mathbb{F}|$ on the probability. Thus, the overall probability is bounded by $(D^2 + 2D)/|\mathbb{F}| < (D+1)^2/|\mathbb{F}| \le (n+1)^2/|\mathbb{F}|$. ∎

The next lemma shows that if there is a multilinear $\Sigma\Pi\Sigma(2)$ circuit $C = M_1 + M_2$, of degree $D > D_4$, for $\mathrm{sim}(f)$, then it is unique and that if S and ρ are such that ρ is good for C, then there is a unique multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)|_\rho$ (and that circuit is $M_1|_\rho + M_2|_\rho$). This lemma generalizes Corollary 7 (by taking $S = [n]$ and ρ to be the "empty" substitution, we get the claim regarding the uniqueness of the circuit for $\mathrm{sim}(f)$).

LEMMA 15. *Let $\mathrm{sim}(f)$ be computed by a multilinear $\Sigma\Pi\Sigma(2)$ circuit $C = M_1 + M_2$. For a set S and a good assignment ρ, if $\deg(\mathrm{sim}(f)|_\rho) > D_4$, then there is a unique simple multilinear $\Sigma\Pi\Sigma(2)$ circuit that computes $\mathrm{sim}(f)|_\rho$. Moreover, this circuit is $M_1|_\rho + M_2|_\rho$.*

Proof. By our assumption we have that $\deg(\mathrm{sim}(f)|_\rho) > D_4$. Assume for a contradiction that there is a multilinear $\Sigma\Pi\Sigma(2)$ circuit $C' = A_1 + A_2$ such that $C' = \mathrm{sim}(f)|_\rho$ and $C' \neq M_1|_\rho + M_2|_\rho$. Therefore the circuit $\tilde{C} = M_1|_\rho + M_2|_\rho - A_1 - A_2$ is identically zero and of degree $> D_4$. Notice that as ρ is a good assignment, we have that $\tilde{C}$ is simple. By Corollary 4 we are assured that it is not minimal, and so we must have that $A_1 = M_1|_\rho$ or $A_1 = M_2|_\rho$, which is a contradiction. ∎

Given the two lemmas it is easy to explain steps 1 and 2 of the algorithm. Let C be *the* unique multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)$ of degree higher than D_4. Let S be a set of size $2D_4$ such that the degree of C in the variables of S is larger than D_4 (think of the other variables as constants and view C as a circuit over the variables in S and compute its degree). By Lemma 14 we get that with probability $\geq 1-(n+1)^2/|F|$ the substitution ρ is such that $C|_\rho$ is the unique circuit for $\mathrm{sim}(f)|_\rho$. Thus, for this set S we get that with probability $\geq 1-(n+1)^2/|F|$ steps 1 and 2 find the unique circuit $C|_\rho$ (and this circuit is of degree greater than D_4). We now continue and show that if such S and good ρ were found, then steps 3 and 4 return the circuit C.

As ρ is good for C, it immediately follows that $\rho_{i,j}$ is good for C (and $S \cup \{i,j\}$). Thus the circuits found in step 3 are the unique circuits computing the different $f|_{\rho_{i,j}}$'s (i.e., those circuits are identical to the $C|_{\rho_{i,j}}$'s). We now explain how step 4 is performed. Consider a linear function L that belongs to M_1. There are two possibilities. Either $L|_\rho$ is constant or not. If $L|_\rho$ is not constant, then for every variable x_i we will find its coefficient in L by considering the circuit, say, $C|_{\rho_{1,i}}$ (assuming that $i \neq 1$; otherwise consider the circuit $C|_{\rho_{1,2}}$). If L was restricted to a constant, then denote $L = \sum_{i=1}^n \alpha_i \cdot x_i$. Notice that in $C|_{\rho_{i,j}}$ there will be a linear function of the form $c\cdot(\alpha_i \cdot x_i + \alpha_j \cdot x_j)$, for some nonzero field element c. In particular, if for some chosen x_i that appears in L, we chose to normalize its coefficient to always be 1, then we can find in this manner all the coefficients of the other variables appearing with it in L and so reconstruct (a nonzero multiple of) L. In this way we can find all the linear functions appearing in M_1 and M_2. Note that we found all the linear factors up to some nonzero multiple, so it only remains to find a constant that will give the correct normalization of each M_i. This can be easily done by querying $\mathrm{sim}(f)$ on two points, one on which M_1 vanishes and not M_2 and vice versa. We note that we do not have to verify that we found the correct circuit for $\mathrm{sim}(f)$, as by Lemma 15 and step 2 we are guaranteed that we have the correct circuit for $\mathrm{sim}(f)|_\rho$ (assuming that ρ is good) and it is clear that starting from this circuit we find the unique circuit C for $\mathrm{sim}(f)$. Thus, with probability at least $1-(n+1)^2/|F|$ the algorithm finds the unique multilinear circuit for $\mathrm{sim}(f)$.

It only remains to analyze the running time of the algorithm. Given a set S and an assignment ρ we check whether $\mathrm{sim}(f)|_\rho$ has new linear factors by applying Theorem 8, which requires polynomial time and succeeds w.h.p. In step 2 we have to go over all possible simple multilinear $\Sigma\Pi\Sigma(2)$ circuits for $\mathrm{sim}(f)|_\rho$ of degree greater than D_4 (and at most $2D_4$). Note that as there are only $|\mathbb{F}|^{2D_4+1}$ linear functions in the variables $\{x_i\}_{i\in S}$, there are polynomially many such circuits ($|\mathbb{F}|^{2D_4\cdot(2D_4+1)}$ is an obvious upper bound). Among all those circuits we shall find, by going over all $0,1$ assignments to $\{x_i\}_{i\in S}$, a circuit that computes $\mathrm{sim}(f)|_\rho$. As the size of S is constant, this costs only a constant factor in the running time. Verifying that the circuit is simple is pretty obvious and requires time polynomial in the degree of the circuit and linear in n. Next, for each pair (i,j) the complexity of step 3 is similar to the complexity of step 2. Finally, the procedure described above for "gluing" the different $C|_{\rho_{i,j}}$'s requires polynomial time (in n). Hence, the overall running time is

polynomial in n and $|\mathbb{F}|$, and with probability $\geq 1-(n+1)^2/|F|$ the algorithm outputs the unique multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\mathrm{sim}(f)$. Note that failure can occur only if we failed to find a good ρ. This completes the proof of Theorem 13. □

We now combine the preprocessing step and Theorems 10 and 13 to get Theorem 1 (the argument is straightforward).

3.4. Proof of Theorem 1. The proof of the theorem is easy given the two algorithms. First, we find $\mathrm{Lin}(f)$ and compute $D = d - \deg(\mathrm{Lin}(f))$. If $D \leq D_4$, then we run the low degree algorithm. If $D > D_4$, then we run the high degree algorithm. We are assured that if we get an answer, then it is going to be correct. The probability of failure in both algorithms is polynomially small in n (and can be easily decreased, e.g., by repeating the algorithm) and so the theorem follows.

4. General circuits. In this section we prove Theorem 2. For convenience we repeat it here.

THEOREM 2. *Let f be an n-variate polynomial computed by a $\Sigma\Pi\Sigma(2)$ circuit of degree d, over a field $\mathbb{F}$. Then there is a randomized interpolation algorithm that, given black-box access to f and the parameters d and n, runs in quasi-polynomial time (in n, d, $|\mathbb{F}|$) and has the following properties:*

- *If $\mathrm{rank}(f) = \Omega\left(\log^2(d)\right)$, then with probability $1-o(1)$ the algorithm outputs the (unique) $\Sigma\Pi\Sigma(2)$ circuit for f.*
- *If $\mathrm{rank}(f) = O\left(\log^2(d)\right)$, then the algorithm outputs, with probability $1-o(1)$, a polynomial $\mathrm{Lin}(f)$, a polynomial $Q(y_1,\ldots,y_k)$, and k linear functions $L_1,\ldots,L_k$, where $k \leq \mathrm{rank}(f)$, such that $\mathrm{Lin}(f)$ is the product of all the linear factors of f and $\mathrm{Lin}(f)\cdot Q(L_1,\ldots,L_k) = f$.*

When $|\mathbb{F}| < \max(d^5, n^5)$, the algorithm is allowed to make queries to f from a polynomial size extension field of $\mathbb{F}$.

From now on we shall assume, w.l.o.g., that the underlying field $\mathbb{F}$ is of size greater than $\max(d^5, n^5)$. We shall have the following representation of f in mind:

$$f(\bar{x}) = \prod_{i=1}^{d} L_i^{(1)}(\bar{x}) + \prod_{i=1}^{d} L_i^{(2)}(\bar{x}). \tag{8}$$

Note that in order to ease the notations we assume that the degrees of both multiplication gates are equal. As described in section 1.4 there are two conceptual steps in the proof. First, we restrict the inputs to come from a random subspace V of dimension $O(\log^2(d))$, where $d = \deg(f)$. We then learn the restriction of f to V which we denote by $f|_V$. The second step is to learn the restriction of f to larger subspaces and then glue the different representations together. While this is the general picture, there is a difference in the way that we handle functions with high rank and functions with low rank. We start by describing the algorithm for the low rank case.

4.1. Interpolation for the low rank case. In this section we give an interpolation algorithm for polynomials that are computed by $\Sigma\Pi\Sigma(2)$ circuits of degree d and that have rank at most $10R_4 \cdot \log^2(d)$ (recall the definition of R_4 from Corollary 6). Let f be such a polynomial. Algorithm 3 is an interpolation algorithm for f. Before giving the algorithm, we describe each of its steps.

Let C be any $\Sigma\Pi\Sigma(2)$ circuit computing f, e.g., the one given by (8). We start by computing $\mathrm{Lin}(f)$. This can be easily done using Theorem 8. After this step we can assume, as before, that we get oracle access to $\mathrm{sim}(f) = f/\mathrm{Lin}(f)$.[11] Note, that as

[11] Unlike the multilinear case, it may be the case that $\mathrm{sim}(C)$ does not compute $\mathrm{sim}(f)$.

$\text{sim}(f) = f/\text{Lin}(f)$ divides $\text{sim}(C)$ and $\text{rank}(f) \leq 10R_4 \cdot \log^2(d)$, we get that $\text{sim}(f)$ can be represented as a polynomial in $\leq 10R_4 \cdot \log^2(d)$ linear functions. Namely, there exists a polynomial $P(y_1, \ldots, y_k)$ and linearly independent linear functions $\{L_i\}_{i\in[k]}$ such that $\text{sim}(f) = P(L_1, \ldots, L_k)$ (we will always think of k as the minimal possible integer allowing such a representation). The goal of Algorithm 3 is to find such a representation for $\text{sim}(f)$. There are two main steps for doing that. First, we pick a random subspace V of dimension $O(R_4 \cdot \log^2(d))$ and look for a polynomial Q and linear functions $\ell_1, \ldots, \ell_k$ such that $\text{sim}(f)|_V = Q(\ell_1, \ldots, \ell_k)$ (we do not necessarily find the polynomial P and restricted linear functions $\{L_i|_V\}$ given above; however, the two different representations are closely related, as shown in Lemma 23). In the second step we "lift" the ℓ_i's and find ℓ_i^*'s such that for every i, $\ell_i^*|_V = \ell_i$ and $\text{sim}(f) = Q(\ell_1^*, \ldots, \ell_k^*)$. This may seem like a big leap from the outcome of the first step; however, we manage take it so by exposing one variable at a time (in a similar way to step 4 of Algorithm 1). The important ingredients of this "lifting" procedure are Lemmas 23 and 24 which basically show that if V' is a subspace containing V, then there exist unique linear functions (that can be easily found) $\{\ell_i'\}_{i=1}^k$, over V', that satisfy $\text{span}(\{L_i|_{V'}\}) = \text{span}(\{\ell_i'\})$, $\ell_i'|_V = \ell_i$, and $Q(\ell_1', \ldots, \ell_k') = \text{sim}(f)|_{V'}$. We now give the formal description of the algorithm.

ALGORITHM 3 (general $\Sigma\Pi\Sigma(2)$ circuits of low rank).

1. **Computing Lin(f):** Find the linear factors of f using the factoring algorithm of Theorem 8.
2. **Interpolating on a low dimensional random subspace:** Pick a random subspace V of dimension $s = 20R_4 \cdot \log^2(d) + \log^2(n)$. Find $k' \leq 10R_4 \cdot \log^2(d)$ linearly independent linear functions $\{\ell_i\}_{i\in[k']}$, such that $\text{sim}(f)|_V = Q(\ell_1, \ldots, \ell_{k'})$, for some polynomial Q of degree $\deg(Q) \leq d$, and such that no such representation is possible for any $k'' < k'$. In other words, do the following: For $k' = 1, \ldots, 10R_4 \cdot \log^2(d)$ consider all sets of k' linearly independent linear functions over V. For every such set $\{\ell_i\}_{i\in[k']}$ find, using brute force interpolation, a polynomial $Q(y_1, \ldots, y_{k'})$ of degree at most d, such that $Q(\ell_1, \ldots, \ell_{k'}) = \text{sim}(f)|_V$, if such a polynomial exists. Finally, output the first set $\{\ell_i\}_{i\in[k']}$ for which such a polynomial Q was found.
3. **Lifting from V to $\mathbb{F}^n$:** Find linear functions $\{\ell_i^*\}_{i\in[k']}$, over $\mathbb{F}^n$, such that $Q(\ell_1^*, \ldots, \ell_{k'}^*) = \text{sim}(f)$ using the following procedure (a more detailed explanation will be given when analyzing the algorithm):
   (a) Let $\{v_i\}_{i\in[n]}$ be a basis to $\mathbb{F}^n$, such that $\{v_i\}_{i=1}^s$ is a basis to V. For every $i \in [n-s]$ let $V_i = \text{span}(V \cup \{v_{s+i}\})$. For each V_i find a representation for $\text{sim}(f)|_{V_i}$ of the form $\text{sim}(f)|_{V_i} = Q(\ell_1^{(i)}, \ldots, \ell_k^{(i)})$, where for each i and j we have that $\ell_j^{(i)}|_V = \ell_j$.
   (b) Find the unique linear functions $\{\ell_i^*\}_{i\in[k]}$ that satisfy $\ell_j^*|_{V_i} = \ell_j^{(i)}$, for each i and j, in a way analogous to step 4 of Algorithm 1.

   If no such representation is found, then output "fail."

The next theorem shows the correctness of Algorithm 3 and analyzes its running time.

THEOREM 16. *Let f be a polynomial that can be computed by a $\Sigma\Pi\Sigma(2)$ circuit of* $\text{rank} \leq 10R_4 \cdot \log^2(d)$. *Then with probability $\geq 1 - |\mathbb{F}|^{-\Omega(\log^2(n))}$ Algorithm 3, when given oracle access to f, computes* $\text{Lin}(f)$, *a natural number $k \leq 10R_4 \cdot \log^2(d)$, a polynomial $Q(y_1, \ldots, y_k)$, and k linear functions $\{\ell_i^*\}_{i=1,\ldots,k}$ such that $Q(\ell_1^*, \ldots, \ell_k^*) =$* $\text{sim}(f)$. *The running time of the algorithm is quasi polynomial in n, d, and $|\mathbb{F}|$ (i.e., it is* $\exp(\text{poly}(\log|\mathbb{F}|, \log n, \log d))$).

The proof of correctness of the algorithm is composed of three parts. First, we study properties of polynomials that can be written as functions of exactly k linear functions. In particular we show that this property is preserved (w.h.p.) when we restrict the polynomial to a random subspace of sufficient dimension (Corollary 22). Then we consider $\mathrm{sim}(f)$ and a random subspace V of dimension $10R_4 \cdot \log^2(d)$ and show how to find the relevant linear functions for it. This will complete the analysis of step 2. After that we prove a lemma mentioned above (Lemma 24) that shows how we can "lift" the linear functions found in the previous step to functions over $\mathbb{F}^n$. This will complete the analysis of step 3. The analysis of the running time will be quite simple. For sake of modularity, in the next subsection we discuss properties of polynomials that depend on k linear functions and then proceed with the proof.

4.1.1. Polynomials depending on k linear functions. In this subsection we study the notion of a polynomial that depends on exactly k linear functions. We first define this property in a formal manner.

DEFINITION 17. *Let $h(x_1, \ldots, x_n)$ be a polynomial. We say that h is a polynomial in* exactly *k linear functions if there is a polynomial $P(y_1, \ldots, y_k)$ and k linear functions $L_1, \ldots, L_k$ such that $h = P(L_1, \ldots, L_k)$, and there is no such representation for h with less than k linear functions. We say that h is a polynomial in k linear functions if there exists a $k' \leq k$ such that h is a polynomial in exactly k' linear functions.*

The following lemma gives a sufficient and necessary condition for being a function of k linear functions.

LEMMA 18. *Let $h(x_1, \ldots, x_n)$ be a polynomial over $\mathbb{F}$. Then h can be written as a polynomial in k linear functions if and only if there is a subspace $V^* \subseteq \mathbb{F}^n$ of dimension $n-k$ such that for every $u \in \mathbb{F}^n$ and $v \in V^*$ we have that $h(u) = h(v+u)$.*

Note that the lemma does not speak about h being a function of exactly k linear functions. However, it is clear that if there is a subspace V^* of dimension $n-k$ that satisfies the condition in the lemma, and there is no subspace U^* of dimension $> n-k$ that satisfies the condition of the lemma, then h is a function of exactly k linear functions.

Proof. Assume w.l.o.g. that h is a polynomial in exactly k linear functions. Denote $h = P(L_1, \ldots, L_k)$, where the L_i's are homogeneous linear functions. Let $V^* = \{\bar{x} \in \mathbb{F}^n : \forall i\ L_i(\bar{x}) = 0\}$. Clearly $\dim(V^*) = n-k$. Let $u \in \mathbb{F}^n$ and $v \in V^*$ be two vectors. We have that $h(v+u) = P(L_1(u+v), \ldots, L_k(v+u)) = P(L_1(u), \ldots, L_k(u)) = h(u)$.

For the other direction, given such V^*, let $L_1, \ldots, L_k$ be k linearly independent linear functions such that for every $v \in V^*$ and $i \in [k]$ we have that $L_i(v) = 0$. Let $V \subset \mathbb{F}^n$ be a k-dimensional subspace such that $V \cap V^* = \{0\}$. Clearly $\mathbb{F}^n = V \oplus V^*$. It is also clear that the linear functions $\{L_i|_V\}$ are linearly independent. As $\dim(V) = k$, it follows that there is some polynomial $Q(y_1, \ldots, y_k)$ such that for every $v \in V$ it holds that $h(v) = Q(L_1(v), \ldots, L_k(v))$. Now, given $u \in \mathbb{F}^n$ write $u = v + v^*$ for $v \in V$ and $v^* \in V^*$. We now have that $h(u) = h(u - v^*) = h(v) = Q(L_1(v), \ldots, L_k(v)) = Q(L_1(v+v^*), \ldots, L_k(v+v^*)) = Q(L_1(u), \ldots, L_k(u))$. Hence, $h = Q(L_1, \ldots, L_k)$. ∎

We now give some easy corollaries of this lemma.

COROLLARY 19. *Let $h(\bar{x}) = P(L_1, \ldots, L_k)$ be a polynomial in exactly k linear functions. Let V be a subspace of $\mathbb{F}^n$. Then $h|_V$ is a polynomial in exactly k linear functions if and only if the restrictions $L_1|_V, \ldots, L_k|_V$ are linearly independent.*

COROLLARY 20. *Let h be a polynomial in exactly k linear functions. Let $V \subseteq \mathbb{F}^n$ be an s-dimensional subspace such that $h|_V$ is a polynomial in $k' < k$ linear functions. Let $v_1, \ldots, v_{n-s}$ be vectors such that $\mathrm{span}(V \cup \{v_i\}_{i \in [n-s]}) = \mathbb{F}^n$. Denote $V_i = \mathrm{span}(V \cup \{v_i\})$. Then for some $i \in [n-s]$ we have that $h|_{V_i}$ is a polynomial in*

more than k' linear functions.

In the analysis of Algorithm 3 we will be interested in restrictions to random subspaces. The following lemmas show that if h is a polynomial in exactly k linear functions and V is a subspace of sufficiently high dimension chosen at random, then w.h.p. $h|_V$ is also a polynomial in exactly k linear functions.

LEMMA 21. *Let $\{\ell_i\}_{i\in[t]}$ be a set of linearly independent linear functions over $\mathbb{F}^n$. Let $V \subseteq \mathbb{F}^n$ be a random s-dimensional subspace. Then the probability that the set $\{\ell_i|_V\}_{i\in[t]}$ is linearly dependent is at most $|\mathbb{F}|^{t-s}$.*

Proof. Clearly, $\ell_1, \ldots, \ell_t$ are linearly dependent on V if and only if there is a nonzero linear combination $\alpha_1\ell_1 + \cdots + \alpha_k\ell_k$ that vanishes on V. Given V let us define $\mathcal{L}^* = \{\ell(x_1, \ldots, x_n) : \ell|_V = 0\}$. Note that as V is random, so is $\mathcal{L}^*$. Namely, $\mathcal{L}^*$ is a random $n-s$-dimensional subspace of linear functions. Thus, $\ell_1, \ldots, \ell_t$ are linearly dependent on V if and only if there is a nonzero function in the span of the ℓ_i's that belongs to $\mathcal{L}^*$. In other words, we have to bound the probability that a random subspace of dimension $n - s$ (i.e., $\mathcal{L}^*$) intersects a given subspace of dimension $\leq t$ (i.e., $\text{span}(\ell_1, \ldots, \ell_t)$). As the probability that a random nonzero vector belongs to a given t-dimensional subspace is $(|\mathbb{F}|^t - 1)/(|\mathbb{F}|^n - 1)$ we get by the union bound that the probability of a nontrivial intersection is upper bounded by $(|\mathbb{F}|^{n-s} - 1) \cdot (|\mathbb{F}|^t - 1)/(|\mathbb{F}^n| - 1) < |\mathbb{F}|^{t-s}$. ∎

From Corollary 19 and Lemma 21 we get the following corollary.

COROLLARY 22. *Let h be a function of exactly k linear forms.*[12] *Let V be a random subspace of dimension $\geq k + \log^2(n)$. Then the probability that $h|_V$ is not a function of exactly k linear forms is at most $|\mathbb{F}|^{-\log^2(n)}$.*

The next lemma shows that if a polynomial in k linear functions, h, has two different representations as a polynomial in exactly k linear functions, then these representations are closely related.

LEMMA 23. *Let $h(\bar{x})$ be a polynomial in exactly k linear functions. Let $P(\ell'_1, \ldots, \ell'_k) = h = Q(\ell_1, \ldots, \ell_k)$ be two different representations for h. Then $\text{span}(\{\ell'_i\}_{i\in[k]}) = \text{span}(\{\ell_i\}_{i\in[k]})$. In particular this implies that there is an invertible linear transformation $T : \mathbb{F}^k \to \mathbb{F}^k$ such that $Q(y_1, \ldots, y_k) = (P \circ T)(y_1, \ldots, y_k)$.*

Proof. Assume for a contradiction that, w.l.o.g., $\ell'_1, \ell_1, \ldots, \ell_k$ are linearly independent. Let W be the codimension 1 subspace on which ℓ'_1 vanishes. It is clear that $\ell_1|_W, \ldots, \ell_k|_W$ are linearly independent, whereas $\ell'_1|_W, \ldots, \ell'_k|_W$ are linearly dependent (as $\ell'_1|_W = 0$). In particular, by Corollary 19 we get that $h|_W$ is a function of at most $k - 1$ linear functions (when considering the representation according to the $\ell'_i|_W$'s) and is a function of exactly k linear functions (when considering the representation according to the $\ell_i|_W$'s), which is a contradiction. Now, let $T : \mathbb{F}^n \to \mathbb{F}^n$ be an invertible linear transformation taking $(\ell_i)_i$ to $(\ell'_i)_i$. Namely, $T(\ell_1(v), \ldots, \ell_k(v)) = (\ell'_1(v), \ldots, \ell'_k(v))$ for every $v \in \mathbb{F}^n$. We get that for every $v \in \mathbb{F}^n$ it holds that

$$P \circ T(\ell_1(v), \ldots, \ell_k(v)) = P(\ell'_1(v), \ldots, \ell'_k(v)) = h(v) = Q(\ell_1(v), \ldots, \ell_k(v)).$$

As the ℓ_i's are linearly independent, it follows that we must have $P \circ T = Q$ (as otherwise we will have two low degree polynomials representing the same function over a large field). ∎

4.1.2. Proof of Theorem 16. We are now ready to give the proof of Theorem 16.

[12] A linear form is a homogeneous linear function.

Proof of Theorem 16. By our assumption that $\text{rank}(f) \leq 10R_4 \cdot \log^2(d)$, we get that $\text{sim}(f)$ can be written as a polynomial in at most $10R_4 \cdot \log^2(d)$ linear functions. For simplicity we shall assume that it is a polynomial of exactly $k = \text{rank}(f)$ linear functions. Let

$$\text{sim}(f) = P(L_1, \ldots, L_k), \tag{9}$$

where $P(y_1, \ldots, y_k)$ is a polynomial and $\{L_i\}_{i=1}^k$ are linear forms. We shall later use this representation. We now analyze separately each step of the algorithm. Step 1 is an immediate application of Theorem 8 and does not require special analysis.

Step 2: Interpolating on a low dimensional random subspace. Recall that in this step we pick a random subspace $V \subseteq \mathbb{F}^n$ of dimension $s = 20R_4 \cdot \log^2(d) + \log^2(n)$, and interpolate $\text{sim}(f)$ over it. By Corollary 22 we get that with probability $\geq 1 - |F|^{-\log^2(n)}$ the polynomial $\text{sim}(f)|_V$ is a polynomial in exactly k linear functions. We now find a representation of the form $\text{sim}(f)|_V = Q(\ell_1, \ldots, \ell_k)$.

The idea is pretty simple. We go over all possible values for k (recall that we do not know k in advance). For each $k' \leq 10R_4 \cdot \log^2(d) + \log^2(n)$ we go over all possible sets of k' linearly independent linear functions over V, and for each set $\{\ell_i\}_{i \in [k']}$ we try to represent $\text{sim}(f)|_V$ as a polynomial $Q(\ell_1, \ldots, \ell_{k'})$. Note that given $\{\ell_i\}_{i \in [k']}$ we can find such a polynomial Q (if such Q exists) by a simple brute force interpolation. As this is routine, we skip it here (for completeness we give a description of this process in Appendix A).

Using the approach above we may get many different representations for $\text{sim}(f)|_V$. Among all those representations we pick one with the smallest possible k'. As mentioned above, by Corollary 22 we get that w.h.p. $k' = k$. Moreover, Corollary 20 gives a way to verify that indeed, $k' = k$. So, to conclude, after the first step of the algorithm we can be sure that[13] $k' = k$ and that we have a representation of the form $\text{sim}(f)|_V = Q(\ell_1, \ldots, \ell_k)$.

The running time of this step is dominated by the number of different subsets of at most $10R_4 \cdot \log^2(d)$ linearly independent linear functions over V. Clearly there are at most $|\mathbb{F}|^{10R_4 \cdot \log^2(d) + \log^2(n)}$ linear functions over V, and so we go over at most $|\mathbb{F}|^{(10R_4 \cdot \log^2(d) + \log^2(n)) \cdot (10R_4 \cdot \log^2(d))}$ such subsets. The brute force interpolation algorithm (as described in Appendix A) is slightly faster and so the total running time of this step is $|\mathbb{F}|^{O(\log^2(d) \cdot (\log^2(d) + \log^2(n)))} = \exp(\log|\mathbb{F}| \cdot \log^2(d) \cdot (\log^2(d) + \log^2(n)))$.

We now move to the next step, in which we show how to lift the representation of $\text{sim}(f)|_V$ to $\mathbb{F}^n$.

Step 3: Lifting from V to $\mathbb{F}^n$. Recall that in this step we look for linear functions $\{\ell_i^*\}_{i \in [k]}$ such that $\ell_i^*|_V = \ell_i$ and $\text{sim}(f) = Q(\ell_1^*, \ldots, \ell_k^*)$. The existence of such functions will follow from the next lemma.

LEMMA 24. *Let $\{L_i\}_{i \in [k]}$ be the linear functions in* (9) *and $\{\ell_i\}_{i \in [k]}$ be the linear functions found in step* 2 *of Algorithm* 3. *Assume that $\{\ell_i^*\}_{i \in [k]}$ satisfy $\ell_i^*|_V = \ell_i$ and $\text{span}\left(\{\ell_j^*\}_{j \in [k]}\right) = \text{span}\left(\{L_j\}_{j \in [k]}\right)$. Then $Q(\ell_1^*, \ldots, \ell_k^*) = \text{sim}(f)$.*

Proof. We shall use the same notations as in (9). By the assumption that $\text{span}\left(\{\ell_j^*\}_{j \in [k]}\right) = \text{span}\left(\{L_j\}_{j \in [k]}\right)$, we see that there is a polynomial $Q'(y_1, \ldots, y_k)$, of degree $\deg(Q') = \deg(P)$, such that $Q'(\ell_1^*, \ldots, \ell_k^*) = P(L_1, \ldots, L_k) = \text{sim}(f)$. By

[13] In the algorithm we do not check whether $k' = k$ (by going one dimension up, using Corollary 20) but we can just as well add such a check without much change to the running time. For the sake of simplicity we did not add this step to the algorithm.

considering the restriction to V, we get

$$Q(\ell_1,\ldots,\ell_k) = \mathrm{sim}(f)|_V = P(L_1,\ldots,L_k)|_V = Q'(\ell_1^*,\ldots,\ell_k^*)|_V$$

$$= Q'(\ell_1^*|_V,\ldots,\ell_k^*|_V) = Q'(\ell_1,\ldots,\ell_k).$$

As the functions $\{\ell_i\}_{i\in[k]}$ are linearly independent, and the degrees of Q and Q' are smaller than the size of the field, we get that $Q \equiv Q'$. In other words, $Q(y_1,\ldots,y_k)$ and $Q'(y_1,\ldots,y_k)$ are the same polynomial. In particular $Q(\ell_1^*,\ldots,\ell_k^*) = Q'(\ell_1^*,\ldots,\ell_k^*) = \mathrm{sim}(f)$, which is what we wanted to prove. □

With the help of the lemma, we now explain why step 3 indeed finds the required representation for $\mathrm{sim}(f)$. In that step we first construct $n-s$ linear spaces $\{V_i\}_{i\in[n-s]}$ of dimension $s+1$, such that $\cap_i V_i = V$ and $\mathrm{span}(\cup_i V_i) = \mathbb{F}^n$. For each such subspace we look for linear functions $\{\ell_j^{(i)}\}_{j\in[k]}$ such that $\ell_j^{(i)}|_V = \ell_j$, for every $j \in [k]$, and $Q(\ell_1^{(i)},\ldots,\ell_k^{(i)}) = \mathrm{sim}(f)|_{V_i}$. The reason that such $\ell_j^{(i)}$'s exist follows from Lemmas 23 and 24. Indeed, Lemma 23 implies that $\mathrm{span}(\{L_i|_V\}_{i\in[k]}) = \mathrm{span}(\{\ell_i\}_{i\in[k]})$. Hence, if we consider the $k \times n$ matrix A, whose rows correspond to the L_t's and columns correspond to the basis vectors $\{v_i\}_{i\in[n]}$, in which the (t,j)th position equals $L_t(v_j)$, then there exists an invertible linear transformation $T : \mathbb{F}^k \to \mathbb{F}^k$, such that for $1 \le j \le k$, the (t,j)th entry in $T \cdot A$ is equal to $\ell_t(v_j)$. It follows that if we consider the linear functions $\ell_t^{(i)}$ defined by $\ell_t^{(i)}(v_j) = (T\cdot A)_{t,j}$, for $j \in [k]\cup\{i\}$, then this gives the required $\ell_t^{(i)}$'s. In particular such $\ell_t^{(i)}$'s exist and we can find them by a simple interpolation. Therefore we are guaranteed that step 3(a) succeeds. Moreover, this argument also shows the uniqueness of the $\ell_j^{(i)}$'s (this follows from the fact that T is invertible). Hence, by the explanation above, it is clear that we can easily construct the matrix $T\cdot A$ by simply letting $(T\cdot A)_{t,j} = \ell_t^{(j)}(v_j)$ for $k < j$, and $(T\cdot A)_{t,j} = \ell_t(v_j)$ for $j \le k$. If we now pick ℓ_i^* to be the linear function defined by $\ell_i^*(v_j) = (T\cdot A)_{i,j}$, for $j \in [n]$, then by Lemma 24 we are guaranteed that $Q(\ell_1^*,\ldots,\ell_k^*) = \mathrm{sim}(f)$. This shows the correctness of step 3(b) (and the algorithmic way for constructing the ℓ_i^*'s). An interesting thing to notice is that we have no knowledge of neither the linear functions $\{L_i\}$ nor the linear transformation T, yet we are able to construct the matrix $T\cdot A$ using the (simple) conclusions of Lemmas 23 and 24.

The running time of step 3 is at most a polynomial factor times the running time of step 2. The main factor is repeating step 2 for each V_i. Then, finding the ℓ_i^*'s is quite simple and can be done in time polynomial in n, as described above. This completes the proof of Theorem 16. □

We note that as $k = O(\log^2(d))$ and $\deg(Q) \le d$, then $Q(\ell_1^*,\ldots,\ell_k^*)$ can be represented as a depth-3 circuit with quasi-polynomially many multiplication gates.

4.2. Interpolation for the high rank case. In this section we deal with the case that f has a circuit of rank higher than $10R_4\cdot\log^2(d)$. We shall use the notation $f = M_1+M_2$, where M_1 and M_2 are the two multiplication gates in the $\Sigma\Pi\Sigma(2)$ circuit for f. We note that from Corollary 7 and the assumption that $\mathrm{rank}(f) \ge 10R_4\cdot\log^2(d)$, we get that f has a unique $\Sigma\Pi\Sigma(2)$ circuit of degree d.

Before giving the algorithm itself we first describe its main steps. The algorithm follows the sketch of section 1.4. We first restrict the circuit to a low dimensional space V (where low means $\mathrm{poly}(\log d)$). Then we reconstruct $f|_V$, which is the main technical difficulty of the algorithm. Finally, we lift the circuit for $f|_V$ to a circuit for f. This step is done in a similar manner to the lifting process of Algorithm 2

(steps 3 and 4 there) and is based on the uniqueness of the circuit for f. We now say a few more words on the way to find a circuit for $f|_V$. The idea is to find a large set (of size $O(\log^2(d))$) of linearly independent linear functions $\{\ell_i\}$ that all divide, say, $M_1/\gcd(C)$. Such a set exists because of the assumption on the rank of f. Then we consider each of those linear functions and define the subspace $V_i = V|_{\ell_i=0}$ (i.e., the subspace of V on which ℓ_i vanishes).[14] Note that for every i it holds that $M_1|_{V_i} = 0$ and so by querying f on V_i we get oracle access to $M_2|_{V_i}$. Thus, by simple factoring we get all the circuits $M_2|_{V_i}$. The challenge now is to combine all of them to get the circuit $M_2|_V$. We later discuss this step in more detail. We now give the formal algorithm (Algorithm 4).

For convenience we divide the analysis of the three steps of the algorithm into three different subsections. We begin with the analysis of step 1.

4.2.1. Step 1: Interpolating on a low dimensional subspace I.

LEMMA 25. *With probability* $\geq 1 - |F|^{-\Omega(\log^2 n)}$, *over the choice of* V, *step* 1 *of Algorithm* 4, *when given oracle access to* f, *finds*[15] *a set of* $t = 100 \log d$ *linearly independent linear functions* $\{\ell_i\}_{i\in[t]}$ *such that* $\prod_{i=1}^{t} \ell_i | M_j$ *for some* $j \in \{1,2\}$, *but no* ℓ_i *divides* M_{3-j} *(the other multiplication gate). The running time of step* 1 *is* $\exp(\text{poly}(\log n, \log d, \log |\mathbb{F}|))$.

ALGORITHM 4 (general circuits of high rank).

1. **Interpolating on a low dimensional subspace I:** Pick a random subspace $V \subseteq \mathbb{F}^n$ of dimension $s \stackrel{\triangle}{=} 20R_4 \cdot \log^2(d) + \log^2(n)$. Consider the restriction $f|_V$. For every set of $t = 100\log(d)$ linearly independent linear functions $\{\ell_i\}_{i\in[t]}$ over V, check whether *for every* $i \in [t]$ the restriction of f to the (affine) subspace $V_i \stackrel{\triangle}{=} V \cap \{x : \ell_i(x) = 0\}$ is equal to a product of linear functions (by factoring).
2. **Interpolating on a low dimensional subspace II:** For each choice of $\{\ell_i\}_{i\in[t]}$, for which factoring was possible, merge (again, only if possible) the different factors into one multiplication gate (using Algorithm 5). For each multiplication gate, M, found check whether $f|_V - M$ is a product of linear functions (using the factoring algorithm from Theorem 8). If this is the case, then output the representation found for $f|_V$. If no such representation is found, then output "fail."
3. **Lifting from V to $\mathbb{F}^n$:** Lift the representation found over V to a representation over $\mathbb{F}^n$ (using the same method as in steps 3 and 4 of Algorithm 2). Namely, if $\{v_i\}_{i\in[s]}$ is a basis for V and $\{v_i\}_{i\in[n]}$ is a basis for $\mathbb{F}^n$ that extends the basis of V, then let $W_i = \text{span}(V \cup v_i)$ for $s < i \leq n$. Repeat steps 1 and 2 for each W_i to get the corresponding circuit C_i. If none of the steps failed for any of the W_i's, then combine the different C_i's to get C. Combining the circuits is done as follows. Let $\{z_i\}_{i\in[n]}$ be the dual basis to $\{v_i\}_{i\in[n]}$, i.e., $z_i(v_j) = \delta_{i,j}$. Each linear function in $C \cup \{C_i\}_{i=s+1}^{n}$ can be written as a linear combination of the z_i's and the constant function. Given a linear function ℓ in C, find the unique ℓ_i in C_i such that $\ell_i|_{z_i=0} = \ell$ (possibly after multiplying ℓ_i by a constant from $\mathbb{F}$). If no such ℓ_i exists, or if it is not unique (i.e., there is $\ell_i' \not\sim \ell_i$ with $\ell_i'|_{z_i=0} = \ell$), then output "fail." Otherwise, if $\ell = \ell_i + \alpha_i \cdot z_i$, then let $\ell' = \ell + \sum_{i=s+1}^{n} \alpha_i \cdot z_i$. Now, replace each linear function ℓ in C with the corresponding ℓ'. Denote the resulting circuit with C'. Output C'.

[14] This is actually an affine subspace, but for simplicity we assume that it is a linear space.

[15] Of course many other sets are found but only those sets interest us.

Proof. In the heart of the proof of the lemma is the fact that w.h.p., over the choice of V, we have that $\text{rank}(f)|_V \geq 10R_4 \cdot \log^2(d)$. We prove this fact in Lemma 26. It will be easy to show that when $\text{rank}(f|_V)$ is high, then the required set $\{\ell_i\}$ exists. Recall that $\gcd(C) = \text{g.c.d.}(M_1, M_2)$, and that

$$\text{sim}(C) = C/\gcd(C) = M_1' + M_2' \tag{10}$$

is a $\Sigma\Pi\Sigma(2)$ circuit. From Corollary 7 and the assumption that $\text{rank}(f) \geq 10R_4 \cdot \log^2(d)$ we get that $\text{sim}(C)$ is the unique $\Sigma\Pi\Sigma(2)$ circuit for $f/\gcd(C)$. Let $V \subseteq \mathbb{F}^n$ be a random subspace of dimension $s = 20R_4 \cdot \log^2(d) + \log^2(n)$. The following lemma shows that w.h.p. $\text{rank}(f|_V)$ is high (the intuition is the same as in Corollary 22).

LEMMA 26. $\Pr[\text{rank}(f|_V) \leq 10R_4 \cdot \log^2(d)] \leq |\mathbb{F}|^{-\Omega(\log^2 n)}$.

Proof of Lemma 26. To prove the claim we show that w.h.p. the rank of the circuit $M_1|_V + M_2|_V$ is not too small, and then, by Corollary 7, we get that this is actually the *unique* representation of $f|_V$ as a $\Sigma\Pi\Sigma(2)$ circuit. The lemma will follow from this fact.

LEMMA 27.

$$\Pr\left[\text{rank}\left(\frac{M_1|_V + M_2|_V}{\text{g.c.d.}(M_1|_V, M_2|_V)}\right) \leq 10R_4 \log^2(n)\right] \leq |\mathbb{F}|^{-\Omega(\log^2(n))}.$$

Proof of Lemma 27. In order to show that the rank of $M_1|_V + M_2|_V$ does not decrease by much, we first show that w.h.p., $\text{g.c.d.}(M_1|_V, M_2|_V) = \text{g.c.d.}(M_1, M_2)|_V$, and then using Lemma 21 we complete the proof.

LEMMA 28. $\Pr[\text{g.c.d.}(M_1|_V, M_2|_V) \neq \text{g.c.d.}(M_1, M_2)|_V] \leq d^2(|\mathbb{F}| + 1)/|\mathbb{F}|^s = |\mathbb{F}|^{-\Omega(\log^2(n))}$.

Proof of Lemma 28. It is not hard to see that $\text{g.c.d.}(M_1|_V, M_2|_V)$ is larger than $\text{g.c.d.}(M_1, M_2)|_V$ only if there are two linearly independent linear functions L_1, L_2, such that $L_i | M_i$, and such that $L_1|_V \sim L_2|_V$. The probability, over the choice of V, that this will happen is at most $(|\mathbb{F}| + 1)/|\mathbb{F}|^s$. As there are at most d^2 such pairs of linear functions, the overall probability that the degree of the g.c.d. increases is at most $d^2(|\mathbb{F}| + 1)/|\mathbb{F}|^s$. □

We continue with the proof of Lemma 27. We know that w.h.p., $\text{g.c.d.}(M_1|_V, M_2|_V) = \text{g.c.d.}(M_1, M_2)|_V$. Therefore, w.h.p.,

$$(M_1|_V + M_2|_V)/\text{g.c.d.}(M_1|_V, M_2|_V) = M_1'|_V + M_2'|_V$$

(as defined in (10)). By the assumption on $\text{rank}(f)$, we get that the rank of the linear functions in $M_1' + M_2'$ is at least $10R_4 \cdot \log^2(d)$. The result now follows by using Lemma 21 with the parameters[16] $s = 20R_4 \cdot \log^2(d) + \log^2(n)$ and $t = 10R_4 \cdot \log^2(d)$. The lemma implies that

$$\Pr\left[\text{rank}\left(M_1'|_V + M_2'|_V\right) \leq 10R_4 \cdot \log^2(d)\right] \leq |\mathbb{F}|^{-\log^2(n)}.$$

This completes the proof of Lemma 27. □

We now continue with the proof of Lemma 26. Notice that whenever $\text{rank}(M_1|_V + M_2|_V) > 10R_4 \cdot \log^2(d)$ we get by Corollary 7 that $M_1|_V + M_2|_V$ is the unique $\Sigma\Pi\Sigma(2)$ circuit for $f|_V$ of degree $\leq d$. In this case we of course get that

[16] Although the rank is at least $10R_4 \cdot \log^2(d)$, we consider only a subset of the linear functions of dimension exactly $10R_4 \cdot \log^2(d)$.

$\operatorname{rank}(f) > 10R_4 \cdot \log^2(d)$. As $\Pr\left[\operatorname{rank}\left(M_1'|_V + M_2'|_V\right) \leq 10R_4 \cdot \log^2(d)\right] \leq |\mathbb{F}|^{-\log^2(n)}$ we get that $\Pr\left[\operatorname{rank}(f) \leq 10R_4 \cdot \log^2(d)\right] \leq |\mathbb{F}|^{-\log^2(n)}$. This completes the proof of Lemma 26. □

Now that we established that (w.h.p.) $\operatorname{rank}(f|_V)$ is high we continue with the proof of Lemma 25. Assuming that $\operatorname{rank}(f|_V) \geq 10R_4 \cdot \log^2(d)$ we get that, w.l.o.g., there are at least $5R_4 \cdot \log^2(d)$ linearly independent linear functions dividing $M_1|_V$ and not $M_2|_V$. In particular there exist $t = 100 \log d$ linearly independent linear functions $\{\ell_i\}_{i\in[t]}$ such that $\prod_{i=1}^t \ell_i$ divides M_1 but no ℓ_i divides M_2.

We bound the running time in the obvious manner. During the step, we simply have to run over all sets of $t = 100 \log d$ linear functions and, for each such set $\{\ell_i\}$, check whether the functions are linearly independent and whether the restriction of f to $V_i = V \cap \{x : \ell_i(x) = 0\}$ completely factorizes to a product of linear functions. Clearly the running time is equal (up to a polynomial factor in $n, |\mathbb{F}|$) to the number of such sets. As the dimension of V is $s = 20R_4 \log^2(d) + \log^2(n)$, the number of sets that we have to consider is $|\mathbb{F}|^{100\log(d)\cdot s}$, which is quasi polynomial in n, d, and $|\mathbb{F}|$. This completes the proof of Lemma 25. □

4.2.2. Step 2: Interpolating on a low dimensional subspace II. As we showed in Lemma 25, we can assume that one of the sets of linearly independent linear functions $\{\ell_i\}_{i\in[t]}$ found in step 1 satisfies that, w.l.o.g., $\prod_{i=1}^t \ell_i$ divides $M_1|_V$ but no ℓ_i divides $M_2|_V$. When studying step 2 we will focus on this set to show that one of the output circuits is indeed $C|_V$.

Before giving the analysis of step 2 we first explain how to computes $M_2|_V$ from the different $M_2|_{V_i}$'s (given that the ℓ_i's satisfy the property above). The idea behind the reconstruction algorithm is the following. We have to reconstruct the set of linear functions appearing in $M_2|_V$. Since a linear function can divide M_2 several times we can speak about the *multiset* of linear functions dividing $M_2|_V$, which we denote with $\mathcal{L}$. As we have oracle access to each V_i, using the factoring algorithm of Theorem 8, we can assume that we have as input the t multisets $\{\mathcal{L}_i\}_{i=1}^t$, where $\mathcal{L}_i$ is the multiset composed of the linear functions dividing $M_2|_{V_i}$. What the algorithm actually does is merge the different $\mathcal{L}_i$'s to get $\mathcal{L}$. To simplify notations we shall make several assumptions on V and $\{\ell_i\}_{i\in[t]}$ that will be w.l.o.g. and will ease the reading of the algorithm.[17] First, we assume that

$$V = \{(a_1, \ldots, a_s, 0, \ldots, 0) \ : \ \forall i \ a_i \in \mathbb{F}\}.$$

Hence, every linear function defined on V is of the form

$$L(v) = \sum_{i=1}^{s} \alpha_i a_i + \alpha_0$$

for the vector $v = (a_1, \ldots, a_s, 0, \ldots, 0)$. We shall also assume that

$$\ell_i(v) = \ell_i\left((a_1, \ldots, a_s, 0, \ldots, 0)\right) = a_i.$$

In other words, we assume that ℓ_i is a projection on the ith coordinate. Notice that as we can apply linear transformations to $\mathbb{F}^n$, this can be assumed w.l.o.g.[18] Thus, by

[17] If we were to describe the most general case, then we had to use the dual basis as in step 3.

[18] Actually, ℓ_i can be an affine function and so should be of the form $a_i + \nu_i$ for some constant ν_i. However, this will not change the algorithm and will only add unnecessary complications to the presentation.

our assumption we get that $V_i = \{(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_s, \ldots, 0) \ : \ \forall j \ a_j \in \mathbb{F}\}$. Thus, the multiset $\mathcal{L}_i$ is the multiset obtained after deleting the ith variable from all the linear functions in the multiset $\mathcal{L}$. Hence, from our assumptions (that can be made w.l.o.g.) we see that our goal is to reconstruct a multiset of linear functions $\mathcal{L}$, from its t different projections $\{\mathcal{L}_\rangle\}_{i\in[t]}$ on $s-1 \geq t-1$ variables. Algorithm 5 solves exactly this task. In fact the algorithm assumes that we have only t variables (and not s) and is still able to solve the problem. Hence, using Algorithm 5 we are able to reconstruct the set $\mathcal{L}$ and from it get $M_2|_V$.

To explain how Algorithm 5 works we shall use the following terminology. We say that a linear function T has multiplicity k in a multiset $\mathcal{T}$ if there are exactly k linear functions in $\mathcal{T}$ that are linearly dependent on T (that is, that are equal to some multiple of T). The idea behind the algorithm is the following. First, it finds a linear function $L \in \mathcal{L}_1$ that has the following property: assume that L appears k times in $\mathcal{L}_1$. Then there exists $i \in \{2, \ldots, t\}$ such that $L|_{x_i=0}$ also has multiplicity k in $\mathcal{L}_1|_{x_i=0}$. In other words, L has the property that there is no other linear function $L' \in \mathcal{L}_1$ such that L and L' together span x_i. Note that it is not clear why such a linear function L should exist; however, using known lower bounds on *locally decodable codes* (see Appendix B) we can prove its existence. Now, given such L, we note that by considering $\mathcal{L}_i$ there are exactly k linear functions $\{L_i\}_{i=1}^k$ that, after erasing their first coordinate, are equal to (a constant multiple of) $L|_{x_i=0}$. Thus, we are assured that there are exactly k linear function $\{\ell_i\}_{i=1}^k$ in $\mathcal{L}$, such that $\ell_i|_{x_1=0} = L$ and $\ell_i|_{x_i=0} = L_i$. In particular, to get ℓ_i we simply add the first coordinate of L_i to L for each $i \in [k]$ (after making the appropriate normalization). Now, we can remove the k projections of $\{\ell_i\}_{i=1}^k$ from the different $\mathcal{L}_i$'s and repeat the algorithm until no linear function is left. From this description it is clear that the main technical point is proving that such a linear function $L \in \mathcal{L}_1$ exists. We now give the formal description of Algorithm 5 and then analyze it. After that it will be clear how to analyze step 2.

ALGORITHM 5 (gluing the different $M_2|_{V_i}$'s together).

Input: Multisets of linear functions $\mathcal{L}_i = \mathcal{L}|_{x_i=0}$, for $i = 1, \ldots, t$, containing $m < \exp((t-1)/60 - 1)$ linear functions each.

Output: a multiset $\tilde{\mathcal{L}}$.

Set $\tilde{\mathcal{L}} \leftarrow \emptyset$.

1. Find a linear function $L \in \mathcal{L}_1$ and an index $1 < i \leq t$ such that the following holds: denote by k the multiplicity of L in $\mathcal{L}_1$. Then the number of $L' \in \mathcal{L}_i$ such that $L'|_{x_1=0} \sim L|_{x_i=0}$ is exactly k (and in particular, $L|_{x_i=0}$ is not a constant).
2. Denote by $\{L_j\}_{j=1}^k \subset \mathcal{L}_i$ those k linear functions. Let $\{c_j\}_{j=1}^k$ be such that $L|_{x_i=0} = c_j \cdot L_j|_{x_1=0}$. Let $a_{j,1}$ be the coefficient of x_1 in the linear function $c_j \cdot L_j$. Let $\ell_j \stackrel{\Delta}{=} L + a_{j,1} \cdot x_1$, for $j = 1, \ldots, k$.
3. Set $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{\ell_j\}_{j\in[k]}$.
4. For each $l \in [t]$, set $\mathcal{L}_l \leftarrow \mathcal{L}_l \setminus \{\ell_j|_{x_l=0}\}_{j=1}^k$ (more accurately, remove from $\mathcal{L}_l$ k linear functions $\ell'_1, \ldots, \ell'_k$ such that $\ell'_j = \ell_j|_{x_l=0}$).
5. If the $\mathcal{L}_i$'s are empty, then return $\tilde{\mathcal{L}}$; otherwise go to step 1.

The following theorem proves correctness and gives the analysis of Algorithm 5.

THEOREM 29. *Let $\mathcal{L} = \{L_i\}_{i=1}^m$ be a multiset of linear functions in $t > 60 \log m + 61$ variables; i.e., a linear function may appear more than once in $\mathcal{L}$. Denote by $\mathcal{L}_i \stackrel{\Delta}{=} \mathcal{L}|_{x_i=0}$ the multiset resulting from $\mathcal{L}$ after removing the ith variable from all the linear functions in it. Then, given the multisets $\{\mathcal{L}_i\}_{i\in t}$, Algorithm 5 reconstructs*

a multiset $\tilde{\mathcal{L}}$ such that $\prod_{\ell\in\tilde{\mathcal{L}}}\ell = c\cdot\prod_{\ell'\in\mathcal{L}}\ell'$, for some nonzero constant c, in time polynomial in m.

Proof. First, we show that if a linear function L was found in step 1, and its multiplicity in $\mathcal{L}_1$ is k, then there are exactly k linear functions $\{\tilde{\ell}_j\}_{j\in[k]}$ in $\mathcal{L}$ that satisfy $\tilde{\ell}_j|_{x_1=0}\sim L$ for each $j\in[k]$. Moreover, these linear functions are equal, up to a multiplicative factor, to the linear functions $\{\ell_j\}_{j\in[k]}$ that were found in step 2 of the algorithm. Namely, possibly after reordering it holds that $\tilde{\ell}_j\sim\ell_j$ for $j\in[k]$. Indeed, let L and i be the linear function and index that were found in step 1 (of Algorithm 5). Denote by k the multiplicity of L in $\mathcal{L}_1$ and let $\{\tilde{\ell}_j\}_{j\in[k]}\subset\mathcal{L}$ be such that $\tilde{\ell}_j|_{x_1=0}\sim L$. From the definition of i these are exactly the functions in $\mathcal{L}$ that satisfy that $\tilde{\ell}_j|_{x_1=0,x_i=0}\sim L|_{x_i=0}$. We therefore have the following. Let a_j be such that $a_j\cdot\tilde{\ell}_j|_{x_1=0}=L$. Let c_j and L_j (for $j\in[k]$) be as in step 2. It must be the case that (possible after reordering) $a_j\cdot\tilde{\ell}_j|_{x_i=0}=c_j\cdot L_j$. In particular if we define $\ell_j=\alpha_{j,1}\cdot x_1+L$, where $\alpha_{j,1}$ is the coefficient of x_1 in $c_j\cdot L_j$, then it must be the case that $\tilde{\ell}_j\sim\ell_j$, as required (note that a different way of defining ℓ_j is by letting $\ell_j=c_j\cdot L_j+\alpha_i\cdot x_i$, where α_i is the coefficient of x_i in L). In particular this shows that step 3 adds to $\tilde{\mathcal{L}}$ linear functions that are equal to the $\tilde{\ell}_j$'s up to a constant factor and step 4 removes the "images" of those functions from the different $\mathcal{L}_i$'s. Thus, if we can always find such L and i, then we are guaranteed that the output will be a set $\tilde{\mathcal{L}}$ for which there is a 1-1 correspondence between the linear functions in $\mathcal{L}$ and those in $\tilde{\mathcal{L}}$ that matches functions that are equal up to a (nonzero) constant factor. This completes the first part of our proof.

We now prove that if $m<\exp((t-1)/60-1)$, then such L and i exist. Assume for a contradiction that no such L and i exist. Then it must be the case that for every $L\in\mathcal{L}_1$ and i there is $L'\in\mathcal{L}_1$ such that $x_i\in\operatorname{span}(L,L')$. In other words, if we consider the mapping $E:\mathbb{F}^{t-1}\to\mathbb{F}^m$ that maps $(x_2,\ldots,x_t)$ to $\{L(x_2,\ldots,x_t)\}_{L\in\mathcal{L}_1}$, then we have that for every $j\in\{2,\ldots,t\}$ there are, say, at least $m/3$ disjoint pairs of linear functions $L,L'\in\mathcal{L}_1$ such that $x_j\in\operatorname{span}(L,L')$. In other words, we can reconstruct x_j by looking at one of the $m/3$ disjoint pairs of indices in the mapping E. A mapping that satisfies this property is called a 2-query locally decodable code, and it is known (Theorem 33) that in such a code it must be the case that $m\geq\exp((t-1)/60-1)$. However, we assumed that $m<\exp((t-1)/60-1)$, and so we reached a contradiction. Hence a linear function L and an index i can be found. For completeness we discuss locally decodable codes in Appendix B.

The claim regarding the running time of Algorithm 5 is clear. This completes the proof of Theorem 29. ∎

To conclude, from Theorem 29 and the discussion at the beginning of section 4.2.2 we see that in step 2 of Algorithm 4, for one of the sets $\{\ell_i\}_{i\in[t]}$ we manage to reconstruct the multiplication gate $M_2|_V$. To be more accurate, note that for the multiset $\tilde{\mathcal{L}}$ computed in Algorithm 5 we may have that $\prod_{\ell\in\tilde{\mathcal{L}}}\ell=c\cdot\prod_{\ell'\in\mathcal{L}}\ell'$ for some nonzero constant c. However, c can be easily found by comparing $\prod_{\ell\in\tilde{\mathcal{L}}}\ell|_{x_1=0}$ to $M_2|_{x_1=0}$. Therefore, we can multiply some $\ell\in\tilde{\mathcal{L}}$ by c to get that $\prod_{\ell\in\tilde{\mathcal{L}}}\ell=\prod_{\ell'\in\mathcal{L}}\ell'=M_2|_V$.

Now, given $M_2|_V$ we can use the oracle to f to factor $f|_V-M_2|_V$ and get $M_1|_V$. In particular we compute the circuit $C|_V=M_1|_V+M_2|_V$ in step 2. We now notice that if V is such that $\operatorname{rank}(f|_V)\geq R_4\log^2(d)$ (which happens w.h.p. according to Lemma 26), then by Corollary 6 this is the only possible representation for $f|_V$ (as a $\Sigma\Pi\Sigma(2)$ circuit) and so the circuit computed at the end of this step is indeed

$M_1|_V + M_2|_V$. The running time of step 2 is equal to the number of different sets $\{\ell_i\}_{i\in[t]}$ considered times the running times of Algorithm 5 and the factoring algorithm of Theorem 8. Thus, we just proved the following theorem.

THEOREM 30. *Step 2 of Algorithm 4 runs in time* $\exp(\mathrm{poly}(\log n, \log d, \log|\mathbb{F}|))$, *and if V is such that* $\mathrm{rank}(f|_V) \geq R_4 \log^2(d)$, *then the circuit computed at the end of the step is* $C|_V = M_1|_V + M_2|_V$.

4.2.3. Step 3: Lifting. We first give the intuition behind the way that the algorithm combines the different circuits for $f|_{W_i}$. For convenience let us assume w.l.o.g., again, that

$$V = \{v = (a_1, \ldots, a_s, 0, \ldots, 0) \mid \forall i\ a_i \in \mathbb{F}\}.$$

Also assume w.l.o.g. that for $1 \leq i \leq n-s$ we have that

$$W_i = \{w = (a_1, \ldots, a_s, 0, \ldots, 0, a_{s+i}, 0, \ldots, 0) \mid \forall j\ a_j \in \mathbb{F}\}.$$

Note that as we are working with the v_i's and their dual basis $\{z_i\}$, we can make this assumption. Let us assume that $\mathrm{rank}(f|_V) \geq R_4 \cdot \log^2(d)$. Theorem 30 now guarantees that the algorithm computes correctly the circuits $C_i = M_1|_{W_i} + M_2|_{W_i}$ for $s < i \leq n$. Assume that our original $\Sigma\Pi\Sigma(2)$ for f has the following form:

$$f(\bar{x}) = \prod_{i=1}^{d} L_i^{(1)}(\bar{x}) + \prod_{i=1}^{d} L_i^{(2)}(\bar{x}),$$

where $M_1 = \prod_{i=1}^{d} L_i^{(1)}(\bar{x})$ and $M_2 = \prod_{i=1}^{d} L_i^{(2)}(\bar{x})$. Denote

$$L_i^{(k)} = \sum_{j=1}^{n} \alpha_{i,j}^{(k)} x_j + \alpha_{i,0}^{(k)}$$

for $k = 1, 2$. If $\mathrm{rank}(f|_V) > R_4 \cdot \log^2(d)$, then by Theorem 30 we get that

$$f|_{W_i} = \prod_{k=1}^{d} L_k^{(1)}(\bar{x})|_{W_i} + \prod_{k=1}^{d} L_k^{(2)}(\bar{x})|_{W_i}$$

$$= \prod_{k=1}^{d} \left(\sum_{j=1}^{s} \alpha_{k,j}^{(1)} x_j + \alpha_{k,s+i}^{(1)} x_{s+i} + \alpha_{k,0}^{(1)} \right) + \prod_{k=1}^{d} \left(\sum_{j=1}^{s} \alpha_{k,j}^{(2)} x_j + \alpha_{k,s+i}^{(2)} x_{s+i} + \alpha_{k,0}^{(2)} \right).$$

Therefore all that we have to do is to find all the linear functions of the form $\sum_{j=1}^{s} \alpha_{k,j}^{(1)} x_j + \alpha_{k,s+i}^{(1)} x_{s+i} + \alpha_{k,0}^{(1)}$, for $i \in [n-s]$, and glue them together to get all the linear functions $\{\sum_{j=1}^{n} \alpha_{k,j}^{(1)} x_j\}_k$ (similarly with $\sum_{j=1}^{s} \alpha_{k,j}^{(2)} x_j + \alpha_{k,s+i}^{(2)} x_{s+i} + \alpha_{k,0}^{(2)}$ and $\{\sum_{j=1}^{n} \alpha_{k,j}^{(2)} x_j\}_k$). To do so we must be sure that there are no two linearly independent linear functions in f that their restrictions to V are linearly dependent. Note that if such a pair exists, e.g., if $\ell|_V \sim \ell'|_V$, where ℓ and ℓ' are linearly independent, then for some W_i and constants $\alpha, \alpha', \alpha''$ such that $\alpha \neq \alpha''$, the circuit C_i will contain two different functions $\tilde{\ell} = \ell + \alpha \cdot x_i$ and $\tilde{\ell}' = \ell' + \alpha' \cdot x_i \sim \ell + \alpha'' \cdot x_i$, and we will not know which of them to attach to ℓ and which to attach to ℓ' when combining appropriate linear functions from the different C_i's.

Now, if no such pair exists, then it is very easy to combine the different circuits C_i into one circuit C' for f: for every linear function ℓ in C, that does not divide both its multiplication gates, there is a unique linear function ℓ_i in C_i such that their restrictions to $x_i = 0$ are linearly dependent (note that by Lemma 28 we can assume that g.c.d.$(M_1|_V, M_2|_V) =$ g.c.d.$(M_1, M_2)|_V$. Hence, if ℓ divides both multiplication gates in C, then there is a unique ℓ_i that divides both multiplication gates in C_i, and the continuation is the same for this case as well). Therefore there is only one way of generating a linear function L such that $\ell'|_V \sim \ell$ and $\ell'|_{x_i=0} \sim \ell_i$ for every i. This explanation shows that in step 3 we combine the different circuits in the only possible way. Moreover, it is clear that combining the C_i's into a single circuit can be done efficiently. It remains only to justify the assumption that there are no two linearly independent linear functions in f that their restrictions to V are linearly dependent.

To see that, we notice that as we picked V to be a random subspace of dimension $\Omega(\log^2(d))$, Lemma 21 assured us that with probability greater than $1-\exp(-\log^2(n))$ no two linearly independent linear functions from C become linearly dependent when restricted to V. Thus, w.h.p. we can complete this step. Notice that by the above explanation it is also very easy to verify that indeed, no two linearly independent linear functions from the circuit for f became dependent on V. The following theorem summarizes what we have shown in this section.

THEOREM 31. *Step* 3 *of Algorithm* 4 *runs in time* $\mathrm{poly}(n, d, \log|\mathbb{F}|))$ *and w.h.p. over the choice of* V *outputs the unique* $\Sigma\Pi\Sigma(2)$ *circuit for* f.

Thus, Algorithm 4 outputs w.h.p. over the choice of V the unique $\Sigma\Pi\Sigma(2)$ for f. We now show how to combine Algorithms 3 and 4 to get Theorem 2.

4.3. Completing the proof of Theorem 2. We note that if we know $\mathrm{rank}(f)$, then the algorithms described in sections 4.1 and 4.2 give the required result. As we don't know what the rank is, we run the high rank algorithm for every rank between $10R_4 \cdot \log^2(d)$ and d. Once the algorithm outputs a $\Sigma\Pi\Sigma(2)$ circuit we verify that its rank is indeed larger than $10R_4 \cdot \log^2(d)$. If this is the case, then we stop and output that circuit. Theorem 31 guarantees that w.h.p. we have found the unique $\Sigma\Pi\Sigma(2)$ circuit C' that computes f. If none of the executions of the high rank algorithm resulted in a $\Sigma\Pi\Sigma(2)$ circuit, then we run the low rank algorithm for each rank between 1 and $10R_4 \cdot \log^2(d)$.

To analyze the error probability of the algorithm we note that the possible errors are in the choice of V and in the factoring algorithm. Indeed, when we are at the "correct" rank and V is such that the rank of $f|_V$ is the same as $\mathrm{rank}(f)$ and no two linearly independent linear functions from C become linearly dependent when restricted to V, then we are assured that we compute a correct representation for f. Recall that V is "bad" for us with probability $1/p(n, d)$, where p is some quasi-polynomial function of n and d. In addition we can repeat the factoring algorithm of Theorem 8 polynomially many times to reduce the error probability. Thus the overall error is quasi-polynomially small (and can be further reduced to be exponentially small by repeating the algorithm several times). This completes the proof of the theorem.

Appendix A. Brute force interpolation. In this section we show how to interpolate a polynomial that can be written as a polynomial in a few linear functions. In fact we need to consider a slightly more complicated scenario as described in the following lemma.

LEMMA 32. *Let*[19] $h(x_1,\ldots,x_s) = \mathrm{Lin}(h)\cdot Q(\ell_1,\ldots,\ell_k)$ *be a polynomial, where* Q *is a polynomial and* $\{L_i\}_{i\in[k]}$ *are linear functions. Let* $d = \deg(h) < |\mathbb{F}|$. *Then, there is a deterministic algorithm that, given oracle access to* h, $\mathrm{Lin}(h)$, d, *and the set of linear functions* $\{\ell_i\}_{i\in[k]}$, *finds* Q. *The running time of the algorithm is* $\mathrm{poly}(|\mathbb{F}|^s, d^k)$.

Proof. Consider a generic degree d polynomial Q' in the ℓ_i's. Such a polynomial has $< (d+1)^k$ monomials of the form $\prod_{i=1}^k {\ell_i}^{e_i}$, such that $\sum_{i=1}^k e_i \le d$. Thus we have a number of unknown coefficients which we can find by a simple interpolation procedure that we now describe: let $\ell_{k+1},\ldots,\ell_s$ be linear functions such that[20] $\ell_1,\ldots,\ell_s$ form a basis to the space of linear functions over $\mathbb{F}^s$. Let

$$Q'(y_1,\ldots,y_k) = \sum_{\{\bar{e}=(e_1,\ldots,e_k):\sum e_i\le d\}} c_{\bar{e}}\cdot\prod {y_i}^{e_i}. \tag{11}$$

Next we represent $\mathrm{Lin}(h)$ as a polynomial in the ℓ_i's. Let $P_{\mathrm{Lin}}(y_1,\ldots,y_s)$ be a polynomial satisfying

$$\mathrm{Lin}(h) = P_{\mathrm{Lin}}(\ell_1,\ldots,\ell_s) = \sum_M \alpha_M\cdot M(\ell_1,\ldots,\ell_s), \tag{12}$$

where the sum is over all monomials M of degree at most d. Note that as we have oracle access to $\mathrm{Lin}(h)$, it is easy to find the α_M's (by the usual interpolation process). We get that h can be written as

$$h = P_{\mathrm{Lin}}(\ell_1,\ldots,\ell_s)\cdot Q'(\ell_1,\ldots,\ell_k) = \sum_M \gamma_M\cdot M(\ell_1,\ldots,\ell_s), \tag{13}$$

where each coefficient γ_M is a linear function in the $c_{\bar{e}}$'s. By querying h on all the points in $\mathbb{F}^s$ and using the standard interpolation method, we can easily find all the coefficients γ_M, and from them we can get the coefficients $c_{\bar{e}}$'s by solving a system of linear equations. We note that once we know the γ_M's, there is a unique solution to the $c_{\bar{e}}$'s (in case that such a solution exists). The reason is that if two different solutions exist, then they give rise to two polynomials $P_{\mathrm{Lin}}\cdot Q_1$ and $P_{\mathrm{Lin}}\cdot Q_2$ that are equal over $\mathbb{F}^s$. However, the degree of each of the polynomials is at most d, which is smaller than the size of the field that we are working with, and so the two polynomials must be the same. In particular we have that $Q' = Q$. ∎

Appendix B. Locally decodable codes. In this section we define the notion of locally decodable codes and state the result that we need. The reader interested in a more complete background is referred to [KT00, GKST06]. We start with the definition of locally decodable codes. We fix in advance some of the parameters to constants in order to simplify the definition. Let $E:\mathbb{F}^t\to\mathbb{F}^n$ be a linear map. We say that E is a 2-query locally decodable code if there exists a probabilistic oracle

[19] In section 4.1.2 we needed to interpolate $\mathrm{sim}(f)|_V$ for an s-dimensional space V, so we assume that h is a polynomial in s variables.

[20] We assume w.l.o.g. that $\ell_1,\ldots,\ell_k$ are linearly independent.

machine A such that

- A makes at most two queries (nonadaptively) to the oracle.
- for every $x \in \mathbb{F}^t$, for every $y \in \mathbb{F}^n$ with $\Delta(y, E(x)) < n/10$, and for every $i \in [t]$, we have $\Pr[A^y(i) = x_i] \geq 2/3$, where the probability is taken over the internal coin tosses of A, and $\Delta(\cdot, \cdot)$ is the Hamming distance function.

The following theorem of [DS06] gives a lower bound on the length of such locally decodable codes over arbitrary fields.

THEOREM 33 (Theorem 1.2 of [DS06]). *Let $\mathbb{F}$ be a field, and let $E : \mathbb{F}^t \to \mathbb{F}^n$ be a linear 2-query locally decodable code, as in the definition above; then $n \geq 2^{\frac{t}{60}-1}$.*

We note that Goldreich et al. [GKST06] were the first to prove a lower bound for linear locally decodable codes with two queries; however, for large fields their result is not optimal. In particular, [GKST06] proved a lower bound of the form $n \geq 2^{\Omega(t)-\log|\mathbb{F}|}$ on the length of such codes over $\mathbb{F}$, which deteriorates with the field size.

Acknowledgments. I would like to thank Amir Yehudayoff and Nader Bshouty for many helpful discussions, Erich Kaltofen and Joachim von zur Gathen for answering my questions regarding factorization algorithms, and Avi Wigderson and the anonymous reviewers for helpful comments. Thanks to Oded Goldreich and Ran Raz for their encouragement. The seed for this work was planted in the BIRS workshop "Recent Advances in Computation Complexity." I thank the organizers for inviting me and thank BIRS for hosting the workshop.

REFERENCES

[BB98] D. BSHOUTY AND N. H. BSHOUTY, *On interpolating arithmetic read-once formulas with exponentiation*, J. Comput. System Sci., 56 (1998), pp. 112–124.

[BBB+00] A. BEIMEL, F. BERGADANO, N. H. BSHOUTY, E. KUSHILEVITZ, AND S. VARRICCHIO, *Learning functions represented as multiplicity automata*, J. ACM, 47 (2000), pp. 506–530.

[BBTV97] F. BERGADANO, N. H. BSHOUTY, C. TAMON, AND S. VARRICCHIO, *On learning branching programs and small depth circuits*, in Proceedings of the 3rd European Conference on Computational Learning Theory, Lecture Notes in Comput. Sci. 1208, Springer, Berlin, 1997, pp. 150–161.

[BBV96] F. BERGADANO, N. H. BSHOUTY, AND S. VARRICCHIO, *Learning multivariate polynomials from substitution and equivalence queries*, Electron. Colloq. Comput. Complex 1996, report TR96-008. Available online at http://eccc.hpi-web.de/eccc-reports/1996/TR96-008/index.html.

[BC98] N. H. BSHOUTY AND R. CLEVE, *Interpolating arithmetic read-once formulas in parallel*, SIAM J. Comput., 27 (1998), pp. 401–413.

[BHH95] N. H. BSHOUTY, T. R. HANCOCK, AND L. HELLERSTEIN, *Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates*, J. Comput. System Sci., 50 (1995), pp. 521–542.

[BOT88] M. BEN-OR AND P. TIWARI, *A deterministic algorithm for sparse multivariate polynominal interpolation*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), Chicago, IL, 1988, pp. 301–309.

[DS06] Z. DVIR AND A. SHPILKA, *Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits*, SIAM J. Comput., 36 (2007), pp. 1404–1434.

[FK06] L. FORTNOW AND A. R. KLIVANS, *Efficient learning algorithms yield circuit lower bounds*, in Proceeding of the 19th Annual Conference on Learning Theory (COLT), Pittsburgh, PA, 2006, pp. 350–363.

[GKS94] D. GRIGORIEV, M. KARPINSKI, AND M. F. SINGER, *Computational complexity of sparse rational interpolation*, SIAM J. Comput., 23 (1994), pp. 1–11.

[GKST06] O. GOLDREICH, H. J. KARLOFF, L. J. SCHULMAN, AND L. TREVISAN, *Lower bounds for linear locally decodable codes and private information retrieval*, Comput. Complex., 15 (2006), pp. 263–296.

[HH91] T. R. Hancock and L. Hellerstein, *Learning read-once formulas over fields and extended bases*, in Proceedings of the 4th Annual Conference on Learning Theory (COLT), Amsterdam, 1991, p. 326–336.

[Kal85] E. Kaltofen, *Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization*, SIAM J. Comput., 14 (1985), pp. 469–489.

[Kal95] E. Kaltofen, *Effective Noether irreducibility forms and applications*, J. Comput. System Sci., 50 (1995), pp. 274–295.

[Kha95] M. Kharitonov, *Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution*, J. Comput. System Sci., 50 (1995), pp. 600–610.

[KL01] M. Krause and S. Lucks, *Pseudorandom functions in TC^0 and cryptographic limitations to proving lower bounds*, Comput. Complex., 10 (2001), pp. 297–313.

[KS96] M. Karpinski and I. Shparlinski, *On some approximation problems concerning sparse polynomials over finite fields*, Theoret. Comput. Sci., 157 (1996), pp. 259–266.

[KS01] A. Klivans and D. Spielman, *Randomness efficient identity testing of multivariate polynomials*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), Crete, 2001, pp. 216–223.

[KS06] A. Klivans and A. Shpilka, *Learning restricted models of arithmetic circuits*, Theory Comput., 2 (2006), pp. 185–206.

[KS08] Z. Karnin and A. Shpilka, *Reconstruction of Generalized Depth-3 Arithmetic Circuits with Bounded Top Fan-In*, manuscript.

[KT90] E. Kaltofen and B. M. Trager, *Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators*, J. Symbolic Comput., 9 (1990), pp. 301–320.

[KT00] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error-correcting codes*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM Press, New York, 2000, pp. 80–86.

[KV94] M. J. Kearns and L. G. Valiant, *Cryptographic limitations on learning Boolean formulae and finite automata*, J. ACM, 41 (1994), pp. 67–95.

[Man95] Y. Mansour, *Randomized interpolation and approximation of sparse polynomials*, SIAM J. Comput., 24 (1995), pp. 357–368.

[NR04] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, J. ACM, 51 (2004), pp. 231–262.

[OGM86] S. Goldwasser, O. Goldreich, and S. Micali, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.

[RR97] A. A. Razboeov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.

[Shp07] A. Shpilka, *Interpolation of depth-3 arithmetic circuits with two multiplication gates*, in Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA, 2007, pp. 284–293.

[SS96] R. E. Schapire and L. M. Sellie, *Learning sparse multivariate polynomials over a field with queries and counterexamples*, J. Comput. System Sci., 52 (1996), pp. 201–213.

[SV08] A. Shpilka and I. Volkovich, *Read-once polynomial identity testing*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC), Victoria, British Columbia, 2008, pp. 507–516.

[SW01] A. Shpilka and A. Wigderson, *Depth-3 arithmetic circuits over fields of characteristic zero*, Computat. Complex., 10 (2001), pp. 1–27.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2162–2178

BREAKING A TIME-AND-SPACE BARRIER IN CONSTRUCTING FULL-TEXT INDICES*

WING-KAI HON†, KUNIHIKO SADAKANE‡, AND WING-KIN SUNG§

Abstract. Suffix trees and suffix arrays are the most prominent full-text indices, and their construction algorithms are well studied. In the literature, the fastest algorithm runs in $O(n)$ time, while it requires $O(n \log n)$-bit working space, where n denotes the length of the text. On the other hand, the most space-efficient algorithm requires $O(n)$-bit working space while it runs in $O(n \log n)$ time. It was open whether these indices can be constructed in both $o(n \log n)$ time and $o(n \log n)$-bit working space. This paper breaks the above time-and-space barrier under the unit-cost word RAM. We give an algorithm for constructing the suffix array, which takes $O(n)$ time and $O(n)$-bit working space, for texts with constant-size alphabets. Note that both the time and the space bounds are optimal. For constructing the suffix tree, our algorithm requires $O(n \log^{\epsilon} n)$ time and $O(n)$-bit working space for any $0 < \epsilon < 1$. Apart from that, our algorithm can also be adopted to build other existing full-text indices, such as compressed suffix tree, compressed suffix arrays, and FM-index. We also study the general case where the size of the alphabet Σ is not constant. Our algorithm can construct a suffix array and a suffix tree using optimal $O(n \log |\Sigma|)$-bit working space while running in $O(n \log \log |\Sigma|)$ time and $O(n(\log^{\epsilon} n + \log |\Sigma|))$ time, respectively. These are the first algorithms that achieve $o(n \log n)$ time with optimal working space. Moreover, for the special case where $\log |\Sigma| = O((\log \log n)^{1-\epsilon})$, we can speed up our suffix array construction algorithm to the optimal $O(n)$.

Key words. text indexing, preprocessing, suffix trees, suffix arrays

AMS subject classifications. 68P05, 68P10, 68P30, 68W40

DOI. 10.1137/070685373

1. Introduction.

Due to the advances in information technology and biotechnology, the amount of text data is increasing exponentially. To assist users in locating their required information, the role of indexing data structures has become more and more important. For texts with a word boundary such as English, inverted index [7] is used since it enables fast queries and is space efficient. However, for texts without word boundary, such as DNA/protein sequences or Chinese/Japanese texts, inverted index is not suitable. In this case, we need full-text indices, that is, indexing data structures which make no assumption on the word boundary. Suffix trees [20] and suffix arrays [19] are two fundamental full-text indices in the literature that find applications in numerous areas, including data mining [28] and biological research [8]. For the other full-text indices, almost all of them originate from these two data structures.

*Received by the editors March 15, 2007; accepted for publication (in revised form) September 24, 2008; published electronically February 11, 2009. The work of the first author was supported in part by Hong Kong RGC grant HKU-7024/01E during his stay at the University of Hong Kong. The work of the second author was supported in part by the Grant-in-Aid of the Ministry of Education, Science, Sports, and Culture of Japan. The work of the third author was supported in part by NUS Academic Research grant R-252-000-119-112. A preliminary version of this paper appears in *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, Cambridge, MA, 2003, pp. 251–260.
http://www.siam.org/journals/sicomp/38-6/68537.html

†Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan (wkhon@cs.nthu.edu.tw).

‡Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka, Japan (sada@csce.kyushu-u.ac.jp).

§Department of Computer Science, School of Computing, National University of Singapore, Singapore (ksung@comp.nus.edu.sg).

Suffix trees and suffix arrays are very useful since they allow us to perform pattern searching efficiently. Consider a text with n characters. Given the suffix tree, we can search for a pattern P within the text using $O(|P|)$ time, which is independent of the text size. For the suffix array, the searching time is $O(|P| + \log n)$,[1] which is only a bit slower. One more advantage of the suffix array is that even if this indexing structure is placed in external memory, it still can achieve good I/O performance for searching [3]. In spite of that, suffix trees and suffix arrays cannot be built easily when n is large. The construction algorithms for both of them either are too slow or require too much working space.

For instance, when we optimize the construction time, based on the work from Weiner [30], McCreight [20], Ukkonen [29], and Farach [4], a suffix tree and a suffix array can be built in $O(n)$ time. However, the working space required is $\Omega(n \log n)$ bits.

On the other hand, when we optimize the construction working space, based on the recent work by Lam et al. [18], we can first build the compressed suffix array (CSA) of [6] and then convert it into the suffix tree and the suffix array. Although such an approach reduces the working space to $O(n)$ bits, the execution time is increased to $O(n \log n)$. Another solution is to rely on external memory [3] to control the working space. However, the time complexity, not to mention the increase in I/O burden, is even worse.

It was open whether the suffix tree and the suffix array can be constructed in $o(n \log n)$ time and $o(n \log n)$-bit working space. The need to break this time-and-space barrier is illustrated in a concrete example that arises in practice. Suppose we would like to construct a suffix array for a human genome (of length approximately 3 billion). The fastest known algorithm runs in linear time. However, it requires 40 gigabytes working space [17]. Such a memory requirement far exceeds the capacity of ordinary computers. On the other hand, if we apply the most space-efficient algorithm, the working space required is roughly 3 gigabytes, which is possible to implement on a PC nowadays. The time required, however, is more than 20 hours [18], which is a bit slow.

Apart from suffix trees and suffix arrays, we observe that the other full-text indices also suffer from the same time-and-space barrier during the construction phase. Such a barrier may prevent these indices from becoming useful for large-scale applications.[2] Table 1 summarizes the performance of the best known algorithms for constructing these full-text indices.

1.1. Our results. Our results are based on the following model. First, we assume a unit-cost RAM with a word size of $O(\log U)$ bits, where $n \leq U$, in which standard arithmetic and bitwise boolean operations on word-sized operands can be performed in constant time [1, 9]. Second, to compare our work fairly with the other main-memory algorithms, we add the following assumptions: (1) We restrict our algorithms to running within the main memory, in which no I/O operations are involved in the intermediate steps; (2) for counting the working space, we do not include the space for the output of the full-text indices (this can be justified, as output can be written directly to the secondary storage upon completion without occupying the main memory). Under the above model, this paper proposes the following construction

[1] We use the notation $\log_b^c n = (\log n / \log b)^c$ to denote the cth power of the base-b logarithm of n. Unless specified otherwise, we use $b = 2$.

[2] Zobel, Moffat, and Ramamohanarao [32] and Crauser and Ferragina [3] both mentioned the importance of construction algorithms to the usefulness of the index.

TABLE 1

Construction times for full-text indices. The acronyms ST, SA, CST, CSA, FM represent suffix tree, suffix array, compressed suffix tree, compressed suffix array, and FM-index, respectively. Also, ϵ is any fixed real number with $0 < \epsilon < 1$.

Index	Algorithm	Time	Space (bits)
SA, CSA, or FM	opt time [19]	$O(n)$	$O(n \log n)$
	opt space [18]	$O(n \log n)$	$O(n)$
	this paper	$O(n)$	$O(n)$
ST or CST	opt time [4]	$O(n)$	$O(n \log n)$
	opt space [12]	$O(n \log n)$	$O(n)$
	this paper	$O(n \log^{\epsilon} n)$	$O(n)$

algorithms for full-text indices, where the input text is assumed to be over a constant-size alphabet:

1. An algorithm which constructs the suffix array in $O(n)$ time and $O(n)$-bit working space;
2. an algorithm which constructs the suffix tree in $O(n \log^{\epsilon} n)$ time and $O(n)$-bit working space for $0 < \epsilon < 1$.

To the best of our knowledge, these are the first algorithms known to run in $o(n \log n)$ time and $o(n \log n)$-bit working space.

Our algorithms can actually be adopted to build other full-text indices, including the CSA, CST [12], and FM-index [5]. The performance of our algorithms for constructing these indices is summarized in Table 1. Another application of our algorithm is that it can act as a time- and space-efficient algorithm for block sorting [2], which is a widely used process in various compression schemes, such as `bzip2` [27].

We also study the general case where the alphabet size is not constant. Let Σ be the alphabet, and let $|\Sigma|$ denote its size. Our algorithm can construct the suffix array and the suffix tree using $O(n \log |\Sigma|)$-bit working space, while running in $O(n \log \log |\Sigma|)$ time and $O(n(\log^{\epsilon} n + \log |\Sigma|))$ time, respectively, for any fixed ϵ with $0 < \epsilon < 1$. These are the first algorithms that achieve $o(n \log n)$ time with optimal working space. Moreover, for the special case where $\log |\Sigma| = O((\log \log n)^{1-\epsilon})$, we can apply Pagh's data structure for constant-time rank queries [24] to further improve the running time of the suffix array construction to the optimal $O(n)$.

Remark. Recently, Na and Park [23] proposed another algorithm for the construction of the CSA, FM-index, and the (BW) transform. The running time is $O(n)$ time, which is independent of the alphabet size. The working space is increased slightly to $O(n \log |\Sigma| \log^{\alpha}_{|\Sigma|} n)$ bits, where $\alpha = \log_3 2$.

1.2. The main techniques. To achieve a small working space, we use the Ψ function [6] of the CSA and the BW text [2] as our tools, where they can act as an implicit representation of any suffix tree generated during construction. Both the Ψ function and the BW text can be stored in $O(n)$-bit space. Moreover, given them, we can construct the suffix tree and the suffix array in $O(n \log^{\epsilon} n)$ time and $O(n)$ time, respectively, for $0 < \epsilon < 1$.

Apart from the space concern, another reason for using these two data structures as our tools is that each one's strength complements nicely the other's weaknesses: The Ψ function allows for efficient pattern query, while it is difficult to update the function in response to the change in the underlying suffix tree; the plain BW text can be easily and quickly updated, though it does not support efficient pattern query. In our construction algorithm, efficient queries and fast updates are frequently required, so we use both data structures alternately in order to utilize their strengths.

Another finding that leads to improvement is related to the backward search algorithm, which is used to find a pattern within the text based on the Ψ function. If we apply a known method [26], given the Ψ function for the text, each step of the algorithm requires $O(\log n)$ time in general. This paper presents a novel auxiliary data structure of $O(n)$ bits which supports each backward search step in $O(\log\log|\Sigma|)$ time instead. As our construction algorithm frequently executes the backward search, the overall running time is thus sped up because of this improvement.

Finally, our algorithm borrows the framework of Farach's linear-time suffix tree construction algorithm [4], which first constructs two suffix trees, one for odd-position suffixes and another for even-position suffixes, and then merges the two trees together. The difference, however, lies in the actual implementation, as we have carefully avoided storing the suffix pointers explicitly, thus saving $O(n\log n)$ bits in total.

The remainder of this paper is organized as follows. Section 2 is a preliminary section, which gives the definitions for the CSA and BW text and discusses the relationship between them. Section 3 shows the improved result in the backward search algorithm. Section 4 describes the framework for the construction algorithm, while sections 5 and 6 detail the main steps of the algorithm. In section 7, we give details on the improvement we can achieve when the alphabet size is sufficiently small, precisely when $\log|\Sigma| = O((\log\log n)^{1-\epsilon})$. Finally, we give a conclusion in section 8.

2. Preliminaries. This section is divided into three parts. The first part gives the basic notation and introduces the suffix array, the Ψ function, and the BW text. In the second part, we describe the representation of the Ψ function that is used throughout the paper. Finally, in the third part we discuss the duality between the Ψ function and the BW text.

2.1. Basic notation. First, we review some of the basic notation and assumptions. A text T of length n over an alphabet Σ is denoted by $T[0\ldots n-1]$. Each character of Σ is uniquely encoded by an integer in $[0,|\Sigma|-1]$, which occupies $\log|\Sigma|$ bits. In addition, a character c is alphabetically larger than a character c' if and only if the encoding of c is larger than the encoding of c'. Also, we assume that $T[n-1]$ is a special character that does not appear elsewhere in the text.

For any $i=0,\ldots,n-1$, the string $T[i\ldots n-1]$ is called a suffix of T. The suffix array $\mathrm{SA}[0\ldots n-1]$ of T is an array of integers such that $T[\mathrm{SA}[i]\ldots n-1]$ is lexicographically the ith smallest suffix of T. The Ψ_T function, or simply Ψ, is the main component of the CSA [6]. It is defined as follows:

- $\Psi[i]=\mathrm{SA}^{-1}[\mathrm{SA}[i]+1]$ if $\mathrm{SA}[i]\neq n-1$;
- $\Psi[i]=\mathrm{SA}^{-1}[0]$ otherwise.

Immediately, we have the following observation.

OBSERVATION 1. *For the suffix array and the Ψ function of the text T,*

1. *characters $T[\mathrm{SA}[i]]$ $(i=1,2,\ldots,n)$ are alphabetically sorted.*
2. *suppose that $T[\mathrm{SA}[i]]=T[\mathrm{SA}[j]]$. Then $\Psi[i]<\Psi[j]$ if and only if $i<j$.*

On the other hand, the BW text W [2] is a transformation on T such that $W[i]=T[\mathrm{SA}[i]-1]$ if $\mathrm{SA}[i]>0$, and $W[i]=T[n-1]$ otherwise. Intuitively, $W[i]$ is the preceding character of the ith smallest suffix of T in T. This transformation process is widely used in various compression schemes, such as `bzip2` [27], and it constitutes the main part of the construction of the FM-index [5].

See Figure 1 for an example of the suffix array, the Ψ function, and the BW text of a string $S=$ `acaaccg$`. In the example, we have $\Sigma=\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$, and the last character of S is \$, which is unique among the other characters in S. We also assume that \$ is alphabetically smaller than each character in Σ.

i	$W[i]$	SA$[i]$	$\Psi[i]$	$S[\text{SA}[i] \dots n-1]$
0	g	7	2	$
1	c	2	3	a a c c g $
2	$	0	4	a c a a c c g $
3	a	3	5	a c c g $
4	a	1	1	c a a c c g $
5	a	4	6	c c g $
6	c	5	7	c g $
7	c	6	0	g $

FIG. 1. *Suffix array, the Ψ function, and the BW text W of a string $S =$* acaaccg$.

2.2. Representation of Ψ. Observation 1 implies that the Ψ function is piecewise increasing. In addition, each Ψ value is less than n. Therefore, we can use a function $\rho(c, x) = enc(c) \cdot n + x$ and obtain a total increasing function $\Psi'[i] = \rho(T[SA[i]], \Psi[i])$, where $enc(c)$ denotes the encoding of the character c. Note that value of Ψ' is less than $n|\Sigma|$.

Based on the total increasing property, Ψ' can be stored as follows. We divide each $\Psi'[i]$, which takes $\log n + \log|\Sigma|$ bits, into two parts q_i and r_i, where q_i is the first (or most significant) $\log n$ bits, and r_i is the remaining $\log|\Sigma|$ bits. We encode the values $q_0, q_1 - q_0, \dots, q_{n-1} - q_{n-2}$ in a bit-vector B_1 using unary codes. (Recall that the unary code for an integer $x \geq 0$ is encoded as x 0's followed by a 1.) Note that the encoding has exactly n 1's where the $(i+1)$th 1, which corresponds to $\Psi[i]$, is at position $i + q_i$. Also, the total number of 0's is q_{n-1}, which is at most n. Thus, B_1 uses $2n$ bits. The r_i's are stored explicitly in an array $B_2[0 \dots n-1]$, where each entry occupies $\log|\Sigma|$ bits. Thus, B_2 occupies $n \log|\Sigma|$ bits. Moreover, an auxiliary data structure of $O(n/\log\log n)$ bits is constructed in $O(n)$ time to enable constant-time *rank* and *select* queries, and thus supporting the retrieval of any q_i in constant time [13, 22]. Then, the total size is $n(\log|\Sigma| + 2) + o(n)$ bits. Since q_i and r_i can be retrieved in constant time, so can $\Psi'[i] = |\Sigma|q_i + r_i$. This gives the following lemma.

LEMMA 1. *The Ψ' function can be encoded in $O(n \log|\Sigma|)$ bits so that each $\Psi'[i]$ can be retrieved in constant time.*

COROLLARY 1. *The Ψ function can be encoded as Ψ' in $O(n \log|\Sigma|)$ bits so that each $\Psi[i]$ can be retrieved in constant time.*

Proof. The retrieval time follows since $\Psi[i] = \Psi'[i] \bmod n$. □

2.3. Duality between Ψ and W. It is known that Ψ and W are one-to-one corresponding. In the section, we show that the transformation between them can be done in linear time and in $O(n \log|\Sigma|)$-bit space.

We first give a property relating W and Ψ.

DEFINITION 1. *Given an array of characters $x[0 \dots n-1]$, we define the* stable sorting order *of $x[i]$ in x to be the number of characters in x which is alphabetically smaller than $x[i]$, plus the number of characters $x[j]$ with $j < i$ which is equal to $x[i]$. This is in fact the position of $x[i]$ after stable sorting.*

LEMMA 2 (see [2]). *Let k be the stable sorting order of $W[i]$ in W. Then, $\Psi[k] = i$.*

Proof. Let Y_i denote the suffix $T_{\text{SA}[i]-1}$ when SA$[i] > 0$, and the suffix $T[n-1]$ otherwise. Note that when $i < j$, if Y_i and Y_j are starting with the same character, Y_i will be lexicographically smaller than Y_j. The reason is that, by excluding the first character, the remaining part of Y_i (which is $T_{\text{SA}[i]}$) is lexicographically smaller than the remaining part of Y_j (which is $T_{\text{SA}[j]}$). Also, observe that the first character of Y_i

is equal to $W[i]$. Then, it follows that the stable sorting order of $W[i]$ in W is equal to the rank of Y_i among the set of all Y_i's, which is the set of all suffixes of T.

Thus, we have $k = \mathrm{SA}^{-1}[\mathrm{SA}[i] - 1]$ when $\mathrm{SA}[i] > 0$, and $k = \mathrm{SA}^{-1}[n-1]$ otherwise. In the former case, $\mathrm{SA}[k] = \mathrm{SA}[i] - 1 < n - 1$, so $\Psi[k] = \mathrm{SA}^{-1}[\mathrm{SA}[k] + 1] = \mathrm{SA}^{-1}[\mathrm{SA}[i]] = i$. For the latter case, we have $i = \mathrm{SA}^{-1}[0]$ and $\mathrm{SA}[k] = n - 1$. Thus we have $\Psi[k] = \mathrm{SA}^{-1}[0] = i$. In summary, $\Psi[k] = i$ for all cases, and the lemma follows. □

The next two lemmas show the linear-time conversion between W and Ψ.

LEMMA 3. *Given W, we can store Ψ in $O(n)$ time and in $O(n \log |\Sigma|)$ bits. The working space is $O(n \log |\Sigma|)$ bits.*

Proof. The conversion is simply based on counting sort. We present the details below for completeness. We construct Ψ' in section 2.2 from W as an encoding of Ψ (Corollary 1). To construct Ψ', we create a bit-vector $B_1[0 \ldots 2n-1]$ and initialize all bits to 0. We also create an array $B_2[0 \ldots n-1]$, where each entry occupies $\log |\Sigma|$ bits.

Now, we show how to compute the stable sorting order of $W[i]$ in W, for $i = 0, 1, 2, \ldots$. To do so, we use three auxiliary arrays. The first array is L_1 such that $L_1[c]$ stores the number of occurrences of the character c in W. This array can be initialized by scanning W once. The second array is L_2 such that $L_2[c]$ stores the number of occurrences of a character that is smaller than c in W. This array can be initialized by scanning L_1 once. Finally, the third array is L_3 such that $L_3[c]$ stores the number of occurrences of c seen so far. Initially, all entries of L_3 are initialized to 0.

Now, we proceed to read $W[0]$, $W[1]$, and so on. Note that during the process, when we read a character c, we maintain the correctness of L_3 by increasing $L_3[c]$ just before the next character is read. Thus, at the beginning of step i, the counter $L_3[W[i]]$ will be storing the number of occurrences of $W[i]$ in $W[0 \ldots i-1]$, and the stable sorting order of $W[i]$ can be computed by $L_2[W[i]] + L_3[W[i]]$.

Let k be the stable sorting order of $W[i]$ that is computed at step i in the above algorithm. By Lemma 2, $\Psi[k] = i$. Thus, we have $\Psi'[k] = x = \rho(T[\mathrm{SA}[k]], \Psi[k]) = \rho(W[i], i)$. By our scheme, x is divided into two parts q and r, where $q = x \text{ div } |\Sigma|$ is the first $\log n$ bits, and $r = x \bmod |\Sigma|$ is the remaining bits. For q, a 1 is stored at $B_1[k+q]$. For r, it is stored at $B_2[k]$.

As the stable sorting order of each $W[i]$ is different, all possible $\Psi[k]$'s will be eventually computed and stored. A summary of the overall algorithm is shown in Figure 2. It is easy to see that the overall time is $O(n + |\Sigma|)$. For the space complexity, note that L_1, L_2, and L_3 each occupies $|\Sigma| \log n$ bits, which is at most $n \log |\Sigma|$ bits because $|\Sigma| \leq n$. Thus, we use $O(n \log |\Sigma|)$-bit working space. □

LEMMA 4. *Given Ψ and T, we can store W in $O(n)$ time and in $O(n \log |\Sigma|)$ bits. The working space is $O(n \log |\Sigma|)$ bits.*

Proof. Let $t = \mathrm{SA}^{-1}[0]$. Then, we have $\Psi[t] = \mathrm{SA}^{-1}[1]$. In general, $\Psi^k[t] = \mathrm{SA}^{-1}[k]$.

Hence, we have $W[\Psi^k[t]] = T[k-1]$. To construct W, we can first compute t. Recall that $T[n-1]$ is a unique character in T. By scanning T, we compare each character of T with $T[n-1]$. Then, we obtain the value $x = \mathrm{SA}^{-1}[n-1]$, which is equal to the number of occurrences of a character in T that is smaller than $T[n-1]$. Then, by definition, $\Psi[x] = \mathrm{SA}^{-1}[0]$, which is equal to t. Thus, t can be found in $O(n)$ time. Afterwards, we iteratively compute $\Psi^i[t]$ and set $W[\Psi^i[t]] = T[i-1]$, for $i = 1 \ldots n$. As $\Psi^i[t]$ corresponds to the rank of a different suffix of T for different i, all

1. Compute $L_1[c]$ to store the number of occurrences of each $c \in \Sigma$.
2. Compute $L_2[c]$ to store the number of occurrences of a character that is smaller than c. That is, $L_2[c] = \sum_{d<c} L_1[c]$ for $c \in \Sigma$.
3. Let L_3 be an array such that $L_3[c]$ stores the number of occurrences of the character c seen so far.
4. Initialize all entries of L_3 to be 0.
5. For $i = 1, 2, \ldots, n$,
   Let $c = W[i]$, $k = L_2[c] + L_3[c]$.
   Compute $x = \rho(c, i)$.
   Let $q = x$ div $|\Sigma|$, $r = x \bmod |\Sigma|$.
   Set $B_1[k + q] = 1$ and $B_2[k] = r$.
   Increase $L_3[c]$ by one.
6. Compute the $O(n/\log\log n)$-bit auxiliary data structure for B_1.

FIG. 2. *Computing* Ψ *from* W.

the characters of W will be eventually computed and stored by the above algorithm. The total time of the algorithm is $O(n)$, and the spaces for W, T, and Ψ are all $O(n \log |\Sigma|)$ bits. The lemma thus follows. □

3. Improving the backward search algorithm. Let S be a text of length m over an alphabet Δ. In this section, we present an $O(m + |\Delta|)$-bit auxiliary data structure for the Ψ function of S that improves each step in the backward search algorithm from $O(\log m)$ time to $O(\log\log |\Delta|)$ time.

We first present in section 3.1 a data structure that supports fast rank query and show that such a data structure can be constructed efficiently in terms of time and working space. Then, in section 3.2, we describe the improved backward search algorithm based on the result of section 3.1.

3.1. Efficient data structure for fast rank query. Let Q be a set of distinct numbers. For any integer x, the *rank* of x in Q is the number of elements in Q smaller than x. We begin with two supporting lemmas prior to the description of our data structure. The first one is on a perfect hash function, which is obtained by rephrasing the result of section 4 of [10] as follows.

LEMMA 5. *Given x b-bit numbers, where $b = \Theta(\log x)$, a data structure of size $O(xb)$ bits supporting an $O(1)$-time existential query can be constructed in $O(x \log x)$ time and $O(xb)$-bit working space.*

The second lemma is derived from a result in [21, 31] based on Lemma 5.

LEMMA 6. *Given z w-bit numbers, where $w = \Theta(\log z)$, a data structure of size $O(zw^2)$ bits supporting $O(\log w)$-time* rank *queries can be constructed in $O(zw \log(zw))$ time and $O(zw^2)$-bit working space.*

Proof. It is shown that *rank* queries can be solved in $O(\log w)$ time if the existential query for all prefixes of the z numbers can be answered in $O(1)$ time [21, 31].[3] The

[3] In the original papers, the results are for another query called *predecessor*, which finds the largest element in the z numbers that is smaller than the input w-bit number k. However, such a result can be modified easily for the *rank* query as follows. For each number i in the z numbers, it is replaced by the number iz + the rank of i (so that the number now has $w + \log z$ bits), and we construct the *predecessor* data structure for these modified numbers. For the intended *rank* query, we first try to find the predecessor for kz in the modified numbers, and if no predecessor is found, the rank of k in the z numbers is 0. Otherwise, let this predecessor be p. It is easy to see that the required result is equal to (p mod z) + 1.

idea is that, given a w-bit number k, its longest common prefix with the z numbers can be found by binary search (on the length) using $O(\log w)$ existential queries, and such a prefix uniquely determines the *rank* of k.

Notice that only $O(zw)$ strings can be a prefix of the z numbers, and each can be represented in $\Theta(w)$ bits. Applying Lemma 5 on this set of strings (with $x = O(zw)$ and $b = \Theta(w)$), we have the required data structure. The lemma thus follows. □

Now, we are ready to describe our new data structure, whose performance is summarized below.

THEOREM 1. *Let Q be a set of n numbers, each of length $\Theta(\log n)$ bits. Then, a data structure of size $O(n \log n)$ bits supporting an $O(\log \log n)$-time* rank *query in Q can be constructed in $O(n)$ time and $O(n \log n)$-bit working space.*

Proof. We construct the following data structure for Q:

1. Let $k_0 < k_1 < \cdots < k_{n-1}$ be n numbers of Q stored in ascending order by an array.
2. Partition the n numbers into n/w^2 lists, each containing w^2 numbers. Precisely, the lists are in the form $\{k_i, k_{i+1}, \ldots, k_{i+w^2-1}\}$, where $i \equiv 0 \pmod{w^2}$.
3. Let the smallest element in each list be its representative. Construct a data structure for *rank* query for these representatives based on Lemma 6.

The above data structure occupies $O(nw)$ bits and can be constructed in $O(n)$ time and $O(nw)$-bit working space. With such a data structure, the *rank* of x among the n numbers can be found in $O(\log w)$ time as follows.

1. Find the *rank* of x among the n/w^2 representatives of the lists. Let this be r.
2. Then, the *rank* of x among the n numbers must now lie in $[rw^2, (r+1)w^2 - 1]$. Do a binary search on the w^2 elements $\{k_{rw^2}, k_{rw^2+1}, \ldots, k_{(r+1)w^2-1}\}$ to find the *rank* of x.

Both steps thus take $O(\log w)$ time. This completes the proof of Theorem 1. □

Note that, in contrast to the existing data structures for the *rank* query [13, 25], our data structure requires either less space for storage (when compared with [13]) or less time in the construction (when compared with [25]); the drawback is a blow-up in query time. Based on Theorem 1, we can use some extra space to achieve a more generalized result, as shown in the following corollary.

COROLLARY 2. *Let Q' be a set of n values, each of length $\log \ell + \Theta(\log n)$ bits for any ℓ. Then, a data structure of size $O(n \log n + \ell)$ bits supporting an $O(\log \log n)$-time* rank *query in Q' can be constructed in $O(n + \ell)$ time and $O(n \log n + \ell)$-bit working space.*

Proof. The idea is to apply Theorem 1 by transforming the set Q' into another set such that each value takes only $\Theta(\log n)$ bits. First, we scan Q' and create a bit-vector $B[0 \ldots \ell - 1]$ such that $B[i] = \texttt{1}$ if there is some number in Q' whose first $\log \ell$ bits represent a value i, and $B[i] = \texttt{0}$ otherwise. Afterwards, we construct an auxiliary data structure for B of size $o(\ell)$ bits to support constant-time *rank* and *select* queries [13, 22].

Now, we transform each number in Q' as follows: If the first $\log \ell$ bits of the number represent the value i, these bits are replaced by the binary bit-sequence for the rank of i in B. Note that the rank of i is less than n, as there are only n numbers. Thus, after the transformation, each value takes $\Theta(\log n)$ bits, and, in addition, the transformation preserves the ordering among the elements in Q'.

Let the set of the transformed values be Q. We create the data structure of Theorem 1 on Q. To perform a *rank* query for x in Q' (we assume that x has the same length as any number in Q'), we first obtain the first $\log \ell$ bits of x. Suppose that these bits represent the value i_x. Then there are two cases.

Case 1. If $B[i_x] = 1$, we replace the first $\log \ell$ bits of x by the $\log n$-bits that represent the rank of i_x in B, and obtain a new value y. Then, it is easy to see that the rank of x in Q' is equal to the rank of y in Q.

Case 2. Otherwise, we replace the first $\log \ell$ bits of x by the $\log n$-bits that represent the rank of i_x in B, while setting the remaining $\Theta(\log n)$ bits to zeros, and obtain a new value z. Then, it is easy to see that the rank of x in Q' is equal to the rank of z in Q.

Finally, for the time and space complexity, B and its auxiliary data structures can be created in $O(\ell)$ time and stored in $\ell + o(\ell)$ bits, while the data structure for the *rank* query in Q can be created in $O(n)$ time and stored in $O(n \log n)$ bits (by Theorem 1). The lemma thus follows. □

3.2. The improved backward search algorithm. First, a *backward search step* is defined as follows.

DEFINITION 2. *For any pattern P, suppose that the rank of P among all suffixes of S is known. A* backward search step *then computes the rank of cP among the suffixes of S for any character $c \in \Delta$.*

Let Ψ' denote the total increasing function such that $\Psi'[i] = \rho(S[\mathrm{SA}[i]], \Psi[i])$ and $\rho(c, x) = enc(c) \cdot m + x$. Then, we have the following lemma.

LEMMA 7. *Let r be the rank of P among all suffixes of S. Then, the rank of cP among all suffixes of S is equal to $j \in [0, m]$ such that $\Psi'[j-1] < \rho(c, r) \leq \Psi'[j]$. (As a sentinel, we let $\Psi'[-1] = -1$ and $\Psi'[m] = m|\Delta|$.)*

Proof. It is easy to check that for all $i = 0, 1, \ldots, j-1$, the rank-i suffix of S must either start with a character smaller than c or start with c but with the remaining part lexicographically smaller than P. Thus, for all $i = 0, 1, \ldots, j-1$, the rank-i suffix of S is lexicographically smaller than cP. On the other hand, for all $i \geq j$, the rank-i suffix of S is lexicographically greater than or equal to cP. Thus, the rank of cP is j. □

Essentially, a backward search step that computes the rank of cP in the above lemma is equivalent to finding the rank of $\rho(c, r)$ in the set of all Ψ' values. Then based on the data structure for the *rank* query in section 3.1 (Corollary 2), we can obtain the main result of this section as follows.

LEMMA 8. *Let S be a text of length m over an alphabet Δ. Suppose that the Ψ function of S is given, which is stored as Ψ' using the scheme in section 2.2. Then, an auxiliary data structure for the Ψ function of S can be constructed in $O(m+|\Delta|)$ time, which supports each backward search step in $O(\log \log |\Delta|)$ time. The space requirement is $O(m + |\Delta|)$ bits.*

Proof. Let V denote the set of all Ψ' values. To prove the lemma, it suffices to show a data structure of $O(m + |\Delta|)$ bits that supports the *rank* query for any x in V in $O(\log \log |\Delta|)$ time.

First, recall that in our encoding of Ψ', each value in V is stored in two parts, where the first $\log m$ bits are encoded by unary codes in a bit-vector B_1, and the remaining $\log |\Delta|$ bits are encoded in an array B_2 as it is. In addition, there is an auxiliary data structure supporting constant-time *rank* and *select* queries.

Let G_i be the set of Ψ' values whose first $\log m$ bits represent the value i. Among the sets of G_i's, we are concerned with those sets whose size is greater than $\log |\Delta|$. Let $G_{i_1}, G_{i_2}, \ldots, G_{i_k}$ be such sets, where $i_1 < i_2 < \cdots < i_k$.

Note that each of the groups $G_{i_1}, G_{i_2}, \ldots, G_{i_k}$ has size between $\log |\Delta|$ and $|\Delta|$. Now, we combine the groups, from left to right, into supergroups of size $\Theta(|\Delta|)$. More precisely, we start from G_{i_1}, merge it with G_{i_2}, G_{i_3}, and so on, until the size exceeds $|\Delta|$. Then, we merge the next unmerged group with its succeeding group and

so on, until the size exceeds $|\Delta|$. The process is repeated until all groups are within a supergroup. (To ensure that each supergroup has size $\Theta(|\Delta|)$, we add a dummy group $G_m = \{m|\Delta|, m|\Delta|+1, \ldots, (m+1)|\Delta|-1\}$ as a sentinel.)

For each supergroup $\mathcal{G}$, let $v_0, v_1, \ldots, v_p$ be its $\Theta(|\Delta|)$ elements. Now, we pick every $\log|\Delta|$ elements (i.e., $v_0, v_{\log|\Delta|}, v_{2\log|\Delta|}, \ldots$), subtract each of them by v_0, and make them the representatives of this supergroup. Then, we construct the data structure for the *rank* query of Corollary 2 over these representatives.

With the above data structure, a *rank* query for any x in $\mathcal{G}$ can be supported as follows. We first check if $x \leq v_0$. If so, the *rank* of x is 0. Otherwise, we find the *rank* of $x - v_0$ among the representatives, which takes $O(\log\log|\Delta|)$ time. Suppose the rank is r. Then, the rank of x in $\mathcal{G}$ must lie between $r\log|\Delta|$ and $(r+1)\log|\Delta|-1$, and this can be found by a binary search in the elements $\{v_{r\log|\Delta|}, \ldots, v_{(r+1)\log|\Delta|-1}\}$, which takes $O(\log\log|\Delta|)$ time. In summary, the time required is $O(\log\log|\Delta|)$.

Now, let us complete the whole picture to show how to perform the *rank* query for x in V. First, we extract the first $\log m$ bits of x by dividing it with $|\Delta|$. Let $i' = x \text{ div } |\Delta|$ be its value. Next, we determine the size of $G_{i'}$, which can be done in constant time using rank and select queries on B_1. If the size is 0 (i.e., $G_{i'}$ is empty), the rank of x in V can be computed immediately (precisely, the required rank is equal to the number of 1's in $B_1[0 \ldots i'-1]$, which can be computed in constant time using B_1 and its auxiliary data structure). If the size is smaller than $\log|\Delta|$, the rank of x can be found by performing a binary search with the elements in $G_{i'}$, which takes $O(\log\log|\Delta|)$ time. Finally, if the size is greater than $\log|\Delta|$, we locate the supergroup $\mathcal{G}$ that contains the elements of $G_{i'}$, and we retrieve the rank r of its smallest element v_0 in V. Then, the required rank is r plus the rank of x in $\mathcal{G}$. We now claim that locating the supergroup and retrieving r can be done in constant time (to be proved shortly), so that the total time is $O(\log\log|\Delta|)$.

We prove the above claim as follows. To support finding the smallest element in each supergroup, and retrieval of its rank in V, we use a bit-vector B_1' of $O(m)$ bits, obtained from B_1 by keeping only those 1's whose corresponding Ψ' value is a smallest element in some supergroup. Also, we augment B_1' with constant-time *rank* and *select* data structures. Then, the smallest value of the $(i+1)$th supergroup, and its rank in V, can be found by consulting B_1 and B_1' in constant time. In addition, for any G_i (with size greater than $\log|\Delta|$), the rank of its supergroup among the other supergroups can be found by consulting B_1' in constant time.

On the other hand, to support locating the *rank* data structure of the supergroup, we first analyze the space requirement of these data structures. For a particular supergroup $\mathcal{G} = \{v_0, v_1, \ldots, v_p\}$, the data structure is built for $p/\log\Delta = \Theta(\Delta/\log\Delta)$ elements, each of which has value in $[0, v_p - v_0]$, so that the space is $O(v_p - v_0 + \frac{p}{\log|\Delta|} \cdot \log|\Delta|)$ bits (by Corollary 2), which is $O(v_p - v_0)$ bits since $p \leq v_p - v_0$. Thus, the total space requirement is $O(m + |\Delta|)$ bits,[4] and we assume that the data structures of the supergroups are stored consecutively according to the rank of its smallest element. Then, we create a bit-vector B_3 whose length is identical to the above data structures, which is used to mark the starting position of each data structure. Also, we augment the bit-vector with an $o(m+|\Delta|)$-bit auxiliary data structure to support constant-time *rank* and *select* queries. Thereafter, when we want to locate a supergroup for G_i, we find its rank r among the other supergroups using B_1', and then this rank-r supergroup can be accessed in constant time using B_3.

[4]The additional $O(|\Delta|)$ bits are due to the dummy group G_m.

In summary, our data structure takes a total space of $O(m+|\Delta|)$ bits and supports each backward search step in $O(\log\log|\Delta|)$ time. For the construction, it takes at most $O(m+|\Delta|)$ time. The lemma thus follows. □

4. The framework for constructing the CSA and the FM-index. Recall that $T[0\ldots n-1]$ is a text of length n over an alphabet Σ, and we assume that $T[n-1]$ is a special character that does not appear elsewhere in T. This section describes how to construct the Ψ function and the BW text W of T in $O(n\log\log|\Sigma|)$ time. Our idea is based on Farach's framework for linear-time construction of the suffix tree [4], which first constructs the suffix tree for even-position suffixes by recursion, based on which it induces the suffix tree for odd-position suffixes, and then merges the two suffix trees to obtain the required one.

For our case, we first assume that the length of T is a multiple of $2^{\lceil\log\log_{|\Sigma|}n\rceil+1}$. (Otherwise, we add enough \$ and a \$′ at the end of T, where \$′ is a character alphabetically smaller than the other characters in T, and proceed with the algorithm. The Ψ of this modified string can be converted into the Ψ of T in $O(n)$ time.) Let h be $\lceil\log\log_{|\Sigma|}n\rceil$. For $0\le k\le h$, we define T^k to be the string over the alphabet Σ^{2^k}, which is formed by concatenating every 2^k characters in T to make one character. That is, $T^k[i]=T[i\cdot 2^k+1\ldots(i+1)\cdot 2^k-1]$, for $1\le i\le n/2^k$. By definition, $T^0=T$.

In addition, we introduce the following definitions associated with a string. For any string $S[0\ldots m-1]$ with even number of characters, denote S_e and S_o to be the strings of length $m/2$ formed by *merging* every 2 characters in $S[0\ldots m-1]$ and $S[1\ldots m-1]S[0]$, respectively; more precisely, $S_e[i]=S[2i]S[2i+1]$ and $S_o[i]=S[2i+1]S[2i+2]$, where we set $S[m]=S[0]$. Intuitively, the suffixes of S_e and S_o correspond to the even-position and odd-position suffixes of S, respectively. We have the following observation.

OBSERVATION 2. $T_e^i=T^{i+1}$.

Also, note that the last characters of T_o^i and T_e^i are unique among the corresponding string. This makes the results in sections 2 and 3 applicable for both texts.

Our basic framework is to use a bottom-up approach to construct Ψ of T^i, or Ψ_{T^i}, for $i=\lceil\log\log_{|\Sigma|}n\rceil$ down to 0, thereby obtaining Ψ of T in the end. Precisely,

- for step $i=\lceil\log\log_{|\Sigma|}n\rceil$, Ψ_{T^i} is constructed by first building the suffix tree for T^i using Farach's algorithm [4] and then converting it back into the Ψ function.
- for the remaining steps, we construct the Ψ_{T^i} based on the $\Psi_{T^{i+1}}$, the latter of which is in fact $\Psi_{T_e^i}$ by Observation 2. We first obtain $\Psi_{T_o^i}$ based on T^i and $\Psi_{T_e^i}$. Afterwards, we merge $\Psi_{T_o^i}$ and $\Psi_{T_e^i}$ to give Ψ_{T^i}. The complete algorithm is shown in Figure 3.

1. For $i=\lceil\log\log_{|\Sigma|}n\rceil$,
   (a) construct suffix tree for T^i.
   (b) construct Ψ_{T^i} from the suffix tree.
2. For $i=\lceil\log\log_{|\Sigma|}n\rceil-1$ to 0,
   (a) construct $\Psi_{T_o^i}$ based on $\Psi_{T_e^i}$. (Note: $\Psi_{T_e^i}=\Psi_{T^{i+1}}$.)
   (b) construct Ψ_{T^i} based on the $\Psi_{T_o^i}$ and $\Psi_{T_e^i}$.

FIG. 3. *The construction algorithm of Ψ function of T.*

Sections 5 and 6 describe in details how to obtain $\Psi_{T_o^i}$ from $\Psi_{T_e^i}$ and T^i, and how to merge $\Psi_{T_o^i}$ and $\Psi_{T_e^i}$ to obtain Ψ_{T^i}, respectively. This gives the main theorem of this section.

THEOREM 2. *The Ψ function and the BW text W of T can be constructed in $O(n \log\log|\Sigma|)$ time and $O(n \log|\Sigma|)$-bit working space.*

Proof. We refer to the algorithm in Figure 3, which has two phases. For phase 1, we have $i = \lceil \log\log_{|\Sigma|} n \rceil$. We first construct the suffix tree for T^i whose size is $n/2^{\lceil \log\log_{|\Sigma|} n \rceil} \leq n \log|\Sigma|/\log n$. This requires $O(n \log|\Sigma|/\log n)$ time and $O(n \log|\Sigma|)$-bit space by using Farach's suffix tree construction algorithm [4]. Then, Ψ_{T^i} can be constructed in $O(n \log|\Sigma|/\log n)$ time and $O(n \log|\Sigma|)$-bit working space. Thus, phase 1 in total takes $O(n)$ time and $O(n \log|\Sigma|)$-bit space.

For every step i in phase 2, we construct Ψ_{T^i}. Let Δ_i be the alphabet of T^i. Then, part (a) takes $O(|T^i| + |\Delta_i|)$ time (Lemma 12), and part (b) takes $O(|T^i| \log\log|\Delta_i| + |\Delta_i|)$ time (Lemma 14). For the space, both require $O(|T^i| \log|\Delta_i| + |\Delta_i|)$ bits. Note that $|T^i| = n/2^i$ and $|\Delta_i| \leq |\Sigma|^{2^i} \leq n$, so the space used by step i is $O(|T^i| \log|\Delta_i| + |\Delta_i|) = O(n \log|\Sigma|)$ bits, and the time is $O(|T^i| \log\log|\Delta_i| + |\Delta_i|) = O((n/2^i) \cdot (i + \log\log|\Sigma|) + |\Sigma|^{2^i})$. In total, the space for phase 2 is $O(n \log|\Sigma|)$ bits and the time is

$$\sum_{i=1}^{\lceil \log\log_{|\Sigma|} n \rceil - 1} O\left(\frac{n}{2^i}(i + \log\log|\Sigma|) + |\Sigma|^{2^i}\right) = O(n \log\log|\Sigma|).$$

The whole algorithm for constructing Ψ of T therefore takes $O(n \log\log|\Sigma|)$ time and $O(n \log|\Sigma|)$-bit space. Finally, the BW text W can be constructed from Ψ using Lemma 4 in $O(n)$ time and $O(n \log|\Sigma|)$-bit space. This completes the proof. □

Once the BW transformation is completed, the FM-index can be created by encoding the transformed text W using move-to-front encoding and run-length encoding [5]. When the alphabet size is small, precisely, when $|\Sigma| \log|\Sigma| = O(\log n)$, move-to-front encoding and run-length encoding can be done in $O(n)$ time based on a precomputed table of $o(n)$ bits. In summary, this encoding procedure takes $O(n)$ time using $o(n)$-bit space in addition to the output index. Thus, we have the following result.

THEOREM 3. *The FM-index of T can be constructed in $O(n \log\log|\Sigma|)$ time and $O(n \log|\Sigma|)$-bit working space, when $|\Sigma| \log|\Sigma| = O(\log n)$.*

4.1. Further discussion. The compressed suffix tree (CST) [12] is a compact representation of the suffix tree taking $O(n \log|\Sigma|)$ bits of space. The core of the CST consists of (1) the CSA of the input text, (2) parentheses encoding of the tree structure of the suffix tree, and (3) an *Hgt* array that enables efficient computation of the longest common prefix (LCP) query. It is shown in [12, 11] that once the CSA of the input text is computed, the CST can be constructed in $O(n \log^\epsilon n)$ time and $O(n \log|\Sigma|)$-bit working space, for any fixed ϵ with $0 < \epsilon < 1$.

Once the CST is constructed, we can simulate a preorder traversal of the original suffix tree in $O(n(\log^\epsilon n + \log|\Sigma|))$ time [12, 11], thereby constructing the original suffix tree along the traversal. Summarizing, we have the following result.

THEOREM 4. *The CST and suffix tree of T can be constructed in $O(n \log^\epsilon n)$ time and $O(n(\log^\epsilon n + \log|\Sigma|))$ time, respectively, for any fixed ϵ with $0 < \epsilon < 1$. Both construction algorithms require $O(n \log|\Sigma|)$-bit working space.*

		X_i		
i	$y[i]$	$x[i]$	$S_e[SA_e[i] \dots m/2-1]$	$S[0]$
0	$a	c	aa cc g$	a
1	cg	$	ac aa cc g$	a
2	ca	a	cc g$	a
3	ac	c	g$	a

(a)

i	$y \to C_o$	$x \to$ sorted x
0	cg	$
1	ca	a
2	$a	c
3	ac	c

(b)

FIG. 4. *Consider* $S =$ acaaccg$. (a) *The relationship between* $x[i]$, $y[i]$, *and* X_i. *Note that* X_i *corresponds to a suffix of* S_o. (b) *After stable sorting on the array* x, *the array* y *becomes* C_o.

5. Constructing Ψ_{S_o}. Given $S[0 \dots m-1]$ and Ψ_{S_e}, this section describes how to construct Ψ_{S_o}. Our approach is indirect, as prior to obtaining Ψ_{S_o}, we need to construct the BW text C_o of S_o.

Let Δ be the alphabet of S. Define $x[0 \dots m/2-1]$ to be an array of characters such that $x[i] = S[2SA_e[i]-1]$, where $2SA_e[i]-1$ is computed in modulo-m arithmetic. Let X_i be the string $x[i]S_e[SA_e[i] \dots m/2-1]S[0]$.

OBSERVATION 3. X_i *is a suffix of* S_o *if* $SA_e[i] \neq 0$. *Otherwise, the first character of* X_i *is* $S[m-1]$, *which is unique among other characters in* S.

Let X be the set $\{X_k | 0 \leq k \leq m/2-1\}$. Intuitively, X is the same as the set of suffixes of S_o. See Figure 4(a) for an example of X_i.

LEMMA 9. *The stable sorting order of* $x[i]$ *in* x *equals the rank of* X_i *in* X.

Proof. By omitting the first character of every X_i, we see that these X_i's are of the form $S_e[SA_e[i] \dots m/2-1]S[0]$, which are already sorted. Thus, the rank of X_i is equal to the stable sorting order of $x[i]$ in x. □

LEMMA 10. *Given* Ψ_{S_e} *and* S, *we can construct* C_o *in* $O(m + |\Delta|)$ *time and* $O(m \log |\Delta| + |\Delta|)$*-bit space.*

Proof. Let $y[0 \dots m/2-1]$ be an array such that $y[i]$ stores the two characters that immediately precede $x[i]$ in S (i.e., $S[2SA_e[i]-3]S[2SA_e[i]-2]$). In fact, $y[i]$ is the preceding character of X_i in S_o. Using a similar approach as in Lemma 4, x and y can be computed in $O(m)$ time, and both arrays occupy $O(m \log |\Delta|)$ bits.

To construct C_o, we perform a stable sort on x as in Lemma 3, and we iteratively compute the stable sorting order k of $x[i]$, which is equal to the rank of X_i by Lemma 9. During the process, we set $C_o[k] = y[i]$. The total time is $O(m + |\Delta|)$ and the total space is $O(m \log |\Delta| + |\Delta|)$ bits. See Figure 4 for an example. □

LEMMA 11. Ψ_{S_o} *can be constructed from* C_o *in* $O(m+|\Delta|)$ *time and* $O(m \log |\Delta| + |\Delta|)$*-bit space.*

Proof. The proof is similar to that of Lemma 3. □

Thus, we conclude this section with the following lemma.

LEMMA 12. *Given* Ψ_{S_e} *and* S, *we can construct* Ψ_{S_o} *in* $O(m + |\Delta|)$ *time and* $O(m \log |\Delta| + |\Delta|)$*-bit space.*

6. Merging Ψ_{S_o} and Ψ_{S_e}. In this section, we construct Ψ_S from Ψ_{S_o} and Ψ_{S_e}. The idea is to determine the rank of any suffix of S among all suffixes of S, and based on this information, we construct the BW text C of S. Finally, we convert C to Ψ_S by Lemma 3.

Let s be any suffix of S. Observe that the rank of s among the suffixes of S is equivalent to the sum of the rank of s among the odd-position suffixes and that among the even-position suffixes of S. Based on this observation, we can construct the C array (the BW transformation of S) as follows.

First, we construct the auxiliary data structures of Lemma 8 for Ψ_{S_o} and for Ψ_{S_e}. Next, we perform backward searches for S_e on Ψ_{S_o} and Ψ_{S_e} simultaneously by Lemma 7, so that at step i, we obtain the ranks of $S_e[m/2-i \ldots m/2-1]$ among the odd-position suffixes and even-position suffixes of S, respectively. By summing these two ranks, we get the rank k of $S_e[m/2-i \ldots m/2-1]$ among all suffixes of S. Then, we set $C[k]$ to be $S[m-2i-1]$, which is the preceding character of the suffix $S[m-2i \ldots m-1] = S_e[m/2-i \ldots m/2-1]$.

Similarly, we perform a simultaneous backward search for S_o on Ψ_{S_o} and Ψ_{S_e} to complete the remaining entries of C. Thus, we obtain C by $O(m)$ backward search steps. The algorithm is depicted as MERGECSA in Figure 5.

MERGECSA

1. Construct the auxiliary data structures for Ψ_{S_o} and for Ψ_{S_e} to support efficient backward search.
2. Backward search for S_e on Ψ_{S_o} and Ψ_{S_e} simultaneously, and
   (a) at step i, we obtain the rank of $S_e[m/2-i \ldots m/2-1]$ among the odd-position suffixes and that among the even-position suffixes of S. Let the sum of the ranks be k.
   (b) Set $C[k] = S[m-2i-1]$.
3. Backward search for S_o on Ψ_{S_o} and Ψ_{S_e} simultaneously, and fill in $C[k]$ accordingly.

FIG. 5. *Merging* Ψ_{S_o} *and* Ψ_{S_e}.

The following lemma shows the correctness of our algorithm.

LEMMA 13. *The algorithm* MERGECSA *in Figure* 5 *constructs* $C[0 \ldots m-1]$ *correctly.*

Proof. Recall that for every suffix $S[i \ldots m-1]$, $C[SA^{-1}[i]]$ equals the preceding character of $S[i \ldots m-1]$. For every even-position suffix $S[i \ldots m-1] = S_e[i/2 \ldots m/2-1]$, step 2 computes its rank k among all odd-position and even-position suffixes. By definition, $k = SA^{-1}[i]$. Therefore, step 2 correctly assigns $C[k]$ to be the preceding character of $S[i \ldots m-1]$. By the same argument, step 3 handles the odd-position suffixes and correctly assigns $C[SA^{-1}[i]]$ to be the preceding character of $S[i \ldots m-1]$.

Therefore, after steps 2 and 3, MERGECSA completely constructs $C[0 \ldots m-1]$. The lemma thus follows. □

By Lemma 8, the auxiliary data structures can be constructed in $O(m+|\Delta|)$ time and $O(m+|\Delta|)$-bit space, and then each backward search step is done in $O(\log\log|\Delta|)$ time. On the other hand, the Ψ function occupies $O(m\log|\Delta|)$-bit space. Thus, we have the following lemma.

LEMMA 14. *Given* Ψ_{S_o} *and* Ψ_{S_e}, *we can construct* Ψ_S *in* $O(m\log\log|\Delta|+|\Delta|)$ *time and* $O(m\log|\Delta|+|\Delta|)$*-bit space.*

7. Improvement when $\log|\Sigma| = O((\log\log n)^{1-\epsilon})$. In case the alphabet size is small, precisely, when $\log|\Sigma| = O((\log\log n)^{1-\epsilon})$, we can improve the construction time of the CSA and the FM-index to $O(n)$, which is optimal. The improvement is based on the following data structure of Pagh for supporting constant-time rank queries [24].

THEOREM 5 (see [24]). *Given* n *distinct numbers in* $[0, m-1]$ *such that* $m = n\log^{O(1)} n$, *a data structure of size* $B + O(\frac{n(\log\log n)^2}{\log n})$ *bits supporting constant-*

time rank *queries can be constructed in* $O(n)$ *time and* $O(B)$*-bit space, where* $B = \lceil \log \binom{m}{n} \rceil = n \log \frac{m}{n} + O(n)$.

We apply the same algorithm as in section 4 for the construction of the CSA, but we make changes only in the encodings of Ψ_{T^i}, for $i < \log\log\log|\Sigma|$. For those values of i, we have $|T^i| = n/2^i$, and the alphabet size of T^i is $|\Sigma|^{2^i}$. When $\log|\Sigma| = O((\log\log n)^{1-\epsilon})$, we have $|\Sigma|^{2^i} = \log^{O(1)}|T^i|$.[5] Thus, the total increasing sequence of such Ψ'_{T^i}'s can be encoded by Theorem 5, and each backward search step on these Ψ functions can be done in constant time. This gives the following theorem.

THEOREM 6. *If* $\log|\Sigma| = O((\log\log n)^{1-\epsilon})$*, the CSA, FM-index and BW text* W *of* T *can be constructed in* $O(n)$ *time and* $O(n \log|\Sigma|)$*-bit working space.*

Proof. We refer to the algorithm in Figure 3. After the change in the encodings of Ψ_{T^i} for $i < \log\log\log|\Sigma|$, the time required by each phase is as follows.

- Phase 1 takes $O(n)$ time;
- for $i \geq \log\log\log|\Sigma|$, step i in phase 2 takes $O((n/2^i)\cdot(i + \log\log|\Sigma|) + |\Sigma|^{2^i})$ time;
- for $i < \log\log\log|\Sigma|$, step i in phase 2 takes $O(n/2^i + |\Sigma|^{2^i})$ time.

It follows that the total time required is $O(n)$. For the space complexity, it remains $O(n \log|\Sigma|)$ bits. Thus, the CSA and the BW text W of T can be constructed in the stated time and space, while the FM-index can be constructed in $O(n)$ time and $O(n \log|\Sigma|)$-bit space once W is obtained. This completes the proof. □

8. Concluding remarks. We have shown that suffix trees, suffix arrays, and other full-text indices can be constructed in $o(n \log n)$ time and $o(n \log n)$-bit space, giving a positive answer to an open problem.

Recently, linear-time algorithms for constructing suffix arrays have been proposed [15, 16, 14]. Though using interesting techniques, those algorithms require $O(n \log n)$-bit working space, and they will imply only $O(n \log^{\epsilon} n)$ time algorithms for constructing suffix arrays if the working space is limited to $O(n \log|\Sigma|)$ bits. Thus, the algorithm proposed in this paper is best suited for suffix array construction under practical consideration, where the input text is very long but the alphabet size is small.

One of the open problems remaining is whether we can construct a suffix tree in optimal $O(n)$ time for texts with general alphabet, while using optimal $O(n \log|\Sigma|)$-bit working space. Another direction of research is to further reduce the working space for constructing full-text indices from $O(n \log|\Sigma|)$ bits to an input-dependent $O(nH)$ bits, where H is the entropy of the input text.

Acknowledgments. The authors would like to thank Roberto Grossi, Tak-Wah Lam, Takeshi Tokuyama, and the anonymous reviewers for helpful comments, and we thank Ankur Gupta for sending us his paper.

REFERENCES

[1] P. BEAME AND F. E. FICH, *Optimal bounds for the predecessor problem and related problems*, J. Comput. System Sci., 65 (2002), pp. 38–72. Preliminary version appears in Proceedings of the ACM Symposium on Theory of Computing (STOC), 1999.

[2] M. BURROWS AND D. J. WHEELER, *A Block-Sorting Lossless Data Compression Algorithm*, Technical report 124, Digital Equipment Corporation, Palo Alto, CA, 1994.

[5] This can be seen by considering the boundary case of $i = \log\log\log|\Sigma|$, so that $|T^i|$ becomes smallest while the alphabet size becomes largest.

[3] A. Crauser and P. Ferragina, *A theoretical and experimental study on the construction of suffix arrays in external memory*, Algorithmica, 32 (2002), pp. 1–35.

[4] M. Farach, *Optimal suffix tree construction with large alphabets*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Miami Beach, FL, 1997, pp. 137–143.

[5] P. Ferragina and G. Manzini, *Indexing compressed text*, J. ACM, 52 (2005), pp. 552–581. Preliminary version appears in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), 2000.

[6] R. Grossi and J. S. Vitter, *Compressed suffix arrays and suffix trees with applications to text indexing and string matching*, SIAM J. Comput., 35 (2005), pp. 378–407. Preliminary version appears in Proceedings of the ACM Symposium on Theory of Computing (STOC), 2000.

[7] D. A. Grossman and O. Frieder, *Information Retrieval: Algorithms and Heuristics*, Kluwer Academic Publishers, Boston, MA, 1998.

[8] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, UK, 1997.

[9] T. Hagerup, *Sorting and searching on the word RAM*, in Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS), Springer, Berlin, 1998, pp. 366–398.

[10] T. Hagerup, P. B. Miltersen, and R. Pagh, *Deterministic dictionaries*, J. Algorithms, 41 (2001), pp. 69–85.

[11] W. K. Hon, *On the Construction and Application of Compressed Text Indexes*, Ph.D. Thesis, University of Hong Kong, Hong Kong, 2004.

[12] W. K. Hon and K. Sadakane, *Space-economical algorithms for finding maximal unique matches*, in Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM), Springer, Berlin, 2002, pp. 144–152.

[13] G. Jacobson, *Space-efficient static trees and graphs*, in Proceedings of the ACM Symposium on Foundations of Computer Science (STOC), 1989, pp. 549–554.

[14] J. Kärkkäinen, P. Sanders, and S. Burkhardt, *Linear work suffix array construction*, J. ACM, 53 (2006), pp. 918–936. Preliminary version appears in Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP), 2003.

[15] D. Kim, J. Sim, H. Park, and K. Park, *Constructing suffix arrays in linear time*, J. Discrete Algorithms, 3 (2005), pp. 126–142. Preliminary version appears in Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM), 2003.

[16] P. Ko and S. Aluru, *Space efficient linear time construction of suffix arrays*, J. Discrete Algorithms, 3 (2005), pp. 143–156. Preliminary version appears in Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM), 2003.

[17] S. Kurtz, *Reducing the space requirement of suffix trees*, Software Practice and Experiences, 29 (1999), pp. 1149–1171.

[18] T. W. Lam, K. Sadakane, W. K. Sung, and S. M. Yiu, *A space and time efficient algorithm for constructing compressed suffix arrays*, in Proceedings of the International Conference on Computing and Combinatorics, Springer, Berlin, 2002, pp. 401–410.

[19] U. Manber and G. Myers, *Suffix arrays: A new method for on-line string searches*, SIAM J. Comput., 22 (1993), pp. 935–948.

[20] E. M. McCreight, *A space-economical suffix tree construction algorithm*, J. ACM, 23 (1976), pp. 262–272.

[21] K. Mehlhorn, *Data Structures and Algorithms* 1: *Sorting and Searching*, Springer-Verlag, Heidelberg, Germany, 1984.

[22] J. I. Munro, *Tables*, in Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science, Springer, Berlin, 1996, pp. 37–42.

[23] J. C. Na and K. Park, *Alphabet-independent linear-time construction of compressed suffix arrays using $o(n \log n)$-bit working space*, Theoret. Comput. Sci., 385 (2007), pp. 127–136. Preliminary version appears in Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM), 2005.

[24] R. Pagh, *Low redundancy in static dictionaries with constant query time*, SIAM J. Comput., 31 (2001), pp. 353–363.

[25] R. Raman, V. Raman, and S. S. Rao, *Succinct indexable dictionaries with applications to encoding k-ary trees and multisets*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2002, pp. 233–242.

[26] K. Sadakane, *New text indexing functionalities of the compressed suffix arrays*, J. Algorithms, 48 (2003), pp. 294–313. Preliminary version appears in Proceedings of the Annual International Symposium on Algorithms and Computation (ISAAC), 2000.

[27] J. Seward, *The* `bzip2` *and* `libbzip2` *Official Home Page*, http://www.bzip.org/ (2008).
[28] S. Shimozono, H. Arimura, and S. Arikawa, *Efficient discovery of optimal word association patterns in large text databases*, New Generation Computing, 18 (2000), pp. 49–60.
[29] E. Ukkonen, *On-line construction of suffix trees*, Algorithmica, 14 (1995), pp. 249–260.
[30] P. Weiner, *Linear pattern matching algorithms*, in Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, Iowa City, IA, 1973, pp. 1–11.
[31] D. E. Willard, *Log-logarithmic worst-case range queries are possible in space* $\Theta(N)$, Inform. Process. Lett., 17 (1983), pp. 81–84.
[32] J. Zobel, A. Moffat, and K. Ramamohanarao, *Guidelines for presentation and comparison of indexing techniques*, SIGMOD Record, 25 (1996), pp. 10–15.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2179–2197

LINEAR-TIME HAPLOTYPE INFERENCE ON PEDIGREES WITHOUT RECOMBINATIONS AND MATING LOOPS*

MEE YEE CHAN†, WUN-TAT CHAN‡, FRANCIS Y. L. CHIN†, STANLEY P. Y. FUNG§, AND MING-YANG KAO¶

Abstract. In this paper, an optimal linear-time algorithm is presented to solve the haplotype inference problem for pedigree data when there are no recombinations and the pedigree has no mating loops. The approach is based on the use of graphs to capture SNP, Mendelian, and parity constraints of the given pedigree. This representation allows us to capture the constraints as the edges in a graph, rather than as a system of linear equations as in previous approaches. Graph traversals are then used to resolve the parity of these edges, resulting in an optimal running time.

Key words. computational biology, haplotype inference, pedigree, recombination

AMS subject classifications. 05C85, 68W01, 11Y16, 92D15

DOI. 10.1137/080680990

1. Introduction.

The modeling of human genetic variation is critical to the understanding of the genetic basis for complex diseases. *Single nucleotide polymorphisms* (SNPs) [5] are the most frequent form of this variation, and it is useful to analyze *haplotypes*, which are sequences of linked SNP genetic markers (small segments of DNA) on a single chromosome. In diploid organisms, such as humans, chromosomes come in pairs, and experiments often yield *genotypes*, which blend haplotypes for the chromosome pair. This gives rise to the problem of inferring haplotypes from genotypes.

Before defining our problem, some preliminary definitions are needed. The physical position of a marker on a chromosome is called a *locus* and its state is called an *allele*. Without loss of generality, the allele of a *biallelic* SNP can be denoted by 0 and 1, and a haplotype with m loci is represented as a length-m string in $\{0,1\}^m$, and a genotype as a length-m string in $\{0,1,2\}^m$. Haplotype pair $\langle h_1, h_2\rangle$ is *SNP-consistent* with genotype g if where the two alleles of h_1 and h_2 are the same at the same locus, say 0 (respectively, 1), the corresponding locus of g is also 0 (respectively, 1), which denotes a *homozygous* locus; otherwise, where the two alleles of h_1 and h_2 are different, the corresponding locus of g is 2, which denotes a *heterozygous* locus (i.e., SNP). A genotype with s heterozygous loci can have 2^{s-1} SNP-consistent haplotype solutions. For example, genotype $g = 012212$ with $s = 3$ has four SNP-consistent haplotype pairs: $\{\langle 01\underline{1}1\underline{1}1, 01\underline{0}0\underline{1}0\rangle, \langle 01\underline{1}1\underline{1}0, 01\underline{0}0\underline{1}1\rangle, \langle 01\underline{1}0\underline{1}1, 01\underline{0}1\underline{1}0\rangle, \langle 01\underline{1}0\underline{1}0, 01\underline{0}1\underline{1}1\rangle\}$.

A *pedigree* is a fundamental connected structure used in genetics. Figure 1 shows the pictorial representation of a pedigree with four nodes, with a square representing

*Received by the editors January 26, 2007, accepted for publication (in revised form) September 15, 2008; published electronically March 4, 2009.
http://www.siam.org/journals/sicomp/38-6/68099.html
†Department of Computer Science, University of Hong Kong, Hong Kong (mychan@cs.hku.hk, chin@cs.hku.hk). The third author's research was supported by Hong Kong RGC grant HKU-7119/05E and HKU Strategic Research Team Fund.
‡Department of Computer Science, King's College, University of London, London, UK (joseph.chan@kcl.ac.uk).
§Department of Computer Science, University of Leicester, Leicester, UK (pyfung@mcs.le.ac.uk).
¶Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 (kao@northwestern.edu).

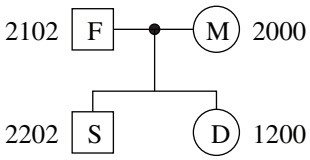


FIG. 1. *Example of a pedigree with four nodes.*

a male node and a circle representing a female node and children placed under their parents: in particular, a *father* (node F), a *mother* (node M), and two *children* (son node S and daughter node D). Each node in the pedigree is associated with a genotype. In Figure 1, for example, 2102 is the genotype for F and 2000 is the genotype for M. We assume that there are no *mating loops*; i.e., the pedigree does not contain loops. For example, marriage between descendants of a common ancestor forms a mating loop. However, polygamy or remarriage is allowed in the sense that stepchildren can exist. A precise definition of a mating loop will be given in section 2. Note that mating loops are rare in real data sets, especially for humans [2].

A *consistent haplotype configuration* (with no recombinations) for a given pedigree is an assignment of a pair of haplotypes to each individual node such that (i) all the haplotype pairs are SNP-consistent with their corresponding genotypes and (ii) the haplotypes of each child are *Mendelian-consistent*; i.e., one of the child's haplotype is exactly the same as one of its father's and the other is the same as one of its mother's.

Haplotyping Pedigree Data (with No Recombinations) Problem (HPD-NR): Given a pedigree P where each individual node of P is associated with a genotype, find a consistent haplotype configuration (CHC) for P.

Wijsman [7] proposed a 20-rule algorithm, and O'Connell [4] described a genotype elimination algorithm, both of which can be used for solving the HPD-NR problem. Li and Jiang [2] formulated the problem as a system of linear equations with $O(mn)$ equations and $O(mn)$ variables, where n is the number of individuals in the pedigree and m is the number of loci for each individual. The equations are then solved by Gaussian elimination. This gives a $O(m^3n^3)$ time algorithm. Xiao, Liu, Xia, and Jiang [8] later improved the time complexity to $O(mn^2 + n^3 \log^2 n \log\log n)$. For the case without mating loops, their algorithm runs in $O(mn^2 + n^3)$ time.

It has long been conjectured that an $O(mn)$ time algorithm exists, but it should be appreciated that finding such an algorithm has been elusive and far less straightforward than many researchers have initially thought.

In this paper, we propose a new 4-stage algorithm that can either find a CHC solution or report "no solution" in optimal $O(mn)$ time when the pedigree has no mating loops. The main idea of our algorithm is to construct a tree to model the predigree. Each vertex in the tree represents a haplotype; i.e., each genotype corresponds to a pair of vertices. Different types of edges are added between the nodes to enforce the SNP and Mendelian consistencies. This is carried out gradually in Stages 1 and 2 in our algorithm, and Stage 3 is to add more edges to unify vertices of the same haplotype so that a connected tree is formed. The main difficulty in our approach is to resolve the correct alleles at the heterozygous loci in the CHC solution. We have developed a routine to resolve the heterozygous loci for tree vertices in a connected component, which is executed in Stages 2, 3, and 4. As a connected tree is formed after Stage 3, either a CHC solution is constructed or "no solution" is reported. This approach allows us to find a CHC solution without (directly) solving a system of linear equations, as in previous approaches [2, 8].

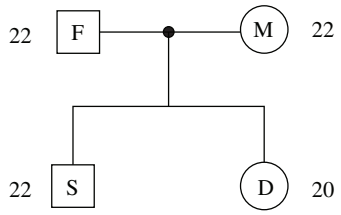


FIG. 2. *Pedigree with a family problem.*

2. Preliminaries.

DEFINITION 1 (pedigree graph and mating loop). *A* pedigree graph *is a graph derived naturally from the pedigree as follows. Each individual in the pedigree becomes a node in the graph. Whenever two individuals mate and produce children, there is an additional* mating node*, and there are undirected edges connecting the mating node to the two parents as well as to each of the children. A* mating loop *is a cycle in the pedigree graph, and a pedigree does not have a mating loop if the associated pedigree graph does not have cycles; i.e., the pedigree graph is a tree. A* trio *consists of a father, a mother and one of their children. A* nuclear family *consists of a father, a mother, and all of their (shared) children.*

We assume without loss of generality that the pedigree, and hence the pedigree graph, is connected.

DEFINITION 2 (family problem). *If there exists a family with father* F*, mother* M*, and two children* C_1 *and* C_2 *in the pedigree and two loci i and j such that i and j are heterozygous in* F*,* M*, and* C_1 *but are homozygous and heterozygous, respectively, in* C_2*, then we say that the pedigree has a* family problem*.*

Figure 2 gives a simple example of a pedigree with a family problem. It can be easily checked that this, and any other pedigree with a family problem, has no CHC solution.

For each trio T, we define $het(T)$ as the set of all loci that are heterozygous for the father, the mother, and the child in T, and $hom(T)$ as the set of all loci that are heterozygous for the father and the mother but homozygous for the child. These two sets for all trios can be computed easily in $O(mn)$ time.

Consider a nuclear family, which consists of a number of trios. The following observation is crucial: the nuclear family has no family problem if and only if for any two trios T_i, T_j in the family, $het(T_i)$ and $het(T_j)$ are either identical or disjoint. Note that $het(T_i) \cup hom(T_i) = het(T_j) \cup hom(T_j)$. Using this observation, we can check the pedigree for family problems in $O(mn)$ time as follows.

LEMMA 1. *The family problems in the pedigree can be identified in* $O(mn)$ *time.*

Proof. Consider each nuclear family in the pedigree. We maintain sets $S_0, S_1, \ldots$ of trios where trios belonging to the same S_i have identical $het()$'s and trios belonging to different S_i's have disjoint $het()$'s. We extend the definition of $het()$ and $hom()$ to S_i naturally: $het(S_i)$ is the set of loci in $het(T)$ for $T \in S_i$, and $hom(S_i)$ is similarly defined. S_0 is a special set of trios with empty $het()$, and initially $S_1 = \{T_1\}$ where T_1 is a trio with non-empty $het()$. We then consider each trio one by one, and do the following:

For each new trio T,
- (a) If $het(T)$ is empty, add T to S_0 and go to next T.
- (b) Otherwise, for each S_i, $i \geq 1$,
  - (i) If some loci in $het(S_i)$ are in $hom(T)$ but some are in $het(T)$, report *"family problem"* and halt.

(ii) Else if all loci in $het(S_i)$ are in $hom(T)$, go to next i. If this is the last i, create a new S_j with T as a member, and go to next T.
(iii) Else (all loci in $het(S_i)$ are in $het(T)$) check whether all other loci in T are homozygous. If this is false, then report *"family problem"* and halt. If this is true, add this T as a member of S_i, and skip to next T (no need to test the other S_j's).

The processing of each trio takes $O(m)$ time even though it may need to compare with all S_i (and there can be $O(n)$ of them), because the running time of steps (b)(i) and (b)(ii) is $O(|het(S_i)|)$, and the sum of all $|het(S_i)|$ is at most m. Thus the checking of family problems over the entire pedigree takes $O(mn)$ time. □

3. The algorithm.

3.1. Stage 1—setting up the local graph *G*.

Stage 1A—Checking for family problems. Our algorithm begins by checking for family problems. Only if there are no family problems will the algorithm continue; otherwise, "no solution" is reported.

Stage 1B—Generating vector-pairs. For each trio in the given pedigree, let the respective genotypes of the father F, the mother M, and the child C be: $x_1x_2\dots x_m$, $y_1y_2\dots y_m$, and $z_1z_2\dots z_m$ where $x_i, y_i, z_i \in \{0,1,2\}$. We determine a pair of vectors (or vector-pair) each for the father, the mother, and the child, namely: $\langle f_1, f_2\rangle$, $\langle m_1, m_2\rangle$, and $\langle c_1, c_2\rangle$, respectively, where $f_1 = x_{1,1}x_{1,2}\dots x_{1,m}$ and $f_2 = x_{2,1}x_{2,2}\dots x_{2,m}$; $m_1 = y_{1,1}y_{1,2}\dots y_{1,m}$ and $m_2 = y_{2,1}y_{2,2}\dots y_{2,m}$; $c_1 = z_{1,1}z_{1,2}\dots z_{1,m}$ and $c_2 = z_{2,1}z_{2,2}\dots z_{2,m}$. The vector-pairs are determined in the following manner.

1. For each locus i, for f_1 and f_2:
   (a) If $x_i = 0$, then $x_{1,i} = x_{2,i} = 0$.
   (b) If $x_i = 1$, then $x_{1,i} = x_{2,i} = 1$.
   (c) If $x_i = 2$ and $z_i = 0$, then $x_{1,i} = 0$ and $x_{2,i} = 1$.
   (d) If $x_i = 2$ and $z_i = 1$, then $x_{1,i} = 1$ and $x_{2,i} = 0$.
   (e) If $x_i = 2$ and $z_i = 2$ and $y_i = 0$, then $x_{1,i} = 1$ and $x_{2,i} = 0$.
   (f) If $x_i = 2$ and $z_i = 2$ and $y_i = 1$, then $x_{1,i} = 0$ and $x_{2,i} = 1$.
   (g) If $x_i = 2$ and $z_i = 2$ and $y_i = 2$, then $x_{1,i} = ?$ and $x_{2,i} = ?$.
2. m_1 and m_2 are similarly determined.
3. Set $\langle c_1, c_2\rangle = \langle f_1, m_1\rangle$. Check that $\langle c_1, c_2\rangle$ is consistent with C's genotype $z_1z_2\dots z_m$; otherwise, report *"no solution."* □

The vector-pairs are the initial assignment of haplotypes, assuming that C inherits f_1 from F and m_1 from M, i.e., $\langle c_1, c_2\rangle = \langle f_1, m_1\rangle$. For example, in Step 1(c), $z_i = 0$ and, hence, $z_{1,i} = z_{2,i} = 0$. Since c_1 is inherited from f_1, we can conclude $x_{1,i} = 0$ and, therefore, $x_{2,i} = 1$. When there is not enough information to deduce the value of a locus in the haplotype, a ? is used.

Observe that if a particular node N in the pedigree belongs to k different trios, then k vector-pairs, or $2k$ vectors, will be created for N in Stage 1B. These need to be unified eventually as a single vector-pair, because there is only one pair of haplotype for each node. This will be handled later when Endgame-consistency is defined, but first notice that we can define SNP-consistency and Mendelian-consistency in terms of vector-pairs. Let Φ(N) be the multiset comprised of these k vector-pairs of a node N. It is sometimes convenient to refer to the vectors rather than the vector-pairs. Thus, we let Γ(N) be the multiset of $2k$ vectors, containing the two vectors of each vector-pair in Φ(N).

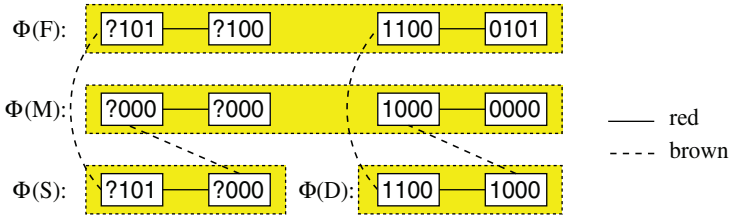


FIG. 3. *Graph G for Example* 1.

DEFINITION 3 (SNP-consistency condition). SNP-consistency is said to be maintained *if and only if, for all nodes* N *in the pedigree, each vector-pair in* Φ(N) *is SNP-consistent with* N*'s genotype. Vector-pair* $\langle h_1, h_2 \rangle$ *is said to be* SNP-consistent *with genotype g if either* (i) *if* h_1 *and* h_2 *are both 0 (respectively,* 1*) at the same locus, then the corresponding locus of g is also* 0 *(respectively,* 1*); or* (ii) *if* h_1 *is* 0 *(respectively,* 1*) and* h_2 *is* 1 *(respectively,* 0*) at the same locus, then the corresponding locus of g is* 2.

DEFINITION 4 (Mendelian-consistency condition [1, 6]). Mendelian-consistency is said to be maintained *if and only if, for all nodes* N *in the pedigree,* N *is a child in a trio comprised of* F, M, *and* N, *then* Φ(N) *contains a vector-pair* $\langle c_1, c_2 \rangle = \langle f_1, m_1 \rangle$ *where* $f_1 \in \Gamma(\mathrm{F})$ *and* $m_1 \in \Gamma(\mathrm{M})$.

Stage 1C—Constructing the local graph $G = (V, E)$. Let V be the multiset of all the vectors created in Stage 1B, and let E be the set of *red* and *brown* edges defined below:

1. A **red** edge is introduced to join the two vectors of each vector-pair generated in Stage 1A. It indicates that a ? appearing at locus i of both vectors must be resolved differently in the later stages of the algorithm (the two vectors can be different or the same at the other non-? locus positions). The red edges enforce SNP-consistency.
2. For each F-M-C trio, let $\langle f_1, f_2 \rangle$, $\langle m_1, m_2 \rangle$, and $\langle c_1, c_2 \rangle$ be vector-pairs in Φ(F), Φ(M), and Φ(C), respectively, associated with this trio. Two **brown** edges are introduced, one connecting c_1 and f_1, and the other connecting c_2 and m_1. A brown edge between two vectors means that the two vectors must be the same at *all* locus positions. The brown edges enforce Mendelian-consistency. □

Example 1. Consider the pedigree with F (father), M (mother), S (son), and D (daughter) shown in Figure 1. Stage 1 produces the graph G in Figure 3 with 12 vertices and 10 edges (6 red and 4 brown), comprised of two connected components, one for each of the two trios, F-M-S and F-M-D, in the pedigree.

DEFINITION 5. *For any locus i in a connected component* $\mathcal{G}$ *of G, we say*

1. *Locus i is* resolved *in* $\mathcal{G}$ *if and only if all vectors in* $\mathcal{G}$ *have* 0 *or* 1 *at locus i.*
2. *Locus i is* unresolved *in* $\mathcal{G}$ *if and only if all vectors in* $\mathcal{G}$ *have* ? *at locus i.*
3. *Otherwise, locus i is* mixed *(it is a mix of* ? *and non-*? *at i).*

In Example 1, the connected component for trio F-M-S has one unresolved locus (locus 1) and three resolved loci (loci 2, 3, and 4). Meanwhile, the component for trio F-M-D has no unresolved loci and four resolved loci (loci 1, 2, 3, and 4).

LEMMA 2. *The time complexity of Stage* 1 *(Stages* 1*A,* 1*B, and* 1*C) is* $O(mn)$, *where n is the number of nodes in the pedigree and m is the number of loci in each genotype. Furthermore, after Stage* 1*, all loci are either resolved or unresolved in each*

connected component of G (no mixed loci), and each connected component has six vertices. G has $O(n)$ vertices and edges and is acyclic.

Proof. Checking for family problems takes $O(mn)$ time (Lemma 1). With $O(n)$ trios and six vectors generated per trio with each vector having m loci, Stage 1B takes $O(mn)$ time. With $O(n)$ trios and six vertices and five edges (three red and two brown) per trio introduced in G, Stage 1C takes $O(n)$ time altogether, and furthermore, G has $O(n)$ vertices and edges. G is acyclic because each connected component of G is a path consisting of 6 vectors, 3 red edges, and 2 brown edges of a trio. □

In later stages of our algorithm, no vector-pairs will be added to or deleted from each $\Phi(\mathrm{N})$, and all loci resolved in Stage 1 will remain unchanged. Components of G will later be merged with the addition of **green** (added in Stage 2) or **white** (added in Stage 3) edges until G becomes a single connected component. Before we explain why and how these edges are added, we first note that these new edges can always be added between two vectors in the same $\Gamma(\mathrm{N})$. This structured way of adding edges to make G connected is possible given Lemma 3 below.

LEMMA 3. *If G has more than one connected component, then there exists a node* N *such that there are two vector-pairs in* $\Phi(\mathrm{N})$ *which belong to two different connected components.*

Proof. Suppose to the contrary that, for all N, the vector-pairs in $\Phi(\mathrm{N})$ are all connected. We make use of the fact that the brown edges in G preserve the connectivity of any two nodes in the pedigree, which we have assumed to be connected. Therefore, if vector-pairs in $\Phi(\mathrm{N})$ are all connected for all N, then all vectors are connected together in a single connected component, which contradicts the assumption that G has more than one connected component. □

There are two reasons why we need to merge the connected components of G. First, each multiset $\Phi(\mathrm{N})$ may contain more than one vector-pair; precisely, it contains k vector-pairs if N belongs to k different trios. However, by the time all loci are resolved, for all nodes N, each multiset $\Phi(\mathrm{N})$ must contain k copies of one unique vector-pair $\langle h_1, h_2\rangle$, which represents the haplotype-pair in a CHC for N. The green and white edges enforce this constraint by connecting vectors in $\Gamma(\mathrm{N})$ that are supposed to be identical (because they have identical values at some resolved heterozygous loci). For example, consider two vector-pairs $\langle u_1, u_2\rangle$, $\langle v_1, v_2\rangle$ of a node N. If at a heterozygous locus i, u_1 is 0 and v_1 is also 0, then we know u_1 must be identical to v_1 (and u_2 identical to v_2). However, if there is another heterozygous locus j where u_1 is 0 and v_1 is 1, then it is impossible to give a unique vector-pair, and there is no CHC solution. We capture this observation by defining the following type of consistency:

DEFINITION 6 (endgame-consistency condition). *Vector-pairs $\langle u_1, u_2\rangle$, $\langle v_1, v_2\rangle \in \Phi(\mathrm{N})$ of a node N are said to be* Endgame-inconsistent *if the vector values at some heterozygous loci i and j $(i \neq j)$ for u_1, u_2, v_1, and v_2 are a permutation of the four possibilities:* $00, 01, 10$, *and* 11, *and* Endgame-consistent *otherwise. The node* N *is said to be* Endgame-inconsistent *if there exist vector-pairs in* $\Phi(\mathrm{N})$ *that are Endgame-inconsistent, and* Endgame-consistent *otherwise.* Endgame-consistency is said to be maintained *if and only if, for all nodes* N *in the pedigree,* N *is Endgame-consistent.*

The second reason of adding green and white edges to connect the components of G is to further resolve the unresolved loci of each connected component of G with SNP-consistency and Mendelian-consistency maintained: vectors connected by green or white edges are supposed to be identical, and if a locus is resolved in one of the connected components but not in the other, the new connection allows us to resolve the locus in the other connected component as well. In the next subsection we will develop a procedure for resolving the values of loci in a connected component of G.

Our algorithm achieves a solution if, at the end of Stage 4, (a) graph G comprises a single connected component; (b) all loci are resolved in G; and (c) SNP-consistency, Mendelian-consistency, and Endgame-consistency are maintained. However, our algorithm may report "no solution" if some N is Endgame-inconsistent before the end of Stage 4.

3.2. Stage 2—Adding green edges. One of the most important aspects of our algorithm is that, at all stages, we maintain the property that each connected component of G has only resolved and unresolved loci (i.e., no mixed loci). In order to do this, we make extensive use of a subroutine called LOCUS_RESOLVE. LOCUS_RESOLVE($\mathcal{G}$) attempts to resolve ?'s in a connected component $\mathcal{G}$ of G. It looks at each locus in turn, identifies a resolved locus, and uses this to resolve the locus at other vertices in the connected component by traversing in a manner consistent with the colors of the edges. We do not lose any feasible solution in this procedure because any feasible solution must satisfy SNP-consistency, Mendelian-consistency, and Endgame-consistency, which are specified by the colors of the edges.

Define $v(i)$ to be the value of locus i ($= 0, 1$, or ?) at vector v.

LOCUS_RESOLVE($\mathcal{G}$):

For each locus i:

1. Traverse the connected component $\mathcal{G}$ to find a vector v where $v(i)$ is resolved ($= 0$ or 1). If no such v exists, go to the next locus.
2. Traverse $\mathcal{G}$ using a linear-time graph traversal procedure (such as depth-first search), starting at v. For any edge $e = (v_1, v_2)$ traversed where $v_1(i) = x$ (0 or 1) and $v_2(i) = ?$,
   (i) If e is a red edge, set $v_2(i) = 1 - x$.
   (ii) Else set $v_2(i) = x$.

LEMMA 4. *LOCUS_RESOLVE($\mathcal{G}$) runs in $O(|\mathcal{G}|m)$ time, where $|\mathcal{G}|$ is the number of vectors in $\mathcal{G}$. All loci in $\mathcal{G}$ are either resolved or unresolved after running the procedure.*

Proof. LOCUS_RESOLVE performs m graph traversals, one for each locus, and each traversal takes $O(\mathcal{G})$ time. Hence the time complexity. At any locus i, if at least one vector is resolved at i, this will be identified in Step 1, and since $\mathcal{G}$ is connected, all other vectors will then be resolved at locus i in Step 2. □

Stage 2 will consider the nuclear families in the pedigree one by one and will try to connect the trios within the same nuclear family, in such a way as to respect Endgame-consistency. Specifically, green edges are added to connect two unconnected vectors in Γ(N) that have the value 0 at heterozygous locus i of N, where N is the father or mother of the nuclear family. Green edges are like brown edges requiring that the ?s in the two vectors connected by the edge to be resolved the same.

There are two types of nuclear families: namely, the **Type A** families where there exists a locus that is heterozygous in either one of, but not both, the parents; and the **Type B** families where each locus is either heterozygous in both parents or homozygous in both parents. Stage 2 consists of Stages 2A and 2B, which process all Type A and Type B nuclear families, respectively.

For each Type A nuclear family, there exists a locus i that is heterozygous in either of, but not both of, F and M. Observe that locus i will be resolved in all vectors of the nuclear family comprised of father F, mother M and their children (that is how Stage 1B works). We can, therefore, connect all the vector-pairs into one single connected component by adding green edges between vectors in Γ(N) with 0 at locus i, where N is the parent with heterozygous locus i. This we do in Stage 2A.

Stage 2A—Processing Type A families. For each nuclear family, which is comprised of, say, father F, mother M, and their children, where there exists a locus i that is heterozygous in either (not both) of F and M, do the following:

1. Let $V = \{v \in \Gamma(\mathrm{N}) \mid v(i) = 0$ and v belongs to this nuclear family$\}$ where N is the parent (either M or F) such that i is heterozygous in N. Pick one vector from V and call it u.
2. For each vector $v \in V$, $v \neq u$, add a green edge to join u and v. After this, the vectors in the nuclear family are connected as a single connected component.
3. Run LOCUS_RESOLVE($\mathcal{G}$) where $\mathcal{G}$ is the connected component containing u.
4. Check for Endgame-consistency within the family, reporting *"no solution"* if it is not maintained. □

For each Type B nuclear family, we make use of the sets $het()$ and $hom()$ defined in section 2. If there is a trio T where $hom(T)$ is empty, then $het(T)$ contains all loci (where F and M are also heterozygous). Since all distinct $het()$'s are disjoint, this implies any other trio T' in the nuclear family either has $het(T')$ empty, or $het(T') = het(T)$ (and $hom(T')$ is empty). In this case all trios with empty $hom()$ cannot be connected (in Stage 2B). So in Stage 2B we consider only trios where $hom(T)$ is nonempty.

We first consider each S_i as defined in the proof of Lemma 1, and connect all trios in the same S_i by adding green edges between them. All trios in the same S_i have identical $het()$ and hence identical (and nonempty) $hom()$; thus they share a resolved locus, which allows us to add a green edge correctly.

Then we will add edges connecting S_i and S_{i+1} for all i. If S_i and S_{i+1} share some common locus in their $hom()$'s, then this allows us to add a green edge correctly using the resolved loci. If they do not share any common locus in their $hom()$'s, then this implies that the union of their $het()$'s equals the set of all loci (where F and M are heterozygous). This means that they are the only two S_i's (call them S_1 and S_2) which have nonempty $het()$. If there are trios in S_0 (which has an empty $het()$), then S_0 shares some common resolved locus with both S_1 and S_2 and all S_i's can be connected by green edges. Otherwise if there are no trios in S_0, then S_1 and S_2 cannot be connected together.

Stage 2B—Processing Type B families. For each nuclear family, which is comprised of, say, father F, mother M, and their children, where each locus is either homozygous in both F and M, or heterozygous in both F and M:

1. If there exists a locus i that is heterozygous in F (and also M), then do the following:
   (a) Let the sets of loci $het()$ and $hom()$ of each trio and the S_i's be as defined in the proof of Lemma 1.
   (b) For each S_i with corresponding trios $T_1, T_2, \ldots, T_k$:
      i. Pick a locus x in $hom(T_1)$. This is also in $hom(T_j)$ for all other j. If such x does not exist, go to the next S_i.
      ii. For each trio T_j other than T_1, add a green edge to join u and v, where $u, v \in \Gamma(\mathrm{F})$ are vectors for trios T_1 and T_j with value 0 at locus x.
   (c) For each pair of S_i and S_{i+1}:
      (i) Pick a T in S_i and a T' in S_{i+1}.
      (ii) If $hom(T)$ and $hom(T')$ share some common locus x, then add a green edge to join u and v, where $u, v \in \Gamma(\mathrm{F})$ are vectors for trios T and T' with value 0 at locus x.

(iii) Otherwise (i.e., no such common locus exists), S_i and S_{i+1} are the only two such sets with a nonempty $het()$. Call them S_1 and S_2. If there are trios in S_0, then add a green edge between S_1 and S_0, and between S_2 and S_0, as in Step (c)(ii). If there are no trios in S_0, then no green edges are added.

(d) Run LOCUS_RESOLVE() on each connected component of G of the nuclear family with the green edges added above.

(e) Check for Endgame-consistency within the family, reporting *"no solution"* if it is not maintained.

2. Otherwise, no green edges are added for this family. □

LEMMA 5. *The time complexity of Stage* 2 *(Stages* 2*A and* 2*B) is* $O(mn)$. *Furthermore, after Stage* 2*, all loci are either resolved or unresolved in each connected component of* G*, and* G *has* $O(n)$ *vertices and edges, and is acyclic.*

Proof. Stage 2A considers the nuclear families of the pedigree one by one. For each nuclear family, with, say, k children: locus i can be determined in $O(m)$ time; Step 1 takes $O(k)$ time with V containing one vector per trio of the family; Step 2 takes $O(k)$ time; Step 3 takes $O(km)$ time; and Step 4, which checks for Endgame-consistency, can be done in $O(km)$ time. Therefore, the total time complexity of Stage 2A is $O(mn)$.

Stage 2B processes the nuclear families similarly. Each addition of a green edge in Steps 1(b) and 1(c) takes $O(m)$ time, and thus for a nuclear family with k children, Steps 1(b) and 1(c) take $O(km)$ time. Steps 1(d) and 1(e) take $O(mn)$ time as in Stage 2A. Hence the time complexity of Stage 2B is also $O(mn)$.

The execution of LOCUS_RESOLVE after green edges are added ensures all loci are either resolved or unresolved in each connected component of G. No vertices are added to G and only up to $k-1$ green edges are added for each family with k children. Thus, G continues to have $O(n)$ vertices and edges. All green edges are added between vectors of the same individual node, and within a nuclear family, green edges are added in either the father or the mother but not both. Hence, in the absence of mating loops, G remains acyclic. □

LEMMA 6. *If a connected component* $\mathcal{G}$ *of* G *has only resolved and unresolved loci (no mixed loci), then all possible ways of resolving ?'s in vectors in* $\mathcal{G}$ *such that SNP-consistency and Mendelian-consistency are maintained will either all make all vector-pairs Endgame-consistent or all make some vector-pairs Endgame-inconsistent.*

Proof. Consider a particular resolution of ?'s in the vectors in $\mathcal{G}$ such that SNP-consistency and Mendelian-consistency are maintained. Suppose Endgame-inconsistency occurs at node N; i.e., there exist two vector-pairs $\langle x_1, x_2\rangle$, $\langle y_1, y_2\rangle \in \Phi(\mathrm{N})$ such that the vector values at some heterozygous loci i and j $(i \neq j)$ for x_1, x_2, y_1, and y_2 are a permutation of the four possibilities: 00, 01, 10, and 11. We can assume, without loss of generality, that the value at such i and j for x_1, x_2, y_1, and y_2 are 00, 11, 01, and 10, respectively. Consider the following three cases for the state of loci i and j prior to the resolution:

Case 1: Loci i and j were both unresolved in $\mathcal{G}$. Then, for all other possible resolutions, the values at loci i and j for x_1, x_2, y_1, and y_2 would either be 00, 11, 01, and 10 respectively, or 11, 00, 10, and 01, respectively, and Endgame-consistency would also be violated.

Case 2: Only one of locus i and j was unresolved, say i, in $\mathcal{G}$. Then, for all other possible resolutions, the values at loci i and j for x_1, x_2, y_1, and y_2 would either be 00, 11, 01, and 10 respectively, or 10, 01, 11, and 00, respectively, and Endgame-consistency would also be violated.

Case 3: Both loci i and j were resolved. Then, Endgame-inconsistency existed prior to any resolution of ?'s. □

The green edges are essential to the next stage of our algorithm by ensuring that certain trios are connected. This is established in Lemmas 7 and 8 below.

LEMMA 7. *Consider a nuclear family within the pedigree with father* F*, mother* M*, and their children. If there exists a locus i that is heterozygous in either one (not both) of* F *and* M*, then after Stage* 2*A, the family will be represented by a single connected component in G.*

Proof. This follows straightforwardly from how Stage 2A works. □

LEMMA 8. *Consider a nuclear family within the pedigree comprised of father* F*, mother* M*, and their children and with no family problems. If there exists a locus that is heterozygous in both* F *and* M *but homozygous in both* C_1 *and* C_2*,* C_1 *and* C_2 *being children of* F *and* M*, then the components for trios* F-M-C_1 *and* F-M-C_2 *will become connected during Stage* 2*B.*

Proof. All trios with a nonempty *hom*() will be connected to other trios within the same S_i in Step 1(b) of Stage 2B, which in turn will be connected to all other S_j's in Step 1(c) of Stage 2B. All such S_i's will be connected to a single connected component since any two of them must share at least one common locus in their *hom*() (since otherwise, all trios will form into two S_1 and S_2 with disjoint *het*() and *hom*(); see the discussion just before Stage 2B is defined). Thus the two trios, which share a common locus in their *hom*()'s, will be connected in Stage 2B. □

The previous two lemmas lead up to Lemma 9 below, which defines the Mother-Father Property. The Mother-Father Property helps us from not having to check for Endgame-consistency for the mother if the father is Endgame-consistent, or vice versa. This will become important in Stage 3.

LEMMA 9 (mother-father property). *Suppose (a)* M *and* F *are the mother and father of two unconnected trios in G after Stage* 2 *and (b) the given pedigree has no family problems. Then, for all possible way(s) of resolving ?s in vectors in the two trios such that SNP-consistency and Mendelian-consistency are maintained,* M *and* F *are either both Endgame-consistent or both Endgame-inconsistent.*

Proof. Suppose F is Endgame-inconsistent. Without loss of generality, let the values at loci i and j for x_1, x_2, y_1, and y_2 be 00, 11, 01, and 10, respectively, where $\langle x_1, x_2\rangle$, $\langle y_1, y_2\rangle \in \Phi(\mathrm{F})$. This means that loci i and j are heterozygous loci for F. Since the two trios are not connected by a green edge, loci i and j are also heterozygous for M (Lemma 7). Let C_1 and C_2 be the two respective children of F connected to $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ by brown edges. In the absence of family problems and green edges connecting the two trios, there are only three cases to consider (there is no need to consider i or j being homozygous for both C_1 and C_2 according to Lemma 8): (i) when loci i and j are both heterozygous for both C_1 and C_2; (ii) when loci i and j are both heterozygous for C_1 and both homozygous for C_2; and (iii) when locus i is heterozygous for C_1 and homozygous for C_2 while locus j is homozygous for C_1 and heterozygous for C_2. It can be readily shown that in all three cases, M is also Endgame-inconsistent.

The argument is similar supposing M is Endgame-inconsistent. Thus, if F (or M) is Endgame-inconsistent, then M (respectively, F) is Endgame-inconsistent, and the contrapositive implies that, if M (or F) is Endgame-consistent, then F (respectively, M) is Endgame-consistent. The lemma follows. □

3.3. Stage 3—Adding white edges. After Stage 2, suppose G is left with more than one connected component. The idea of Stage 3 is to connect components

of G together with white edges, so that a single connected component results and loci can be further resolved. Before we present the various substages of Stage 3 formally, we first give an intuitive idea.

Suppose a pedigree has a CHC solution. Then for any node N with vector-pairs $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$, if these vector-pairs are not already connected by green edges in Stage 2, then we need to add a white edge to connect either u_1 to v_1, or connect u_1 to v_2. These represent the two different ways of resolving the haplotypes in N so as to maintain Endgame-consistency (i.e., either u_1 should be identical to v_1, or it should be identical to v_2). Thus white edges are analogous to green edges and they are treated as "nonred" edges by LOCUS_RESOLVE.

While it may appear at first sight that each of these white edges can be added arbitrarily, it turns out that this is not true when multiple white edges are considered together, and we need a way to determine which of the two ways is the correct way of connecting the vectors. To do this, we first construct a **support graph *H*** in Stage 3A. The support graph contains unlabeled edges, each corresponding to a white edge in G, and which will be labeled with either 0 or 1 in Stage 3C. Suppose e is an unlabeled edge in H corresponding to the white edge between the vector-pairs $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ as defined in the previous paragraph. A label of 0 on e denotes that the white edge in G should connect u_1 and v_1, while a label of 1 denotes that the white edge should connect u_1 to v_2. This is how H is used. In order to find this labeling, we will construct another graph J in Stage 3B which captures the constraints on how the unlabeled edges can be labeled.

We start with the construction of H in Stage 3A.

Stage 3A—Constructing the support graph *H*.

1. For each nuclear family:
   If the vector-pairs in the family consist of $k > 1$ connected components,[1] then do the following. Pick a vector from each of the connected components in either the father or the mother, but not both (all vectors must be from the same parent). Create a vertex in H for each such vector. Add $k-1$ *unlabeled edges* to join these vertices in H.
2. For each pedigree node N:
   Suppose this individual N belongs to k' different nuclear families. Within each such nuclear family, any two vectors of N in G are either already connected in Stage 2, or they belong to connected components with corresponding vertices in H that are connected in Step 1 above. Vectors of N from different nuclear families are, however, not connected in either G or H. Pick one vector from N from each of these nuclear families. Create a vertex in H for each such vector. Add $k'-1$ new *unlabeled edges* to connect them in H.
3. For each connected component $\mathcal{G}$ in G:
   (a) Let k'' be the number of vectors in $\mathcal{G}$ that are chosen as vertices in H in Steps 1 or 2 above.
   (b) Join these vertices in H with $k''-1$ *labeled edges*. The label is 0 if the path between the two vectors in G has an even number of red edges, and 1 otherwise. □

[1] After Stage 2, Type A families have $k = 1$, and vector-pairs of each Type B family are connected into at most two connected components except those trios whose $hom()$ is empty, which are still unconnected.

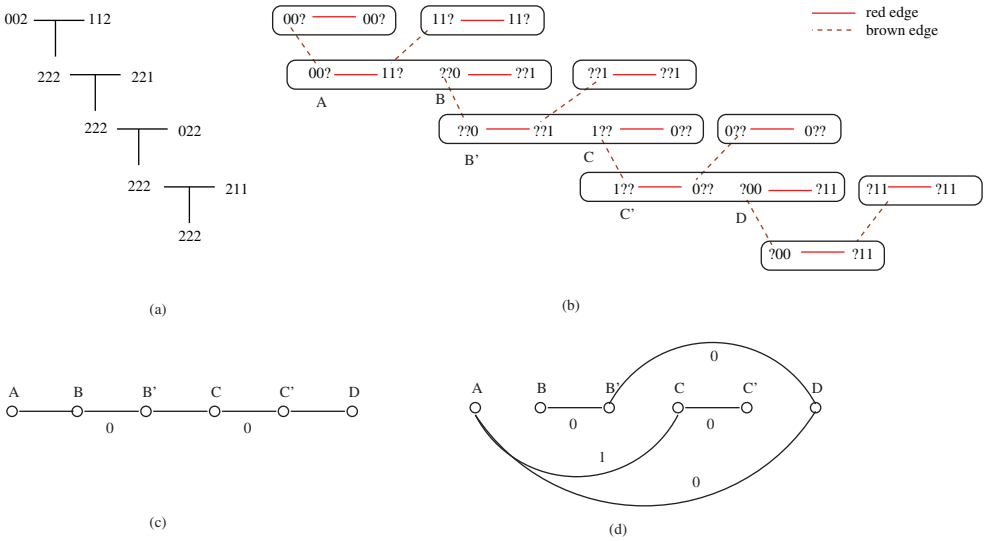


FIG. 4. *An example showing the steps in Stage* 3. (a) *The pedigree.* (b) *The local graph* G. (c) *The support graph* H. *Two edges are labeled with* 0. (d) *The parity constraint graph* J. *Three constraints are added.*

Figures 4(a) and 4(b) show a pedigree and the corresponding local graph G. Figure 4(c) shows the support graph H. In this example, since each nuclear family has only one connected component, Step 1 is skipped. Unlabeled edges (A, B), (B', C), and (C', D) are added in Step 2, and labeled edges (B, B') and (C, C') with label 0 are added in Step 3. In this case H happens to be a path, but it can be more general. Lemma 10 shows that H is always acyclic.

LEMMA 10. *If there are no mating loops in the pedigree, then* H *is acyclic.*

Proof. If there is a cycle in H, it cannot involve only vectors in one nuclear family, by our construction (Step 1 of Stage 3A). Any other cycle is impossible without mating loops. □

LEMMA 11. H *is connected, has* $O(n)$ *vertices and edges, and can be constructed in* $O(n)$ *time.*

Proof. Vertices in H which correspond to vectors in the same connected component in G are connected by labeled edges. The different connected components of G can always be connected by adding edges joining vectors in the same Φ(N) for some node N (Lemma 3); these correspond to the unlabeled edges in H. Hence, H is connected.

H clearly has $O(n)$ vertices since the set of vertices is a subset of those of G. Since H has no cycles by Lemma 10, it has $O(n)$ edges.

Steps 1 and 2 of Stage 3A take $O(n)$ time since the time to process each nuclear family or individual is proportional to the number of vectors in them. We can check for connectivity easily by preprocessing (e.g., traversing G to identify connected components). Step 3 also takes $O(n)$ time, where the only tricky part is computing the labels on the labeled edges. This can be done in constant time per label, once the following preprocessing step is done. By traversing each connected component $\mathcal{G}$ of G, we can compute, for each vertex v in $\mathcal{G}$, whether the number of red edges in the path from a fixed vertex t in $\mathcal{G}$ is odd or even, i.e., the parity. Since G has $O(n)$

edges, this can be done in $O(n)$ time and is only done once as a preprocessing step. Then, the parity of the path in G between any pair of vertices u and v in the same connected component can be computed in constant time from the parity of the path between u and t and that between t and v. □

In Stage 3B, we construct a **parity constraint graph J** to represent the constraints on the labeling of H. One of the essential differences between H and J is that H shows connections between "neighboring" components while J captures all parity constraints between far-apart components.

Figure 4(d) shows the graph J which is derived from the graph H in Figure 4(c). The vertices in J are the same as the vertices in H. Since the vertices in H correspond to vertices (vectors) in G, we can extend the terminology of vectors to the vertices in H: for example, we say that a vertex u in H is heterozygous at locus i when i is heterozygous in a pedigree node N where $u \in \Gamma(\mathrm{N})$. Simlarly we can speak of a vertex as homozygous, resolved, unresolved, etc., at a locus i.

Assume white edges have been added to G. G remains acyclic by a reasoning similar to showing that G is acyclic after adding green edges (Lemma 5). Hence, a path between any two vectors u and v in G is unique. If u and v are heterozygous and resolved (have 0 or 1) at locus i but all other vectors (if any) in the path between u and v are unresolved at locus i, then there is a constraint on how the unresolved loci can be resolved (equivalently, how the white edges should be added): namely, the number of red edges in the path in G must be even (or odd) if u and v have the same (respectively, different) parity of resolved loci at a locus i. This is because the unresolved loci at the two ends of each red edge must have different parity. To represent this constraint, we add an edge (u, v) labeled L between u and v in J, where L is 1 if u and v are resolved differently at locus i, and 0 otherwise.

A straightforward implementation of the above idea will lead to too many edges. Stage 3B below adds only $O(mn)$ edges to J, and Lemma 12 shows that this is sufficient to represent all parity constraints.

Stage 3B—Constructing the parity constraint graph J.

1. The vertices in J are the same as the vertices in H.
2. Add an edge between two vectors u and v in J if (u, v) is labeled in H. Furthermore, the label of this edge in J is the same as its label in H.
3. For each locus i, consider the tree H as if it is separated into subtrees at all vertices where i is homozygous. That is, each subtree does not contain any vertex homozygous at i. For each subtree, we root the subtree at an arbitrary vertex that is heterozygous and resolved at i. (If there is no such vertex, go to the next subtree.) Traverse the subtree and find, for each vertex v that is heterozygous and resolved at locus i, its lowest ancestor u that is also heterozygous and resolved at locus i. Add an edge between u and v in J. Label this edge with 0 (or 1) if u and v have same (respectively, different) parity of locus value. Note that there may already be such an edge in J, with the same or different labels, due to other loci. If it is the same label, do not add the edge (which is redundant). If the label is different, report *"no solution."*
4. Check whether all cycles in J have an even number of edges labeled 1. Report *"no solution"* and stop if there is a cycle in J with an odd number of edges labeled 1.
5. Note that J may not be connected. To make J connected, add edge (u, v) to J if u and v are in different connected components in J and (u, v) is an edge in H. This is always possible because H is a connected graph, and J and H

have the same set of vectors as the vertices. Arbitrarily label this edge with 0. We call the corresponding edge in H a **free edge** because we have the freedom to label (u, v) with 1 instead. We continue adding edges until J is connected. □

In effect, the graph J represents a set of linear equations modulo 2; in the example in Figure 4(d) the equations are $x_{AB} + x_{B'C} = 1, x_{B'C} + x_{C'D} = 0$, and $x_{AB} + x_{B'C} + x_{C'D} = 0$. The free edges in Step 5 correspond to the edges in H that are still unlabeled after Step 4, and are the result of the degrees of freedom in the system of equations. These unlabeled edges are "free" by themselves, in the sense that they can be assigned either 0 or 1, but once an assignment is made on one of the free edges, the other free edges may become nonfree. For example, consider Figure 4(d). There are no free edges since H is one connected component (and the system of linear equations has no degree of freedom). If we assume the edge connecting A and D does not exist, then there are two connected components, and AB, $B'C$, and $C'D$ are all potential free edges. But there is only one degree of freedom since if we assign AB to, say, 0, then all other "free" edges have their values fixed.

Lemma 12 below shows that Step 3 in Stage 3B adds sufficient edges in J to represent the parity constraints.

LEMMA 12. *Suppose there is a path between two vertices u and v in H and there is a locus i such that both u and v are heterozygous and resolved at i while all other vertices in this path are not resolved at i. Let $L = 0$ if u and v have the same resolved value at i, and 1 otherwise. Then, there is a path in J connecting u and v so that the result of applying the logical operation exclusive or (XOR) to all labels in this path equals L.*

Proof. Since all other vertices in this path are not resolved, u and v must be in the same subtree in Step 3 of Stage 3B. If one of u or v is an ancestor of the other in the rooted subtree (for locus i), not necessarily the lowest ancestor, then by the construction in Step 3 of Stage 3B, there is a sequence of edges $(u, v_0), \ldots, (v_k, v)$ in J connecting u and v, such that all these vertices are resolved at locus i. If this path is a single edge (in the case of the lowest ancestor), then we are done. Otherwise, we can assume by induction that the XOR of the labels on the path between u and v_k is equal to the parity difference of u and v_k. Adding the edge from v_k to v, with the label equal to the parity difference of v_k and v, the claim follows.

Otherwise, if neither u nor v is an ancestor of the other in the subtree, let w be the lowest common ancestor of u and v which is resolved at i. We then have two paths, one from w to u and the other from w to v, which by a similar argument to the above, have the correct labels. Hence, there is a path from u to v (through w) in J, and the XOR of the labels on this path is equal to the parity difference of u and w XORed with the parity difference of w and v, the result of which is the parity difference of u and v. □

LEMMA 13. *If Steps 3 and 4 of Stage 3B report "no solution," then there is no CHC solution. Otherwise, J has no odd cycle (an odd cycle is a cycle where the number of 1-labeled edges is odd).*

Proof. In Step 3, if "no solution" is reported, there are two conflicting constraints. In Step 4, if "no solution" is reported, then there is an odd cycle. Each edge with label 1 denotes that the resolved loci at two ends of the edge must be of different parity (and label 0 implies same parity). It follows that there is no way of resolving the loci consistently along an odd cycle. □

The free edges introduced in Step 5 are to make J connected so that we can determine the parity difference between any two nodes in J and completely label all

edges in H. The following lemma shows that these free edges can be added freely and labeled arbitrarily without affecting the existence of the CHC solution.

LEMMA 14. *If there is a CHC solution of the pedigree, then any label (*0 *or* 1*) on the free edges introduced in Step* 5 *of Stage* 3*B will make all vector-pairs SNP-consistent, Mendelian-consistent, and Endgame-consistent. On the other hand, if there is no CHC solution, none of the labels on the free edges can achieve Endgame-consistency for all vector-pairs.*

Proof. It is obvious that SNP-consistency and Mendelian consistency will always be maintained because of the red and brown edges. Since a free edge (u, v) is only added when J is not connected, u and v must be two vectors in different connected components of J and there must not exist two vectors, one in each connected component, where locus i is resolved and the vectors in the path in H between these two vectors are unresolved at locus i.

As edge (u, v) is in H, u and v must be in the same $\Phi(\mathrm{N})$ for some node N whose loci are either homozygous or heterozygous. If N is heterozygous at locus i, then locus i will be unresolved in all vectors in either u's connected component, or v's connected component, or both connected components (otherwise J cannot be disconnected). In all these cases, it can be shown that, from Lemma 6, Endgame-consistency will be maintained or not maintained no matter whether the free edge is labeled with 0 or 1. Thus the lemma is proved. □

LEMMA 15. *J has $O(mn)$ edges and can be constructed in $O(mn)$ time.*

Proof. There are $O(n)$ vertices and edges in H. Thus, Steps 1 and 2 of Stage 3B can be done in $O(n)$ time.

Step 3 can be done in $O(mn)$ time using a recursive traversal of each subtree of H for each locus as follows. We start at the root noting itself as the lowest resolved ancestor, and recursively traverse each child, passing down the ancestor information in the recursive calls. At each child, its lowest resolved ancestor is the lowest resolved ancestor of the parent. If the child itself is resolved heterozygous, then the child notes itself as the lowest resolved ancestor in subsequent traversal of its own children. Thus, it takes $O(n)$ time to perform such traversal for each locus.

Each traversal of a locus adds at most $O(n)$ edges to J, so J has at most $O(mn)$ edges.

In Step 4 we need to identify cycles with an odd number of edges labeled 1. If we imagine contracting every edge in J with label 0, then the problem reduces to checking whether the contracted graph has an odd cycle, which amounts to checking bipartiteness. Thus Step 4 can be done in $O(mn)$ time.

Step 5 can also be done in $O(mn)$ time as follows. First, for each vertex x in J, keep a list $\mathrm{LIST}(x)$ of vertices adjacent to x in H. Next, we perform a two-pass traversal as follows. Start with an arbitrary vertex X, traverse the connected component, and label the vertices traversed as "marked." Then we go back and traverse the connected component again starting at X. When traversing vertex x, we check whether all the vertices in $\mathrm{LIST}(x)$ are marked. If there is a vertex y that is unmarked, we add (x, y) to J and perform the same two-pass traversal for the connected component of y; in effect we are doing a traversal within traversal. In this way, we grow from one connected component of J and add free edges to connect to other connected components. The two-pass approach is employed to prevent adding edges which are not actually free after other free edges are added. Since the graph has $O(mn)$ edges, Step 5 takes $O(mn)$ time.

Thus, the total time complexity of Stage 3B is $O(mn)$. □

Next, we use J to complete the labeling of H.

Stage 3C—Completing the labeling of *H*.

1. Traverse J, computing, for each vertex v in J, the parity (odd or even) of the number of 1-labeled edges in the path from a fixed vertex t in J.
2. For each unlabeled edge (u, v) in H, if u and v have the same parity in J, then label edge (u, v) in H with 0, otherwise with 1. □

LEMMA 16. *All edges in H can be labeled with 0 or 1 in $O(mn)$ time in Stage 3C, and the labels in H are consistent with the parity constraints specified in J in the sense that the parity between any two vectors u and v specified in J is the same as the parity of the number of 1-labeled edges in the path between u and v in H.*

Proof. Note that J specifies a unique parity between any two vertices because J has no odd cycle. Consider a path $P = (v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ in H. Each unlabeled edge (v_i, v_{i+1}) in P receives a label equal to the parity between v_i and v_{i+1} in J in Step 1 of Stage 3C, which is equal to the parity of the number of 1-labeled edges between v_i and v_{i+1} in J. Each labeled edge (v_i, v_{i+1}) in P has the same edge (with the same label) in J, and hence the label of this edge in H is also equal to the parity of the number of 1-labeled edges between v_i and v_{i+1} in J. Hence, the parity of the number of 1-labeled edges in P is equal to the XOR of the labels on all edges in P, which in turn is equal to the XOR of the parity between v_i and v_{i+1} in J over all i, which is the parity between v_0 and v_k in J.

As far as the time complexity is concerned, since H has $O(n)$ edges and J has $O(mn)$ edges, the total time complexity of Stage 3C is $O(mn)$. □

Stage 3D—Adding white edges to *G*.

1. For each edge (u, v) in H that became labeled during Stage 3C:
   (a) If the edge is labeled 1, then let x be the vector adjacent to v by a red edge; otherwise, let x be v.
   (b) Add a white edge between u and x.
2. G now becomes a single connected component. Run LOCUS_RESOLVE(G).

LEMMA 17. *Stage 3D can be done in $O(mn)$ time, and after Stage 3D, G will be a single connected component with only unresolved and resolved loci.*

Proof. Step 1 of Stage 3D considers each of the $O(n)$ edges of H one by one, each taking constant time; thus this step takes $O(n)$ time. Step 2 takes $O(mn)$ time. The time complexity thus follows.

H is a connected graph that contains at least one vertex from each connected component of G, and the newly labeled edges in H are between vertices that were not connected in G (while the edges that were already labeled are between vertices that were already connected in G). Therefore, each white edge added results in one fewer connected component in G, and G will become a single connected component after Step 1 finishes.

LOCUS_RESOLVE ensures that G, as a single connected component, has only unresolved and resolved loci. □

Figure 5 shows the graph H of the same example in Figure 4 after the edges are labeled. The resulting white edges are added to G and the loci resolved.

LEMMA 18. *If the pedigree has a CHC solution, Stage 3D maintains Endgame-consistency.*

Proof. Suppose, to the contrary, that some node N becomes Endgame-inconsistent after Stage 3D. Without loss of generality, let the values at loci i and j for x_1, x_2, y_1, and y_2 be 00, 11, 01, and 10, respectively, where $\langle x_1, x_2\rangle, \langle y_1, y_2\rangle \in \Phi(\mathrm{N})$.

Consider the situation prior to Stage 3D. Since the pedigree has a CHC solution, given Lemma 6, the vector-pairs in each connected component are Endgame-consistent. Thus $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ must belong to different connected components;

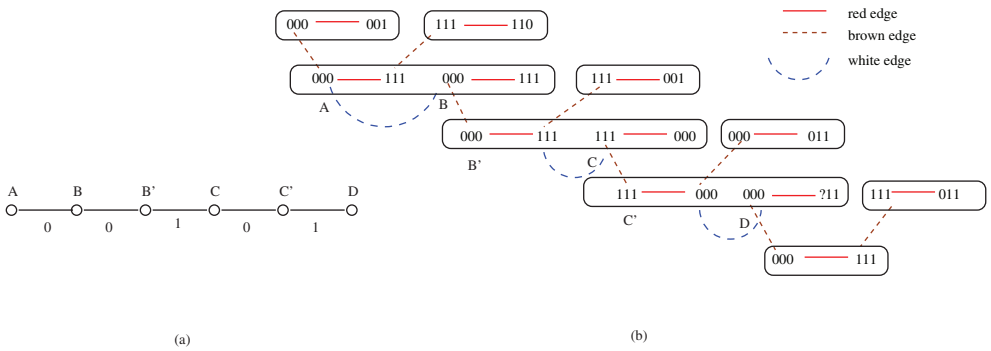


FIG. 5. *The same example from Figure* 4 *showing the result of Stages* 3*C and* 3*D.* (a) *The graph H with the correct labels. In this case there is a unique solution.* (b) *The local graph G after addition of white (dashed) edges. Loci values are also resolved.*

call them G_1 and G_2, respectively. Now suppose $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ become connected during Stage 3D after the addition of a white edge e, which connects G_1 and G_2. There are four cases to consider:

Case 1: e connects $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$. White edge e corresponds to an edge in H, and since H is acyclic, it is the unique edge between the vector-pairs and is labeled with a unique parity. Without loss of generality, suppose e connects x_1 and y_1 and is labeled 0. This white edge will make x_1 and y_1 equal and, therefore, the value of loci i and j cannot possibly become 00 for x_1 and 01 for y_1.

Case 2: e connects $\langle x_3, x_4\rangle$ in G_1 and $\langle y_3, y_4\rangle$ in G_2 where $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle \in \Phi(\mathrm{N})$. Since the pedigree has a CHC solution, and G_1 has only resolved and unresolved loci, according to Lemma 6, vector-pairs of N that are in G_1 must be Endgame-consistent. This implies that $\langle x_1, x_2\rangle$ and $\langle x_3, x_4\rangle$, which are in G_1, are Endgame-consistent. Likewise, $\langle y_1, y_2\rangle$ and $\langle y_3, y_4\rangle$ must also be Endgame-consistent. By the argument in Case 1, $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle$ must also be Endgame-consistent. This makes it impossible for $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$ to be Endgame-inconsistent.

Case 3: e connects $\langle x_3, x_4\rangle$ in G_1 and $\langle y_3, y_4\rangle$ in G_2 where $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle \in \Phi(\mathrm{M})$ and M is N's spouse. Suppose $\langle u_1, u_2\rangle \in \Phi(\mathrm{M})$ belongs to the same trio as $\langle x_1, x_2\rangle$ and suppose $\langle v_1, v_2\rangle \in \Phi(\mathrm{M})$ belongs to the same trio as $\langle y_1, y_2\rangle$. According to Lemma 9, $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ are also Endgame-inconsistent. Thus, we can consider $\langle u_1, u_2\rangle$ and $\langle v_1, v_2\rangle$ instead of $\langle x_1, x_2\rangle$ and $\langle y_1, y_2\rangle$, and accordingly, apply the arguments of Case 2.

Case 4: e connects $\langle x_3, x_4\rangle$ in G_1 and $\langle y_3, y_4\rangle$ in G_2 where $\langle x_3, x_4\rangle$ and $\langle y_3, y_4\rangle \in \Phi(\mathrm{M})$ and M is neither N nor N's spouse. Assuming no mating loops, this case does not exist. □

3.4. Stage 4—Finishing up. At this point, our graph G has only one connected component, and it only has resolved or unresolved (no mixed) loci, since this is the property we maintain by our locus resolve procedures. If all loci are resolved, then of course we are done. For those loci that are still unresolved, Lemma 6 tells us that any way of resolving makes no difference: we can arbitrarily resolve them in an SNP-consistent and Mendelian-consistent manner, and it will not affect Endgame-consistency in the sense that either all vector-pairs will be Endgame-consistent for all resolutions, or there will be Endgame-inconsistent vector-pairs for all resolutions. This means the algorithm does not need to try all possibilities; any one will do. This is crucial for avoiding an exponential blow-up.

Hence, we do the following as the final stage of our algorithm.

Stage 4—Dealing with a single connected component.

1. Arbitrarily pick a vector u of G. For all unresolved loci i, assign a value of 0 to each of them. Then run LOCUS_RESOLVE(G).
2. For all N, check Φ(N) for Endgame-consistency and report *"no solution"* if it is not maintained.

LEMMA 19. *Stage* 4 *runs in* $O(mn)$ *time.*

Proof. After Stage 3D, G is a single connected component and has no mixed loci. By Lemma 6, we do not have to try all possible resolutions; any one will do. Let s be the number of unresolved loci in G. The time complexity of resolving all of these unresolved loci is $O(sn)$ since LOCUS_RESOLVE runs in $O(n)$ time per locus. Checking all N for Endgame-consistency can be done in $O(mn)$ time. ∎

Note that assigning a value of 1 instead of 0 to any locus before running LOCUS_RESOLVE would work equally well (Lemma 6); the effect is that all 1's become 0 and all 0's become 1 at each such locus, giving another solution. In general, if there are s unresolved loci after Stage 3D and the pedigree admits a consistent solution, then there are 2^s different CHC solutions. However, if every node in the pedigree has exactly s heterozygous loci, then there are only 2^{s-1} different CHC solutions due to symmetry.

THEOREM 1. *For a given pedigree, we can either achieve a solution that represents a CHC for the given pedigree, or report "no solution" when there is no solution, in* $O(mn)$ *time where* n *is the number of nodes in the pedigree and* m *is the number of loci.*

Proof. Each of the stages of our algorithm runs in $O(mn)$ time. The algorithm reports "no solution" only when there can be no CHC solution. If the algorithm does not report "no solution," then after Stage 4, all loci are resolved and SNP-consistency, Mendelian-consistency, and Endgame-consistency are all maintained. Thus, the resolved loci values represent a CHC solution. ∎

4. Open problems. In this paper, an optimal linear-time algorithm is presented to solve the haplotype problem for pedigree data when there are no recombinations and the pedigree has no mating loops. It remains an open problem to extend the algorithm to handle mating loops. For the haplotyping problem with recombinations, the problem becomes intractable even when at most one recombination is allowed at each haplotype of a child, or when the problem is to find a feasible haplotype with the minimum number of recombinations (even without mating loops) [3]. However, there is still much scope for further study. For example, in practice, pedigree data often contain a significant amount of missing alleles (up to 14–15% of the alleles belonging to a block could be missing in the pedigree data studied). In some cases, the deduction of the missing information on alleles is possible. The goal is then to devise an efficient algorithm to determine as many missing alleles as possible.

Acknowledgments. We thank Tao Jiang and the referees for their valuable comments.

REFERENCES

[1] R. COX, N. BOUZEKRI, S. MARTIN, L. SOUTHAM, A. HUGILL, M. GOLAMAULLY, R. COOPER, A. ADEYEMO, F. SOUBRIER, R. WARD, G. M. LATHROP, F. MATSUDA, AND M. FARRALL, *Angiotensin-*1*-converting enzyme (ACE) plasma concentration is influenced by multiple ACE-linked quantitative trait nucleotides*, Human Molecular Genetics, 11 (2002), pp. 2969–2977.

[2] J. Li and T. Jiang, *Efficient inference of haplotypes from genotypes on a pedigree*, J. Bioinformatics and Computational Biology, 1 (2003), pp. 41–69.

[3] J. Li and T. Jiang, *An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming*, in Proceedings of 8th International Conference on Computational Molecular Biology, 2004, pp. 20–29.

[4] J. R. O'Connell, *Zero-recombinant haplotyping: Applications to fine mapping using SNPs*, Genet. Epidemiology, 19 Suppl 1:S64-70, 2000.

[5] E. Russo and P. Smaglik, *Single nucleotide polymorphism: Big pharmacy hedges its bets*, The Scientist, 13, 1999.

[6] N. Wang, J. M. Akey, K. Zhang, K. Chakraborty, and L. Jin, *Distribution of recombination crossovers and the origin of haplotype blocks: The interplay of population history, recombination, and mutation*, Amer. J. Human Genetics, 11 (2002), pp. 1227–1234.

[7] E. M. Wijsman, *A deductive method of haplotype analysis in pedigrees*, Amer. J. Human Genetics, 41 (1987), pp. 356–373.

[8] J. Xiao, L. Liu, L. Xia, and T. Jiang, *Fast elimination of redundant linear equations and reconstruction of recombination-free Mendelian inheritance on a pedigree*, in Proceedings of 18th ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 655–664.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2198–2219

EFFICIENT ALGORITHMS FOR RECONSTRUCTING ZERO-RECOMBINANT HAPLOTYPES ON A PEDIGREE BASED ON FAST ELIMINATION OF REDUNDANT LINEAR EQUATIONS*

JING XIAO†, LAN LIU‡, LIRONG XIA§, AND TAO JIANG¶

Abstract. Computational inference of haplotypes from genotypes has attracted a great deal of attention in the computational biology community recently, partially driven by the international HapMap project. In this paper, we study the question of how to efficiently infer haplotypes from genotypes of individuals related by a pedigree, assuming that the hereditary process was free of mutations (i.e., the Mendelian law of inheritance) and recombinants. The problem has recently been formulated as a system of linear equations over the finite field of $F(2)$ and solved in $O(m^3n^3)$ time by using standard Gaussian elimination, where m is the number of loci (or markers) in a genotype and n the number of individuals in the pedigree. We give a much faster algorithm with running time $O(mn^2 + n^3 \log^2 n \log\log n)$. The key ingredients of our construction are (i) a new system of linear equations based on some spanning tree of the pedigree graph and (ii) an efficient method for eliminating redundant equations in a system of $O(mn)$ linear equations over $O(n)$ variables. Although such a fast elimination method is not known for general systems of linear equations, we take advantage of the underlying pedigree graph structure and recent progress on low-stretch spanning trees.

Key words. haplotype inference, pedigree analysis, system of linear equations, low-stretch spanning tree

AMS subject classifications. 11D04, 68R10, 68W05

DOI. 10.1137/070687591

1. Introduction.

For centuries, human beings have fought the battle against deadly diseases, such as diabetes, cancer, stroke, heart disease, depression, and asthma. Genetic factors are believed to play a significant role for preventing, diagnosing, and treating these diseases. In recent years, *gene mapping* [2, 20, 31], whose goal is to establish connections between diseases and some specific genetic variations, has become one of the most active areas of research in human genetics. In October 2002, a multicountry collaboration, namely, the international *HapMap* project was launched [18]. One of the main objectives of the HapMap project is to identify the *haplotype* (i.e., the states of genetic markers from a single chromosome) structure of humans and common haplotypes among various populations. This information will greatly facilitate the mapping of many important disease-susceptible genes. However, the human genome is a *diploid* (i.e., its chromosomes come in pairs, with one being paternal and

*Received by the editors April 9, 2007; accepted for publication (in revised form) September 15, 2008; published electronically March 4, 2009. An extended abstract of this paper, entitled "Fast elimination of redundant linear equations and reconstruction of recombination-free Mendelian inheritance on a pedigree," appeared in *Proceedings of the* 18*th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, New Orleans, LA, 2007. This research was supported in part by NSF grant CCR-0309902, NIH grant LM008991-01, NSFC grants 60528001 and 60553001, National Key Project for Basic Research (973) grants 2002CB512801 and 2007CB807901, and a fellowship from the Center for Advanced Study, Tsinghua University. The authors wish it to be known that the first two authors should be regarded as joint first authors of this paper.

http://www.siam.org/journals/sicomp/38-6/68759.html

†Department of Computer Science and Technology, Tsinghua University, Beijing 100080, China (xiaojing00@mails.tsinghua.edu.cn).

‡Google, Inc., 1600 Amphitheater Parkway, Mountain View, CA (lanliu@google.com).

§Department of Computer Science, Duke University, Durham, NC 27708 (xialirong@gmail.com).

¶Department of Computer Science and Engineering, University of California, Riverside, CA 92507 (jiang@cs.ucr.edu).

the other maternal), and, in practice, haplotype data are not collected directly, especially in large-scale sequencing projects, mainly due to cost considerations. Instead, *genotype* data (i.e., the states of genetic markers from all chromosomes, without specifying which chromosome gives rise to each particular marker state) are collected routinely. Hence, combinatorial algorithms and statistical methods for the inference of haplotypes from genotypes, which is also commonly referred to as *phasing*, are urgently needed and have been intensively studied.

This paper is concerned with the inference of haplotypes from genotypes of individuals related by a *pedigree*, which describes the parent-offspring relationship among the individuals. Figure 1 gives an illustrative example of pedigree, genotype, and haplotype, as well as *recombination*, where the haplotypes of a parent recombine to produce a haplotype of her child. (See the appendix for more detailed definitions of these concepts.) Pedigree data is often collected in family-based gene association/mapping studies in addition to genotype data. It is generally believed that haplotypes inferred from pedigrees are more accurate than those from population data. Moreover, some family-based statistical gene association tests such as TDT (i.e., *transmission disequilibrium test*) and its variants (e.g., [32, 39], among others) require access to haplotype information for each member in a pedigree.

By utilizing some biological assumptions, such as the *Mendelian law of inheritance*, i.e., one haplotype of each child is inherited from the father while the other is inherited from the mother free of mutations, and the *minimum-recombination principle*, which says that genetic recombination is rare for closely linked markers and thus haplotypes with fewer recombinants should be preferred in haplotype inference [29, 30], several combinatorial approaches for inferring haplotypes from genotypes on a pedigree have been proposed recently and shown to be powerful and practical [5, 8, 23, 24, 25, 29, 30, 33, 36, 38]. These methods essentially propose polynomial-time heuristics or exponential-time exact algorithms for the so-called the *minimum-recombinant haplotype configuration (MRHC)* problem, which requires a haplotype solution for the input pedigree with the minimum number of recombinants (i.e., recombination events) and is known to be NP-hard [23]. (See the appendix for a more formal definition of the MRHC problem.)

A closely related problem, called the *zero-recombinant haplotype configuration (ZRHC)* problem, where we would like to enumerate all haplotype solutions requiring no recombinant (if such solutions exist), was studied in [23]. The ZRHC problem was proposed under a more stringent biological assumption that the pedigree is also recombination-free. (See the appendix for a more formal definition of the ZRHC problem.) ZRHC is interesting because recent genetic research has shown that human genomic DNAs can be partitioned into long blocks (called *haplotype blocks*) such that recombination within each block is rare or even nonexistent [7, 11, 19], especially when restricted to a single pedigree [24, 25]. An efficient algorithm for ZRHC could also be useful for solving the general MRHC problem as a subroutine, when the number of recombinants is expected to be small. We note in passing that recent work on haplotype inference for population data based on perfect phylogenies also assumes the data is recombination-free [10, 15, 16, 17]. Observe that, when the solution for ZRHC is not unique, it would really be useful to be able to enumerate all of the solutions instead of finding only one feasible solution, so that the solutions can be examined in subsequent analysis (e.g., likelihood distribution of haplotypes [24, 25, 28], linkage between different haplotype blocks [1, 14, 21], etc.) by geneticists.

The algorithmic problem and our result. The ZRHC problem can also be stated abstractly as a simple inheritance reconstruction problem as follows. We have

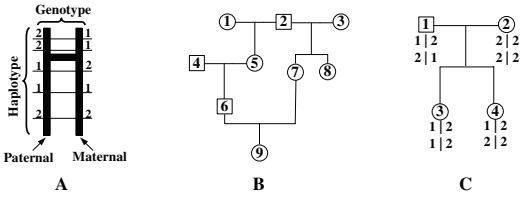


FIG. 1. A. *The structure of a pair of chromosomes from a mathematical point of view. In the figure, each numeric value (*1 *or* 2*) represents a marker state or an* allele. *The haplotype inherited from the father (or the mother) is called paternal haplotype (or maternal haplotype, respectively). The paternal and maternal haplotypes are thus strings* 22112 *and* 11212*, and they form the genotype* $\{1,2\}\{1,2\}\{1,2\}\{1,1\}\{2,2\}$*, which is a string of unordered pairs of alleles at each locus.* B. *An illustration of a pedigree with* 9 *members with a* mating loop*, where circles represent females and boxes represent males. Children are shown under their parents with line connections. For example, individuals* 7 *and* 8 *are children of individuals* 2 *and* 3*. Individuals without parents, such as individuals* 1 *and* 2*, are called* founders*. A pedigree with no mating loops is called* tree pedigree *conventionally.* C. *An example of recombination event where the haplotypes of individual* 1 *recombine to produce the paternal haplotype of individual* 3*. The numbers inside the circles/boxes are individual IDs. Here, a "|" is used to indicate the phase of the two alleles at a marker locus, with the left allele being paternal and the right maternal. Both loci of individual* 2 *and the second locus of individual* 4 *are* homozygous*, while all of the other loci in the pedigree are* heterozygous.

a pedigree connecting n individuals where each individual j has two haplotypes (i.e., strings) defined on m marker loci $a_{j,1}\cdots a_{j,m}$ and $b_{j,1}\cdots b_{j,m}$ inherited from j's father and mother, respectively. That is, if individuals j_1 and j_2 are the parents of j, then $a_{j,1}\cdots a_{j,m} \in \{a_{j_1,1}\cdots a_{j_1,m}, b_{j_1,1}\cdots b_{j_1,m}\}$ and $b_{j,1}\cdots b_{j,m} \in \{a_{j_2,1}\cdots a_{j_2,m}, b_{j_2,1}\cdots b_{j_2,m}\}$. The haplotypes are unknown, but the genotype of each individual j is given to us in the form of string $\{a_{j,1}, b_{j,1}\}\cdots\{a_{j,m}, b_{j,m}\}$. We would like to reconstruct all of the haplotype solutions that could have resulted in the genotypes.

Li and Jiang presented an $O(m^3n^3)$ time algorithm for ZRHC by formulating it as a system of $O(mn)$ linear equations with mn variables over the finite field of $F(2)$ and applying Gaussian elimination [23]. Although this algorithm is polynomial, it is inadequate for large-scale pedigree analysis where both m and n can be in the order of tens or even hundreds, and we may have to examine many pedigrees and haplotype blocks. There are, for example, over five million SNP markers in the public database dbSNP [18]. This challenge motivates us to find more efficient algorithms for ZRHC. Several attempts have been made recently in [4, 26], but the authors failed to prove the correctness of their algorithms in all cases, especially when the input pedigree has mating loops. Chan et al. proposed a linear-time algorithm in [3], but the algorithm works only for pedigrees without mating loops (i.e., the tree pedigrees).

In this paper, we present a much faster algorithm for ZRHC with running time $O(mn^2+n^3\log^2 n\log\log n)$. Our construction begins with a new system of linear equations over $F(2)$ for ZRHC. Although the system still has $O(mn)$ variables and $O(mn)$ equations, it can be effectively reduced to an equivalent system with $O(mn)$ equations and at most $2n$ variables, by exploring the underlying pedigree graph structure. By using standard Gaussian elimination, this already gives an improved algorithm with running time $O(mn^3)$. We then show how to reduce the number of equations further to $O(n\log^2 n\log\log n)$ (assuming that $m \geq \log^2 n\log\log n$, which usually holds in practice), by giving an $O(mn)$ time method for eliminating redundant equations in the system. Although such a fast elimination method is not known for general systems of linear equations, we again take advantage of the underlying pedigree graph structure and recent progress on low-stretch spanning trees in [9]. In particular, the

low-stretch spanning tree result helps upper bound the number of equations that need to be kept in the elimination process. We also show that our algorithm actually runs in $O(mn^2 + n^3)$ time when the input pedigree is a tree pedigree with no mating loops (which is often true for human pedigrees) or when there is a locus that is heterozygous across the entire pedigree. Moreover, our algorithm produces a general solution[1] to the original system of linear equations at the end that represents all feasible solutions to the ZRHC problem.

Related work on solving systems of (sparse) linear equations. The search for efficient algorithms for solving systems of linear equations is a classical problem in linear algebra. Besides Gaussian elimination, methods based on fast matrix multiplication algorithms have been proposed and could achieve an asymptotic speed of $O(n^{2.376})$ on n equations with n unknowns [6, 35]. However, these methods are only of theoretical interest since they are hard to implement and do not outperform Gaussian elimination unless n is very large. Moreover, they assume that the coefficient matrix is of full rank, which is an unreasonable assumption in ZRHC (considering the linear systems derived for ZRHC so far).

Observe that the linear system given in [23] for ZRHC is actually very sparse since each of its equations has at most four variables. Thus, a plausible way to speed up is to utilize fast algorithms for solving sparse linear systems. The Lanczos and conjugate gradient algorithms [13] and the Wiedemann algorithm [37] are some of the best known algorithms for solving sparse linear system over finite fields. The Wiedemann algorithm runs in (expected) quadratic time (which is in fact slower than our algorithm when applied to linear systems for ZRHC), while the Lanczos and conjugate gradient algorithms are only heuristics [22]. However, they use randomization and do not find all solutions. Furthermore, the algorithms cannot check if the system has no solution [12]. A randomized algorithm with quadratic expected time for certifying inconsistency of linear systems is given in [12].

The rest of our paper is organized as follows. We will describe a new system of linear equations for ZRHC and some useful graphs derived from a pedigree in section 2. The $O(mn^3)$ time algorithm is presented in section 3, and the $O(mn^2 + n^3 \log^2 n \log\log n)$ time algorithm is given in section 4. Some concluding remarks are given in section 5. The appendix contains some related biological definitions concerning MRHC and an example execution of the main algorithm.

2. A system of linear equations for ZRHC and the pedigree graph.

In this section, we first present a new formulation of ZRHC in terms of linear equations and then define some graph structures which will be used in our algorithm.

2.1. The linear system.

Throughout this paper, n denotes the number of the individuals (or members) in the input pedigree and m the number of marker loci. Without loss of generality, suppose that each allele in the given genotypes is numbered numerically as 1 or 2 (i.e., the markers are assumed to be *biallelic*, which makes the hardest case for MRHC/ZRHC [23]), and the pedigree is free of genotype errors (i.e., the two alleles at each locus of a child can always be obtained from her respective parents). Hence, we can represent the genotype of member j as a ternary vector $\mathbf{g}_j$ as follows: $g_j[i] = 0$ if locus i of member j is homozygous with both alleles being 1's, $g_j[i] = 1$ if the locus is homozygous with both alleles being 2's, and $g_j[i] = 2$ otherwise

[1] A general solution of any linear system is denoted by the span of a basis in the solution space to its associated homogeneous system, offset from the origin by a vector, namely by any particular solution.

(i.e., the locus is heterozygous). For any heterozygous locus i of member j, we use a binary variable $p_j[i]$ to denote the *phase* at the locus as follows: $p_j[i] = 1$ if allele 2 is paternal, and $p_j[i] = 0$ otherwise. When the locus is homozygous, the variable is set to $g_j[i]$ for some technical reasons (so that the equations below involving $p_j[i]$ will hold). Hence, the vector $\mathbf{p}_j$ describes the paternal and maternal haplotypes of member j. Observe that the vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$ represent a complete haplotype configuration of the pedigree. In fact, the sparse linear system in [23] was based on these vectors. Also for technical reasons, define a vector $\mathbf{w}_j$ for member j such that $w_j[i] = 0$ if its ith locus is homozygous and $w_j[i] = 1$ otherwise.

Suppose that member j_r is a parent of member j. We introduce an auxiliary binary variable $h_{j_r,j}$ to indicate which haplotype of j_r is passed to j. If j_r gives its paternal haplotype to j, then $h_{j_r,j} = 0$; otherwise, $h_{j_r,j} = 1$. Suppose that j is a nonfounder member with her father and mother being j_1 and j_2, respectively. We can define two linear (constraint) equations over $F(2)$ to describe the inheritance of paternal and maternal haplotypes at j, respectively, following the Mendelian law of inheritance and zero-recombinant assumption:

$$\mathbf{p}_{j_1} + h_{j_1,j} \cdot \mathbf{w}_{j_1} = \mathbf{p}_j \qquad \text{and} \qquad \mathbf{p}_{j_2} + h_{j_2,j} \cdot \mathbf{w}_{j_2} = \mathbf{p}_j + \mathbf{w}_j. \tag{2.1}$$

If we let $\mathbf{d}_{j_1,j}$ denote the vector $\mathbf{0}$ and $\mathbf{d}_{j_2,j}$ denote $\mathbf{w}_j$, then the above equations can be unified into a single equation as

$$\mathbf{p}_{j_r} + h_{j_r,j} \cdot \mathbf{w}_{j_r} = \mathbf{p}_j + \mathbf{d}_{j_r,j} \qquad (r = 1,\ 2). \tag{2.2}$$

Formally, we can express the ZRHC problem as a system of linear equations:

$$\begin{cases} p_k[i] + h_{k,j} \cdot w_k[i] = p_j[i] + d_{k,j}[i], & 1 \le i \le m,\ 1 \le j,k \le n, k \text{ is a parent of } j, \\ p_j[i] = g_j[i], & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2, \\ w_j[i] = 1, & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] = 2, \\ w_j[i] = 0, & 1 \le i \le m,\ 1 \le j \le n,\ g_j[i] \ne 2, \\ d_{k,j}[i] = w_j[i], & 1 \le i \le m,\ 1 \le j,k \le n,\ k \text{ is the mother of } j, \\ d_{k,j}[i] = 0, & 1 \le i \le m,\ 1 \le j,k \le n,\ k \text{ is the father of } j, \end{cases} \tag{2.3}$$

where $g_j[i], w_j[i], d_{k,j}[i]$ are all constants depending on the input genotypes, and $p_j[i], h_{k,j}$ are the unknowns. Note that the number of p-variables is exactly mn and the number of h-variables is at most $2n$ since every child has two parents and there are at most n children in the pedigree.

Remark. Observe that, for any member j, if the member itself or one of its parents is homozygous at locus i, then $p_j[i]$ is fixed based on (2.3). In the rest of this paper, we will assume that all such variables $p_j[i]$ are *predetermined* (without any conflict) and use them as "anchor points" to define some new constraints about the h-variables.

2.2. The pedigree graph and locus graphs. To apply combinatorial techniques, we transform the input pedigree into a graph, called the *pedigree graph*, by connecting each parent directly to her children. See Figure 2(B) for an example. Although the edges in the pedigree represent the inheritance relationship between a parent and a child and are directed, we will think of the pedigree graph, and, more importantly, the subsequent locus graphs, as undirected in future definitions and constructions. This is because each edge (j,k) of the pedigree graph (and locus graphs) will be used to represent the constraint between the vectors $\mathbf{p}_j$ and $\mathbf{p}_k$ (i.e., the phases

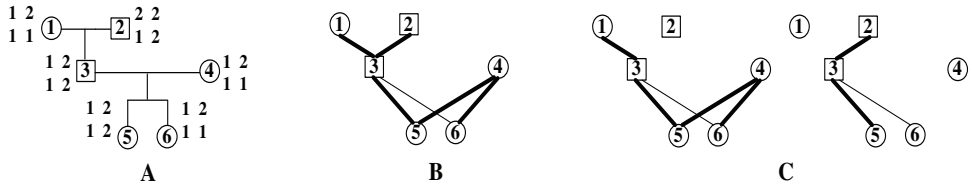


FIG. 2. A. *An example pedigree with genotype data. Here, the alleles at a locus are ordered according to their ID numbers instead of phase (which is unknown).* B. *The pedigree graph with a spanning tree. The tree edges are highlighted. Observe that there lies a* cycle *of length* 4 *in the given tree pedigree graph.* C. *The locus graphs. The left graph is for the first locus, which has a cycle, while the right graph is for the second locus. The locus forests are highlighted.*

at j and k) via the variable $h_{j,k}$, which is symmetric, as can be seen from Lemma 1 and Corollaries 2 and 3 below.[2]

Clearly, such a pedigree graph $G = (V, E)$ may be cyclic due to mating loops or multiple children shared by a pair of parents. Let $\mathcal{T}(G)$ be any spanning tree of G. $\mathcal{T}(G)$ partitions the edge set E into two subsets: *tree edges* and *nontree edges*. For simplicity, the nontree edges will be called *cross edges*. Let $E^{\mathbb{X}}$ denote the set of cross edges. Since $|E| \leq 2n$ and the number of edges in $\mathcal{T}(G)$ is $n-1$, we have $|E^{\mathbb{X}}| \leq n+1$. Figure 2(B) gives an example of tree edges and cross edges.

For any fixed locus i, the value $w_k[i]$ can be viewed as the weight of each edge $(k, j) \in E$, where k is a parent of j. We construct the *ith locus graph* G_i as the subgraph of G induced by the edges with weight 1. Formally, $G_i = (V, E_i)$, where $E_i = \{(k, j) |\ k \text{ is a parent of } j, w_k[i] = 1\}$. The ith locus graph G_i induces a subgraph of the spanning tree $\mathcal{T}(G)$. Since the subgraph is a forest, it will be referred to as the *ith locus forest* and denoted by $\mathcal{T}(G_i)$. Figure 2(C) shows the locus graphs and the locus forests of the given pedigree.

The locus graphs can be used to identify some implicit constraints on the h-variables as follows. First, we need to "symmetrize" of the h-variables and d-constants: for any edge $(k, j) \in E$, define $h_{k,j} = h_{j,k}$ and $\mathbf{d}_{k,j} = \mathbf{d}_{j,k}$.

LEMMA 1. *For any path $\mathcal{P} = j_0, \ldots, j_k$ in locus graph G_i connecting vertices j_0 and j_k, we have*

$$p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} (h_{j_r, j_{r+1}} + d_{j_r, j_{r+1}}[i]) = 0.$$

Proof. The equation follows easily from (2.3) and an induction on the length k of the path. □

Note that the above constraint remains the same no matter in which direction path $\mathcal{P}$ is read, since the addition is over field $F(2)$ and the h-variables and d-constants are symmetric. From the lemma, we can see that for a cycle in G_i the summation of all of the h-variables corresponding to the edges on the cycle is a constant. The constant is said to be *associated* with the cycle.

COROLLARY 2. *For any cycle $\mathcal{C} = j_0, \ldots, j_k, j_0$ in G_i, there exists a binary constant b defined as $b = \sum_{r=0}^{k} d_{j_r, j_{r+1 \bmod k+1}}[i]$ such that $\sum_{r=0}^{k} h_{j_r, j_{r+1 \bmod k+1}} = b$.*

[2]The reader can also verify that the direction of an edge will not affect the graph traversal and the ensuing treatment of constraint equations to be discussed in the next two sections.

Proof. This follows from Lemma 1 and the fact that $p_{j_0}[i] + p_{j_0}[i] = 0$. ∎

From Lemma 1, we can easily see that if the p-variables at the endpoints of a path are predetermined, then the summation of all of the h-variables corresponding to the edges on the path is a constant. The constant is said to be *associated* with the path. We construct constraints on h-variables as follows. Again, notice that the following constant b does not depend on the direction that path $\mathcal{P}$ is read.

COROLLARY 3. *Suppose that $\mathcal{P} = j_0, \ldots, j_k$ is a path in G_i connecting vertices j_0 and j_k, and the variables $p_{j_0}[i]$ and $p_{j_k}[i]$ are predetermined. Then there exists a binary constant b defined as $b = p_{j_0}[i] + p_{j_k}[i] + \sum_{r=0}^{k-1} d_{j_r,j_{r+1}}[i]$ such that $\sum_{r=0}^{k-1} h_{j_r,j_{r+1}} = b$.*

3. An $O(mn^3)$ time algorithm for ZRHC. Since the number of h-variables is at most $2n$, our key idea is to first derive a system of $O(mn)$ linear equations on the h-variables. We use paths and cycles in G_i and the predetermined p-variables as mentioned in the last section to build the linear system, and then we find a *general* solution to the system by using Gaussian elimination so that all inherent freedom in (2.3) is kept. This new system of equations about the h-variables is clearly necessary for (2.3). The crux of the construction is to show that it is also sufficient, and thus the p-variables can be determined from the values of the h-variables by a simple traversal of the locus graphs.

3.1. Linear constraints on the h-variables. We will introduce constraint equations to "cover" all of the edges in each locus graph. As mentioned above, these equations connect the p-variables and will suffice to help determine their values. Note that, since the edges broken in each locus graph involve predetermined p-variables, we do not have to introduce constraints to cover them. The constraints can be classified into two categories with respect to the spanning tree $\mathcal{T}(G)$: constraints for cross edges and constraints for tree edges.

Cross edge constraints. Adding a cross edge e to the spanning tree $\mathcal{T}(G)$ yields a cycle $\mathcal{C}$ in the pedigree graph G. Let $length(\mathcal{C})$ denote the length of cycle $\mathcal{C}$. Suppose that the edge e exists in the ith locus graph G_i, and consider two cases of the cycle $\mathcal{C}$ with respect to graph G_i.

Case 1. The cycle exists in G_i. We introduce a constraint along the cycle as in Corollary 2. This constraint is called a *cycle constraint*. The set of such cycle constraints for edge e in all locus graphs is denoted by $C^{\mathbb{C}}(e)$, i.e.,

$$C^{\mathbb{C}}(e) = \{(b, e) \mid b \text{ is associated with the cycle in } \mathcal{T}(G_i) \cup \{e\},\ 1 \le i \le m\}.$$

The set of cycle constraints for all cross edges is denoted by $C^{\mathbb{C}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{C}}(e)$.

Case 2. Some of the edges of the cycle do not exist in G_i. This means that the cycle $\mathcal{C}$ is broken into several disjoint paths in G_i by the predetermined vertices. Since e exists in G_i, exactly one of these paths, denoted as $\mathcal{P}$, contains e. Observe that both endpoints of $\mathcal{P}$ are predetermined, and thus Corollary 3 could give us a constraint concerning the h-variables along the path. Such a constraint will be called a *path constraint*. The set of such path constraints for e in all locus graphs G_i is denoted by $C^{\mathbb{P}}(e)$, i.e.,

$$C^{\mathbb{P}}(e) = \left\{ (k, j, b, e) \;\middle|\; \begin{array}{l} \text{in } \mathcal{T}(G_i) \cup \{e\},\ b \text{ is associated with the path containing } e \\ \text{connecting two predetermined vertices } k \text{ and } j,\ 1 \le i \le m \end{array} \right\}.$$

The set of path constraints for all cross edges is denoted by $C^{\mathbb{P}} = \biguplus_{e \in E^{\mathbb{X}}} C^{\mathbb{P}}(e)$.

Please refer to **Procedure** CROSS_EDGE_CONSTRAINTS in Figure 3 to see the generation of $C^{\mathbb{C}}$ and $C^{\mathbb{P}}$.

Tree edge constraints. By Corollary 3, there is an implicit constraint concerning the h-variables along each path between two predetermined vertices in the same connected component of $\mathcal{T}(G_i)$. Therefore, for each connected component $\mathcal{T}$ of $\mathcal{T}(G_i)$, we arbitrarily pick a predetermined vertex in the component as the *seed* vertex, and generate a constraint for the unique path in $\mathcal{T}(G_i)$ between the seed and each of the other predetermined vertices in the component, as in Corollary 3. Such a constraint will be called a *tree constraint*. Notice that if there exists any component having no predetermined vertices, then locus i must be heterozygous across the entire pedigree and $\mathcal{T}(G_i)$ is actually a spanning tree. Such a locus will be referred to as an *all-heterozygous* locus. For such a locus i, we arbitrarily pick a vertex in $\mathcal{T}(G_i)$ as the seed, but we will not generate at tree constraints.

To conform with the notation of path constraints and for the convenience of presentation, we arbitrarily pick a tree edge denoted as e_0 and write the set of tree constraints at all loci as

$$C^{\mathbb{T}} = \left\{ (k, j, b, e_0) \;\middle|\; \begin{array}{l} \text{in a connected component of } \mathcal{T}(G_i) \text{ with seed } k,\ b \text{ is} \\ \text{associated with the path connecting vertices } k \text{ and} \\ \text{a predetermined vertex } j,\ 1 \le i \le m \end{array} \right\}.$$

Note that e_0 is the same for all of the tree constraints and will be used as an *indicator* to distinguish tree constraints from path constraints defined by cross edges. The formal construction of $C^{\mathbb{T}}$ is described in **Procedure** Tree_Edge_Constraints in Figure 3.

Again, we need to symmetrize path constraints and tree constraints: given any constraint (k, j, b, e) generated for a path connecting two predetermined vertices k and j in a locus graph, define $(k, j, b, e) = (j, k, b, e)$. The above constructions of $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, and $C^{\mathbb{T}}$ are more formally described as pseudocode in Figure 3. We can easily see that the following holds.

Lemma 4. $|C^{\mathbb{C}}| + |C^{\mathbb{P}}| + |C^{\mathbb{T}}| = O(mn)$.

3.2. Solving the linear system for ZRHC using the new constraints. We now describe how to solve the system in (2.3) in $O(mn^3)$ time. The pseudocode for solving the system is formally given as Algorithm **ZRHC_Phase** in Figure 4. Here, we first construct the cycle, path, and tree constraints on the h-variables, and pick a vertex as the seed for every connected component in the locus forests $\mathcal{T}(G_i)$, as described in the last subsection. Then we solve these constraints by using Gaussian elimination to obtain a general solution of the h-variables, which may contain some *free* h-variables. Next, for each connected component with no predetermined vertices, we set the p-variable of the seed as a *free* variable and treat it as a determined value. Finally, we perform a breadth-first search (BFS) on the spanning forest $\mathcal{T}(G_i)$ of each locus graph G_i. For each connected component of $\mathcal{T}(G_i)$, we start from the seed and propagate its p-variable value to the undetermined vertices in the component by using the solution for the h-variables, which will result in functions of the *free* h-variables and at most one *free* p-variable. Note that, in the last step of the algorithm, $p_k[i]$ is expressed as a linear combination of the free variables in $p_j[i]$ and the free h-variables with an appropriate constant term.

To show the correctness of the algorithm, we need only show that the solution found by the algorithm is a feasible solution for (2.3) and vice versa. Since we determine the p-variables based on the linear system for the h-variables derived from (2.3), any feasible solution to (2.3) will be included in the (general) solution found by our algorithm. In other words, we do not lose any degrees of freedom in the solution process. Hence, it suffices to show that our solution satisfies (2.3).

Procedure CROSS_EDGE_CONSTRAINTS
input: locus graphs $G_{[1..m]}$ and the spanning tree $\mathcal{T}(G)$
output: cross edge constraint sets $C^{\mathbb{C}}, C^{\mathbb{P}}$
begin
  $C^{\mathbb{C}} = C^{\mathbb{P}} = \emptyset$;
  for each cross edge e
    Suppose that $\mathcal{C}$ is the cycle in $\mathcal{T}(G) \cup \{e\}$;
    $C^{\mathbb{P}}(e) = C^{\mathbb{C}}(e) = \emptyset$;
    for each locus i
      if $\mathcal{C}$ is connected in G_i
        Let $b = \sum_{(k,j) \in \mathcal{C}} d_{k,j}[i]$;
        $C^{\mathbb{C}}(e) = C^{\mathbb{C}}(e) \biguplus \{(e, b)\}$;
      else
        Suppose $\mathcal{P}$ is the path containing e in $\mathcal{T}(G_i) \cup \{e\}$;
        Let vertices j_1 and j_2 be the endpoints of $\mathcal{P}$;
        Define $b = p_{j_1}[i] + p_{j_2}[i] + \sum_{(k,j) \in \mathcal{P}} d_{k,j}[i]$;
        $C^{\mathbb{P}}(e) = C^{\mathbb{P}}(e) \biguplus \{(j_1, j_2, b, e)\}$;
    $C^{\mathbb{C}} = C^{\mathbb{C}} \biguplus C^{\mathbb{C}}(e)$;
    $C^{\mathbb{P}} = C^{\mathbb{P}} \biguplus C^{\mathbb{P}}(e)$;
end.

Procedure TREE_EDGE_CONSTRAINTS
input: locus forests $\mathcal{T}(G_{[1..m]})$ and a fixed tree edge e_0
output: tree edge constraint set $C^{\mathbb{T}}$
begin
  $C^{\mathbb{T}} = \emptyset$;
  for each locus i
    for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
      if $\mathcal{T}$ has no predetermined vertices
        Arbitrarily pick a vertex j_0 in $\mathcal{T}$ as the seed of $\mathcal{T}$;
      else
        Arbitrarily pick a predetermined vertex j_0 in $\mathcal{T}$ as the seed of $\mathcal{T}$;
      for each predetermined vertex $j_1 \neq j_0$ in $\mathcal{T}$
        Let $\mathcal{P}$ be the path between j_0 and j_1;
        Define $b = p_{j_0}[i] + p_{j_1}[i] + \sum_{(k,j) \in \mathcal{P}} d_{k,j}[i]$;
        $C^{\mathbb{T}} = C^{\mathbb{T}} \biguplus \{(j_0, j_1, b, e_0)\}$;
end.

FIG. 3. *The procedure for generating constraints.*

LEMMA 5. *The p-variables and h-variables determined by Algorithm* **ZRHC_Phase** *satisfy the linear system in* (2.3).

Proof. Denote the solution found by our algorithm as $\mathbf{p}$ and $\mathbf{h}$. We need only care about the first equations in (2.3). If the $w_k[i]$ in such an equation is 0, then the equation involves only two predetermined p-values, which holds trivially since Step 1 of our algorithm explicitly takes care of predetermined p-values. Otherwise, each such equation corresponds to an edge in some locus graph. Let $e = (j_1, j_2)$ be an edge in locus graph G_i. It represents an equation $p_{j_1}[i] + h_{j_1,j_2} = p_{j_2}[i] + d_{j_1,j_2}[i]$, i.e., the first equation on edge e in (2.3).

Algorithm ZRHC_Phase
[**Improved ZRHC_Phase**]
input: pedigree $G = (V, E)$ and genotype $\{\mathbf{g}_j\}$
output: a general solution of $\{\mathbf{p}_j\}$
begin
Step 1. *Preprocessing*
Construct a [**low-stretch**] spanning tree $\mathcal{T}(G)$ on G;
Let e_0 be an arbitrary tree edge;
for each locus i
Generate the locus graph G_i;
Generate the locus forest $\mathcal{T}(G_i)$;
Identify the predetermined nodes;
Step 2. *Constraint generation*
Cross_Edge_Constraints($G_{[1..m]}, \mathcal{T}(G),\ C^{\mathbb{C}},\ C^{\mathbb{P}}$);
Tree_Edge_Constraints($\mathcal{T}(G_{[1..m]}),\ C^{\mathbb{T}}, e_0$);
[**Step 2′. *Redundant constraint elimination***
Compact_Constraints($C^{\mathbb{C}},\ C^{\mathbb{P}},\ C^{\mathbb{T}},\ e_0$ **)**;]
Step 3. *Solve the h-variables*
Apply Gaussian elimination on $C^{\mathbb{C}} \uplus C^{\mathbb{P}} \uplus C^{\mathbb{T}}$
to get a general solution of the h-variables;
Step 4. *Solve the p-variables by propagation*
for each locus i
for each connected component $\mathcal{T}$ in $\mathcal{T}(G_i)$
if $\mathcal{T}$ has no predetermined vertices
Set the p-variable of the seed as a free variable and
treat it as a determined value;
Traverse $\mathcal{T}$ by BFS starting from the seed;
for each edge (j, k) in $\mathcal{T}$
if $p_j[i]$ is determined but $p_k[i]$ is undetermined
$p_k[i] = p_j[i] + h_{j,k} + d_{j,k}[i]$;
return $\{\mathbf{p}_j\}$;
end.

FIG. 4. *The $O(mn^3)$ time algorithm* ZRHC_Phase *and the $O(mn^2 + n^3 \log^2 n \log\log n)$ time algorithm* Improved_ZRHC_Phase. *The additional instructions in* Improved_ZRHC_Phase *are highlighted by bold font in square brackets. In order to save running time, we use disjoint union (i.e., $\uplus$) in Algorithms* ZRHC_Phase *and* Improved_ZRHC_Phase.

Given any two vertices j_s and j_t in a same connected component of $\mathcal{T}(G_i)$, we denote by $\mathcal{P}(j_s, j_t)$ the unique path in the component connecting j_s and j_t. Suppose that vertex j_0 is the seed of the component. The key observation here is that for any vertex j_t, regardless of whether vertex j_t is predetermined or not, our solution satisfies the equation

$$p_{j_t}[i] \ = \ p_{j_0}[i] \ + \sum_{(k,\ j)\ \in\ \mathcal{P}(j_0,\, j_t)} (h_{k,j} + d_{k,j}[i]). \tag{3.1}$$

More precisely, if vertex j_t is predetermined, then (3.1) holds because of the tree constraint for path $\mathcal{P}(j_0, j_t)$ defined in Step 2 of the algorithm. Otherwise, $p_{j_t}[i]$ is assigned the value as given in (3.1) during the traversal from the seed j_0 to j_t in Step 4. More generally, we can build a relationship between the p-values of two vertices j_s and j_t from (3.1) as follows:

$$p_{j_t}[i] \;=\; p_{j_s}[i] \;+ \sum_{(k,\ j)\ \in\ \mathcal{P}(j_t,\, j_s)} (h_{k,j} + d_{k,j}[i]). \tag{3.2}$$

Notice that, as long as j_s and j_t are in the same connected component of $\mathcal{T}(G_i)$, the above equation is satisfied by our solution. With the help of (3.2), we now show that our solution satisfies the equation represented by edge e.

Case 1. e is a tree edge. Clearly, vertices j_1 and j_2 belong to the same connected component of $\mathcal{T}(G_i)$. We replace j_s and j_t by j_1 and j_2, respectively, in (3.2) and obtain $p_{j_1}[i] \;=\; p_{j_2}[i] \;+\sum_{(k,\ j)\ \in\ \mathcal{P}(j_1,\, j_2)}(h_{k,j} + d_{k,j}[i]) \;=\; p_{j_2}[i] \;+ h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the tree edge e, since the constant $w_{j_1}[i]$ is assumed to be 1.

Case 2. e is a cross edge. Denote by $\mathcal{C}$ the cycle in $\mathcal{T}(G_i) \cup \{e\}$. There are two subcases.

Case 2.1. $\mathcal{C}$ exists in G_i. In this case, we have a cycle constraint for $\mathcal{C}$ in G_i. Suppose that the cycle constraint is $0 = \sum_{(k,j)\in\mathcal{C}}(h_{k,j} + d_{k,j}[i])$. Observe that vertices j_1 and j_2 are in the same connected component of $\mathcal{T}(G_i)$; thus we have $p_{j_1}[i] \;=\; p_{j_2}[i] \;+\sum_{(k,\ j)\ \in\ \mathcal{P}(j_1,\, j_2)}(h_{k,j} + d_{k,j}[i])$ as shown in (3.2). Observe that cycle $\mathcal{C}$ can be decomposed into path $\mathcal{P}(j_1, j_2)$ and cross edge $e = (j_1, j_2)$. Therefore, summing up the above two equations, we can get $p_{j_1}[i] \;=\; p_{j_2}[i] + h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the cross edge e.

Case 2.2. $\mathcal{C}$ is broken into several disjoint paths in G_i by predetermined vertices. Suppose that the unique path containing e is $\mathcal{P}$. Without loss of generality, suppose that vertices k_1 and k_2 are the endpoints of $\mathcal{P}$, and $\mathcal{P}$ has the form $\mathcal{P}(k_1, j_1); e; \mathcal{P}(j_2, k_2)$. Since vertices k_1 and k_2 are predetermined, our algorithm defines a path constraint for $\mathcal{P}$ in G_i, i.e., $p_{k_1}[i] \;=\; p_{k_2}[i] \;+\sum_{(k,\ j)\ \in\ \mathcal{P}}(h_{k,j} + d_{k,j}[i])$. Observe that vertices k_1 and j_1 are in the same connected component of $\mathcal{T}(G_i)$. Therefore, $p_{k_1}[i] = p_{j_1}[i] + \sum_{(k,\ j)\ \in\ \mathcal{P}(k_1,\, j_1)}(h_{k,j} + d_{k,j}[i])$ based on (3.2). Similarly, vertices j_2 and k_2 are in the same connected component of $\mathcal{T}(G_i)$, and $p_{j_2}[i] = p_{k_2}[i] + \sum_{(k,\ j)\ \in\ \mathcal{P}(j_2,\, k_2)}(h_{k,j} + d_{k,j}[i])$. Since path $\mathcal{P} = \mathcal{P}(k_1, j_1); e; \mathcal{P}(j_2, k_2)$, summing up the above three equations yields equality $p_{j_2}[i] = p_{j_1}[i] + h_{j_1,j_2} + d_{j_1,j_2}[i]$, which means that our solution satisfies the equation represented by the cross edge e.

In conclusion, the solution produced by Algorithm **ZRHC_Phase** satisfies the equations on all edges, and thus the lemma holds. ∎

THEOREM 6. *The running time of Algorithm* **ZRHC_Phase** *is* $O(mn^3)$.

Proof. Step 1 needs $O(n)$ time for each locus, which takes $O(mn)$ time in total. In Step 2, we need at most $O(n)$ time for generating a constraint. Since there are at most $O(mn)$ constraints, it takes at most $O(mn^2)$ time. In Step 3, the system has $O(n)$ h-variables and $O(mn)$ constraints. So, Gaussian elimination requires $O(mn^3)$ time. In Step 4, every edge is visited just once in the traversal. Since each h-variable is expressed as a linear combination of at most n free h-variables at the end of Step 3,every p-variable is expressed as at most n free h-variables and at most one free p-

variable on the same locus. Thus, this steps takes $O(mn^2)$ time altogether. Therefore, the entire process takes $O(mn^3)$ time. □

4. Speeding up the algorithm by fast elimination of redundant equations. The bottleneck in the above algorithm is Step 3, where we have to spend $O(mn^3)$ time to solve a system of $O(mn)$ equations over $O(n)$ variables. Clearly, most of the equations are redundant and can be expressed as linear combinations of other equations. The question is how to detect and eliminate these redundant equations (without using Gaussian elimination, of course). To the best of our knowledge, there are no methods that would eliminate redundant equations for any system of linear equations over any field faster than Gaussian elimination asymptotically in the worst case. Here, we give such a method taking advantage of the underlying pedigree structure. We first give a general method to compact path and tree constraints that correspond to paths on a cycle in the pedigree graph.

Let $\mathcal{C}$ be a cycle in the pedigree graph G induced by cross edge e_1. For convenience, we say that a path/tree constraint is on the cycle $\mathcal{C}$ if it corresponds to a path/edge on $\mathcal{C}$. The following lemma shows that the path/tree constraints on a cycle can be greatly compacted and is the key to our algorithm to eliminate redundant constraints.

LEMMA 7. *Given a set C of path/tree constraints on cycle $\mathcal{C}$, we can reduce C to an equivalent constraint set of size at most $2 \cdot length(\mathcal{C})$ in time $O(|C|)$.*

Proof. Recall that e_0 represents the fixed tree edge introduced in subsection 3.1 for defining the tree constraints. We use $\widehat{C}$ to denote the equivalent constraint set (to be constructed). Initially, we set $\widehat{C} = \emptyset$.

For convenience, we say that a path/tree constraint *connects* vertices j and k if the constraint has the form (k, j, b, e). To depict a more clear picture of the relationship between the constraints in C, we define a *constraint graph* G^*[3] as follows. For each vertex k of the cycle $\mathcal{C}$, we create a vertex in G^*. For each path/tree constraint in C connecting k and j, we build an edge connecting k and j in G^*. Observe that the connected components in G^* naturally partition the constraints in C into disjoint subsets. We will compact the constraints in each of these disjoint subsets separately and put the resultant equivalent constraint sets into $\widehat{C}$. More precisely, for each connected component of G^*, we pick an arbitrary vertex as the *root* of the component and construct new constraints connecting the root and the other vertices in the component. The details of the construction will be given in the next paragraph. Here, the term root is meant to be synonymous to the term seed defined in subsection 3.1, although each seed is defined for a single locus, whereas a root may be used to deal with constraints concerning multiple loci.

Now, we give the details of how to construct $\widehat{C}$. Consider each connected component S of G^*. Suppose that its root is k_0. We process the constraints of C induced by S in the order of increasing distance between the root and the vertices connected by the constraints. In other words, we traverse S by BFS starting from the root. Suppose that we are now visiting vertex j. For each edge (k_0, j), we directly put into $\widehat{C}$ the constraints that created the edge (k_0, j) in G^*. For each edge (k, j) where k $(k \neq k_0)$ is visited before j, our construction guarantees that $\widehat{C}$ will have constraints connecting the root k_0 and k. Suppose that one of such constraints is $c' = (k_0, k, b', e')$. Let $c = (k, j, b, e) \in C$ denote the constraint that created the edge (k, j) in G^*. We generate a new constraint $c'' = (k_0, j, b'', e'')$, where $b'' = b + b'$ and e'' is defined as

[3]Note that a constraint graph might actually be a multigraph, but this will not affect the correctness of Lemma 7.

follows:

$$e'' = \begin{cases} e_1 & \text{if } \{e\} \cup \{e'\} = \{e_0, e_1\}, \\ e_0 & \text{otherwise.} \end{cases}$$

Because c can be represented as the summation of c'' and $c' \in \widehat{C}$, c is equivalent to c'' given $\widehat{C}$. Then we add c'' to $\widehat{C}$. Here, again the fixed tree e_0 and cross edge e_1 are used to indicate tree and path constraints, respectively. Observe that the above constraint c'' resulted from the combination of constraints c' and that c involves only tree edges if and only if both or none of c' and c corresponds to paths containing e. Otherwise, c'' corresponds to some path across e_1.

The BFS creates an equivalent constraint in $\widehat{C}$ for every constraint in C, and thus $\widehat{C} \equiv C$. Recall that each constraint in $\widehat{C}$ has the form (k_0, j, b, e), where j is a vertex in $\mathcal{C}$, e is either e_0 or e_1, and k_0 is the root of the connected component of G^* containing j. It is easy to see that two constraints (k_0, j, b', e) and (k_0, j, b'', e) that differ only in the associated b-constants are consistent with each other if and only if $b' = b''$. Hence, $\widehat{C}$ has at most $2 \cdot length(\mathcal{C})$ different constraints, or otherwise the input linear system has no feasible solutions. The construction can be done in $O(|C|)$ time, as more formally described by Procedure **Compact_PT_Const** in Figure 5. Hence, the lemma holds. □

The reader may refer to the example in Appendix B (Step 2′) for a simple illustration of how the construction in the above proof works. An immediate application of Lemma 7 is to remove redundancy from each path constraint set $C^{\mathbb{P}}(e)$, since the path constraints in $C^{\mathbb{P}}(e)$ are all on the cycle induced by e.

COROLLARY 8. *Given the path constraint set $C^{\mathbb{P}}(e)$, we can reduce it to an equivalent constraint set of size at most $2 \cdot length(\mathcal{C})$ in time $O(|C^{\mathbb{P}}(e)|)$, where $\mathcal{C}$ is the cycle induced by cross edge e.*

We can also use Lemma 7 to remove redundant tree constraints. Note that the construction in the proof of Lemma 7 still works if the constraints in C are all tree constraints involving no cross edge e_1. Moreover, the resultant set $\widehat{C}$ contains only constraints of the form (k_0, j, b, e_0). This implies that $|\widehat{C}| \leq n$. Therefore, the following corollary holds.

COROLLARY 9. *Given the tree constraint set $C^{\mathbb{T}}$, we can reduce it to an equivalent constraint set of size at most n in $O(|C^{\mathbb{T}}|)$ time.*

4.1. Elimination of redundant cycle constraints. Each cross edge e induces a unique cycle $\mathcal{C}$. Since every constraint in $C^{\mathbb{C}}(e)$ concerns the same set of h-variables corresponding to the edges on $\mathcal{C}$, each $C^{\mathbb{C}}(e)$ contains only one independent constraint. Moreover, these constraints are consistent with each other if and only if their associated constants are identical, which can be checked in $O(m)$ time. Because the total number of cross edges are at most $n + 1$ we have the following lemma.

LEMMA 10. *Given the cycle constraint set $C^{\mathbb{C}}$, we can reduce it to an equivalent constraint set of size at most $n + 1$ in $O(mn)$ time.*

4.2. Elimination of redundant path constraints. We will show how to reduce the path constraints $C^{\mathbb{P}}$ on a general pedigree to an equivalent set of path constraints with size $O(n \log^2 n \log \log n)$ in $O(mn)$ time (assuming $\log^2 n \log \log n < m$). Furthermore, for tree pedigrees (i.e., pedigrees with no mating loops) we can make the equivalent constraint set as small as $O(n)$. For pedigrees with an all-heterozygous locus across the entire pedigree, we can first transform $C^{\mathbb{P}}$ into an equivalent tree con-

```
Procedure COMPACT_PT_CONST
input: a set C of path/tree constraints
       on cycle C induced by cross edge e_1,
       and a fixed tree edge e_0
output: a compact constraint set Ĉ ≡ C
begin
 Construct the constraint graph G* for C;
 Ĉ = ∅;
 for each connected component S of G*
   Pick an arbitrary vertex k_0 ∈ S as the root of S;
   Traverse S by BFS starting from k_0;
   while there exists unvisited vertices in G*
     Visit an unvisited vertex, say k, in the BFS order;
     for each constraint c = (k_0, j, b, e) in C
       Ĉ = Ĉ ⊎ {c};
     for each constraint c = (k, j, b, e) in C
                 s.t. vertex k ≠ k_0 is visited before j
       for each constraint c' = (k_0, k, b', e') in Ĉ
         b'' = b + b';
         if {e} ∪ {e'} = {e_0, e_1}
           e'' = e_1;
         else
           e'' = e_0;
         Construct a new constraint c'' = (k_0, j, b'', e'');
         if there exists a constraint (k_0, j, b'' + 1, e'') ∈ Ĉ
           exit "The input genotypes are inconsistent.";
         if c'' ∉ Ĉ
           Ĉ = Ĉ ⊎ {c''};
 return Ĉ;
end.
```

FIG. 5. *The procedure for compacting path and tree constraints on a cycle.*

straint set with size $O(mn)$, and then we will remove its redundancy via Corollary 9. We first start with the special cases.

Elimination of redundant path constraints on tree pedigrees. Observing that the length of each (simple) cycle in the pedigree graph of a tree pedigree is a constant (i.e., 4, of which an example is given in Figure 2(B)), we can upper bound the total number of path constraints as follows.

LEMMA 11. *Given the path constraint set $C^{\mathbb{P}}$ on a tree pedigree, we can reduce it to an equivalent path constraint set of size $O(n)$ in $O(mn)$ time.*

Proof. By Corollary 8, we can reduce $C^{\mathbb{P}}(e)$ for each cross edge e to an equivalent set of at most eight path constraints in $O(m)$ time. Since there are at most $n+1$ cross edges, the set $C^{\mathbb{P}}$ can be reduced to an equivalent set of size $O(n)$ in $O(mn)$ time, and thus the lemma holds. □

Transformation of path constraints on pedigrees with an all-heterozygous locus. Observe that for a pedigree with an all-heterozygous locus i each cross edge induces a cycle that exists in the locus graph G_i and has a cycle constraint in the (reduced) set $C^{\mathbb{C}}$. This allows us to transform all of the path constraints into tree constraints given the cycle constraints as follows.

COROLLARY 12. *Given the path constraint set $C^{\mathbb{P}}$ on a pedigree with an all-heterozygous locus, we can construct an equivalent tree constraint set of size $O(mn)$ in $O(mn)$ time.*

Proof. Let us first focus on a cross edge e. Suppose that the cycle $\mathcal{C}$ is induced by e in $\mathcal{T}(G) \cup \{e\}$, the compact $C^{\mathbb{C}}(e)$ is $\{(b', e)\}$, and e_0 is the fixed tree edge defined in subsection 3.1. Notice that the cycle $\mathcal{C}$ has two disjoint paths connecting each pair of two vertices on the cycle. One of them consists of only tree edges and may be used to define a tree constraint, while the other contains the cross edge and may be used to represent a path constraint. Therefore, with the help of the cycle constraint (b', e), we can transform the path constraint set $C^{\mathbb{P}}(e)$ into a tree constraint set $C(e)$ of the same size as follows:

$$C(e) = \{(k, j,\ b - b',\ e_0) \mid (k, j,\ b,\ e) \in C^{\mathbb{P}}(e)\}.$$

It is not hard to see that $C(e) \cup C^{\mathbb{C}}(e) \equiv C^{\mathbb{P}}(e) \cup C^{\mathbb{C}}(e)$, and $C(e)$ can be constructed in $|C^{\mathbb{P}}(e)|$ time.

Let $C = \cup_{e \in E^{\mathbb{X}}} C(e)$. Obviously, $C \equiv C^{\mathbb{P}}$ and $|C| = O(mn)$ since $|C^{\mathbb{P}}| = O(mn)$. Hence, the lemma holds. ∎

Elimination of redundant path constraints on a general pedigree. Now, we consider how to compact path constraints in the general case. As shown in Corollary 8, given a cross edge e inducing cycle $\mathcal{C}$, we can compact the constraints in $C^{\mathbb{P}}(e)$ so that at most $2 \cdot length(\mathcal{C})$ constraints are kept. Clearly, the compact $C^{\mathbb{P}}$ has size at most $O(n^2)$ since the number of cross edges is at most $n + 1$ and the length of a cycle containing a cross edge is at most n. This bound can be improved by observing that the total length of all cycles in G is related to the average *stretch* [9] of G with respect to the spanning tree $\mathcal{T}(G)$. Hence, we can obtain a sharper upper bound on $|C^{\mathbb{P}}|$ by using a low-stretch spanning tree $\mathcal{T}(G)$ as constructed in [9].

We first give a formal definition of the *stretch* of an unweighted connected graph with respect to a spanning tree. Given a spanning tree $\mathcal{T}$ on an unweighted connected graph $G = (V, E)$ (e.g., the pedigree graph), we define the stretch of an edge $(k, j) \in E$, denoted as $stretch_{\mathcal{T}}(k, j)$, to be the length of the unique path (i.e., the number of edges on the path) in $\mathcal{T}$ between k and j. The average stretch of G with respect to $\mathcal{T}$ is then defined as $avg\text{-}stretch_{\mathcal{T}}(E) = \frac{1}{|E|} \sum_{(k,j) \in E} stretch_{\mathcal{T}}(k, j)$.

LEMMA 13. *Given a pedigree G, we can build a low-stretch spanning tree $\mathcal{T}(G)$ in $O(n \log n)$ time such that $|C^{\mathbb{P}}| = O(n \log^2 n \cdot \log \log n)$ after compacting.*

Proof. In [9], Elkin et al. showed that every unweighted connected graph $G = (V, E)$ contains a spanning tree, into which each edge of the graph can be embedded with an average stretch of $O(\log^2 n \log \log n)$. Moreover, this tree can constructed in $O(|E| \log |V|)$ time. In our situation, such a low-stretch spanning tree $\mathcal{T}(G)$ can be built in $O(n \log n)$ time because $|E| \leq 2n$ and $|V| = n$. The following inequality establishes the relationship between $|C^{\mathbb{P}}|$ and the average stretch of E with respect to

$\mathcal{T}(G)$. From Corollary 8, we have

$$\begin{aligned}
|C^{\mathbb{P}}| &= \sum_{e \in E^{\mathbb{X}}} |C^{\mathbb{P}}(e)| \\
&\leq \sum_{\mathcal{C} \text{ induced by } e \in E^{\mathbb{X}}} 2 \cdot length(\mathcal{C}) \\
&= \sum_{e \,=\, (k,j) \in E^{\mathbb{X}}} 2 \cdot \big(stretch_{\mathcal{T}}(k,j) + 1\big) \\
&\leq 2 \sum_{(k,j) \in E} stretch_{\mathcal{T}}(k,j) + 2n \\
&\leq 2 \cdot |E| \cdot avg\text{-}stretch_{\mathcal{T}}(E) + 2n \\
&\leq 2 \cdot (2n) \cdot O(\log^2 n \log\log n) + 2n \\
&= O(n \cdot \log^2 n \log\log n) \qquad \square
\end{aligned}$$

4.3. Elimination of redundant tree constraints and the final algorithm. After we process (i.e., compact or transform) the path constraints, we eliminate redundant tree constraints and obtain a compact tree constraint set containing at most n constraints as shown in Corollary 9. The complete algorithm for eliminating redundant cycle, path, and tree constraints is given in Figure 6 as Procedure **Compact_Constraints**. The next theorem summarizes the above discussion.

THEOREM 14. *Given the constraint sets $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, and $C^{\mathbb{T}}$ on a pedigree, we can reduce them to an equivalent constraint set of size $O(n \cdot \log^2 n \log\log n)$ in $O(mn)$ time. In particular, for tree pedigrees and pedigrees with an all-heterozygous locus, the equivalent constraint set has size $O(n)$.*

We can incorporate the above redundant constraint elimination procedure **Compact_Constraints** into the $O(mn^3)$ time algorithm for ZRHC in order to obtain an improved algorithm **Improved_ZRHC_Phase** as shown in Figure 4. (An example of how **Improved_ZRHC_Phase** works is given in the appendix.) The following theorem is obvious given Theorem 14.

THEOREM 15. *Algorithm* **Improved_ZRHC_Phase** *solves the ZRHC problem correctly on any pedigree in $O(mn^2 + n^3 \log^2 n \log\log n)$ time. Moreover, it solves ZRHC on tree pedigrees or pedigrees with an all-heterozygous locus in $O(mn^2 + n^3)$ time.*

5. Concluding remarks. It remains interesting if the time complexity for ZRHC on general pedigrees can be improved to $O(mn^2+n^3)$ or lower. Another open question is how to use the algorithm to solve MRHC on pedigrees that require only a small (constant) number of recombinants.

Appendix.

A. Some related biological definitions. The genome of an organism consists of *chromosomes* that are double strand DNAs. Locations on a chromosome can be labelled using *markers*, which are small segments of DNA with some specific features. A physical position of a marker on a chromosome is called a *marker locus* and a marker state is called an *allele*. In *diploid* organisms, chromosomes come in pairs. The status of two alleles at a particular marker locus of a pair of chromosomes is called a *marker genotype*. The genotype information at a locus will be denoted using a set, e.g., $\{a, b\}$. If the two alleles a and b are the same, then the genotype is *homozygous*.

Procedure COMPACT_CONSTRAINTS
input: $C^{\mathbb{C}}$, $C^{\mathbb{P}}$, $C^{\mathbb{T}}$, and a fixed tree edge e_0
output: compact $C^{\mathbb{C}}, C^{\mathbb{P}}$, and $C^{\mathbb{T}}$
begin

Step 1. *Removing redundant cycle constraints*

for each cross edge e
  Pick an arbitrary constraint, say $c = (e, b)$, from $C^{\mathbb{C}}(e)$;
  if there exists a constraint $(e, b+1) \in C^{\mathbb{C}}(e)$
    exit "The input genotypes are inconsistent.";
  $C^{\mathbb{C}} = C^{\mathbb{C}} - C^{\mathbb{C}}(e) \biguplus \{c\}$;

Step 2. *Processing path constraints*

if G is a tree pedigree
  for each cross edge e
    $\widehat{C}^{\mathbb{P}}(e) = \emptyset$;
    for each constraint $c = (k, j, b, e) \in C^{\mathbb{P}}(e)$
      if there exists a constraint $(k, j, b+1, e) \in C^{\mathbb{P}}(e)$
        exit "The input genotypes are inconsistent.";
      $\widehat{C}^{\mathbb{P}}(e) = \widehat{C}^{\mathbb{P}}(e) \biguplus \{c\}$;
    $C^{\mathbb{P}} = C^{\mathbb{P}} - C^{\mathbb{P}}(e) \biguplus \widehat{C}^{\mathbb{P}}(e)$;

else if G has an all-heterozygous locus
  for each cross edge e
    Let (b', e) be the cycle constraint for e in $C^{\mathbb{C}}$;
    for each constraint $c = (k, j, b, e)$ in $C^{\mathbb{P}}(e)$
      Construct a new constraint $c' = (k, j, b - b', e_0)$;
      $C^{\mathbb{T}} = C^{\mathbb{T}} \biguplus \{c'\}$;

else (i.e., G is a general pedigree)
  for each cross edge e
    $\widehat{C}^{\mathbb{P}}(e) =$ COMPACT_PT_CONST($C^{\mathbb{P}}(e)$, e_0);
    $C^{\mathbb{P}} = C^{\mathbb{P}} - C^{\mathbb{P}}(e) \biguplus \widehat{C}^{\mathbb{P}}(e)$;

Step 3. *Removing redundant tree constraints*

$C^{\mathbb{T}} =$ COMPACT_PT_CONST($C^{\mathbb{T}}$, e_0);

end.

FIG. 6. *The procedure for removing redundant constraints.*

Otherwise, it is *heterozygous*. A *haplotype* consists of all alleles, one from each locus, that are on the same chromosome. Figure 1(A) illustrates the above concepts, where alleles are represented by their numerical IDs.

A pedigree can be defined formally as follows.

DEFINITION 16. *A pedigree graph is a weakly connected directed acyclic graph (DAG) $G = (V, E)$, where $V = M \cup F$, M stands for the male nodes, and F stands for the female nodes. The in-degree of each node is* 0 *(founders) or* 2 *(nonfounders). If the in-degree of a node is* 2*, then one edge must start from a male node (called father) and the other edge from a female node (called mother), and the node itself is a child of its parents (father and mother).*

A *mating loop* consists of two distinct paths from a node x to a node y. Figure 1(B) illustrates an example pedigree with a mating loop. The Mendelian law of inheritance states that the alleles of a child must come from the alleles of its parents at each marker locus (i.e., assuming no mutations within a pedigree). In other words, the two alleles at each locus of the child have different origins: one is from its father (which is called the *paternal* allele) and the other from its mother (which is called the *maternal* allele). Usually, a child inherits a complete haplotype from each parent. However, *recombination* may occur, where the two haplotypes of a parent get shuffled due to a crossover of chromosomes and one of the shuffled copies is passed on to the child. Such an event is called a recombination event, and its result is called a *recombinant*. Since markers are usually short DNA sequences, we assume that recombination occurs only between markers. Figure 1(C) illustrates an example where the paternal haplotype of member 3 is the result of a recombinant. The paternal allele and maternal allele at each locus is separated by a "|" in this figure.

We use the term *haplotype configuration* to describes not only the paternal and maternal haplotypes of an individual but also the grandpaternal or grandmaternal origin of each allele on the haplotypes. Observe that the number of recombinants required in a pedigree can be easily computed once the haplotype configuration of each member of the pedigree is given. The MRHC problem is defined as follows.

DEFINITION 17 (MRHC). *Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires the minimum number of recombinants.*

Namely, the ZRHC problem is a special case of MRHC with the following definition.

DEFINITION 18 (ZRHC). *Given a pedigree and genotype information for each member of the pedigree, find a haplotype configuration for the pedigree that requires no recombinant (if such solution exists).*

B. An example execution of Algorithm Improved_ZRHC_phase. The example in Figure 7 aims to demonstrate how Algorithm **Improved_ZRHC_phase** works (see Figure 6 for the pseudocode of the algorithm). The input pedigree with genotype data is shown in Figure 7(A), and the corresponding pedigree graph is in Figure 7(B).

In Step 1, we generate the locus graphs (or forests) as illustrated in Figure 7(C) and identify the predetermined vertices in Figure 7(D). Moreover, we arbitrarily pick a tree edge, say $e_{1,4}$, as the *indicator* to distinguish tree constraints from path constraints, which is defined in subsection 3.1.

In Step 2, we generate cycle, path, and tree constraints as follows. For example, given the cycle $\mathcal{C} = v_2v_5v_3v_6v_2$ in the second locus graph of Figure 7(D), we denote the cycle constraint $h_{2,5} + h_{3,5} + h_{3,6} + h_{2,6} = 0$ by the form $(0, e_{2,6})$. Afterwards, we have $C^{\mathbb{C}} = \{(0, e_{2,6}), (0, e_{2,6})\}$, $C^{\mathbb{P}} = \{(v_4, v_9, 0, e_{4,9}), (v_9, v_8, 1, e_{4,9})\}$ and $C^{\mathbb{T}} = \{(v_6, v_8, 0, e_{1,4}), (v_6, v_9, 1, e_{1,4}), (v_4, v_8, 1, e_{1,4})\}$.

In Step 2′, we first remove redundant cycle constraints and obtain $C^{\mathbb{C}} = \{(0, e_{2,6})\}$. Next, we need to take care of path constraints. For instance, given the path constraints $(v_4, v_9, 0, e_{4,9})$ and $(v_9, v_8, 1, e_{4,9})$ induced by the cross edge $e_{4,9}$, we draw the constraint graph as illustrated in Figure 7(E) to help remove redundant path constraints and obtain $C^{\mathbb{P}} = \{(v_4, v_9, 0, e_{4,9}), (v_9, v_8, 1, e_{4,9})\}$ based on Lemma 7. Then we construct the constraint graph as shown in Figure 7(E) for tree constraints and obtain $C^{\mathbb{T}} = \{(v_4, v_8, 1, e_{1,4}), (v_4, v_6, 1, e_{1,4}), (v_4, v_9, 0, e_{1,4})\}$, according to Lemma 7.

In Step 3, we apply Gaussian elimination to the following linear system, which is

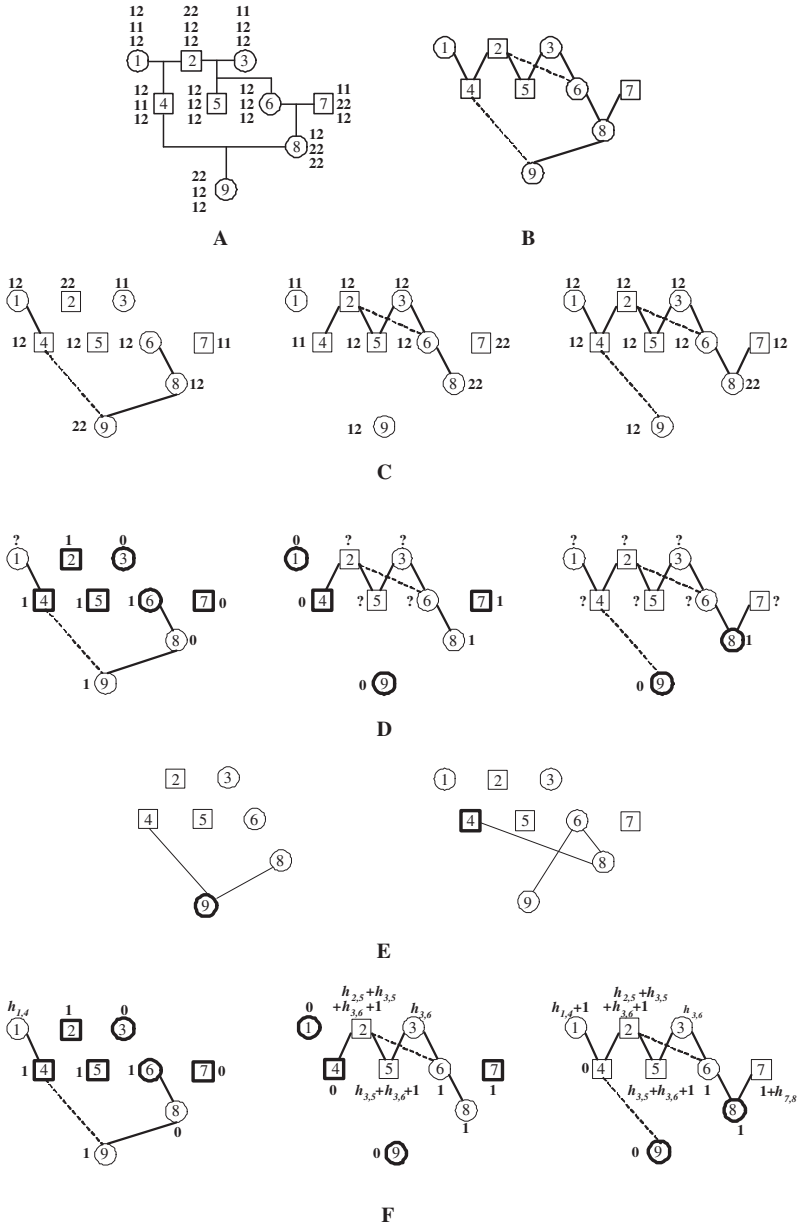


FIG. 7. A. *An example input pedigree with genotype data. Here, the alleles at a locus are ordered according to their ID numbers instead of phase (which is unknown).* B. *The pedigree graph with a spanning tree. The tree edges are highlighted.* C. *Locus graphs for the three loci, respectively. The locus forests are highlighted.* D. *The predetermined nodes. In the locus graphs, the predetermined nodes are indicated by their corresponding p-value (i.e., the number* 0 *or* 1 *near the nodes), while the undetermined nodes are accompanied by the question mark (i.e., "?"). The seeds in the graphs are highlighted by thick borders.* E. *The constraint graphs. The left graph is for removing redundant path constraints generated by the cross edge* $e_{4,9}$*, while the right graph is for tree constraints. The roots (see Lemma* 7 *for the definition) in the graphs are highlighted by thick borders.* F. *The locus graphs with propagated p-values. The notations are the same as those in Figure* 7(D)*, except that the question marks on the undermined nodes are replaced by their resolved p-values. The resolved p-values are expressed as a linear combination of the free variables in* $p_j[i]$ *and the free h-variables with an appropriate constant term.*

equivalent to $C^{\mathbb{C}} \biguplus C^{\mathbb{P}} \biguplus C^{\mathbb{T}}$:

$$\begin{cases} h_{2,5}+h_{3,5}+h_{3,6}+h_{2,6}=0, & \text{i.e., the cycle constraint } (0, e_{2,6}),\\ h_{4,9}=0, & \text{i.e., the path constraint } (v_4, v_9, 0, e_{4,9}),\\ h_{4,9}+h_{2,4}+h_{2,5}+h_{3,5}+h_{3,6}+h_{6,8}=1, & \text{i.e., the path constraint } (v_9, v_8, 1, e_{4,9}),\\ h_{2,4}+h_{2,5}+h_{5,3}+h_{3,6}+h_{6,8}=1, & \text{i.e., the tree constraint } (v_4, v_8, 1, e_{1,4}),\\ h_{2,4}+h_{2,5}+h_{5,3}+h_{3,6}=1, & \text{i.e., the tree constraint } (v_4, v_6, 1, e_{1,4}),\\ h_{2,4}+h_{2,5}+h_{5,3}+h_{3,6}+h_{6,8}+h_{8,9}=0, & \text{i.e., the tree constraint } (v_4, v_9, 0, e_{1,4}). \end{cases}$$

Then we obtain the following general solution:

$$\begin{cases} h_{4,9}=0,\\ h_{6,8}=0,\\ h_{8,9}=1,\\ h_{2,6}=h_{2,5}+h_{3,5}+h_{3,6},\\ h_{2,4}=h_{2,5}+h_{3,5}+h_{3,6}+1, \end{cases}$$

where $h_{2,5}, h_{3,5}, h_{3,6}, h_{1,4}$, and $h_{7,8}$ are free variables.

In Step 4, we solve the unknown p-values by propagation from the seeds and obtain all p-values as shown in Figure 7(F).

Acknowledgments. We would like to thank Jing Li and Cliff Zhang for many useful discussions on ZRHC, and the anonymous referees for several constructive suggestions.

REFERENCES

[1] G. R. Abecasis, S. S. Cherny, W. O. Cookson, and L. R. Cardon, *Merlin–rapid analysis of dense genetic maps using sparse gene flow trees*, Nat. Genet., 30 (2002), pp. 97–101.

[2] M. C. Campbell and S. A. Tishkoff, *African genetic diversity: Implications for human demographic history, modern human origins, and complex disease mapping*, Annu. Rev. Genomics Hum. Genet., 9 (2008), pp. 403–433.

[3] M. Y. Chan, W. Chan, F. Chin, S. Fung, and M. Kao, *Linear-time haplotype inference on pedigrees without recombinations*, in Proceedings of the 6th Annual Workshop on Algorithms in Bioinformatics (WABI), 2006.

[4] F. Chin and Q. Zhang, *Haplotype Inference on Tightly Linked Markers in Pedigree Data*, manuscript, 2005.

[5] F. Chin, Q. Zhang, and H. Shen, *k-recombination haplotype inference in pedigrees*, in Proceedings of the International Conference on Computational Science (ICCS), Springer, Berlin, 2005, pp. 985–993.

[6] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic programming*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[7] M. J. Daly, J. D. Rioux, S. F. Schaffner, T. J. Hudson, and E. S. Lander, *High-resolution haplotype structure in the human genome*, Nat. Genet., 29 (2001), pp. 229–232.

[8] K. Doi, J. Li, and T. Jiang, *Minimum recombinant haplotype configuration on tree pedigrees*, in Proceedings of the 3rd Annual Workshop on Algorithms in Bioinformatics (WABI), Springer, Berlin, 2003, pp. 339–353.

[9] M. Elkin, Y. Emeky, D. A. Spielman, and S. Teng, *Lower-stretch spanning trees*, in Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), 2005, pp. 494–503.

[10] E. Eskin, E. Halperin, and R. M. Karp, *Large scale reconstruction of haplotypes from genotype data*, in Proceedings of the 7th Annual Conference on Research in Computational Molecular Biology (RECOMB), ACM, New York, 2003, pp. 104–113.

[11] S. B. Gabriel, S. F. Schaffner, H. Nguyen, J. M. Moore, J. Roy, B. Blumenstiel, J. Higgins, M. DeFelice, A. Lochner, M. Faggart, S. N. Liu-Cordero, C. Rotimi, A. Adeyemo, R. Cooper, R. Ward, E. S. Lander, M. J. Daly, and D. Altshuler, *The structure of haplotype blocks in the human genome*, Science, 296 (2002), pp. 2225–2229.

[12] M. Giesbrecht, A. Lobo, and B. Saunders, *Certifying inconsistency of sparse linear systems*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1998, pp. 113–119.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The John Hopkins University Press, Baltimore, MD, 1996.
[14] D. F. Gudbjartsson, K. Jonasson, M. L. Frigge, and A. Kong, *Allegro, a new computer program for multipoint linkage analysis*, Nat. Genet., 25 (2000), pp. 12–13.
[15] D. Gusfield, *Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions*, in Proceedings of the 6th Annual Conference on Research in Computational Biology (RECOMB), ACM, New York, 2002, pp. 166–175.
[16] D. Gusfield, *An overview of combinatorial methods for haplotype inference*, in Computational Methods for SNPs and Haplotype Inference, Lecture Notes in Comput. Sci. 2983, Springer, Berlin, 2004, pp. 9–25.
[17] B. V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail, *A survey of computational methods for determining haplotypes*, in Computational Methods for SNPs and Haplotype Inference, Lecture Notes in Comput. Sci. 2983, Springer, Berlin, 2004, pp. 26–47.
[18] The international HapMap Consortium, *International HapMap project*, Nature, 426 (2003), pp. 789–796.
[19] L. Helmuth, *Genome research: Map of the human genome* 3.0, Science, 293 (2001), pp. 583–585.
[20] W. Hou, H. Li, B. Zhang, M. Huang, and R. Wu, *A nonlinear mixed-effect mixture model for functional mapping of dynamic traits*, Heredity, 101 (2008), pp. 321–328.
[21] L. Kruglyak, M. J. Daly, M. P. Reeve-Daly, and E. S. Lander, *Parametric and nonparametric linkage analysis: A unified multipoint approach*, Am. J. Hum. Genet., 58 (1996), pp. 1347–1363.
[22] B. LaMacchia and A. Odlyzko, *Solving large sparse linear systems over finite fields*, in Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology, Lecture Notes in Comput. Sci. 537, Springer, Berlin, 1990, pp. 109–133.
[23] J. Li and T. Jiang, *Efficient rule-based haplotyping algorithm for pedigree data*, in Proceedings of the 7th Annual Conference on Research in Computational Molecular Biology (RECOMB), ACM, New York, 2003, pp. 197–206.
[24] J. Li and T. Jiang, *An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming*, in Proceedings of the 8th Annual Conference on Research in Computational Biology (RECOMB), ACM, New York, 2004, pp. 20–29.
[25] J. Li and T. Jiang, *Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming*, J. Comput. Biol., 12 (2005), pp. 719–739.
[26] X. Li, Y. Chen, and J. Li, *An Efficient Algorithm for the Zero-Recombinant Haplotype Configuration Problem*, manuscript, 2006.
[27] L. Liu, X. Chen, J. Xiao, and T. Jiang, *Complexity and approximation of the minimum recombination haplotype configuration problem*, in Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC), Springer, Berlin, 2005, pp. 370–379.
[28] S. Lin and T. P. Speed, *An algorithm for haplotype analysis*, J. Comput. Biol., 4 (1997), pp. 535–546.
[29] J. R. O'Connell, *Zero-recombinant haplotyping: Applications to fine mapping using SNPs*, Genet. Epidemiol., 19 (2000), pp. 64–70.
[30] D. Qian and L. Beckmann, *Minimum-recombinant haplotyping in pedigrees*, Am. J. Hum. Genet., 70 (2002), pp. 1434–1445.
[31] D. K. Santra, X. M. Chen, M. Santra, K. G. Campbell, and K. K. Kidwell, *Identification and mapping QTL for high-temperature adult-plant resistance to stripe rust in winter wheat (Triticum aestivum L.) cultivar "Stephens,"* Theor. Appl. Genet., 117 (2008), pp. 793–802.
[32] H. Seltman, K. Roeder, and B. D. Devlin, *Transmission/disequilibrium test meets measured haplotype analysis: Family-based association analysis guided by evolution of haplotypes*, Am. J. Hum. Genet., 68 (2001), pp. 1250–1263.
[33] E. Sobel, K. Lange, J. O'Connell, and D. Weeks, *Haplotyping algorithms*, in Genetic Mapping and DNA Sequencing, IMA Vol. Math Appl. 81, T. Speed and M. Waterman, eds., Springer, New York, 1996, pp. 89–110.
[34] E. Sobel and K. Lange, *Descent graphs in pedigree analysis: Applications to haplotyping, location scores, and marker-sharing statistics*, Am. J. Hum. Genet., 58 (1996), pp. 1323–1337.
[35] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
[36] P. Tapadar, S. Ghosh, and P. P. Majumder, *Haplotyping in pedigrees via a genetic algorithm*, Hum. Hered., 50 (2000), pp. 43–56.

[37] D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory, IT-32 (1986), pp. 54–62.

[38] E. M. Wijsman, *A deductive method of haplotype analysis in pedigrees*, Am. J. Hum. Genet., 41 (1987), pp. 356–373.

[39] S. Zhang, Q. Sha, H. S. Chen, J. Dong, and R. Jiang, *Transmission/disequilibrium test based on haplotype sharing for tightly linked markers*, Am. J. Hum. Genet., 73 (2003), pp. 566–579.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2220–2272

POLYLOGARITHMIC INDEPENDENCE CAN FOOL DNF FORMULAS*

LOUAY M. J. BAZZI†

Abstract. We show that any k-wise independent probability distribution on $\{0,1\}^n$ $O(m^{2.2}2^{-\sqrt{k}/10})$-fools any boolean function computable by an m-clause disjunctive normal form (DNF) (or conjunctive normal form (CNF)) formula on n variables. Thus, for each constant $e > 0$, there is a constant $c > 0$ such that any boolean function computable by an m-clause DNF (or CNF) formula is m^{-e}-fooled by any $c \log^2 m$-wise probability distribution. This resolves up to an $O(\log m)$ factor the depth-2 circuit case of a conjecture due to Linial and Nisan [*Combinatorica*, 10 (1990), pp. 349–365]. The result is equivalent to a new characterization of DNF (or CNF) formulas by low degree polynomials. It implies a similar statement for probability distributions with the small bias property. Using known explicit constructions of small probability spaces having the limited independence property or the small bias property, we directly obtain a large class of explicit pseudorandom generators of $O(\log^2 m \log n)$-seed length for m-clause DNF (or CNF) formulas on n variables, improving previously known seed lengths.

Key words. limited independence, DNF formulas, harmonic analysis, posets, pseudorandomness, polynomial approximation

AMS subject classifications. 06A07, 42B05, 60C05, 60E15, 68R05, 68W20

DOI. 10.1137/070691954

1. Introduction.

If μ is a probability distribution on $\{0,1\}^n$ and $g : \{0,1\}^n \to \{0,1\}$ is a boolean function, we say that μ *ϵ-fools* g [8, 27] if

$$|Pr_{x\sim\mu}[g(x) = 1] - Pr_{x\in\{0,1\}^n}[g(x) = 1]| \leq \epsilon,$$

where the second probability is with respect to the uniform probability distribution on $\{0,1\}^n$.

Let μ be a probability distribution on $\{0,1\}^n$, and let $k \geq 0$ be an integer. We say that μ is *k-wise independent* (e.g., [15, 26]) if any k or fewer of the underlying n binary random variables are statistically independent and each is equally likely to be zero or one.[1]

A *disjunctive normal form (DNF) formula* on n variables $x_1, \ldots, x_n$ is an OR of AND gates, called *clauses*, on the *literals* $x_1, \neg x_1, \ldots, x_n, \neg x_n$. Similarly, a *conjunctive normal form (CNF) formula* is an AND of OR gates.

We consider in this paper the following problem: how large should k be in terms of ϵ, n, and m so that any k-wise independent probability distribution on $\{0,1\}^n$ ϵ-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables?

The main contribution of this paper is the following theorem.

*Received by the editors May 18, 2007; accepted for publication (in revised form) September 24, 2008; published electronically March 4, 2009. An extended abstract of this paper appeared in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007, pp. 63–73.

http://www.siam.org/journals/sicomp/38-6/69195.html

†Department of Electrical and Computer Engineering, American University of Beirut, Beirut 1107 2020, Lebanon (Louay.Bazzi@aub.edu.lb).

[1]For technical convenience, we allow $k = 0$ in the sense that any probability distribution on $\{0,1\}^n$ is 0-wise independent.

THEOREM 1.1. *Any k-wise independent probability distribution on $\{0,1\}^n$ $(16m^{2.2}2^{-\sqrt{k}/10})$-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables.*

The proof is based on harmonic and poset analysis techniques. It uses Hastad's switching lemma [9] indirectly via the Linial–Mansour–Nisan (LMN) energy bound [13], applied to many DNF formulas derived from the DNF formula under consideration. The proof can be regarded as a sequence of reductions between some L_1- and L_2-approximations of DNF formulas and auxiliary functions by low degree polynomials with real coefficients.

COROLLARY 1.2. *For each constant $e > 0$, there is a constant $c > 0$ such that any boolean function computable by an m-clause DNF (or CNF) formula is m^{-e}-fooled by any $c\log^2 m$-wise probability distribution.*

The above problem was first proposed by Linial and Nisan [14]. Motivated by this problem, they derived a general bound on approximate inclusion-exclusion from which they concluded that any boolean function computable by an m-clause DNF (or CNF) formula is $o(1)$-fooled by any $\lfloor\sqrt{m}\log m\rfloor$-wise independent probability distribution. They conjectured that any boolean function computable by a size-M depth-d unbounded-fanin AND/OR circuit is ϵ-fooled by any $\log^{d-1} M$-wise independent probability distribution, where $\epsilon = 0.1$. Corollary 1.2 resolves up to an $O(\log m)$ factor this conjecture for depth-2 circuits, reducing the $\sqrt{m}\log m$ bound of [14] to $O(\log^2 m)$. Note that the conjecture's strict parameters are not correct: Luby and Velickovic [16] reported a counterexample which exhibits for each power m of 2 and for all $n \geq \log m$ a function $f : \{0,1\}^n \to \{0,1\}$ computable by an m-clause DNF formula, and a $\log m$-wise independent probability distribution μ on $\{0,1\}^n$ such that μ does not $\frac{1}{2}$-fool f. Theorem 1.1 leaves the region between $O(\log m)$ and $o(\log^2 m)$ open for depth-2 circuits.

Next we explain the linear programming (LP)-dual of Theorem 1.1 which is a new approximation of DNF (or CNF) formulas by low degree polynomials and compare it with the related literature.

1.1. Dual problem. Let $g : \{0,1\}^n \to \{0,1\}$ be a boolean function, $k \geq 0$ an integer, and $\epsilon \geq 0$. Then saying that "any k-wise independent probability distribution ϵ-fools g" is equivalent to saying "there exist $g_l, g_u : \{0,1\}^n \to \mathbb{R}$ such that

- (*low degree*[2]) $deg(g_l) \leq k$ and $deg(g_u) \leq k$;
- (*sandwiching polynomials*) $g_l \leq g \leq g_u$;
- (*small L_1-approximation error*) $E(g - g_l) \leq \epsilon$ and $E(g_u - g) \leq \epsilon$,

where the expectation is over the uniform probability distribution."

We show this in section 4 (see Theorem 4.2). Thus the dual problem is about an L_1-approximation of DNF (or CNF) formulas by low degree sandwiching polynomials with real coefficients. Via this duality, Theorem 1.1 is (up to a constant factor) equivalent to the following theorem.

THEOREM 1.3. *Let $g : \{0,1\}^n \to \{0,1\}$ be a boolean function computable by an m-clause DNF (or CNF) formula, and let $k \geq 0$ be an integer. Then there exist two real-valued functions $g_l, g_u : \{0,1\}^n \to \mathbb{R}$ each of degree at most k such that $g_l \leq g \leq g_u$, $E(g - g_l) = O(m^{2.2}2^{-\sqrt{k}/10})$, and $E(g_u - g) = O(m^{2.2}2^{-\sqrt{k}/10})$, where the expectation is over the uniform probability distribution.*

We will actually establish the dual statement.

[2]The degree of a function $f : \{0,1\}^n \to \mathbb{R}$ is the smallest degree of a polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ such that $p(x) = f(x)$ for all $x \in \{0,1\}^n$.

Theorem 1.3 implies the following weaker L_1-approximation.

COROLLARY 1.4. *Let $g : \{0,1\}^n \to \{0,1\}$ be a boolean function computable by an m-clause DNF (or CNF) formula, and let $k \geq 0$ be an integer. Then there exists a real-valued function $p : \{0,1\}^n \to \mathbb{R}$ of degree at most k such that $E|g-p| = O(m^{2.2}2^{-\sqrt{k}/10})$.*

Proof. Set $p = g_l$ or g_u. □

Small constant-depth unbounded-fanin AND/OR circuits can be approximated by low degree polynomials with real coefficients in different ways [1, 7, 13]. They can be also approximated by low degree polynomials with coefficients over finite fields [22]. We compare our sandwiching L_1-approximation with [1, 7, 13] specialized to depth-2 circuits. The approximation in [13] is an L_2-approximation based on Hastad's switching lemma.

THEOREM 1.5 (see [13]). *Let $g : \{0,1\}^n \to \{0,1\}$ be computable by an m-clause DNF (or CNF) formula, and let $k \geq 0$ be an integer. Then there exists a real-valued function $p : \{0,1\}^n \to \mathbb{R}$ of degree at most k such that $E(g-p)^2 \leq 2m2^{-\sqrt{k}/20}$.*

The proof of Theorem 1.1 uses a variation of this L_2-approximation (see Theorem 9.1) applied to many DNF formulas derived from the DNF formula under consideration.

The approximation in [1, 7] does not use Hastad's switching lemma and is in terms of probabilistic polynomials.

THEOREM 1.6 (see [1, 7]). *Let $g : \{0,1\}^n \to \{0,1\}$ be computable by an m-clause DNF (or CNF) formula. For all $\epsilon > 0$, there exists a (finite) family of polynomials $\{p_\alpha\}_\alpha$, where each p_α is a real polynomial (with integer coefficients) on the variables $x_1, \ldots, x_n$ of degree $O(\log^2(m/\epsilon)\log^2 m \log^2 n)$ such that $Pr_\alpha[p_\alpha(x) \neq g(x)] \leq \epsilon$ for all $x \in \{0,1\}^n$. Thus, in particular, $Pr_x[p_\alpha(x) \neq g(x)] \leq \epsilon$ for some α.*

The polynomials do not give a good L_1-approximation ($E_x|p_\alpha(x)-g(x)|$ is potentially as large as $2^{O(\log^2(m/\epsilon)\log^2 m \log^2 n)}$), but we believe that they probably can be used indirectly to establish a weak version of Theorem 1.1 which is naturally extensible to AC_0 circuits. See [5, sections 5.7 and 5.8] for a work in this direction.

A final remark is that one cannot hope to obtain a good L_∞-approximation of DNF formulas by low degree polynomials. This follows from [20].

1.2. Paper outline. In section 2, we give direct applications of Theorem 1.1. Section 3 contains Fourier transform preliminaries. Section 4 highlights the dual characterization of the class of boolean functions that are fooled by the limited independence property. The remainder of the paper is about the proof of Theorem 1.1, starting with the proof outline in section 5. The proof consists of sections 5, 6, 7, 8, and 9. The proof depends on sections 3 and 4, but does not use section 2 or section 10, which branches from the proof and ends with an open problem.

2. Some direct applications. This section can be skipped without loss of continuity. In section 2.1, we conclude from Theorem 1.1 that probability spaces with quasi-polynomially small[3] bias also fool all polynomial size DNF (or CNF) formulas. Using known explicit constructions of small probability spaces having the limited independence property or the small bias property, we directly obtain in section 2.2 a large class of explicit pseudorandom generators (PRGs) of $O(\log^2 m \log n)$-seed length for m-clause DNF (or CNF) formulas on n variables, improving previously known (unconditional) seed lengths. Finally, we highlight in section 2.3 a direct application of

[3] By quasi-polynomially small we mean $2^{-\log^{\Theta(1)}(n)}$.

Theorem 1.1 to the distribution of patterns in linear codes.

2.1. Extension to small bias probability spaces. We can conclude from Theorem 1.1 that probability spaces with quasi-polynomially small bias also fool all polynomial size DNF (or CNF) formulas.

Let μ be a probability distribution on $\{0,1\}^n$, $k \geq 0$ be an integer, and $\delta > 0$. We say that μ is (δ, k)-*biased* [18] if μ δ-fools all parity functions on k or fewer of the n binary variables. This is a relaxation of the k-wise independent property since the latter is equivalent to the $(0, k)$-bias property. If μ is (δ, n)-biased, it is called δ-*biased* [18].

We need the following relation.

THEOREM 2.1 (see [3]). *Any (δ, k)-biased probability distribution μ on $\{0,1\}^n$ is $n^k\delta$-close to a k-wise independent probability distribution μ' on $\{0,1\}^n$ in the sense that $|\mu(A) - \mu'(A)| \leq n^k\delta$ for all $A \subset \{0,1\}^n$. Thus if $f : \{0,1\}^n \to \{0,1\}$ is such that any k-wise independent probability distribution on $\{0,1\}^n$ ϵ-fools f, then any (δ, k)-biased probability distribution on $\{0,1\}^n$ $(\epsilon + \delta n^k)$-fools f.*

Using Theorems 1.1 and 2.1, we get the following corollary.

COROLLARY 2.2. *Any (δ, k)-biased probability distribution on $\{0,1\}^n$ $(16m^{2.2}$ $2^{-\sqrt{k}/10} + \delta n^k)$-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables.*

COROLLARY 2.3. *There is a function $\delta(m, n, \epsilon) = 2^{-\Theta(\log^2 \frac{m}{\epsilon} \log n)}$ such that for all positive integers m and n, and all $0 < \epsilon < 1$, any $\delta(m, n, \epsilon)$-biased probability distribution on $\{0,1\}^n$ ϵ-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables.*

Proof. Set $k = \lceil (10 \log \frac{32m^{2.2}}{\epsilon})^2 \rceil$ in Corollary 2.2 so that $16m^{2.2}2^{-\sqrt{k}/10} \leq \epsilon/2$ and $\delta n^k \leq \epsilon/2$ if $\delta \leq \epsilon 2^{-\lceil (10 \log \frac{32m^{2.2}}{\epsilon})^2 \rceil \log n - 1} \stackrel{\text{def}}{=} \delta(n, n, \epsilon)$. □

Related previously known bounds in [4, 16, 25] work for DNF formulas with very small fanins. We call a DNF formula an s-*DNF* if each clause contains at most s literal. We call a probability distribution on $\{0,1\}^n$ k-*wise* δ-*dependent* if it δ-fools all AND gates on at most k literals. Ajtai and Wigderson [4], and Luby and Velickovic [16] show that for all integers $1 \leq s \leq n$, any k-wise δ-dependent probability distribution μ on $\{0,1\}^n$ $(e^{-k/(s2^s)} + 2^s\delta)$-fools all boolean functions computable by s-DNF (or s-CNF) formulas on n variables. This implies that there is a function $\delta(s, \epsilon) = 2^{-O(s2^s \log(1/\epsilon))}$ such that for all $0 < \epsilon < 1$ and all integers $1 \leq s \leq n$, any $\delta(s, \epsilon)$-biased probability distribution μ on $\{0,1\}^n$ ϵ-fools all boolean functions computable by s-DNF (or s-CNF) formulas on n variables [25]. The bound is good for $s = O(1)$ and is nontrivial only if the condition $k > s2^s$ is satisfied. For general DNF formulas, we can restrict our attention to the case when $s = \log m + O(\log \frac{1}{\epsilon})$ (see section 5.4), but that will not help here since then the condition $k > s2^s$ would require $k > m \log m$.

2.2. A large class of PRGs for DNF formulas. The problem of derandomizing AC_0 circuits (polynomial-size constant-depth unbounded-fanin AND/OR circuits) was first studied by Ajtai and Wigderson [4]. Using Hastad's parity lower bound [9], Nisan [19] constructed a quasi-polynomial complexity[4] PRG for AC_0 circuits of seed length $O(\log^{2d+6} n)$, where d is the circuit depth and n is the input length. This initiated the hardness-versus-randomness approach which was developed in [21, 10] and others. Nisan's generator was optimized by Luby, Velickovic, and Wigderson [17] for depth-2 circuits reducing the seed length from $O(\log^{10} n)$ to $O(\log^4 mn)$, where m

[4] By quasi-polynomial complexity we mean $2^{\log^{\Theta(1)}(n)}$.

is the number of clauses of the DNF (or CNF) formula. Using classical linear code–based constructions of small probability spaces having the k-wise independent or the δ-bias property, we directly obtain from Theorem 1.1 a large class of explicit PRGs for depth-2 circuits of seed length $O(\log^3 mn)$.

Small probability spaces having the k-wise independence property can be constructed from linear codes. The following construction is folklore. If $C \subset \{0,1\}^n$ is a binary linear code whose dual $C^\perp \stackrel{\text{def}}{=} \{x \in \{0,1\}^n : \sum_i x_i y_i = 0 \pmod 2$ for all $y \in C\}$ has minimum distance greater than k, then the uniform distribution on the codewords of C is k-wise independent as a probability distribution on $\{0,1\}^n$. Classical linear code–explicit constructions achieve $|C| = n^{\Theta(k)}$.

COROLLARY 2.4. *For all positive integers m, n and every $\epsilon > 0$, there are an integer $t = O(\log^2 \frac{m}{\epsilon} \log n)$ and an explicit generator $G : \{0,1\}^t \to \{0,1\}^n$, constructible in poly$(n)$-time and computable in $O(tn)$-time, such that for every boolean function $g : \{0,1\}^n \to \{0,1\}$ computable by an m-clause DNF (or CNF) formula on n variables, we have*

$$|Pr_{x\in\{0,1\}^t}[g(G(x)) = 1] - Pr_{x\in\{0,1\}^n}[g(x) = 1]| \le \epsilon.$$

Proof. Without loss of generality, assume that $\log^2 \frac{m}{\epsilon} \log n = o(n)$ (otherwise, set $t = n$ and let G be the identity map). Given n, m, and ϵ, let

$$k \stackrel{\text{def}}{=} \left\lceil \left(10 \log \frac{16m^{2.2}}{\epsilon} \right)^2 \right\rceil;$$

thus $16m^{2.2}2^{-\sqrt{k}/10} \le \epsilon$. Construct in *poly*$(n, k)$-time the parity check matrix $H_{t\times n}$ of an explicit binary linear code D of block length n, message length $n - t$, and minimum distance at least $k + 1$, where $t = O(k \log n) = O(\log^2 \frac{m}{\epsilon} \log n)$. We can achieve $t = O(k \log n)$ using, for instance, a Reed–Solomon code or an algebraic geometry code reduced to a binary code by plain binarization or by concatenation, and punctured if necessary. Thus the probability distribution on $\{0,1\}^n$ resulting from choosing a random codeword from the dual $C = D^\perp$ of D is k-wise independent. This distribution is induced by the uniform distribution on $\{0,1\}^t$ via the $\mathbb{F}_2$-linear map $G : \{0,1\}^t \to \{0,1\}^n$ defined by $G(x) = xH$. □

Probability distributions with the δ-bias property can be explicitly constructed from linear codes with support size $\left(\frac{n}{\delta}\right)^{\Theta(1)}$ [18, 2]. Using those constructions and Corollary 2.3, we get a variation of Corollary 2.4 of asymptotically the same seed length, i.e., $t = O(\log^2 \frac{m}{\epsilon} \log n)$ (see also Corollary 11.2). Explicit constructions of (δ, k)-biased probability distributions of support size $(\frac{k \log n}{\delta})^{\Theta(1)}$ [18, 2] can also be used via Corollary 2.2 to achieve asymptotically the same seed length for $k = \lceil (10 \log \frac{32m^{2.2}}{\epsilon})^2 \rceil$ and $\delta = \frac{\epsilon}{2n^k}$.

For $\epsilon = n^{-O(1)}$, Corollary 2.4 and its variations give us PRGs of $O(\log^2 m \log n)$-seed lengths for m-clause DNF (or CNF) formulas.

Note that, when $\epsilon = n^{-O(1)}$, each of the above PRGs leads to a $2^{O(\log^2 m \log n)}$-time algorithm for the *DNF formula approximate counting problem* (given a DNF formula and $\epsilon > 0$, approximate the fraction of its satisfying assignments within $\pm\epsilon$ additive error). We should mention here that this does not improve the best known time for the DNF formula approximate counting problem; the algorithm of Luby and Velickovic [16], which is not based on a PRG, solves this problem in $(m \log n)^{exp(O(\sqrt{\log \log n}))}$-time when ϵ is constant.

The problems of constructing a logarithmic seed length (unconditional) PRG for DNF formulas or finding a polynomial-time algorithm for the DNF formula approximate counting problem remain open.

2.3. Patterns in binary linear codes. If $I \subset [n]$ and $\alpha \in \{0,1\}^I$, we call the pair (I,α) an *n-pattern*. We say that a string $x \in \{0,1\}^n$ *contains* an n-pattern $p = (I,\alpha)$ if $x_i = \alpha_i$ for all $i \in I$.

If $C \subset \{0,1\}^n$ is a binary linear code whose dual has minimum distance greater than k, then the uniform distribution on the codewords of C is k-wise independent as a probability distribution on $\{0,1\}^n$. Specialized to such k-wise independent distributions, Theorem 1.1 can be rephrased as an estimate of the probability that a random codeword of C contains a pattern from a given set of patterns.

COROLLARY 2.5. *Let A be a set consisting of m n-patterns, and let $C \subset \{0,1\}^n$ be a linear code whose dual has minimum distance greater than k. Then*

$$\left| Pr_{x\in C}\begin{bmatrix}\exists p \in A \text{ s.t.}\\ x \text{ contains } p\end{bmatrix} - Pr_{x\in\{0,1\}^n}\begin{bmatrix}\exists p \in A \text{ s.t.}\\ x \text{ contains } p\end{bmatrix}\right| \le 16m^{2.2}2^{-\sqrt{k}/10}.$$

Proof. Let μ be the k-wise independent probability distribution resulting from choosing a random codeword of C, and let F be the DNF formula whose clauses correspond to the patterns in A, i.e.,

$$F = \bigvee_{(I,\alpha)\in A}\left(\bigwedge_{i\in I:\alpha_i=1} x_i \wedge \bigwedge_{i\in I:\alpha_i=0}\neg x_i\right).$$

Apply Theorem 1.1 on μ and F. □

For instance, consider the following concrete case.

COROLLARY 2.6. *Let $C \subset \{0,1\}^n$ be a linear code whose dual has minimum distance greater than k, and let $1 \le t \le n$ be an integer. Then*

$$\left| Pr_{x\in C}\begin{bmatrix}x \text{ contains } t\\ \text{consecutive ones}\end{bmatrix} - Pr_{x\in\{0,1\}^n}\begin{bmatrix}x \text{ contains } t\\ \text{consecutive ones}\end{bmatrix}\right| \le 16(n-t+1)^{2.2}2^{-\sqrt{k}/10}.$$

Note that if the maximum size s of a pattern in A in Corollary 2.5 is $o(\sqrt{k})$, the bound can be improved via Corollary 9.6.

3. Fourier transform preliminaries. The study of boolean function using harmonic analysis methods dates back to the late 1960s. See, for instance, [12, 11, 13].

We assemble below some needed preliminaries: Fourier transform definition, Parseval's equality, the degree of a boolean function, and the Fourier truncation operator.

We identify the hypercube $\{0,1\}^n$ with the group $\mathbb{Z}_2^n \stackrel{\text{def}}{=} (\mathbb{Z}/2\mathbb{Z})^n$. The *characters* of the abelian group $\mathbb{Z}_2^n$ are $\{\mathcal{X}_y\}_{y\in\mathbb{Z}_2^n}$, where $\mathcal{X}_y(x) \stackrel{\text{def}}{=} (-1)^{\sum_{i=1}^n x_i y_i}$. Those characters form an orthogonal basis of the space of real-valued functions defined on $\mathbb{Z}_2^n$. They are orthogonal with respect to the uniform distribution; i.e., $E\mathcal{X}_y\mathcal{X}_{y'} = 0$ if $y \ne y'$ and 1 otherwise. If g is a real-valued function on $\mathbb{Z}_2^n$, we denote by $\widehat{g}$ the *Fourier transform* of g with respect to the characters of the abelian group $\mathbb{Z}_2^n$. That is, if $g : \{0,1\}^n \to \mathbb{R}$, its Fourier transform $\widehat{g} : \{0,1\}^n \to \mathbb{R}$ is given by the coefficients of the expansion of g in terms of the $\{\mathcal{X}_z\}_z$ basis:

$$g(x) = \sum_y \widehat{g}(y)\mathcal{X}_y(x) \quad \text{and} \quad \widehat{g}(y) = \frac{1}{2^n}\sum_x g(x)\mathcal{X}_y(x).$$

Parseval's equality relates the expected value of the square of a boolean function $g : \{0,1\}^n \to \mathbb{R}$ to the L_2-norm of its Fourier transform:

$$Eg^2 = \sum_y \widehat{g}(y)^2 = \|\widehat{g}\|_2^2.$$

The equality follows from the orthogonality of the characters $\{\mathcal{X}_y\}_y$.

If $x \in \mathbb{Z}_2^n$, the *weight* of x, which we denote by $|x|$, is the number of nonzero coordinates of x. The *degree* of $g : \{0,1\}^n \to \mathbb{R}$ is the smallest degree of a polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ such that $p(x) = g(x)$ for all $x \in \{0,1\}^n$. Equivalently, in terms of the basis $\{\mathcal{X}_y\}_y$, the *degree* of g is equal to the maximal weight of $y \in \{0,1\}^n$ such that $\widehat{g}(y) \neq 0$.

Finally, we define the Fourier truncation operator. Denote by $L(\{0,1\}^n)$ the space of real-valued functions on $\{0,1\}^n$. If $t \geq 0$ is an integer, define the *Fourier truncation operator* $\mathrm{Trn}_t : L(\{0,1\}^n) \to L(\{0,1\}^n)$ by

$$\mathrm{Trn}_t\ g = \sum_{y:|y|\leq t} \widehat{g}(y)\mathcal{X}_y.$$

The truncation operator Trn_t kills the high frequencies of g and produces a function $\mathrm{Trn}_t\ g$ of degree at most t. Equivalently, $\mathrm{Trn}_t\ g$ can be defined as the optimal solution f^* of the following L_2-approximation problem: given g and t, minimize $E(g-f)^2$ over the choice of $f : \{0,1\}^n \to \mathbb{R}$ of degree at most t. This follows from solving the underlying least square problem in the orthogonal basis $\{\mathcal{X}_y\}_y$. Note that, by Parseval's equality, the smallest L_2-approximation error is

$$E(g - \mathrm{Trn}_t g)^2 = \sum_{y:|y|>k} \widehat{g}(y)^2.$$

4. LP duality perspective.

DEFINITION 4.1. *We say that a distribution* property ϵ-fools *a function* $g : \{0,1\}^n \to \{0,1\}$ *if any probability distribution on* $\{0,1\}^n$ *with this property* ϵ*-fools* g.

We note that LP duality gives a purely analytical characterization of the class of boolean functions that are fooled by the k-wise independence property. The characterization is in terms of L_1-approximability by sandwiching polynomials of degree at most k. In particular, we show that the following theorem holds.

THEOREM 4.2. *Let* $g : \{0,1\}^n \to \{0,1\}$, $k \geq 0$ *be an integer, and* $\epsilon \geq 0$. *Then the* k*-wise independence property* ϵ*-fools* g *if and only if there exist* $g_l, g_u : \{0,1\}^n \to \mathbb{R}$ *such that*

(i) (low degree) $deg(g_l) \leq k$ *and* $deg(g_u) \leq k$;
(ii) (sandwiching polynomials) $g_l \leq g \leq g_u$;
(iii) (small L_1-approximation error) $E(g - g_l) \leq \epsilon$ *and* $E(g_u - g) \leq \epsilon$, *where the expectation is over the uniform probability distribution.*

We show the LP duality calculations in Appendix A in the more general context of the (δ, k)-bias property. Since the k-wise independence property is the $(0,k)$-bias property, Theorem 4.2 follows from Theorem A.1 in Appendix A by setting $\delta = 0$.

The proof of the main result of this paper in Theorem 1.1 depends only on the if part of Theorem 4.2. We give in this section a direct verification of the if part which does not involve LP duality calculations.

In terms of the $\{\mathcal{X}_y\}_y$ basis, the definition of k-wise independence can be rephrased as follows. Let μ be a probability distribution on $\{0,1\}^n$, and let $k \geq 0$ be an integer. Then the following are equivalent:

(a) μ is k-wise independent;
(b) $E_\mu \mathcal{X}_y = 0$ for each nonzero y in $\{0,1\}^n$ whose weight is less than or equal to k;
(c) $E_\mu p = Ep$ for each $p : \{0,1\}^n \to \mathbb{R}$ such that $deg(p) \leq k$, where the second expectation is with respect to the uniform probability distribution.

The equivalence between (a) and (b) is immediate. To relate to (c), write p as $p = \sum_{y:|y|\leq k} \widehat{p}(y)\mathcal{X}_y$; thus $E_\mu p = \widehat{p}(0) + \sum_{y\neq 0:|y|\leq k} \widehat{p}(y) E_\mu \mathcal{X}_y$.

This equivalence is the key relation between k-wise independent probability distributions and polynomials of degree at most k. Using this relation, we establish below the if part of Theorem 4.2. Let $g : \{0,1\}^n \to \{0,1\}$, $k \geq 0$ be an integer, and $\epsilon \geq 0$. Assume the existence of sandwiching polynomials g_l and g_u satisfying (i), (ii), and (iii). Let μ be a k-wise independent probability measure. We want to show that μ ϵ-fools g. Since g_u has degree at most k, $E_\mu g_u = Eg_u$. Hence

$$\begin{aligned} Pr_{x\sim\mu}[g(x) = 1] - Pr_{x\in\{0,1\}^n}[g(x) = 1] &= E_\mu g - Eg \\ &= E_\mu(g - g_u) + E(g_u - g) \leq E(g_u - g) \leq \epsilon, \end{aligned}$$

where the first inequality follows from the fact that $g_u \geq g$. Similarly, using g_l, we get

$$\begin{aligned} -Pr_{x\sim\mu}[g(x) = 1] + Pr_{x\in\{0,1\}^n}[g(x) = 1] &= -E_\mu g + Eg \\ &= E_\mu(g_l - g) + E(g - g_l) \leq E(g - g_l) \leq \epsilon. \end{aligned}$$

That is, μ ϵ-fools g.

5. Outline of proof. We outline and overview in this section the proof of Theorem 1.1, restated below.

THEOREM 1.1 *Any k-wise independent probability distribution on $\{0,1\}^n$ $(16m^{2.2}2^{-\sqrt{k}/10})$-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables.*

The proof is based on harmonic and poset analysis techniques. It uses Hastad's switching lemma [9] indirectly via the LMN energy bound [13], applied to many DNF formulas derived from the original DNF formula. The proof can be regarded as a sequence of reductions between some L_1- and L_2-approximations of DNF formulas and auxiliary functions by low degree polynomials with real coefficients. After some simplifications in section 5.1, we define those approximation notions in section 5.2, and then we outline the main steps in the proof in section 5.3.

5.1. Simplifications and notation. Without loss of generality, we restrict our attention to DNF formulas since any CNF formula is the negation of a DNF formula with the same number of clauses, and a probability distribution ϵ-fools a boolean function if and only if it ϵ-fools its negation.

To avoid degenerate cases, we assume that the DNF formula has at least one clause and that each clause has at least one literal. We can do this without loss of generality since any probability distribution 0-fools the identically one boolean function and the identically zero boolean function.

A final notational technicality is that if F is a DNF formula, we abuse notation and also denote by F the boolean function computed by F. That is, if $A_1, \ldots, A_m$ are

the clauses of F, we will denote by F also the boolean function $F : \{0,1\}^n \to \{0,1\}$ given by $F(x) = \bigvee_{c=1}^{m} A_c(x)$.

5.2. Approximation notions used in the proof. The proof uses the following three approximation notions of real-valued functions on the hypercube by low degree polynomials with real coefficients.

DEFINITION 5.1 (sandwiched L_1-approximation: bias). *If $g : \{0,1\}^n \to \{0,1\}$ is a boolean function and $k \geq 0$ is an integer, define the* k-bias of g, *denoted by bias$(g;k)$, to be the minimum value of ϵ such that there exist $g_l, g_u : \{0,1\}^n \to \mathbb{R}$ each of degree at most k such that $g_l \leq g \leq g_u$, $E(g_u - g) \leq \epsilon$, and $E(g - g_l) \leq \epsilon$. We call g_l and g_u sandwiching polynomials of g.*

Equivalently, by LP duality (Theorem 4.2), bias$(g;k)$ is the minimum value of ϵ such that any k-wise independent probability distribution μ on $\{0,1\}^n$ ϵ-fools g.

The term bias should not be confused with its other various meanings in the literature.

DEFINITION 5.2 (L_2-approximation: energy). *If $g : \{0,1\}^n \to \mathbb{R}$ is a real-valued function and $t \geq 0$ is an integer, define the* t-energy of g *to be energy$(g;t) = \min_f E(g-f)^2$ over the choice of a polynomial $f : \{0,1\}^n \to \mathbb{R}$ of degree at most t.*

Equivalently (see section 3),

$$\textit{energy}(g;t) = \sum_{y:|y|>t} \widehat{g}(y)^2.$$

That is, energy$(g;t)$ is the high energy content of g at frequencies above t, and hence the "energy" terminology.

We are allowing g to take real values since eventually we will be working with nonboolean-valued functions derived from DNF formulas.

DEFINITION 5.3 (constrained L_2-approximation: zero-energy). *If $g : \{0,1\}^n \to \{0,1\}$ is a boolean function and $t \geq 0$ is an integer, define the* t-zero-energy of g *to be zeroEnergy$(g;t) = \min_f E(g-f)^2$ over the choice of a polynomial $f : \{0,1\}^n \to \mathbb{R}$ of degree at most t satisfying the* zeros-constraint*: $f = 0$ whenever $g = 0$; i.e., $f(x) = 0$ for each $x \in \{0,1\}^n$ such that $g(x) = 0$.*

The zero-energy has no natural interpretation in the Fourier domain. The terminology is motivated by the above energy terminology and the zeros-constraint.

5.3. Main steps in the proof. Given an m-clause DNF formula F and $k \geq 0$, we want to bound its sandwiched L_1-approximation error bias$(F;k)$. The bound claimed in Theorem 1.1 is bias$(F;k) \leq 16m^{2.2}2^{-\sqrt{k}/10}$.

To get a concrete sense of the parameters, keep in mind the typical case when $k = \Theta(\log^2 m)$, and note that, although the number n of variables of F does not appear in the above bound or the subsequent ones, we care about the typical case when m is polynomial in n.

The proof can be regarded as a sequence of reductions between the above approximation notions in the context of DNF formulas and auxiliary functions. At a high level, we reduce the DNF sandwiched L_1-approximation problem to the DNF L_2-approximation problem, to which we apply the LMN energy bound. The DNF constrained L_2-approximation problem serves as an intermediate problem in this reduction.

The LMN energy bound [13] says that for each m-clause DNF formula F and each integer $t \geq 0$, energy$(F;t) \leq 2m2^{-\sqrt{t}/20}$.

To get started, we restrict our attention to DNF formulas with at most s literals per clause, where $s = \Theta(\sqrt{k})$. We call such DNF formulas s-DNF formulas. We can do that without loss of generality by paying a small additive error as noted in section 5.4 of this outline. Note that $s = \Theta(\log m)$ in the typical case when $k = \Theta(\log^2 m)$.

The first step of the proof reduces the problem of estimating the k-bias of an s-DNF formula to that of estimating its t-zero-energy, where $t = \lfloor (k-s)/2 \rfloor \approx k/2$. The argument is short and is in section 5.5 of this outline. Hence $s = \Theta(\sqrt{t})$, and $t = \Theta(\log^2 m)$ in the typical case when $k = \Theta(\log^2 m)$.

The second and more difficult step of the proof is estimating the zero-energy of an s-DNF formula, i.e., the s-DNF constrained L_2-approximation problem. We reduce this problem to the s-DNF L_2-approximation problem. The argument is long and involves two intermediate reductions to L_2-approximation problems of auxiliary nonboolean-valued functions associated with DNF formulas. First, we reduce the problem of bounding the t-zero-energy of an s-DNF formula F to that of bounding the t'-energies of auxiliary real-valued functions associated with DNF formulas derived from F, where $t' = t - s \approx t$. We give an overview of this in sections 5.6, 5.7, and 5.8. Then we reduce the problem of bounding the t'-energies of each of those auxiliary functions to that of bounding the t'-energies of additional derived DNF formulas, which finally enables us to use the LMN energy bound. We give an overview of this in section 5.9. We conclude in sections 5.10 and 5.11.

The arguments in the second step are based on harmonic and poset analysis machinery, which we develop in section 6. In this outline section, we give an overview of the underlying constructions and the end results without using the language of section 6.

5.4. Ignoring large clauses. If $s \geq 1$ is an integer, we call a DNF formula F an *s-DNF* if each clause of F contains at most s literals. By paying a small additive error, we can assume without loss of generality that the DNF formula does not contain very large clauses.

LEMMA 5.4. *Let $k \geq s \geq 1$ be integers and $\epsilon \geq 0$. If bias$(F;k) \leq \epsilon$ for each m-clause s-DNF formula F, then bias$(F;k) \leq \epsilon + m2^{-s}$ for each m-clause DNF formula F.*

At the end, we will set $s = \Theta(\sqrt{k})$. Hence $s = \Theta(\log m)$ in the typical case when $k = \Theta(\log^2 m)$.

Proof. The argument is easy. It is more direct in this lemma to work with the primal definition of the bias. That is, we argue on probability distribution and not on sandwiching polynomials.

Assume that the hypothesis is correct. Let F be an m-clause DNF formula on n variables, and let μ be a k-wise independent probability distribution on $\{0,1\}^n$. We want to show that $|Pr_\mu[F=1] - Pr[F=1]| \leq \epsilon + m2^{-s}$.

Let $A_1, \ldots, A_m$ be the clauses of F; thus $F = \bigvee_{c=1}^m A_c$. Let $C' \subset [m]$ be the set of indices of clauses each containing at most s literals, $F' = \bigvee_{c \in C'} A_c$, $C'' = [m] \backslash C'$, and $F'' = \bigvee_{c \in C''} A_c$.

Since F' is an s-DNF formula, we have $|Pr_\mu[F'=1] - Pr[F'=1]| \leq \epsilon$ because bias$(F';k) \leq \epsilon$ by the lemma hypothesis.

Since $F = F' \vee F''$, we have $Pr_\mu[F'=1] \leq Pr_\mu[F=1] \leq Pr_\mu[F'=1] + Pr_\mu[F''=1]$ and $-Pr[F'=1] - Pr[F''=1] \leq -Pr[F=1] \leq -P[F'=1]$. Thus

$$-\epsilon - Pr[F''=1] \leq Pr_\mu[F=1] - Pr_\mu[F=1] \leq \epsilon + Pr_\mu[F''=1].$$

The lemma then follows from the inequalities $Pr[F''=1] \leq |C''|2^{-(s+1)} \leq m2^{-s}$

and $Pr_\mu[F'' = 1] \leq |C''|2^{-s} \leq m2^{-s}$. The first inequality is immediate since each clause of F'' contains at least $s+1$ literals. To verify the second inequality, construct another DNF formula G from F'' by arbitrarily removing literals from each clause of F'' to make its size equal to s. By construction, G is satisfied by all the satisfying assignments of F; hence $Pr_\mu[F'' = 1] \leq Pr_\mu[G = 1]$. Since μ is k-wise independent and $k \geq s$, each clause of G is satisfied with a probability exactly 2^{-s} with respect to μ. Thus $Pr_\mu[G = 1] \leq |C''|2^{-s}$. □

5.5. From bias to zero-energy. In this section, we reduce the s-DNF sandwiched L_1-approximation problem to the s-DNF constrained L_2-approximation problem. In particular, we reduce the problem of estimating the k-bias of an s-DNF formula to that of estimating its t-zero-energy, where $t = \lfloor \frac{k-s}{2} \rfloor$.

To justify this move from L_1 to L_2, we briefly mention in Appendix B natural L_1-approaches which fall short of bounding the k-bias of s-DNF formulas.

We show that the following lemma holds.

LEMMA 5.5 (bias $\preccurlyeq$ zero-energy). *Let F be an m-clause s-DNF formula, and let $k \geq s$ be an integer. Then bias$(F; k) \leq m \times$ zeroEnergy$(F; t)$, where $t = \lfloor \frac{k-s}{2} \rfloor$.*

Note that $t \approx k/2$ when $s = \Theta(\sqrt{k})$. Moreover, $t = \Theta(\log^2 m)$ in the typical case when $k = \Theta(\log^2 m)$.

Proof. Assume that we have $f : \{0,1\}^n \to \mathbb{R}$ such that $deg(f) \leq \lfloor \frac{k-s}{2} \rfloor$, and that f satisfies the zeros-constraint: $f(x) = 0$ for each $x \in \{0,1\}^n$ such that $F(x) = 0$.

The approach is to construct the sandwiching polynomials f_l and f_u of F as

$$f_l \stackrel{\text{def}}{=} 1 - (1-f)^2 \text{ and}$$
$$f_u \stackrel{\text{def}}{=} 1 - \left(1 - \sum_{c=1}^{m} A_c\right)(1-f)^2,$$

where $A_1, \ldots, A_m$ are the clauses of F realized as polynomials on the variables $x_1, \ldots, x_n$. Since F is an s-DNF, the degree of each A_c is at most s. Hence, by construction, $deg(f_l), deg(f_u) \leq k$.

We want to show that

(i) $f_l \leq F \leq f_u$ and

(ii) $E(F - f_l) \leq mE(F-f)^2$ and $E(f_u - F) \leq mE(F-f)^2$.

To establish (i), let $x \in \{0,1\}^n$, and consider two cases depending on whether $F(x) = 0$ or 1.

If $F(x) = 0$, then $f(x) = 0$ by the zeros-constraint on f. Moreover, none of the clauses of F is satisfied by x; i.e., $A_c(x) = 0$ for each clause A_c of F. Hence $f_l(x) = 1 - (1-0)^2 = 0$ and $f_u(x) = 1 - (1-0)(1-0)^2 = 0$. That is, (i) holds with equality when $F(x) = 0$.

If $F(x) = 1$, there exists at least one clause c such that $A_c(x) = 1$; hence $1 - \sum_c A_c(x) \leq 0$. It follows that

$$f_u(x) = 1 - \left(1 - \sum_c A_c(x)\right)(1 - f(x))^2 \geq 1 = F(x).$$

Moreover, $f_l(x) = 1 - (1-f(x))^2 \leq 1 = F(x)$, which verifies (i) when $F(x) = 1$.

To establish (ii), it is enough to argue that $E(f_u - f_l) \leq mE(F-f)^2$ since (by (i)) $E(f_u - f_l)$ is an upper bound on both $E(F - f_l)$ and $E(f_u - F)$.

We have

$$f_u(x) - f_l(x) = \sum_c A_c(x)(1 - f(x))^2 = \sum_c A_c(x)(F(x) - f(x))^2.$$

To verify the second equality, consider two cases depending on whether $F(x) = 1$ or 0. The $F(x) = 1$ case is immediate. If $F(x) = 0$, then $A_c(x) = 0$ for each c, and hence the equality holds because both terms are zero. It follows that

$$E(f_u - f_l) = E\left(\sum_{c=1}^{m} A_c\right)(F - f)^2 \le mE(F - f)^2. \qquad \square$$

We derive in section 10 a compact form of the optimal solution of the least square problem underlying the definition of the t-zero-energy of an s-DNF formula (we focus on the monotone case for simplicity). Unable to estimate the optimal solution, we leave the problem open and construct next a suboptimal solution.

5.6. Construction overview. Let F be an m-clause s-DNF formula and $t \ge s$ be an integer. We want to upper bound the t-zero-energy of F, i.e., construct $f : \{0,1\}^n \to \mathbb{R}$ of degree at most t such that the mean square error $E(F - f)^2$ is small, and f satisfies the zeros-constraint: $f = 0$ whenever $F = 0$.

In this section we explain a construction of a function f satisfying the zeros-constraints. We do not analyze the corresponding mean square error, but we give some intuition as to why one would speculate that it is small. The mean square error analysis is fully presented in section 7. In a first reading, the reader may skip this overview section without loss of formal continuity and move to the end results in sections 5.7 and 5.8.

The zeros-constraint is behind the difficulty of the problem. It is worth mentioning that it excludes setting f to the truncation $\mathrm{Trn}_t F$ of F by the Fourier truncation operator Trn_t (defined in section 3). This choice minimizes the mean square error, but it is not an option for us because $\mathrm{Trn}_t F$ typically violates the zeros-constraint as it is rarely equal to 0 (or 1). We will not truncate the formula F, but we will apply truncation to carefully chosen components arising from rewriting the formula using inclusion-exclusion as we explain next.

For simplicity, we assume in this overview section that the DNF formula F is monotone. A DNF formula is called *monotone* if none of its clauses contains a negated variable. Represent F by a bipartite graph $F = (C, [n], N)$ between the set $C = [m]$ of clauses and the set $[n]$ of variable indices. For each clause $c \in C$, $N(c)$ is the neighborhood of c consisting of the indices of the variables in c. If $S \subset C$ is a set of clauses, let $N(S)$ be the neighborhood of S; i.e., $N(S) \stackrel{\text{def}}{=} \cup_{c \in S} N(c)$. If $z \subset [n]$ is a set of variable indices, denote the corresponding monotone AND gate by AND_z, i.e.,

$$AND_z(x) \stackrel{\text{def}}{=} \bigwedge_{i \in z} x_i = \prod_{i \in z} x_i,$$

for all $x \in \{0,1\}^n$. Thus

$$F(x) = \bigvee_{c \in C} AND_{N(c)}(x).$$

To construct f, expand F as follows:

$$\begin{aligned} F(x) &= 1 - \prod_{c \in C} \left(1 - \prod_{i \in N(c)} x_i\right) \\ &= \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} \prod_{i \in N(S)} x_i \\ &= \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} AND_{N(S)}(x). \end{aligned} \tag{5.1}$$

A direct way to obtain from this summation a low degree function which satisfies the zeros-constraint is to throw away the terms where $N(S)$ is larger than t. This reduces the degree to t and satisfies the zeros-constraint (since if $F = 0$, then $AND_{N(c)} = 0$ for each $c \in C$, and hence $AND_{N(S)} = 0$ for each $S \neq \emptyset \subset C$). But this does not work since the resulting mean square error may grow exponentially as t grows (the simplest example is when the DNF formula consists of a single OR gate on the n variables, i.e., $C = [n]$ and $N(i) = \{i\}$ for each $i \in C$).

Instead of throwing away the large terms, we will modify them while guaranteeing that each is still zero on the zeros of F. Let $S \subset C$ such that $S \neq \emptyset$, and consider the term corresponding to S. For each $c \in S$, we have

$$AND_{N(S)} = AND_{N(c)} AND_{N(S) \setminus N(c)}.$$

Averaging over all $c \in S$, we trivially get

$$AND_{N(S)} = E_{c \in S} AND_{N(c)} AND_{N(S) \setminus N(c)};$$

hence we can express F as

$$F = \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} E_{c \in S} AND_{N(c)} AND_{N(S) \setminus N(c)}.$$

Consider constructing f by truncating each $AND_{N(S) \setminus N(c)}$ to a degree-$(t - |N(c)|)$ polynomial via the Fourier truncation operator $\mathrm{Trn}_{t-|N(c)|}$. That is, define

$$f \stackrel{\text{def}}{=} \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} E_{c \in S} AND_{N(c)} \mathrm{Trn}_{t-|N(c)|} AND_{N(S) \setminus N(c)}.$$

The degree of each $\mathrm{Trn}_{t-|N(c)|} AND_{N(S) \setminus N(c)}$ is at most $t - |N(c)|$, and the degree of each $AND_{N(c)}$ is $|N(c)|$. Thus, by construction, $deg(f) \leq t$.

The key point is that if $F(x) = 0$, then $AND_{N(c)}(x) = 0$ for each clause $c \in C$, and hence $f(x) = 0$. That is, f satisfies the zeros-constraint.

To sum up, let F be a monotone s-DNF formula and $t \geq s$ be an integer. Then we have the bound

$$\mathrm{zeroEnergy}(F; t) \leq E(F - f)^2 = \|\widehat{F - f}\|_2^2, \tag{5.2}$$

where $F - f$ is the construction error term given by

$$F - f = \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} E_{c \in S} AND_{N(c)} (1 - \mathrm{Trn}_{t-|N(c)|}) AND_{N(S) \setminus N(c)}. \tag{5.3}$$

Needless to say, the signs $(-1)^{|S|}$ in the above summation are critical. That is, using a triangular inequality to upper bound $E(F-f)^2$ does not give a nontrivial bound since we have exponentially many terms whose values are not small enough to make the sum value less than 1.

Estimating the mean square error $E(F-f)^2 = \|\widehat{F-f}\|_2^2$ as t grows is difficult. Before proceeding with that, we give some intuition as to why one would speculate that the mean square error of this construction decays as t grows.

It can be shown that $\widehat{f}(y) = \widehat{\mathrm{Trn}_t F}(y)$ if $|y| \le t-s$ and that $\widehat{f}(y) = \widehat{\mathrm{Trn}_t F}(y) = 0$ if $|y| > t$ (see section 7 for a verification). In the region $t-s < |y| \le t$, $\widehat{f}(y)$ behaves oddly. Since $\widehat{f}(y)$ and $\widehat{\mathrm{Trn}_t F}(y)$ are equal outside this region, and since we know from the LMN energy bound (Theorem 5.8) that $\mathrm{energy}(F;t) = \|F - \widehat{\mathrm{Trn}_t F}\|_2^2$ decays quickly as t grows, we can hope that $\widehat{f}(y)$ is not too bad in the region $t-s < |y| \le t$ and hence speculate that $E(F-f)^2 = \|\widehat{F-f}\|_2^2$ also decays in some way with t. This makes f a potential candidate for bounding the t-zero-energy of F, but unfortunately this intuition does not help in the analysis as the frequencies in the region $t-s < |y| \le t$ are too many to handle separately by trivial bounds.

Using analytical means, we bound $\|\widehat{F-f}\|_2^2$ in section 7 in terms of the energies of auxiliary functions associated with DNF formulas derived from F, which reduces via (5.2) the problem of bounding the t-zero-energy of F to that of bounding the energies of those functions. After defining the auxiliary functions in section 5.7, we state the reduction in section 5.8 without going into the above construction of f.

5.7. Skin and cover auxiliary functions. The proof uses the following auxiliary functions associated with DNF formulas.

DEFINITION 5.6. *Let G be a DNF formula on n variables whose clauses are $A_1, \ldots, A_m$.*

- (Skin) *If $u \ge 0$, define the* u-skin of G *to be the real-valued function $\mathit{skin}_{G,u} : \{0,1\}^n \to \mathbb{R}$ given by*

$$\mathit{skin}_{G,u}(x) \stackrel{\mathrm{def}}{=} 1 - (1-e^{-u})^{\sum_{c=1}^m A_c(x)}.$$

  Note that $\sum_{c=1}^m A_c(x)$ is the number of clauses of G satisfied by x. The function $\mathit{skin}_{G,u}$ is extended to $u=0$ by continuity; i.e., $\mathit{skin}_{G,0} = G$.
- (Cover) *Define the* cover of G *to be the real-valued function $\mathit{cover}_G : \{0,1\}^n \to \mathbb{R}$ given by*

$$\mathit{cover}_G(x) \stackrel{\mathrm{def}}{=} \int_0^\infty \mathit{skin}_{G,u}(x) e^{-u} du = 1 - \frac{1}{1+\sum_{c=1}^m A_c(x)}.$$

To evaluate the integral, note that

$$\int_0^\infty (1-(1-e^{-u})^a) e^{-u} du = \int_0^\infty e^{-u} du - \int_0^1 (1-e^{-u})^a d(1-e^{-u}) = 1 - \frac{1}{1+a}$$

for all $a \ne -1$.

Remark 5.7. The function $\mathrm{skin}_{G,u}$ converges to G from above as u approaches 0, and hence the name u-skin of G. Moreover, $\mathrm{cover}_G \ge G$, and hence the name cover of G. The fact that both functions are greater than G is not used in the proof. Note that we defined cover_G as an integral. The fact that this integral evaluates to $1 - 1/(1+\sum_{c=1}^m A_c)$ is not used either in the proof (it is needed, however, to justify

the "cover" name). The proof is based on those functions in the Fourier domain. The origin of the skin and cover functions is the construction overviewed in section 5.6 above. We show in section 7 that the Fourier transform of the cover function naturally appears when analyzing the Fourier transform of the construction error term given in (5.3), and the skin function naturally appears when trying to recover the cover function from its Fourier transform.

5.8. From zero-energy to the energies of auxiliary functions. In this section, we state the end results of section 7 in which we reduce the problem of bounding the t-zero-energy of an s-DNF formula F to that of bounding the $(t-s)$-energies of cover and skin auxiliary functions associated with DNF formulas derived from F.

First, we reduce the problem of bounding the t-zero-energy of an s-DNF formula F to that of bounding the $(t-s)$-energies of the cover functions of DNF formulas derived from F as follows.

For simplicity, we start by stating the reduction in the context of monotone DNF formulas.

THEOREM 7.3 (zero-energy $\preccurlyeq$ energy of cover). *Let F be an s-DNF formula on the variables $x_1, \dots, x_n$, and let $t \geq s$ be an integer. Let $A_1, \dots, A_m$ be the clauses of F, and let $C = [m]$ be the set of indices of the clauses of F.*

(a) (Monotone case) *Assume that F is monotone and $m \geq 2$. For each clause index $c \in C$, let F_c be the DNF formula on the variables $x_1, \dots, x_n$ whose clauses are $\{A_c \wedge A_d\}_{d \in C \setminus \{c\}}$. That is, F_c is the formula resulting from removing from F the clause A_c and adding the variables of A_c to each of the remaining clauses. Then*

$$zeroEnergy(F; t) \leq m^2 \max_{c \in C} energy(cover_{F_c}; t - s).$$

(b) (General case) *We call two clauses A_c and A_d consistent if they have a common satisfying assignment. If $c \in C$, let C_c be the set of indices of clauses other than A_c which are consistent with A_c; i.e., $C_c = \{d \in C \setminus \{c\} : A_c$ and A_d are consistent$\}$. Let C_{main} be the set of indices of the clauses of F which are consistent with at least one clause of F other than themselves; i.e., $C_{main} = \{c \in C : C_c \neq \emptyset\}$.*

For each clause index $c \in C_{main}$, let F_c be the DNF formula on the variables $x_1, \dots, x_n$ whose clauses are $\{A_c \wedge A_d\}_{d \in C_c}$. That is, F_c is the formula resulting from removing from F the clause A_c and all the clauses not consistent with A_c, and adding the literals of A_c to each of the remaining clauses. Then

$$zeroEnergy(F; t) \leq |C_{main}|^2 \max_{c \in C_{main}} energy(cover_{F_c}; t - s).$$

Thus if F is monotone, then $C_c = C \setminus \{c\}$ for each $c \in C$, and $C_{main} = C$.

The proof of Theorem 7.3 is in section 7 and uses the machinery developed in section 6. The construction underlying the reduction is overviewed in section 5.6.

Note that each F_c is a $2s$-DNF formula on n variables with at most $m-1$ clauses and at least one clause (by the definition of C_{main}). That is, the complexity of each F_c is in the worst case comparable to that of F. Recall from section 5.5 that we care about the case when $s = \Theta(\sqrt{t})$ (since $t = \lfloor \frac{k-s}{2} \rfloor$ and $s = \Theta(\sqrt{k})$); hence $t - s \approx t$. Moreover, typically $t = \Theta(\log^2 m)$ (in the typical case when $k = \Theta(\log^2 m)$).

Therefore, in general, we can now focus on estimating the t-energy of the cover of a DNF formula G, where $t \geq 0$ is an integer. Unable to argue directly on the cover

function, we move to the u-skin function. The fact that $\text{cover}_G = \int_0^\infty \text{skin}_{G,u} e^{-u} du$ immediately reduces the problem of estimating the t-energy of the cover function of a DNF to that of estimating the t-energies of its u-skin functions for all $u \geq 0$. In particular, we have the following bound, which is verified via the Cauchy–Schwarz inequality in section 7.4.

LEMMA 7.4 (energy of cover $\preccurlyeq$ energy of skin). *Let G be a DNF formula, and let $t \geq 0$ be an integer. Then*

$$\textit{energy}(\textit{cover}_G; t) \leq \sup_{u \geq 0} \textit{energy}(\textit{skin}_{G,u}; t).$$

5.9. Back to DNF formulas. Let G be a DNF formula, $u \geq 0$, and $t \geq 0$ be an integer. We want to estimate the t-energy of the u-skin of G. We bound in section 8 the t-energy of the u-skin of G by the t-energies of DNF formulas derived from G by adding new auxiliary variables, which enables us to use the LMN energy bound.

First we state the reduction in the special case when there exists a nonnegative integer v such that $e^{-u} = 2^{-v}$. We construct from G a new DNF formula G_v by adding v new auxiliary nonnegated variables to each clause of G. Thus, the total number of variables of G_v is $n + mv$. We show in section 8 that $\text{energy}(\text{skin}_{G,u}; t) \leq \text{energy}(G_v; t)$. The proof is based on examining the Fourier transforms of $\text{skin}_{G,u}$ and uses the machinery developed in section 6.

In general, if v is not necessarily an integer, we show in section 8 that the following theorem holds.

THEOREM 8.1 (energy of skin $\preccurlyeq$ energy). *Let G be a DNF formula whose clauses are $A_1, \ldots, A_m$, and let $t \geq 0$ be an integer. If $d \in \mathbb{N}^m$, construct from G a new DNF formula G_d by adding d_c new auxiliary nonnegated variables to each clause A_c. That is, the clauses of G_d are $\{A_c \wedge \bigwedge_{i=1}^{d_c} \ddot{x}_{ci}\}_{c=1}^m$, where $\{\ddot{x}_{ci}\}_{i=1}^{d_c}$ are the new auxiliary variables added to clause A_c.*

Let $u \geq 0$ and let $v \geq 0$ such that $e^{-u} = 2^{-v}$; i.e., $v = u/\ln 2$. Then

$$\textit{energy}(\textit{skin}_{G,u}; t) \leq \max_{d \in \{\lfloor v \rfloor, \lceil v \rceil\}^m} \textit{energy}(G_d; t).$$

Note that for each $d \in \{\lfloor v \rfloor, \lceil v \rceil\}^m$, the formula G_d is an m-clause DNF on $n + \sum_c d_c$ variables.

This enables us to use the bound derived from Hastad's switching lemma in [13] on the t-energy of a DNF formula.

THEOREM 5.8 (LMN energy bound [13]). *Let G be an m-clause DNF formula, and let $t \geq 0$ be an integer; then $\textit{energy}(G; t) \leq 2m2^{-\sqrt{t}/20}$.*

The proof of (an asymptotic version of) Theorem 1.1 follows by first substituting the bound of Theorem 5.8 in Theorem 8.1 and backtracking the bounds till Lemma 5.4. We summarize in section 5.10; then we backtrack the bounds in section 5.11.

5.10. Summary. Tables 1 and 2 below summarize the main definitions and reductions.

The full presentation in sections 7 and 8 uses the machinery in section 6. In sections 6, 7, and 8, we separate between the monotone case and the general case to introduce the arguments in a simple context. We recommend that the reader first traverse the monotone part of the full sections in the following order: sections 6.1, 6.2, 6.3, 7.1, 7.2, 7.4, introduction of section 8.

For future reference in sections 5.11 and 9, we list below compact statements of the above reductions.

TABLE 1

Notion	Terminology	Definition
k-bias of a function $g : \{0,1\}^n \to \{0,1\}$	$\mathrm{bias}(g;k)$	Definition 5.1
Fourier truncation operator	Trn_t	section 3
t-energy of a function $g : \{0,1\}^n \to \mathbb{R}$	$\mathrm{energy}(g;t)$	Definition 5.2
t-zero-energy of a function $g : \{0,1\}^n \to \{0,1\}$	$\mathrm{zeroEnergy}(g;t)$	Definition 5.3
boolean function of a DNF formula G	G by notational abuse	section 5.1
u-skin function of a DNF formula G	$\mathrm{skin}_{G,u}$	Definition 5.6
cover function of a DNF formula G	cover_G	Definition 5.6

TABLE 2

Reduction	Statement	Overview	Full presentation
bias $\preccurlyeq$ zero-energy	Lemma 5.5		section 5.5
zero-energy $\preccurlyeq$ energy of cover	Theorem 7.3	sections 5.6, 5.8	section 7
energy of cover $\preccurlyeq$ energy of skin	Lemma 7.4	section 5.8	section 7
energy of skin $\preccurlyeq$ energy	Theorem 8.1	section 5.9	section 8

- LEMMA 5.4 (focus on s-DNF). Let $k \geq s \geq 1$ be integers, and let $\epsilon \geq 0$. If $\mathrm{bias}(F;k) \leq \epsilon$ for each m-clause s-DNF formula F, then $\mathrm{bias}(F;k) \leq \epsilon + m2^{-s}$ for each m-clause DNF formula F.
  At the end, we will set $s = \Theta(\sqrt{k})$. Thus $s = \Theta(\log m)$ in the typical case when $k = \Theta(\log^2 m)$.
- LEMMA 5.5 (bias $\preccurlyeq$ zero-energy). Let F be an m-clause s-DNF formula, and let $k \geq s$ be an integer. Then $\mathrm{bias}(F;k) \leq m \times \mathrm{zeroEnergy}(F;t)$, where $t = \lfloor \frac{k-s}{2} \rfloor$.
  Note that $t \approx k/2$ for $s = \Theta(\sqrt{k})$. Moreover, $t = \Theta(\log^2 m)$ in the typical case when $k = \Theta(\log^2 m)$.
- THEOREM 7.3 (zero-energy $\preccurlyeq$ energy of cover). Let F be an s-DNF formula on the variables $x_1, \ldots, x_n$, and let $t \geq s$ be an integer. Let $A_1, \ldots, A_m$ be the clauses of F. Let $C = [m]$ be the set of indices of the clauses of F. If $c \in C$, let $C_c = \{d \in C \backslash \{c\} : A_c \text{ and } A_d \text{ are consistent}\}$. Let $C_{main} = \{c \in C : C_c \neq \emptyset\}$. For each $c \in C_{main}$, let F_c be the DNF formula on the variables $x_1, \ldots, x_n$ whose clauses are $\{A_c \wedge A_d\}_{d \in C_c}$. Then

$$\mathrm{zeroEnergy}(F;t) \leq |C_{main}|^2 \max_{c \in C_{main}} \mathrm{energy}(\mathrm{cover}_{F_c}; t-s).$$

  Note that for each $c \in C_{main}$, F_c is a $2s$-DNF formula with at most $m-1$ clauses and least one clause. Moreover, $|C_{main}| \leq |C| = m$. Note also that $t - s \approx t \approx k/2$ for $t = \lfloor \frac{k-s}{2} \rfloor$ and $s = \Theta(\sqrt{k})$.
- LEMMA 7.4 (energy of cover $\preccurlyeq$ energy of skin). Let G be a DNF formula, and let $t \geq 0$ be an integer. Then

$$\mathrm{energy}(\mathrm{cover}_G; t) \leq \sup_{u \geq 0} \mathrm{energy}(\mathrm{skin}_{G,u}; t).$$

- THEOREM 8.1 (energy of skin $\preccurlyeq$ energy). Let G be a DNF formula, whose clauses are $A_1, \ldots, A_m$, and let $t \geq 0$ be an integer. If $d \in \mathbb{N}^m$, construct from G a new DNF formula G_d by adding d_c new auxiliary nonnegated variables to each clause A_c. Let $u \geq 0$ and $v = u/\ln 2$. Then

$$\mathrm{energy}(\mathrm{skin}_{G,u}; t) \leq \max_{d \in \{\lfloor v \rfloor, \lceil v \rceil\}^m} \mathrm{energy}(G_d; t).$$

Note that for each d, the number of clauses of G_d is equal to that of G.

- THEOREM 5.8 (LMN energy bound). Let G be an m-clause DNF formula, and let $t \geq 0$ be an integer; then $\text{energy}(G;t) \leq 2m2^{-\sqrt{t}/20}$.

5.11. Backtracking. In this section, we derive an asymptotic version of Theorem 1.1 by substituting the bound of Theorem 5.8 in Theorem 8.1 and backtracking the bounds via Lemma 7.4, Theorem 7.3, Lemma 5.5, till Lemma 5.4. Namely, we show that if F is an m-clause DNF formula and $k \geq 1$ is an integer, then $\text{bias}(F;k) = O(m^{\Theta(1)}2^{-\Theta(\sqrt{k})})$. Since the bound of Theorem 5.8 does not depend on the maximum number of literals in a clause, the needed calculations are minimal.

Let G be an m-clause DNF formula, and let $t \geq 0$. Substituting the bound of Theorem 5.8 in Theorem 8.1, we get that $\text{energy}(\text{skin}_{G,u};t) \leq 2m2^{-\sqrt{t}/20}$ for all $u \geq 0$. Substituting in Lemma 7.4, we obtain $\text{energy}(\text{cover}_G;t) \leq 2m2^{-\sqrt{t}/20}$. Thus, by Theorem 7.3, if F is an m-clause s-DNF formula and $t \geq s$ is an integer, then $\text{zeroEnergy}(F;t) \leq 2m^2(m-1)2^{-\sqrt{t-s}/20}$. It follows from Lemma 5.5 that if $k \geq s$ is an integer, then $\text{bias}(F;k) \leq 2m^3(m-1)2^{-\sqrt{\lfloor (k-s)/2 \rfloor - s}/20}$. Finally, substituting in Lemma 5.4, we get that if F is an m-clause DNF formula and $k \geq 1$ is an integer, then

$$\text{bias}(F;k) \leq m2^{-s} + 2m^3(m-1)2^{-\sqrt{\lfloor (k-s)/2 \rfloor - s}/20}$$

for all integers s such that $k \geq s \geq 1$. Optimizing on s, we obtain $\text{bias}(F;k) = O(m^4 2^{-\Theta(\sqrt{k})})$ for $s = \Theta(\sqrt{k})$.

The exact bound $m^{2.2}2^{-\sqrt{k}/10}$ of Theorem 1.1 is derived in section 9. It uses another form of the LMN energy bound which is tighter than Theorem 5.8 for s-DNF formulas when s is not relatively large (Theorem 9.1 in section 9), and a sharper form of Lemma 7.4 (part (a) of Lemma 7.4 in section 7).

6. Möbius and Fourier analysis of DNF formulas and auxiliary functions. We develop in this section the proof machinery used in sections 7 and 8.

We develop the monotone machinery in sections 6.2 and 6.3. Monotone DNF formulas and their skin and cover auxiliary functions have natural expansions as linear combinations of monotone AND gates. We are interested in the coefficients of those expansions. The Fourier transforms of those functions can be extracted from those coefficients by a basis change. When expanding a real-valued function f defined on the hypercube as a linear combination of monotone AND gates, it is convenient to view the hypercube as the poset B_n of subsets of $[n]$ ordered by inclusion. We note in section 6.2 that this enables us to interpret the coefficients of the expansion of f as the Möbius transform of f with respect to the poset B_n. We also derive a simple change of basis formula to extract the Fourier transform of a function from its Möbius transform. In section 6.3, we compute the Möbius and Fourier transforms of monotone DNF formulas and their auxiliary functions. The monotone machinery is used in sections 7.1 and 8.

The poset language is essential to generalize to nonnecessarily monotone DNF formulas. The monotone machinery naturally generalizes to the nonnecessarily monotone case by essentially replacing the poset B_n with another poset $B_n^{(2)}$, defined in section 6.4. We develop the general machinery in sections 6.4 and 6.5. It is used in section 7.3.

6.1. Posets preliminaries. For a general reference on posets, see [24]. We need only a few elementary definitions. A *poset* (partially ordered set) X is a set X with

a reflexive, antisymmetric, and transitive binary relation $\leq_X$. We denote $\leq_X$ by $\leq$ when there is no confusion. We implicitly assume that X is finite. We denote the set of real-valued functions on X by $L(X) = \{f : X \to \mathbb{R}\}$. The *zeta function* ζ_X of X is the linear transformation $\zeta_X : L(X) \to L(X)$ given by

$$(\zeta_X f)(x) = \sum_{y \leq x} f(y) = \sum_{y \in X} \zeta_X(y, x) f(y).$$

That is, the matrix coefficients $(\zeta_X(y, x))_{x,y}$ of ζ_X are given by

$$\zeta_X(y, x) = \begin{cases} 1 & \text{if } y \leq x, \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

The zeta function ζ_X is always nonsingular. The inverse ζ_X^{-1} of ζ_X is called the *Möbius function* of X and is denoted by $\mu_X = \zeta_X^{-1}$.

We are interested in two posets defined in sections 6.2 and 6.4.

6.2. Poset B_n. Let $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Let B_n be the poset of subsets of $[n]$ ordered by inclusion. We denote the subset inclusion $a \subset b$ by $a \leq b$ and $b \geq a$.

We identify the hypercube $\{0, 1\}^n$ with the poset B_n by associating $x \in \{0, 1\}^n$ with $\text{support}(x) \stackrel{\text{def}}{=} \{i \in [n] : x_i = 1\} \in B_n$. Thus $x \in B_n$ means both a subset of $[n]$ and a vector in $\{0, 1\}^n$ depending on the context.

If $z \in B_n$, define the monotone AND function $AND_z : B_n \to \{0, 1\}$ by

$$AND_z(x) \stackrel{\text{def}}{=} \bigwedge_{i \in z} x_i.$$

The functions $\{AND_z\}_{z \in B_n}$ form a basis of $L(B_n)$. When working with the $\{AND_z\}_{z \in B_n}$ basis, it is convenient to view the hypercube as the poset B_n since

$$AND_z(x) = \zeta_{B_n}(z, x) \tag{6.2}$$

by (6.1). That is, the functions $\{AND_z\}_{z \in B_n}$ are the rows of the matrix of the zeta function ζ_{B_n} of the poset B_n. Any function $f \in L(B_n)$ can be expressed as

$$f = \sum_{z \in B_n} \widetilde{f}(z) AND_z$$

for some $\widetilde{f} \in L(B_n)$. By (6.2),

$$f(x) = \sum_{z \in B_n} \widetilde{f}(z) \zeta_{B_n}(z, x) = (\zeta_{B_n} \widetilde{f})(x).$$

That is, $f = \zeta_{B_n} \widetilde{f}$, and hence $\widetilde{f} = \mu_{B_n} f$. We call $\widetilde{f} = \mu_{B_n} f$ the *Möbius transform* of f.

Our interest in the Möbius transform on B_n is motivated by the fact that monotone DNF formulas and auxiliary functions have natural expansions in the $\{AND_z\}_z$ basis from which we can extract their Möbius transforms. We show that in section 6.3.

Given the Möbius transform of a function, its Fourier transform can be extracted via a simple weighted summation, which we derive next. Recall the $(\mathbb{Z}/2\mathbb{Z})^n$ group structure on $\{0, 1\}^n$ from section 3. In terms of the identification of $\{0, 1\}^n$ with B_n, B_n is an abelian group under the set exclusive union operation, which we denote by

$\oplus$. In the B_n-terminology, the characters of this abelian group defined in section 3 are $\{\mathcal{X}_y(x) = (-1)^{|x\cap y|}\}_{y\in B_n}$.

To extract the Fourier transform of a function $f \in L(B_n)$ from its Möbius transform, we need a change of basis formula between the $\{AND_z\}_z$ basis and the $\{\mathcal{X}_y\}_y$ basis of $L(B_n)$. We have

$$AND_z(x) = \bigwedge_{i\in z} x_i = \prod_{i\in z} x_i = \prod_{i\in z} \frac{1-(-1)^{x_i}}{2} = \frac{1}{2^{|z|}} \sum_{y\le z} (-1)^{|y|}(-1)^{\sum_{i\in y} x_i}.$$

That is,

$$AND_z(x) = \frac{1}{2^{|z|}} \sum_{y\le z} (-1)^{|y|} \mathcal{X}_y(x). \tag{6.3}$$

Note that if $z = \emptyset$, by convention $\prod_{i\in z} x_i = \bigwedge_{i\in z} x_i = 1$. Thus

$$\begin{aligned} f(x) &= \sum_z \widetilde{f}(z) AND_z(x) \\ &= \sum_z \widetilde{f}(z) 2^{-|z|} \sum_{y\le z} (-1)^{|y|} \mathcal{X}_y(x) \\ &= \sum_y \mathcal{X}_y(x) (-1)^{|y|} \sum_{z\ge y} 2^{-|z|} \widetilde{f}(z). \end{aligned}$$

Therefore, the desired change of basis formula is

$$\widehat{f}(y) = (-1)^{|y|} \sum_{z\ge y} 2^{-|z|} \widetilde{f}(z) \quad \text{for all } y \in B_n \text{ and } f \in L(B_n). \tag{6.4}$$

Conversely, one can verify that

$$\widetilde{f}(z) = (-1)^{|z|} 2^{|z|} \sum_{y\ge z} \widehat{f}(y),$$

but we will not use that.

Finally, if $f : \{0,1\}^n \to \mathbb{R}$, the degree of f is the smallest degree of a polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ such that $p(x) = f(x)$ for all $x \in \{0,1\}^n$. In terms of the $\{\mathcal{X}_y\}_y$ basis, the degree of f is equal to the maximal cardinality of $y \in B_n$ such that $\widehat{f}(y) \neq 0$. In terms of the $\{AND_z\}_z$ basis, the degree of f is equal to the maximal cardinality of $z \in B_n$ such that $\widetilde{f}(z) \neq 0$.

6.3. Monotone DNF formulas and auxiliary functions. In this section we compute the Möbius and Fourier transforms of monotone DNF formulas and auxiliary functions.

A DNF formula is called *monotone* if none of its clauses contains a negated variable. We represent a monotone DNF formula F on n variables by a bipartite graph $F = (C, [n], N)$ between the set C of clauses and the set $[n]$ of variables indices. For each clause $c \in C$, $N(c)$ is the neighborhood of c consisting of the indices of the variables in c. To avoid degenerate cases, we assume that we have at least one clause, i.e., $|C| \ge 1$, and that each clause contains at least one variable, i.e., $N(c) \neq \emptyset$, for each $c \in C$. If $S \subset C$, $N(S)$ denotes the neighborhood of S, i.e., $N(S) = \cup_{c\in S} N(c)$. Finally, if $s \ge 1$ is an integer, we call a monotone DNF formula $F = (C, [n], N)$ an

s-DNF formula if each clause contains at most s variables, i.e., $|N(c)| \leq s$, for each $c \in C$.

We chose the bipartite graph representation to allow for duplicate clauses; formulas possibly containing duplicate clauses will be derived in the proof of Theorem 1.1 (see section 7.2.C).

If $F = (C, [n], N)$ is a monotone DNF formula, we abuse notation and denote by F also the boolean function computed by F. That is, the boolean function $F : B_n \to \{0, 1\}$ is given by

$$F(x) \stackrel{\text{def}}{=} \bigvee_{c \in C} AND_{N(c)}(x) \qquad \text{for } x \in B_n.$$

Monotone DNF formulas and auxiliary functions have natural expansions in the $\{AND_z\}_z$ basis from which we can extract their Möbius transforms. To get the Fourier transforms, we use the change of basis formula (6.4). To warm up, let us compute the Möbius and Fourier transforms of F.

LEMMA 6.1. *Let $F = (C, [n], N)$ be a monotone DNF formula. Then for all $z, y \in B_n$,*

$$\widetilde{F}(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|}, \tag{6.5}$$

$$\widehat{F}(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-|N(S)|}. \tag{6.6}$$

Proof. By expanding $F(x)$ as in (5.1) and grouping terms, we get

$$F(x) = \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} AND_{N(S)}(x) = \sum_{z \in B_n} AND_z(x) \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|},$$

which verifies (6.5). The correctness of (6.6) follows from (6.5) via (6.4):

$$\begin{aligned} \widehat{F}(y) &= (-1)^{|y|} \sum_{z \geq y} 2^{-|z|} \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|} \\ &= (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-|N(S)|}. \end{aligned}$$

□

If $F = (C, [n], N)$ is a monotone DNF formula and $u \geq 0$, recall from Definition 5.6 the auxiliary functions u-skin of F and cover of F: $\text{skin}_{F,u}, \text{cover}_F \in L(B_n)$ are given by

$$\begin{aligned} \text{skin}_{F,u}(x) &\stackrel{\text{def}}{=} 1 - (1 - e^{-u})^{\sum_{c \in C} AND_{N(c)}(x)}, \\ \text{cover}_F(x) &\stackrel{\text{def}}{=} \int_0^\infty \text{skin}_{F,u}(x) e^{-u} du, \end{aligned}$$

where $\text{skin}_{F,u}$ is extended to $u = 0$ by continuity, i.e., $\text{skin}_{F,0} = F$.

We compute below their Möbius and Fourier transforms which play a critical role in the proof of Theorem 1.1 as shown in sections 7 and 8.

LEMMA 6.2. *Let $F = (C, [n], N)$ be a monotone DNF formula, and let $u \geq 0$. Then for all $z, y \in B_n$,*

$$\widetilde{skin}_{F,u}(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|} e^{-u|S|}, \tag{6.7}$$

$$\widetilde{cover}_F(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|} \frac{1}{|S|+1}, \tag{6.8}$$

$$\widehat{skin}_{F,u}(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-|N(S)|} e^{-u|S|}, \tag{6.9}$$

$$\widehat{cover}_F(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-|N(S)|} \frac{1}{|S|+1}. \tag{6.10}$$

Remark 6.3.

1. Our interest in the cover and skin functions originates from the last summation in (6.10). The analysis of the construction error term overviewed in section 5.6 and fully presented in section 7 leads to summations like the right-hand side of (6.10) (see the proof of Lemma 7.1 and section 7.2.A). To interpret this summation, we expressed $\frac{1}{|S|+1}$ as $\frac{1}{|S|+1} = \int_0^\infty e^{-u|S|} e^{-u} du$, which led us to the right-hand side of (6.9). Using the Fourier–Möbius change of basis formula (6.4), we obtained (6.7) which we identified as the Möbius transform of the skin function as shown in the proof below.
2. Comparing Lemmas 6.1 and 6.2, we see that the Möbius and Fourier transforms of the u-skin and cover of F are smoothed or weighted versions of those of F. The smoothing or weighting factor of the u-skin function is $e^{-u|S|}$, and that of the cover function is $\frac{1}{|S|+1}$.

Proof. We start with (6.7). It is enough to verify it under the assumption that $u > 0$. The $u = 0$ case follows from (6.5). We have

$$1 - (1 - e^{-u})^{\sum_{c \in C} AND_{N(c)}(x)} = 1 - \prod_{c \in C} (1 - e^{-u})^{AND_{N(c)}(x)}.$$

Since $u > 0$, we have $(1 - e^{-u})^{AND_{N(c)}(x)} = 1 - e^{-u} AND_{N(c)}(x)$ for all $c \in C$ and all $x \in \{0,1\}^n$ (if $AND_{N(c)}(x) = 0$, both terms are 1; if $AND_{N(c)}(x) = 1$, both terms are $1 - e^{-u}$). Thus

$$\begin{aligned} 1 - (1 - e^{-u})^{\sum_{c \in C} AND_{N(c)}(x)} &= 1 - \prod_{c \in C} (1 - AND_{N(c)}(x) e^{-u}) \\ &= \sum_{S \neq \emptyset \subset C} -(-e^{-u})^{|S|} \prod_{c \in S} AND_{N(c)}(x) \\ &= \sum_{S \neq \emptyset \subset C} -(-1)^{|S|} e^{-u|S|} AND_{N(S)}(x) \\ &= \sum_{z \in B_n} AND_z(x) \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|} e^{-u|S|}, \end{aligned} \tag{6.11}$$

which verifies (6.7). Applying the linear operator μ_{B_n} to $\mathrm{cover}_F = \int_0^\infty \mathrm{skin}_{F,u} e^{-u} du$,

we get

$$\begin{aligned}\widetilde{\mathrm{cover}}_F(z) &= \int_0^\infty \widetilde{\mathrm{skin}}_{F,u}(z)e^{-u}du \\ &= \sum_{S\neq\emptyset\subset C:N(S)=z} -(-1)^{|S|}\int_0^\infty e^{-u(|S|+1)}du \\ &= \sum_{S\neq\emptyset\subset C:N(S)=z} -(-1)^{|S|}\frac{1}{|S|+1},\end{aligned}$$

which verifies (6.10). Finally, as in Lemma 6.1, (6.10) and (6.9) follow immediately from (6.8) and (6.7) via (6.4). □

6.4. The poset $B_n^{(2)}$. To generalize the monotone machinery in sections 6.2 and 6.3 to the nonnecessarily monotone case, we basically only have to replace the poset B_n with another poset $B_n^{(2)}$, which we study in this section. Below, we define this poset and give the corresponding analogues of (6.2) and (6.4).

When the DNF formula is not necessarily monotone, the AND gates are of the form

$$ANDz_{(z',z'')}(x) = \bigwedge_{i\in z'} x_i \wedge \bigwedge_{i\in z'} \neg x_i = ANDz_{z'}(x)\wedge ANDz_{z''}(x^c) \quad \text{for } x\in B_n,$$

where $(z',z'')\in B_n\times B_n$ are such that $z'\cap z''=\emptyset$ and $x^c \stackrel{\text{def}}{=} [n]\backslash x \in B_n$.

This motivates looking at the poset $B_n^{(2)}$ defined as follows. Consider the product poset $B_n^2 \stackrel{\text{def}}{=} B_n\times B_n$; i.e., the order relation on B_n^2 is given by $(x',x'')\le(y',y'')$ if $x'\le y'$ and $x''\le y''$. Let $B_n^{(2)}$ be the poset given by

$$B_n^{(2)} \stackrel{\text{def}}{=} \{(x',x'')\in B_n^2 : x'\cap x''=\emptyset\} \tag{6.12}$$

and ordered via the ordered relation of B_n^2. That is, $B_n^{(2)}$ is the subposet of B_n^2 given by (6.12).

If $x\in B_n^{(2)}$, we denote by x' and x'' the elements of B_n such that $x=(x',x'')$.

If $z\in B_n^{(2)}$, the corresponding AND gate $ANDz_z : B_n\to\{0,1\}$ is given by

$$ANDz_z(x) \stackrel{\text{def}}{=} ANDz_{z'}(x)\wedge ANDz_{z''}(x^c) \quad \text{for } x\in B_n.$$

To get a relation similar to (6.2), lift $ANDz_z$ to $B_n^{(2)}$ as follows. If $z\in B_n^{(2)}$, define $\overline{ANDz}_z : B_n^{(2)}\to\{0,1\}$ by

$$\overline{ANDz}_z(x) \stackrel{\text{def}}{=} ANDz_{z'}(x')\wedge ANDz_{z''}(x'') = \zeta_{B_n^{(2)}}(z,x).$$

Thus

$$ANDz_z(x) = \overline{ANDz}_z(x,x^c) \quad \text{for } x\in B_n,$$

and

$$\overline{ANDz}_z(x) = \zeta_{B_n^{(2)}}(z,x) \quad \text{for } x\in B_n^{(2)}, \tag{6.13}$$

which is the analogue of (6.2) on $B_n^{(2)}$. We are using the bar-notation to indicate that $\overline{AND}_z$ is the *lift* of AND_z from B_n to $B_n^{(2)}$ (the bar-notation should not be confused with negation).

The functions $\{\overline{AND}_z\}_{z\in B_n^{(2)}}$ form a basis for $L(B_n^{(2)})$ since by (6.13) they are the rows of the matrix of the invertible linear transformation $\zeta_{B_n^{(2)}}$. Any function $f \in L(B_n^{(2)})$ can be expressed as

$$f = \sum_{z\in B_n^{(2)}} \widetilde{f}(z)\overline{AND}_z$$

for some $\widetilde{f} \in L(B_n^{(2)})$. Indeed, by (6.13),

$$f(x) = \sum_z \widetilde{f}(z)\zeta_{B_n^{(2)}}(z,x) = (\zeta_{B_n^{(2)}} f)(x).$$

That is, $\widetilde{f} = \mu_{B_n^{(2)}} f$ and $f = \zeta_{B_n^{(2)}} \widetilde{f}$. We call $\widetilde{f} = \mu_{B_n^{(2)}} f$ the *Möbius transform* of f.

We need an analogue of (6.4) on $B_n^{(2)}$. If $f \in L(B_n^{(2)})$, we relate next the Fourier transform of its projection to B_n to the Möbius transform of f.

Consider the injective embedding $B_n \to B_n^{(2)}$, $x \mapsto (x, x^c)$. It induces the linear map $\text{Proj} : L(B_n^{(2)}) \to L(B_n)$ given by $(\text{Proj } f)(x) = f(x, x^c)$. In terms of Proj, we have $AND_z = \text{Proj } \overline{AND}_z$; thus

$$(\text{Proj } f)(x) = \sum_z \widetilde{f}(z) AND_z(x).$$

If $f \in L(B_n^{(2)})$, we show below how to extract $\widehat{\text{Proj } f}$ from $\widetilde{f}$. Let $z \in B_n^{(2)}$. From (6.3), we have

$$AND_{z'}(x) = 2^{-|z'|} \sum_{y'\le z'} (-1)^{|y'|} \mathcal{X}_{y'}(x)$$

and

$$AND_{z''}(x^c) = 2^{-|z''|} \sum_{y''\le z''} (-1)^{|y''|} \mathcal{X}_{y''}(x^c).$$

Now

$$\mathcal{X}_{y''}(x^c) = (-1)^{|y''\cap x^c|} = (-1)^{|y''|-|y''\cap x|} = (-1)^{|y''|}\mathcal{X}_{y''}(x);$$

hence

$$AND_{z''}(x) = 2^{-|z''|} \sum_{y''\le z''} \mathcal{X}_{y''}(x).$$

It follows that

$$\begin{aligned} AND_z(x) &= AND_{z'}(x) AND_{z''}(x^c) \\ &= 2^{-|z'|} \sum_{y'\le z'} (-1)^{|y'|} \mathcal{X}_{y'}(x) 2^{-|z''|} \sum_{y''\le z''} \mathcal{X}_{y''}(x) \\ &= 2^{-|z|} \sum_{y\le z} (-1)^{|y'|} \mathcal{X}_{y'\oplus y''}(x) \\ &= 2^{-|z|} \sum_{y\le z} (-1)^{|y'|} \mathcal{X}_{y'\cup y''}(x), \end{aligned} \tag{6.14}$$

where $y' \oplus y'' = y' \cup y''$ since $y' \cap y'' = \emptyset$. Note that we are working in $B_n^{(2)}$ and not in B_n^2; i.e., if $z \in B_n^{(2)}$, then summing over all $y \le z$ ($y \ge z$, respectively) means summing over all $y \in B_n^{(2)}$ such that $y \le z$ ($y \ge z$, respectively). Thus

$$\begin{aligned}(\text{Proj } f)(x) &= \sum_z \widetilde{f}(z) 2^{-|z|} \sum_{y \le z} (-1)^{|y'|} \mathcal{X}_{y' \cup y''}(x) \\ &= \sum_{w \in B_n} \mathcal{X}_w(x) \sum_{a \le w} (-1)^{|a|} \sum_{z \ge (a, w \setminus a)} 2^{-|z|} \widetilde{f}(z).\end{aligned}$$

That is, the desired analogue of the change of basis formula (6.4) on $B_n^{(2)}$ is

(6.15)
$$\widehat{(\text{Proj } f)}(w) = \sum_{a \le w} (-1)^{|a|} \sum_{z \ge (a, w \setminus a)} 2^{-|z|} \widetilde{f}(z) \qquad \text{for all } w \in B_n \text{ and } f \in L(B_n^{(2)}).$$

Finally, we define some basic notions on $B_n^{(2)}$ used in section 7.3.

- *Size.* If $x \in B_n^{(2)}$, define the *size* or *rank* of x to be $|x| \stackrel{\text{def}}{=} |x' \cup x''| = |x'| + |x''|$, and note that $0 \le |x| \le n$.
- *Consistent elements and union.* We call two elements $x, y \in B_n^{(2)}$ *consistent* if x' and y'' are disjoint and x'' and y' are disjoint. It is straightforward to verify that two elements x and y of $B_n^{(2)}$ are consistent if and only if they have an *upper bound* (i.e., an element z of $B_n^{(2)}$ such that $z \ge x$ and $z \ge y$), or equivalently, a *least upper bound* (i.e., an upper bound $\le$ all upper bounds of x and y). If $x, y \in B_n^{(2)}$ are consistent, their least upper bound, which we also call *union* and denote by $x \cup y$, is given by $x \cup y \stackrel{\text{def}}{=} (x' \cup y', x'' \cup y'') \in B_n^{(2)}$. Let $y, z \in B_n^{(2)}$. If y and z are consistent, then $\overline{AND}_y \overline{AND}_z = \overline{AND}_{y \cup z}$. If y and z are not consistent, then $\overline{AND}_y(x) \overline{AND}_z(x) = 0$ for all $x \in B_n^{(2)}$. The reason is that if $\overline{AND}_y(x) \overline{AND}_z(x) = 1$, then $y \le x$ and $z \le x$. Hence y and z have an upper bound; i.e., they are consistent.
- *Intersection and complement.* To avoid degenerate cases, we define the *intersection* $x \cap y$ and *complement* $x \setminus y$ only for consistent elements $x, y \in B_n^{(2)}$ as follows. Let $x \cap y \stackrel{\text{def}}{=} (x' \cap y', x'' \cap y'') \in B_n^{(2)}$ and $y \setminus x \stackrel{\text{def}}{=} (y' \setminus x', y'' \setminus x'') \in B_n^{(2)}$. Note that $x \cap y \le y$, $y \setminus x \le y$, and $(x \cap y) \cup (y \setminus x) = y$. Note also that if $x \le y$, then x and y are obviously consistent; hence $y \setminus x$ is defined.
- *Separated elements.* We call two elements $x, y \in B_n^{(2)}$ *separated* if x', x'', y', y'' are mutually disjoint. Separated elements are consistent. If x and y are separated, then $|y \cup x| = |y| + |x|$. If x and y are consistent, but not necessarily separated, then $y \setminus x$ and x are separated and $(y \setminus x) \cup x = y \cup x$; hence $|y \cup x| = |y \setminus x| + |x|$.

6.5. General DNF formulas and auxiliary functions. In this section we compute the Möbius and Fourier transforms of general DNF formulas and auxiliary functions.

We represent a DNF formula F on n variables by two bipartite graphs $(C, [n], N')$ and $(C, [n], N'')$ both between the set C of clauses and the set $[n]$ of variable indices. For each clause $c \in C$, $N'(c)$ is the neighborhood of c consisting of the indices of the nonnegated variables of c, and $N''(c)$ is the neighborhood of c consisting of the indices of the negated variables in c. We assume that $N'(c) \cap N''(c) = \emptyset$ for each clause $c \in C$. To avoid degenerate cases, we assume, as in the monotone case, that

we have at least one clause, i.e., $|C| \geq 1$, and that each clause contains at least one literal, i.e., $N'(c) \cup N''(c) \neq \emptyset$ for each $c \in C$. Let $N \stackrel{\text{def}}{=} (N', N'')$, i.e., $N(c) \stackrel{\text{def}}{=} (N'(c), N''(c)) \in B_n^{(2)}$ for each clause $c \in C$, and $N(S) \stackrel{\text{def}}{=} (N'(S), N''(S)) \in B_n^2$ for each set of clauses $S \subset C$. Note that $N(S)$ is not necessarily in $B_n^{(2)}$ since possibly $N'(S) \cap N''(S) \neq \emptyset$. We denote F by $F = (C, [n], N)$. Finally, if $s \geq 1$ is an integer, we call a DNF formula $F = (C, [n], N)$ an *s-DNF* formula if each clause contains at most s literals, i.e., $|N(c)| \leq s$ for each $c \in C$.

Let $F = (C, [n], N)$ be a DNF formula. We have the boolean function computed by F, which by notational abuse we denote by $F \in L(B_n)$. We also have the u-skin and cover functions $\text{cover}_F, \text{skin}_{F,u} \in L(B_n)$ associated with F (for $u \geq 0$). They are given by Definition 5.6 as $F \stackrel{\text{def}}{=} \bigvee_{c \in C} AND_{N(c)}$, $\text{skin}_{F,u} \stackrel{\text{def}}{=} 1 - (1 - e^{-u})^{\sum_{c \in C} AND_{N(c)}}$, $\text{cover}_F \stackrel{\text{def}}{=} \int_0^\infty \text{skin}_{F,u} e^{-u} du$, where $\text{skin}_{F,0} = F$ by taking the limit. By lifting each of the AND gates of the DNF from B_n to $B_n^{(2)}$, we get the following natural lifts of F as a boolean function, cover_F, and $\text{skin}_{F,u}$.

DEFINITION 6.4 (lifted DNF boolean function, skin, and cover). *Let $F = (C, [n], N)$ be a DNF formula and $u \geq 0$, and define $\overline{F}, \overline{cover}_F, \overline{skin}_{F,u} \in L(B_n^{(2)})$ as*

$$\begin{aligned}
\overline{F}(x) &\stackrel{\text{def}}{=} \bigvee_{c \in C} \overline{AND}_{N(c)}(x), \\
\overline{skin}_{F,u}(x) &\stackrel{\text{def}}{=} 1 - (1 - e^{-u})^{\sum_{c \in C} \overline{AND}_{N(c)}(x)}, \\
\overline{cover}_F(x) &\stackrel{\text{def}}{=} \int_0^\infty \overline{skin}_{F,u}(x) e^{-u} du
\end{aligned}$$

for all $x \in B_n^{(2)}$. Thus $F = Proj\ \overline{F}$, $cover_F = Proj\ \overline{cover}_F$, and $skin_{F,u} = Proj\ \overline{skin}_{F,u}$.

Those lifted functions have natural expansions in the $\{\overline{AND}_z\}_z$ basis from which we can extract their Möbius transforms. To get the Fourier transforms of the original functions, we use (6.15).

We have the following analogue of Lemmas 6.1 and 6.2.

LEMMA 6.5. *Let $F = (C, [n], N)$ be a DNF formula, and let $u \geq 0$. Then for all $z \in B_n^{(2)}$ and $w \in B_n$,*

$$\widetilde{\overline{F}}(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|}, \tag{6.16}$$

$$\widetilde{\overline{skin}}_{F,u}(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|} e^{-u|S|}, \tag{6.17}$$

$$\widetilde{\overline{cover}}_F(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|} \frac{1}{|S| + 1}, \tag{6.18}$$

$$\widehat{F}(w) = \sum_{a \leq w} (-1)^{|a|} \sum_{S \neq \emptyset \subset C \,:\, N'(S) \cap N''(S) = \emptyset \;\&\; N(S) \geq (a, w \setminus a)} -(-1)^{|S|} 2^{-|N(S)|}, \tag{6.19}$$

$$\widehat{\overline{skin}}_{F,u}(w) = \sum_{a \leq w} (-1)^{|a|} \sum_{S \neq \emptyset \subset C \,:\, N'(S) \cap N''(S) = \emptyset \;\&\; N(S) \geq (a, w \setminus a)} -(-1)^{|S|} 2^{-|N(S)|} e^{-u|S|}, \tag{6.20}$$

$$\widehat{\overline{cover}}_F(w) = \sum_{a \leq w} (-1)^{|a|} \sum_{S \neq \emptyset \subset C \,:\, N'(S) \cap N''(S) = \emptyset \;\&\; N(S) \geq (a, w \setminus a)} -(-1)^{|S|} 2^{-|N(S)|} \frac{1}{|S| + 1}. \tag{6.21}$$

Proof. For $u \geq 0$, we have

$$1 - \prod_{c\in C}(1 - \overline{AND}_{N(c)}(x)e^{-u}) = \sum_{S\neq\emptyset\subset C} -(-e^{-u})^{|S|} \prod_{c\in S} \overline{AND}_{N(c)}(x)$$
$$= \sum_{S\neq\emptyset\subset C} -(-e^{-u})^{|S|} AND_{N'(S)}(x')AND_{N''(S)}(x'').$$

If $AND_{N'(S)}(x')AND_{N''(S)}(x'') = 1$, then $N'(S) \leq x'$ and $N''(S) \leq x''$, and hence $N'(S)$ and $N''(S)$ must be disjoint since x' and x'' are disjoint; i.e., we can exclude from the sum the sets S such that $N'(S) \cap N''(S) \neq \emptyset$. Thus

$$1 - \prod_{c\in C}(1 - \overline{AND}_{N(c)}(x)e^{-u}) = \sum_{S\neq\emptyset\subset C: N'(S)\cap N''(S)=\emptyset} -(-1)^{|S|}e^{-u|S|}\overline{AND}_{N(S)}(x)$$
$$= \sum_{z\in B_n^{(2)}} \left(\sum_{S\neq\emptyset\subset C: N(S)=z} -(-1)^{|S|}e^{-u|S|} \right) \overline{AND}_z(x).$$

Setting $u = 0$, we get (6.16). To establish (6.17), it is enough to note that, for $u > 0$, we have

$$1 - (1 - e^{-u})^{\sum_{c\in C}\overline{AND}_{N(c)}(x)} = 1 - \prod_{c\in C}(1 - e^{-u})^{\overline{AND}_{N(c)}(x)}$$
$$= 1 - \prod_{c\in C}(1 - \overline{AND}_{N(c)}(x)e^{-u}).$$

This verifies (6.17) for $u > 0$. The $u = 0$ case follows by taking the limit.

Applying the linear operator $\mu_{B_n^{(2)}}$ to $\overline{\text{cover}}_F = \int_0^\infty \overline{\text{skin}}_{F,u}e^{-u}du$, we get

$$\widetilde{\overline{\text{cover}}}_F(z) = \int_0^\infty \widetilde{\overline{\text{skin}}}_{F,u}(z)e^{-u}du$$
$$= \sum_{S\neq\emptyset\subset C: N(S)=z} -(-1)^{|S|} \int_0^\infty e^{-u(|S|+1)}du$$
$$= \sum_{S\neq\emptyset\subset C: N(S)=z} -(-1)^{|S|}\frac{1}{|S|+1},$$

which establishes (6.18).

The correctness of (6.19) follows from (6.16) via (6.15):

$$\widehat{F}(w) = \sum_{a\leq w}(-1)^{|a|} \sum_{z\geq(a,w\setminus a)} 2^{-|z|} \sum_{S\neq\emptyset\subset C: N(S)=z} -(-1)^{|S|}$$
$$= \sum_{a\leq w}(-1)^{|a|} \sum_{S\neq\emptyset\subset C: N'(S)\cap N''(S)=\emptyset \ \ \&N(S)\geq(a,w\setminus a)} -(-1)^{|S|}2^{-|N(S)|}.$$

Similarly, (6.20) and (6.21) follow from (6.17) and (6.18) via (6.15). □

6.6. Miscellaneous remarks. This section can be skipped without loss of continuity. In poset terminology,[5] $B_n^{(2)}$ can be alternatively defined as the order ideal of B_n^2 generated by the antichain $A_n \stackrel{\text{def}}{=} \{(x, x^c) : x \in B_n\} \subset B_n^2$.

[5] An *antichain* of a poset X is a subset A of X such that any two distinct elements of A are incomparable. A subset I of X is called an *order ideal* if $x \in I$ and $y \leq x$; then $y \in I$. We say that an order ideal is *generated by* a subset a A of X if $I = \{x \in X : x \leq y \text{ for some } y \in A\}$. Any order ideal has a generating antichain.

If X is a poset, the matrix coefficients of μ_X are denoted by $(\mu_X(y,x))_{x,y}$; i.e.,

$$(\mu_X f)(x) = \sum_{y \in X} \mu_X(y,x) f(y). \tag{6.22}$$

Since ζ_X is lower triangular, μ_X is also lower triangular; i.e., $\mu_X(y,x) = 0$ if $y \not\leq x$.

It is worth mentioning that the coefficients of the Möbius functions of the posets B_n and $B_n^{(2)}$ have the following simple expressions:

(i) $\mu_{B_n}(y,x) = (-1)^{|x \setminus y|}$ if $y \leq x \in B_n$ (see [24]).

(ii) $\mu_{B_n^{(2)}}(y,x) = (-1)^{|x \setminus y|}$ if $y \leq x \in B_n^{(2)}$. This can be derived from (i) via the fact that $B_n^{(2)}$ is an order ideal of $B_n^2 = B_n \times B_n$. (In general, if I is an order ideal of a poset X regarded as a subposet of X, then $\mu_I(y,x) = \mu_X(y,x)$ for all $x, y \in I$.)

Those expressions are not used in the proof since, instead of using (6.22) to compute the Möbius transforms of DNF formulas and their auxiliary functions, we extracted the Möbius transforms from natural expansions of the functions in the $\{AND_z\}_z$ basis.

7. From zero-energy to energies of auxiliary functions. In this section, we reduce the problem of bounding the t-zero-energy of an s-DNF formula F to that of bounding the $(t-s)$-energies of auxiliary functions derived from F.

Let $F = (C, [n], N)$ be an s-DNF formula, and let $t \geq s$ be an integer. We want to bound the t-zero-energy of F. That is, we want to construct $f \in L(B_n)$ of degree at most t such that the mean square error $E(F-f)^2$ is small and f satisfies the zeros-constraint: $f = 0$ whenever $F = 0$ (i.e., $f(x) = 0$ for each $x \in \{0,1\}^n$ such that $F(x) = 0$). For any such f, we have the bound $\text{zeroEnergy}(F;t) \leq E(F-f)^2 = \|\widehat{F-f}\|_2^2$.

We presented the construction of f in the monotone case in section 5.6 without analyzing its mean square error. In sections 7.1 and 7.2, we analyze the Fourier transform of the construction error term $\widehat{F-f}$ in the monotone case. Then we generalize to arbitrary DNF formulas in section 7.3. The analysis of $\widehat{F-f}$ naturally leads us to the cover and u-skin functions. In Lemmas 7.1 and 7.2, we express $\widehat{F-f}$ in terms of the Fourier transforms of cover functions of DNF derived from F. The analysis in sections 7.1, 7.2, and 7.3 uses the machinery developed in section 6.

In section 7.4, we apply two simple bounds to the obtained expression of $\widehat{F-f}$. The first bound reduces the problem of bounding the t-zero-energy of F to that of bounding the $(t-s)$-energies of the cover functions of the derived DNF formulas. The second bound reduces the latter problem to bounding the $(t-s)$-energies of the u-skin functions of the derived formulas, for all $u \geq 0$.

7.1. Monotone case error analysis. This section uses the monotone machinery developed in sections 6.2 and 6.3.

In this section we analyze the mean square error of the monotone construction defined in section 5.6. Assume that $F = (C, [n], N)$ is monotone and construct f as in section 5.6. That is, define $f \in L(B_n)$ as

$$f \stackrel{\text{def}}{=} \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} E_{c \in S} AND_{N(c)} \text{Trn}_{t-|N(c)|} AND_{N(S) \setminus N(c)}. \tag{7.1}$$

We know from section 5.6 that $deg(f) \leq t$ and f satisfies the zeros-constraint.

The difficult part is estimating the mean square error $E(F-f)^2 = \|\widehat{F-f}\|_2^2$ as t grows. Toward this end, we start analyzing $\widehat{F-f}$ by first computing the Fourier transform of f, which gives some intuition as to why one would speculate that the mean square error of this construction decays as t grows. Then we interpret $\widehat{F-f}$ in Lemma 7.1 in terms of the Fourier transforms of cover functions of DNF formulas derived from F.

Recall from (6.6) that

$$\widehat{F}(y) = (-1)^{|y|} \sum_{S\neq\emptyset\subset C:N(S)\geq y} -(-1)^{|S|}2^{-|N(S)|}. \tag{7.2}$$

We show below that the Fourier transform of f is given by

$$\widehat{f}(y) = (-1)^{|y|} \sum_{S\neq\emptyset\subset C:N(S)\geq y} -(-1)^{|S|}2^{-|N(S)|} Pr_{c\in S}[|y\cup N(c)|\leq t]. \tag{7.3}$$

Proof of (7.3). Recall from (6.3) that

$$AND_z = \frac{1}{2^{|z|}}\sum_{y\leq z}(-1)^{|y|}\mathcal{X}_y.$$

Let $S\neq\emptyset\subset C$, and let $c\in S$. Then

$$AND_{N(c)} = \frac{1}{2^{|N(c)|}}\sum_{y_1\leq N(c)}(-1)^{|y_1|}\mathcal{X}_{y_1},$$

and

$$\mathrm{Trn}_{t-|N(c)|}AND_{N(S)\setminus N(c)} = \frac{1}{2^{|N(S)\setminus N(c)|}}\sum_{y_2\leq N(S)\setminus N(c):|y_2|\leq t-|N(c)|}(-1)^{|y_2|}\mathcal{X}_{y_2}.$$

Thus

$$\begin{aligned} AND_{N(c)}\mathrm{Trn}_{t-|N(c)|}AND_{N(S)\setminus N(c)} &= \frac{1}{2^{|N(S)|}}\sum_{\substack{y_1\leq N(c),\\ y_2\leq N(S)\setminus N(c):\\ |y_2|\leq t-|N(c)|}}(-1)^{|y_1|+|y_2|}\mathcal{X}_{y_1\oplus y_2} \\ &= \frac{1}{2^{|N(S)|}}\sum_{y\leq N(S):|y\cup N(c)|\leq t}(-1)^{|y|}\mathcal{X}_y, \end{aligned} \tag{7.4}$$

where we used the fact that $y_1\oplus y_2 = y_1\cup y_2$ since y_1 and y_2 are disjoint because $N(c)$ and $N(S)\setminus N(c)$ are disjoint.

Substituting (7.4) in (7.1) and writing the expectation operator in (7.1) as $E_{c\in S} = \frac{1}{|S|}\sum_{c\in S}$, we get

$$\begin{aligned} f &= \sum_{S\subset C:S\neq\emptyset} -(-1)^{|S|}\frac{1}{|S|}\sum_{c\in S}2^{-|N(S)|}\sum_{y\leq N(S):|y\cup N(c)|\leq t}(-1)^{|y|}\mathcal{X}_y \\ &= \sum_y \mathcal{X}_y(-1)^{|y|}\sum_{S\neq\emptyset\subset C:N(S)\geq y} -(-1)^{|S|}2^{-|N(S)|}\ \frac{1}{|S|}\sum_{c\in S:|y\cup N(c)|\leq t}1 \end{aligned}$$

after rearranging the summations. Noting that $\frac{1}{|S|}\sum_{c\in S:|y\cup N(c)|\le t} 1 = Pr_{c\in S}[|y \cup N(c)| \le t]$, we obtain (7.3). ∎

Comparing the expressions of $\widehat{F}$ and $\widehat{f}$ in (7.2) and (7.3), and noting that $|N(c)| \le s \le t$ for each $c \in C$ since F is an s-DNF formula, we get

$$\widehat{f}(y) = \begin{cases} \widehat{F}(y) & \text{if } |y| \le t-s, \\ 0 & \text{if } |y| > t. \end{cases}$$

This relation gives some intuition why the mean square error $\|\widehat{F-f}\|_2^2$ of this construction decays as t grows. It says that $\widehat{f}(y) = \widehat{\mathrm{Trn}_t F}(y)$ if $|y| \le t-s$ or $|y| > t$. We know from the LMN energy bound (Theorem 5.8) that $E(F-\mathrm{Trn}_t F)^2 = \|\widehat{F-\mathrm{Trn}_t F}\|_2^2$ decays quickly as t grows. Thus, we can hope that $\widehat{f}(y)$ is not too bad in the region $t-s < |y| \le t$ and accordingly speculate that the mean square error $\|\widehat{F-f}\|_2^2$ also decays as t grows.

Unfortunately, we could not turn this intuition into a bound on $\|\widehat{F-f}\|_2^2$ since the frequencies in the region $t-s < |y| \le t$ are too many to handle separately by trivial bounds. We can think of other equally intuitive constructions, which we briefly mention in section 7.2.4. The reason behind favoring this choice of f is analytical. Using analytical means, we managed to bound $\|\widehat{F-f}\|_2^2$ by interpreting the Fourier transform of the error term $f-F$ in terms of the Fourier transforms of cover functions of DNF formulas derived from F as shown in Lemma 7.1. We consider $f-F$ instead of $F-f$ for technical convenience.

LEMMA 7.1 (error interpretation). *Let $F = (C, [n], N)$ be a monotone s-DNF formula, and let $t \ge s$ be an integer. Assume that $|C| \ge 2$. Let $f \in L(B_n)$ be given by*

$$f = \sum_{S\subset C: S\neq\emptyset} -(-1)^{|S|} E_{c\in S} AND_{N(c)}\, Trn_{t-|N(c)|} AND_{N(S)\setminus N(c)}. \tag{7.5}$$

For each clause $c \in C$, define the new DNF formula $F_c = (C_c, [n], N_c)$ resulting from removing the clause c from F and adding its variables to all the other clauses; i.e., $C_c = C\setminus\{c\}$ and $N_c(d) = N(d) \cup N(c)$ for each $d \in C_c$.

Then (i) *$deg(f) \le t$;* (ii) *f satisfies the zeros-constraint $f = 0$ whenever $F = 0$; and* (iii) *for each $y \in B_n$,*

$$\widehat{(f-F)}(y) = \sum_{c\in C:|y\cup N(c)|>t} \widehat{cover}_{F_c}(y). \tag{7.6}$$

We show in section 7.4 that (7.6) immediately leads to a bound on $\|\widehat{f-F}\|_2^2$ in terms of the $(t-s)$-energies of the covers of the derived DNF formulas.

It is important to note that the following error analysis is the origin of the cover and skin functions. We highlight in section 7.2.1 how the error analysis in the proof below naturally leads to the definitions of skin and cover.

Proof of Lemma 7.1. We have (i) and (ii) from section 5.6. We need to establish (7.6). Let

$$\Delta \stackrel{\text{def}}{=} f - F.$$

Subtracting the summation (7.2) from (7.3), we get

$$\widehat{\Delta}(y) = (-1)^{|y|} \sum_{S\neq\emptyset\subset C: N(S)\ge y} (-1)^{|S|} 2^{-|N(S)|} \frac{1}{|S|} \sum_{c\in S:|y\cup N(c)|>t} 1 \tag{7.7}$$

for all $y \in B_n$.

First, for technical convenience, we note that the summation $\sum_{S\subset C:S\neq\emptyset}$ in (7.7) can be replaced by $\sum_{S\subset C:|S|>1}$. The condition $|S|>1$ is nonrestrictive since the size-1 subsets S of C do not contribute to the summation. Indeed, assume that $|S|=1$; hence $S=\{c_0\}$ for some $c_0 \in C$. Thus the expression in (7.7) is nonzero only if $N(c_0) \geq y$ and $|y \cup N(c_0)| > t$. But then we get $|N(c_0)| > t$ since $y \cup N(c_0) = N(c_0)$ because $N(c_0) \geq y$. This is not possible since $|N(c_0)| \leq s$ because F is an s-DNF and $s \leq t$ by the lemma hypothesis.

If we replace $\sum_{S\subset C:S\neq\emptyset}$ by $\sum_{S\subset C:|S|>1}$ and reverse the order of the summations in (7.7), we get

$$\begin{aligned}\widehat{\Delta}(y) &= \sum_{\substack{c\in C:\\ |y\cup N(c)|>t}} (-1)^{|y|} \sum_{\substack{S\subset C:|S|>1\\ N(S)\geq y\\ c\in S}} (-1)^{|S|} 2^{-|N(S)|} \frac{1}{|S|} \\ &= \sum_{\substack{c\in C:\\ |y\cup N(c)|>t}} (-1)^{|y|} \sum_{z\geq y} \left(\sum_{\substack{S\subset C:|S|>1\\ N(S)=z \ \&\ c\in S}} (-1)^{|S|} \frac{1}{|S|} \right) 2^{-|z|}.\end{aligned}$$

Using the change of basis formula (6.4), we obtain

$$\widehat{\Delta}(y) = \sum_{c\in C:|y\cup N(c)|>t} \widehat{X}_c(y),$$

where $X_c \in L(B_n)$ is a function given by its Möbius transform

$$\widetilde{X}_c(z) = \sum_{\substack{S\subset C:|S|>1\\ N(S)=z\\ c\in S}} (-1)^{|S|} \frac{1}{|S|}.$$

By a change of variables from S to $T = S\backslash\{c\}$, we can write this as

$$\widetilde{X}_c(z) = \sum_{\substack{T\neq\emptyset\subset C\backslash\{c\}:\\ N(T\cup\{c\})=z}} (-1)^{|T|+1} \frac{1}{|T|+1}.$$

By the definition of the formula F_c, we have $C_c = C\backslash\{c\}$ and $N_c(d) = N(d) \cup N(c)$ for each $d \in C_c$. Hence for each $T \subset C_c$, $N_c(T) = \cup_{d\in T} N_c(d) = \cup_{d\in T\cup\{c\}} N(d) = N(T \cup \{c\})$. Thus

$$\widetilde{X}_c(z) = \sum_{T\neq\emptyset\subset C_c:N_c(T)=z} -(-1)^{|T|} \frac{1}{|T|+1}. \tag{7.8}$$

Using (6.8), we identify this as the Möbius transform of the cover of F_c, i.e., $X_c = \text{cover}_{F_c}$, which proves (7.6). □

7.2. Discussion. In this section, we make some remarks related to the above construction.

7.2.1. Origin of the cover and skin functions. Equation (7.6) is the reason behind our interest in the cover and skin functions. We explain below how the analysis of $\widehat{f-F}$ led us to the cover and skin functions. As shown in the proof of Lemma 7.1, $\widehat{f-F}$ can be expressed as

$$(\widehat{f-F})(y) = \sum_{c\in C:|y\cup N(c)|>t} \widehat{X}_c(y)$$

for some function $X_c \in L(B_n)$ whose Möbius transform is given by (7.8). By expressing $\frac{1}{|T|+1}$ as $\frac{1}{|T|+1} = \int_0^\infty e^{-u|T|}e^{-u}du$ we concluded that $X_c = \int_0^\infty Y_{c,u}e^{-u}du$, for some family of functions $Y_{c,u} \in L(B_n)$ whose Möbius transforms are given by

$$\widetilde{Y}_{c,u}(z) = \sum_{T\neq\emptyset\subset C_c:N_c(T)=z} -(-1)^{|T|}e^{-u|T|}.$$

The u-skin function was identified from its Möbius transform via (6.11) with respect to F_c:

$$\sum_{z\in B_n}\left(\sum_{T\neq\emptyset\subset C_c:N_c(T)=z} -(-1)^{|T|}e^{-u|T|}\right) AND_z(x) = 1-(1-e^{-u})^{\sum_{d\in C_c} AND_{N_c(d)}(x)}.$$

That is, we first did the computations in Lemma 6.2 backward on F_c to conclude that $Y_{c,u} = \mathrm{skin}_{F_c,u}$, and hence $X_c = \mathrm{cover}_{F_c}$ by evaluating the integral $\int_0^\infty \mathrm{skin}_{F_c,u}\, e^{-u}du$.

The right way to understand (7.6) is in the Fourier domain. It does not have a simple analogue outside this domain due to the condition $|y\cup N(c)| > t$ on y in the summation.

We do not have an intuitive nonanalytical explanation of (7.6). Recall that we constructed f so that it satisfies the zeros-constraint and the low degree condition. As explained in the discussion preceding the theorem statement, the construction intuition is "hopefully $\widehat{f}(y)$ is not too bad in the region $t-s<|y|\le t$." The same intuition applies to other similar constructions of f which we failed to analyze (see below). The issue is that we could not turn this intuition into an argument. We managed instead in (7.6) to interpret the construction error term using analytical means which led us to the cover and skin functions.

7.2.2. Identifying the components of (7.6). From a big perspective, the components of (7.6) can be identified with the definition of f in (7.5) as follows. Write the expectation operator in (7.5) as $E_{c\in S} = \frac{1}{|S|}\sum_{c\in S}$. The summation on c in $E_{c\in S}$ corresponds to the summation on c in (7.6). The latter summation is subject to the condition $|y\cap N(c)| > t$ which comes from the truncation operator in (7.5). The $\frac{1}{|S|}$ term of the expectation operator $E_{c\in S}$ corresponds to the $\frac{1}{|T|+1}$ weighting factor in (7.8) via the change of variables done in the proof of Lemma 7.1 from S to $T = S\backslash\{c\}$. Note that this $\frac{1}{|T|+1}$ weighting factor is what distinguishes the cover of F_c from F_c in the Möbius and Fourier domains (see item 2 of Remark 6.3).

7.2.3. Duplicate clause issue. It is possible that there exist $c, d_1, d_2 \in C$ such that $N(d_1) \neq N(d_2)$, but $N(c)\cup N(d_1) = N(c)\cup N(d_2)$; i.e., $N_c(d_1) = N_c(d_2)$. Thus it is possible that the DNF formula F_c has duplicate clauses even if F does not. Duplicate clauses in F_c affect the function cover_{F_c}. This is the reason why we chose to represent a DNF formula by a bipartite graph and not a set of clauses, i.e., a subset of B_n (or $B_n^{(2)}$ in the general case).

7.2.4. Alternative constructions. In what follows, we briefly mention two intuitive alternative constructions of f which we failed to analyze. Instead of starting from (5.1), it is natural to try grouping terms first; i.e., express

$$F(x) = \sum_{z \in B_n} \widetilde{F}(z) AND_z(x),$$

where the Möbius transform of F is given in (6.5) by $\widetilde{F}(z) = \sum_{S \neq \emptyset \subset C : N(S) = z} -(-1)^{|S|}$. Starting from this expression, we can define

$$f' = \sum_z \widetilde{F}(z) E_{c \in N^{-1}(z)} AND_{N(c)} \mathrm{Trn}_{t-|N(c)|} AND_{z \setminus N(c)},$$

where $N^{-1}(z) = \{c \in C : c \leq z\}$. Here again the degree of f' is at most t, f' satisfies the zeros-constraint, and the same intuition behind the above construction of f applies to f'. We can also express $\widehat{(f' - F)}(y)$ similarly to (7.6) as a sum over $c \in C$ of the Fourier coefficients of some functions. The issue, however, is that, unlike the covers of the derived formulas, those functions are hard to analyze and they have no clear interpretation.

The second construction is the following. Rather than averaging over all the $c \in S$, we could have fixed an arbitrary map $\alpha : 2^C \to C$ which attaches to each $S \subset C$ a fixed element $c_S \stackrel{\text{def}}{=} \alpha(S) \in S$ (e.g., the smallest clause in S). Then we can define

$$f_\alpha = \sum_{S \subset C : S \neq \emptyset} -(-1)^{|S|} AND_{N(c_S)} \mathrm{Trn}_{t-|N(c_S)|} AND_{N(S) \setminus N(c_S)}.$$

Here again, the degree of f_α is at most t, f_α satisfies the zeros-constraint, and the same intuition behind the above construction of f applies to f_α. The issue is again in the difficulty of interpreting and analyzing the functions in the resulting analogue of (7.6). We can view f, however, as the average of f_α over all the maps α, i.e., $f = E_\alpha f_\alpha$.

7.3. General case construction. This section uses the general machinery in sections 6.4 and 6.5.

If F is not necessarily monotone, the monotone case construction generalizes naturally as follows. Following the derivations in the proof of Lemma 6.5 (for $u = 0$), expand F as

$$\begin{aligned}
F(x) &= 1 - \prod_{c \in C} (1 - AND_{N(c)}(x)) \\
&= \sum_{S \neq \emptyset \subset C} -(-1)^{|S|} \prod_{c \in S} AND_{N(c)}(x) \\
&= \sum_{S \neq \emptyset \subset C} -(-1)^{|S|} AND_{N'(S)}(x') AND_{N''(S)}(x'') \\
&= \sum_{S \neq \emptyset \subset C : N'(S) \cap N''(S) = \emptyset} -(-1)^{|S|} AND_{N'(S)}(x') AND_{N''(S)}(x'') \\
&= \sum_{S \neq \emptyset \subset C : N'(S) \cap N''(S) = \emptyset} -(-1)^{|S|} AND_{N(S)}(x).
\end{aligned}$$

Let $S \subset C$ such that $S \neq \emptyset$ and $N'(S) \cap N''(S) = \emptyset$; i.e., $N(S) \in B_n^{(2)} \backslash \{(\emptyset, \emptyset)\}$. For each $c \in S$, we have

$$AND_{N(S)} = AND_{N(c)} AND_{N(S)\backslash N(c)}.$$

Recall from section 6.4 that we defined $y \backslash x \in B_n^{(2)}$ for consistent elements $x, y \in B_n^{(2)}$, and note that $N(c)$ and $N(S)$ are consistent since $N(c) \leq N(S)$. Averaging over all $c \in S$, we trivially get

$$AND_{N(S)} = E_{c \in S} AND_{N(c)} AND_{N(S)\backslash N(c)};$$

hence

$$F = \sum_{S \neq \emptyset \subset C : N'(S) \cap N''(S) = \emptyset} -(-1)^{|S|} E_{c \in S} AND_{N(c)} AND_{N(S)\backslash N(c)}.$$

We construct f as in the monotone case by truncating each $AND_{N(S)\backslash N(c)}$ to $\mathrm{Trn}_{t-|N(c)|} AND_{N(S)\backslash N(c)}$.

LEMMA 7.2 (error interpretation). *Let $F = (C, [n], N)$ be an s-DNF formula, and let $t \geq s$ be an integer. Let $f \in L(B_n)$ be given by*

$$f = \sum_{S \subset C : S \neq \emptyset \ \& N'(S) \cap N''(S) = \emptyset} -(-1)^{|S|} E_{c \in S} AND_{N(c)} \, Trn_{t-|N(c)|} AND_{N(S)\backslash N(c)}.$$

If $c \in C$, let C_c be the set of clauses other than c which are consistent with c, i.e., $C_c = \{d \in C \backslash \{c\} : N(d)$ and $N(c)$ are consistent$\}$. Let C_{main} be the set of clauses which are consistent with at least one clause of F other than themselves, i.e., $C_{main} = \{c \in C : C_c \neq \emptyset\}$.

For each clause $c \in C_{main}$, define the new DNF formula $F_c = (C_c, [n], N_c)$, where $N_c(d) = N(d) \cup N(c)$ for each $d \in C_c$. That is, F_c is the formula resulting from removing from F the clause c and all the clauses not consistent with c, and adding the literals of c to each of the remaining clauses.

Then (i) *$deg(f) \leq t$;* (ii) *f satisfies the zeros-constraint $f = 0$ whenever $F = 0$; and* (iii) *for each $w \in B_n$*

$$\widehat{(f - F)}(w) = \sum_{c \in C_{main} : |w \cup N'(c) \cup N''(c)| > t} \widehat{cover}_{F_c}(w). \tag{7.9}$$

Proof. We have $deg(f) \leq t$ since the degree of each $\mathrm{Trn}_{t-|N(c)|} AND_{N(S)\backslash N(c)}$ is at most $t - |N(c)|$ and the degree of $AND_{N(c)}$ is $|N(c)|$. Moreover, if $F(x) = 0$, then $AND_{N(c)}(x) = 0$ for each $c \in C$; thus $f(x) = 0$, and hence (ii). We have to establish (7.9). Let

$$\Delta \stackrel{\mathrm{def}}{=} f - F.$$

We have

$$\begin{aligned} \Delta &= \sum_{S \neq \emptyset \subset C : N'(S) \cap N''(S) = \emptyset} (-1)^{|S|} E_{c \in S} AND_{N(c)} (1 - \mathrm{Trn}_{t-|N(c)|}) AND_{N(S)\backslash N(c)} \\ &= \sum_{S \subset C : |S| > 1 \ \& \ N'(S) \cap N''(S) = \emptyset} (-1)^{|S|} E_{c \in S} AND_{N(c)} (1 - \mathrm{Trn}_{t-|N(c)|}) AND_{N(S)\backslash N(c)}. \end{aligned} \tag{7.10}$$

As in the monotone case, we impose the nonrestrictive condition $|S| > 1$ for technical convenience. This is nonrestrictive since if $|S| = 1$, then $S = \{c_0\}$ for some $c_0 \in C$; hence $AND_{N(S)\backslash N(c_0)} = 1$. Therefore $(1 - \mathrm{Trn}_{t-|N(c_0)|})AND_{N(S)\backslash N(c_0)} = 0$ since $t - |N(c_0)| \geq 0$ because $|N(c_0)| \leq s$ as F is an s-DNF and $t \geq s$ by the lemma hypothesis.

Recall from (6.14) that

$$AND_z = \sum_{y \leq z} (-1)^{|y'|} \mathcal{X}_{y' \cup y''},$$

and recall also that we are working in $B_n^{(2)}$ and not in B_n^2; i.e., if $z \in B_n^{(2)}$, then summing over all $y \leq z$ ($y \geq z$, respectively) means summing over all $y \in B_n^{(2)}$ such that $y \leq z$ ($y \geq z$, respectively).

Let $S \neq \emptyset \subset C$ such that $N'(S) \cap N''(S) = \emptyset$, i.e., such that $N(S) \in B_n^{(2)}$, and let $c \in S$. Then

$$AND_{N(c)} = \frac{1}{2^{|N(c)|}} \sum_{y_1 \leq N(c)} (-1)^{|y_1'|} \mathcal{X}_{y_1' \cup y_1''}$$

and

$$(1 - \mathrm{Trn}_{t-|N(c)|})AND_{N(S)\backslash N(c)} = \frac{1}{2^{|N(S)\backslash N(c)|}} \sum_{y_2 \leq N(S)\backslash N(c) : |y_2| > t - |N(c)|} (-1)^{|y_2'|} \mathcal{X}_{y_2' \cup y_2''}.$$

Thus

$$\begin{aligned} &AND_{N(c)}(1 - \mathrm{Trn}_{t-|N(c)|})AND_{N(S)\backslash N(c)} \\ &= \frac{1}{2^{|N(S)|}} \sum_{\substack{y_1 \leq N(c), \\ y_2 \leq N(S)\backslash N(c): \\ |y_2| > t - |N(c)|}} (-1)^{|y_1'| + |y_2'|} \mathcal{X}_{y_1' \cup y_1'' \oplus y_2' \cup y_2''} \\ &= \frac{1}{2^{|N(S)|}} \sum_{y \leq N(S) : |y \backslash N(c)| > t - |N(c)|} (-1)^{|y'|} \mathcal{X}_{y' \cup y''}. \end{aligned} \tag{7.11}$$

To verify (7.11), recall first from section 6.4 the definitions of basic operations in $B_n^{(2)}$. Equation (7.11) follows from the fact that $N(c)$ and $N(S)\backslash N(c)$ are separated and hence y_1 and y_2 are separated. Thus summing over $y_1 \leq N(c)$ and $y_2 \leq N(S)\backslash N(c)$ is equivalent to summing over $y = y_1 \cup y_2 \leq N(c) \cup (N(S)\backslash N(c)) = N(S)$. Since y_1 and y_2 are separated, i.e., y_1', y_1'', y_2', y_2'' are mutually disjoint, we have $y_1' \cup y_1'' \oplus y_2' \cup y_2'' = y_1' \cup y_1'' \cup y_2' \cup y_2'' = y' \cup y''$ and $|y_1'| + |y_2'| = |y_1' \cup y_2'| = |y'|$.

Now, note that

$$|y\backslash N(c)| + |N(c)| = |y \cup N(c)| = |(y \cup N(c))' \cup (y \cup N(c))''| = |y' \cup N'(c) \cup y'' \cup N''(c)|. \tag{7.12}$$

The first equality holds because, in general, if x and y are consistent, then $|y \cup x| = |y\backslash x| + |x|$ as noted at the end of section 6.4 (y and $N(c)$ are consistent as they have the upper bound $N(S)$ in $B_n^{(2)}$). The second (third, respectively) equality follows from the definition of size (union, respectively) in $B_n^{(2)}$.

Substituting (7.12) in (7.11), and then (7.11) in (7.10), we get

$$\Delta = \sum_{\substack{S\subset C:|S|>1\\ N'(S)\cap N''(S)=\emptyset}} (-1)^{|S|}\frac{1}{|S|}\sum_{c\in S}\frac{1}{2^{|N(S)|}} \sum_{\substack{y\le N(S):\\ |(y'\cup y'')\cup N'(c)\cup N''(c)|>t}} (-1)^{|y'|}\mathcal{X}_{y'\cup y''}.$$

By rearranging the summations, we extract the Fourier coefficients of Δ: $\Delta = \sum_{w\in B_n}\widehat{\Delta}(w)\mathcal{X}_w$, where

$$\begin{aligned}\widehat{\Delta}(w) &= \sum_{\substack{S\subset C:|S|>1\\ N'(S)\cap N''(S)=\emptyset}} (-1)^{|S|}\frac{1}{|S|} \sum_{\substack{c\in S:\\ |w\cup N'(c)\cup N''(c)|>t}} \frac{1}{2^{|N(S)|}} \sum_{\substack{a\le w:\\ (a,w\setminus a)\le N(S)}} (-1)^{|a|}\\ &= \sum_{\substack{c\in C:\\ |w\cup N'(c)\cup N''(c)|>t}} \sum_{a\le w}(-1)^{|a|} \sum_{\substack{S\subset C:|S|>1\\ N'(S)\cap N''(S)=\emptyset\\ N(S)\ge(a,w\setminus a)\\ c\in S}} (-1)^{|S|}2^{-|N(S)|}\frac{1}{|S|}\\ &= \sum_{\substack{c\in C:\\ |w\cup N'(c)\cup N''(c)|>t}} \sum_{a\le w}(-1)^{|a|} \sum_{z\ge(a,w\setminus a)} \left(\sum_{\substack{S\subset C:|S|>1\\ N(S)=z\ \&\ c\in S}} (-1)^{|S|}\frac{1}{|S|}\right) 2^{-|z|}.\end{aligned}$$

Using (6.15), we obtain

$$\widehat{\Delta}(w) = \sum_{\substack{c\in C:\\ |w\cup N'(c)\cup N''(c)|>t}} \widehat{\mathrm{Proj}\ X_c}(w), \tag{7.13}$$

where $X_c \in L(B_n^{(2)})$ is given by its Möbius transform

$$\begin{aligned}\widetilde{X}_c(z) &= \sum_{\substack{S\subset C:|S|>1\\ N(S)=z\ \&\ c\in S}} (-1)^{|S|}\frac{1}{|S|}\\ &= \sum_{T\ne\emptyset\subset C\setminus\{c\}:N(T\cup\{c\})=z} (-1)^{|T|+1}\frac{1}{|T|+1}\end{aligned}$$

after a change of variables from S to $T = S\setminus\{c\}$.

We will show that $X_c = \overline{\mathrm{cover}}_{F_c}$ if $c \in C_{main}$, and $X_c = 0$ if $c \in C\setminus C_{main}$.

First we handle the degenerate case when $c \in C\setminus C_{main}$. Assume $c \in C\setminus C_{main}$; thus there is no $d \ne c \in C$ such that $N(d)$ and $N(c)$ are consistent. Hence $T = \emptyset$ is the only $T \subset C\setminus\{c\}$ such that $N(T\cup\{c\}) \in B_n^{(2)}$. But $T = \emptyset$ is not allowed in the summation. It follows that $X_c = 0$.

Now, assume that $c \in C_{main}$. By the definition of the formula F_c, $C_c = \{d \in C\setminus\{c\} : N(d)$ and $N(c)$ are consistent$\} \ne \emptyset$, and $N_c(d) = N(d)\cup N(c)$ for each $d \in C_c$. Hence for each $T \subset C_c$,

$$N_c(T) = (N'_c(T), N''_c(T)) = (N'(T\cup\{c\}), N''(T\cup\{c\})) = N(T\cup\{c\}).$$

Moreover, if $T \subset C\setminus\{c\}$, but $T \not\subset C_c$, then T contains an element d of C such that $N(c)$ and $N(d)$ are not consistent. Consequently, $N'(\{c,d\}) \cap N''(\{c,d\}) \ne \emptyset$,

and hence $N'(T \cup \{c\}) \cap N''(T \cup \{c\}) \neq \emptyset$; i.e., $N(T \cup \{c\}) \notin B_n^{(2)}$. Therefore T does not contribute to the summation. It follows that

$$\widetilde{X}_c(z) = \sum_{T \neq \emptyset \subset C_c : N_c(S) = z} -(-1)^{|T|} \frac{1}{|T|+1}.$$

Using (6.18), we identify this as the Möbius transform of the lifted cover of F_c, i.e., $X_c = \overline{\text{cover}}_{F_c}$. Thus Proj $X_c = \text{cover}_{F_c}$, and hence (7.9) follows from (7.13). ∎

7.4. Bounds. The analysis in sections 7.1 and 7.3 consists of exact derivations not because we are interested in exact evaluations, but because we could not use approximations since the involved summations have exponentially many terms of alternating signs. The expression of $\widehat{f - F}$ in Lemma 7.1 (or Lemma 7.2) can be regarded as a way to hide those huge summations in the Fourier transforms of the cover functions.

In this section, we apply two simple bounds to this expression. The first bound (Theorem 7.3) reduces the problem of bounding the t-zero-energy of F to that of bounding the $(t-s)$-energies of the cover functions of DNF formulas derived from F. The second bound (Lemma 7.4) reduces the latter problem to bounding the $(t-s)$-energies of the u-skin functions of the derived formulas for all $u \geq 0$.

THEOREM 7.3 (zero-energy $\preccurlyeq$ energy of cover). *Let $F = (C, [n], N)$ be an s-DNF formula, and let $t \geq s$ be an integer.*

(a) (Monotone case) *Assume that F is monotone and $|C| \geq 2$. For each clause $c \in C$, define the new DNF formula $F_c = (C_c, [n], N_c)$ resulting from removing the clause c from F and adding its variables to all the other clauses, i.e., $C_c = C \backslash \{c\}$ and $N_c(d) = N(d) \cup N(c)$ for each $d \in C_c$. Then*

$$\textit{zeroEnergy}(F; t) \leq |C|^2 \max_{c \in C} \textit{energy}(\textit{cover}_{F_c}; t-s).$$

(b) (General case) *In general, if $c \in C$, let C_c be the set of clauses other than c which are consistent with c, i.e., $C_c = \{d \in C \backslash \{c\} : N(d)$ and $N(c)$ are consistent$\}$. Let C_{main} be the set of clauses which are consistent with at least one clause of F other than themselves, i.e., $C_{main} = \{c \in C : C_c \neq \emptyset\}$.*
For each clause $c \in C_{main}$, define the new DNF formula $F_c = (C_c, [n], N_c)$, where $N_c(d) = N(d) \cup N(c)$ for each $d \in C_c$. That is, F_c is the formula resulting from removing from F the clause c and all the clauses not consistent with c, and adding the literals of c to each of the remaining clauses. Then

$$\textit{zeroEnergy}(F; t) \leq |C_{main}|^2 \max_{c \in C_{main}} \textit{energy}(\textit{cover}_{F_c}; t-s).$$

Proof. (a) Let $f \in L(B_n)$ be as defined in the statement of Lemma 7.1. Thus

$$\text{zeroEnergy}(F; t) \leq E(F-f)^2 = \|\widehat{f - F}\|_2^2$$

and

$$(\widehat{f - F})(y) = \sum_{c \in C : |y \cup N(c)| > t} \widehat{\text{cover}}_{F_c}(y),$$

for each $y \in B_n$. To hide the dependency of the summation on y, for each $c \in C$, define $a_c \in L(B_n)$ by

$$a_c(y) = \begin{cases} \widehat{\text{cover}}_{F_c}(y) & \text{if } |y \cup N(c)| > t, \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\widehat{f-F} = \sum_{c\in C} a_c$. Applying a triangular inequality, we get

$$\begin{aligned}\|\widehat{f-F}\|_2 &\le \sum_{c\in C} \|a_c\|_2 \\ &= \sum_{c\in C} \left(\sum_{y\in B_n : |y\cup N(c)|>t} \widehat{\mathrm{cover}}_{F_c}(y)^2 \right)^{1/2} \\ &\le \sum_{c\in C} \left(\sum_{y\in B_n : |y|>t-s} \widehat{\mathrm{cover}}_{F_c}(y)^2 \right)^{1/2} \\ &= \sum_{c\in C} \sqrt{\mathrm{energy}(\mathrm{cover}_{F_c}; t-s)},\end{aligned}$$

where the second inequality follows from the fact that $|y\cup N(c)| \le |y|+|N(c)| \le |y|+s$ since F is an s-DNF. If follows that

$$\begin{aligned}\mathrm{zeroEnergy}(F;t) &\le \left(\sum_{c\in C} \sqrt{\mathrm{energy}(\mathrm{cover}_{F_c}; t-s)} \right)^2 \\ &\le |C|^2 \max_{c\in C} \mathrm{energy}(\mathrm{cover}_{F_c}; t-s).\end{aligned}$$

(b) The general case follows from exactly the same argument. Just replace Lemma 7.1 with Lemma 7.2, y with w, C with C_{main}, and $N(c)$ with $N'(c)\cup N''(c)$. ∎

It is not clear how to estimate the t-energy of the cover function without resorting to the u-skin function for all $u \ge 0$.

LEMMA 7.4 (energy of cover $\preccurlyeq$ energy of skin). *Let G be a DNF formula, and let $t \ge 0$ be an integer. Then*

(a)

$$\mathit{energy}(\mathit{cover}_G; t) \le \left(\int_0^\infty \sqrt{\mathit{energy}(\mathit{skin}_{G,u}; t)} e^{-u} du \right)^2,$$

(b)

$$\mathit{energy}(\mathit{cover}_G; t) \le \sup_{u\ge 0} \mathit{energy}(\mathit{skin}_{G,u}; t).$$

Proof. Part (b) follows immediately from part (a) since $(\int_0^\infty e^{-u}du)^2 = 1$.

Part (a) follows from the fact that $\mathrm{cover}_G = \int_0^\infty \mathrm{skin}_{G,u} e^{-u} du$ via the Cauchy–Schwarz inequality as we explain next. Let $a_u = \mathrm{skin}_{G,u}$ and $b = \int_0^\infty a_u e^u du$. Thus $\widehat{b} = \int_0^\infty \widehat{a}_u e^u du$ by the linearity of the Fourier transform operator. Hence

$$\widehat{b}(y)^2 = \int_0^\infty \int_0^\infty \widehat{a}_{u_1}(y)\widehat{a}_{u_2}(y) e^{-u_1-u_2} du_1 du_2$$

for all $y \in \{0,1\}^n$. It follows from the Cauchy–Schwarz inequality that

$$\begin{aligned}\sum_{y:|y|>t} \widehat{b}(y)^2 &= \int_0^\infty \int_0^\infty \left(\sum_{y:|y|>t} \widehat{a}_{u_1}(y)\widehat{a}_{u_2}(y) \right) e^{-u_1-u_2} du_1 du_2 \\ &\le \int_0^\infty \int_0^\infty \sqrt{\sum_{y:|y|>t} \widehat{a}_{u_1}(y)^2} \sqrt{\sum_{y:|y|>t} \widehat{a}_{u_2}(y)^2}\; e^{-u_1-u_2} du_1 du_2 \\ &= \left(\int_0^\infty \sqrt{\sum_{y:|y|>t} \widehat{a}_u(y)^2} e^{-u} du \right)^2 \\ &= \left(\int_0^\infty \sqrt{\mathrm{energy}(a_u;t)} e^{-u} du \right)^2. \qquad \square\end{aligned}$$

PROBLEM 7.5. *Let $G = (C, [n], N)$ be a DNF formula, $t \ge 0$ be an integer, and $u \ge 0$. If G is monotone, it follows from* (6.10) *and* (6.9) *that*

$$\begin{aligned}\mathit{energy}(\mathit{cover}_G; t) &= \sum_{y \in B_n : |y|>t} \left(\sum_{S \subset C : N(S) \ge y} (-1)^{|S|} 2^{-|N(S)|} \frac{1}{|S|+1} \right)^2, \\ \mathit{energy}(\mathit{skin}_{G,u}; t) &= \sum_{y \in B_n : |y|>t} \left(\sum_{S \subset C : N(S) \ge y} (-1)^{|S|} 2^{-|N(S)|} e^{-u|S|} \right)^2.\end{aligned}$$

In general, it follows from (6.21) *and* (6.20) *that*

$$\begin{aligned}\mathit{energy}(\mathit{cover}_G; t) &= \sum_{w \in B_n : |w|>t} \left(\sum_{a \le w} (-1)^{|a|} \sum_{S \subset C : N'(S) \cap N''(S) = \emptyset \;\&\; N(S) \ge (a, w \setminus a)} (-1)^{|S|} 2^{-|N(S)|} \frac{1}{|S|+1} \right)^2, \\ \mathit{energy}(\mathit{skin}_{G,u}; t) &= \sum_{w \in B_n : |w|>t} \left(\sum_{a \le w} (-1)^{|a|} \sum_{S \subset C : N'(S) \cap N''(S) = \emptyset \;\&\; N(S) \ge (a, w \setminus a)} (-1)^{|S|} 2^{-|N(S)|} e^{-u|S|} \right)^2.\end{aligned}$$

Analyze and estimate those sums without using the technique of section 8. *In the monotone case the sums are expressions associated with bipartite graphs. Note also that, experimentally, it is evident that energy$(\mathit{skin}_{G,u}; t)$ exponentially decreases with u for fixed G and t.*

8. Back to DNF formulas. Let G be a DNF formula, $u \ge 0$, and $t \ge 0$ be an integer. We want to estimate the t-energy of the u-skin of G. In this section, we bound the t-energy of the u-skin of G by the t-energies of DNF formulas derived from G by adding new auxiliary variables.

To motivate the technique, assume for simplicity that G is monotone. Let $G = (C, [n], N)$. The key idea behind the reduction can be easily pointed out using the Fourier transform expression of the u-skin function derived in section 6.3.

We have

$$\mathrm{energy}(\mathrm{skin}_{G,u}; t) = \sum_{y:|y|>t} \widehat{\mathrm{skin}}_{G,u}(y)^2$$

with

$$\widehat{\mathrm{skin}}_{G,u}(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-|N(S)|} e^{-u|S|}$$

by (6.9). Recall also from (6.6) that

$$\widehat{G}(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-|N(S)|}.$$

We have a bound on energy$(G;t)$ from the LMN energy bound (Theorem 5.8). Since $G = \mathrm{skin}_{G,0}$, this is a bound on energy$(\mathrm{skin}_{G,u};t)$ for $u = 0$. We need a bound for all $u \geq 0$.

The key observation is the following. Consider the special case when $e^{-u} = 2^{-v}$, where v is a nonnegative integer. Then

$$\widehat{\mathrm{skin}}_{G,u}(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N(S) \geq y} -(-1)^{|S|} 2^{-(|N(S)| + v|S|)}.$$

Construct from G a new monotone DNF formula G_v by adding v new auxiliary variables to each clause $c \in C$. That is, for each $c \in C$, let $\ddot{N}(c)$ be a size-v set of variable indices such that $\ddot{N}(c_1) \cap \ddot{N}(c_2) = \emptyset$ and $\ddot{N}(c_1) \cap [n] = \emptyset$ for all $c_1 \neq c_2 \in C$. Let $\ddot{I} = \cup_{c \in C} \ddot{N}(c)$. Then $G_v = (C, [n] \cup \ddot{I}, N_v)$, where $N_v(c) = N(c) \cup \ddot{N}(c)$ for each $c \in C$.

If $S \subset C$, then $N_v(S) = N(S) \cup \ddot{N}(S)$ and hence $|N_v(S)| = |N(S)| + v|S|$. Moreover, if $y \subset [n]$, then $N(S) \geq y$ if and only if $N_v(S) \geq y$ because $\ddot{I}$ and $[n]$ are disjoint. Thus

$$\widehat{\mathrm{skin}}_{G,u}(y) = (-1)^{|y|} \sum_{S \neq \emptyset \subset C : N_v(S) \geq y} -(-1)^{|S|} 2^{-|N_v(S)|} \qquad \text{for all } y \subset [n].$$

This leads us to the *key Fourier relation*

$$\widehat{\mathrm{skin}}_{G,u}(y) = \widehat{G}_v(y) \qquad \text{for all } y \subset [n], \tag{8.1}$$

which is the key point behind adding new auxiliary variables. It immediately implies that

$$\begin{aligned} \mathrm{energy}(\mathrm{skin}_{G,u};t) &= \sum_{y \subset [n] : |y| > t} \widehat{\mathrm{skin}}_{G,u}(y)^2 \\ &= \sum_{y \subset [n] : |y| > t} \widehat{G}_v(y)^2 \\ &\leq \sum_{y \subset [n] \cup \ddot{I} : |y| > t} \widehat{G}_v(y)^2 \qquad (8.2) \\ &= \mathrm{energy}(G_v;t), \end{aligned}$$

which enables us to use the LMN energy bound. The same argument works if G is not necessarily monotone. The above argument assumes that v is an integer. If v is not necessarily an integer and G is not necessarily monotone, we prove the following theorem.

THEOREM 8.1 (energy of skin $\preccurlyeq$ energy). *Let $G = (C, [n], N)$ be a DNF formula, and let $t \geq 0$ be an integer.*

If $d \in \mathbb{N}^C$, construct from G a new DNF formula G_d by adding d_c new auxiliary nonnegated variables to each clause $c \in C$. That is, for each $c \in C$, let $\ddot{N}(c)$ be a size-d_c set of variables indices such that $\ddot{N}(c_1) \cap \ddot{N}(c_2) = \emptyset$ and $\ddot{N}(c_1) \cap [n] = \emptyset$ for all $c_1 \neq c_2 \in C$. Let $\ddot{I} = \cup_{c \in C} \ddot{N}(c)$. Then $G_d = (C, [n] \cup \ddot{I}, N_d)$, where $N_d(c) = (N'(c) \cup \ddot{N}(c), N''(c))$ for each $c \in C$.

Let $u \geq 0$ and let $v \geq 0$ such that $e^{-u} = 2^{-v}$; i.e., $v = u/\ln 2$.

Then

$$\mathit{energy}(\mathit{skin}_{G,u}; t) \leq \max_{d \in \{\lfloor v \rfloor, \lceil v \rceil\}^C} \mathit{energy}(G_d; t).$$

The underlying analogue of the key Fourier relation in (8.1) is the following. If v is not necessarily an integer, let $0 \leq p \leq 1$ such that $p2^{-\lceil v \rceil} + (1-p)2^{-\lfloor v \rfloor} = 2^{-v} = e^{-u}$. We note in the proof below that $\widehat{\mathrm{skin}_{G,u}}(y) = E_D\, \widehat{G}_D(y)$, for each $y \subset [n]$, where D is a random vector chosen from $\{\lfloor v \rfloor, \lceil v \rceil\}^C$ by independently setting each of its entries to $\lceil v \rceil$ with probability p and to $\lfloor v \rfloor$ with probability $1-p$.

As noted above, the key point behind adding new auxiliary variables is (8.1), which can be easily seen by examining the summation in the expression of the Fourier transform of the u-skin function. We can directly verify (8.1) and its analogue without going into this summation, but with little insight into what is going on. We do that below to avoid the messy summations in the nonnecessarily monotone case.

Remark 8.2.

1. It is not clear how tight the bound is; i.e., it is not clear how much we are losing in (8.2).
2. Experimentally, it is evident that $\mathrm{energy}(\mathrm{skin}_{G,u}; t)$ exponentially decreases with u for fixed G and t. We conjecture that there is a bound on $\mathrm{energy}(\mathrm{skin}_{G,u}; t)$ in terms of u, the number m of clauses, and t which exponentially decreases with u.
   Note that $\mathrm{skin}_{G,u}(x)$ is a strictly decreasing function in u for fixed G and x. But this fact alone is not enough to conclude anything about the variation of its energy $\mathrm{energy}(\mathrm{skin}_{G,u}; t)$ with u for fixed G and t.
3. It is not clear whether the bound of Theorem 8.1 decays exponentially with u for fixed m and t. It is not hard to derive an exponentially decaying bound for $t = 1$. We were not able to do that for larger values of t.
4. It is worth mentioning that the u-skin of G does not simplify to a low degree polynomial under random restrictions, which excludes the possibility of directly adapting the argument in [13] to the u-skin function without going into the process of deriving the formulas G_d from G. The same holds for the cover function.

8.1. Proof of Theorem 8.1. We view G_d and $\widehat{G}_d$ as functions defined on $\{0,1\}^n \times \{0,1\}^{\ddot{I}}$. We denote the elements of $\{0,1\}^n \times \{0,1\}^{\ddot{I}}$ by $(x, \ddot{x})$ or $(y, \ddot{y})$.

Let $0 \leq p \leq 1$ such that $p2^{-\lceil v \rceil} + (1-p)2^{-\lfloor v \rfloor} = 2^{-v} = e^{-u}$. Such p exists since $2^{-\lceil v \rceil} \leq 2^{-v} \leq 2^{-\lfloor v \rfloor}$. Consider the random vector $D = (D_c)_{c \in C} \in \{\lfloor v \rfloor, \lceil v \rceil\}^C$ whose entries are chosen independently by setting each to $\lceil v \rceil$ with probability p and to $\lfloor v \rfloor$ with probability $1-p$. Thus $E_{D_c} 2^{-D_c} = 2^{-v} = e^{-u}$, for each $c \in C$, by the definition of p.

We show below that

$$\widehat{\mathrm{skin}}_{G,u}(y) = E_D \widehat{G}_D(y, 0) \tag{8.3}$$

for each $y \in \{0,1\}^n$.

Theorem 8.1 follows from (8.3) as follows. We have

$$\widehat{\text{skin}}_{G,u}(y)^2 = \left(E_D\ \widehat{G}_D(y,0)\right)^2 \le E_D\ \widehat{G}_D(y,0)^2,$$

since $0 \le E(X - EX)^2 = EX^2 - (EX)^2$ for any random variable X. Thus

$$\begin{aligned}
\text{energy}(\text{skin}_{G,u};t) &= \sum_{y\in\{0,1\}^n:|y|>t} \widehat{\text{skin}}_{G,u}(y)^2 \\
&\le E_D \sum_{y\in\{0,1\}^n:|y|>t} \widehat{G}_D(y,0)^2 \\
&\le E_D \sum_{(y,\ddot{y})\in\{0,1\}^n\times\{0,1\}^{\ddot{\imath}}:|(y,\ddot{y})|>t} \widehat{G}_D(y,\ddot{y})^2 \\
&= E_D\ \text{energy}(G_D;t) \\
&\le \max_{d\in\{\lfloor v\rfloor,\lceil v\rceil\}^C} \text{energy}(G_d;t).
\end{aligned}$$

To establish (8.3), we use an intermediate function. If $d \in \mathbb{N}^C$, define $\text{shell}_{G,d} : \{0,1\}^n \to \mathbb{R}$ as

$$\text{shell}_{G,d}(x) \stackrel{\text{def}}{=} 1 - \prod_{c\in C}(1 - 2^{-d_c} AND_{N(c)}(x)).$$

This is a nonuniform variation of the skin function where the clauses are weighted differently.

LEMMA 8.3. *If $d \in \mathbb{N}^C$, then*

$$\widehat{G_d}(y,0) = \widehat{shell}_{G,d}(y)$$

for each $y \in \{0,1\}^n$.

LEMMA 8.4. *For all $u \ge 0$,*

$$E_D\ shell_{G,D} = skin_{G,u}.$$

Thus, by the linearity of the Fourier transform operator,

$$\begin{aligned}
\widehat{\text{skin}}_{G,u}(y) &= E_x\ \text{skin}_{G,u}(x)\mathcal{X}_y(x) \\
&= E_x E_D\ \text{shell}_{G,D}(x)\mathcal{X}_y(x) \\
&= E_D E_x\ \text{shell}_{G,D}(x)\mathcal{X}_y(x) \\
&= E_D\ \widehat{\text{shell}}_{G,D}(y) \\
&= E_D\ \widehat{G}_D(y,0),
\end{aligned}$$

which verifies (8.3).

Proof of Lemma 8.3. We have

$$\widehat{G}_d(y,\ddot{y}) = E_{(x,\ddot{x})} G_d(x,\ddot{x})\mathcal{X}_{(y,\ddot{y})}(x,\ddot{x}).$$

Thus

$$\begin{aligned}
\widehat{G}_d(y,0) &= E_{(x,\ddot{x})} G_d(x,\ddot{x})\mathcal{X}_{(y,0)}(x,\ddot{x}) = E_{(x,\ddot{x})} G_d(x,\ddot{x})\mathcal{X}_y(x) \\
&= E_{x\in\{0,1\}^n}\left(E_{\ddot{x}\in\{0,1\}^{\ddot{\imath}}} G_d(x,\ddot{x})\right)\mathcal{X}_y(x). \qquad (8.4)
\end{aligned}$$

First note that if $c \in C$ and $(x, \ddot{x}) \in \{0,1\}^n \times \{0,1\}^{\ddot{I}}$, then

$$AND_{N_d(c)}(x, \ddot{x}) = AND_{N(c)}(x) AND_{\ddot{N}(c)}(\ddot{x}).$$

Thus

$$G_d(x, \ddot{x}) = \bigvee_{c \in C} AND_{N_d(c)}(x, \ddot{x}) = 1 - \prod_{c \in C} \left(1 - AND_{N(c)}(x) AND_{\ddot{N}(c)}(\ddot{x})\right).$$

Since each of the new auxiliary variables belongs to one and only one clause, by decomposing $\ddot{x} \in \{0,1\}^{\ddot{I}}$ as $\ddot{x} = (\ddot{x}_c)_{c \in C} \in \prod_{c \in C} \{0,1\}^{\ddot{N}(c)}$, we get

$$\begin{aligned} E_{\ddot{x} \in \{0,1\}^{\ddot{I}}} G_d(x, \ddot{x}) &= 1 - \prod_{c \in C} \left(1 - AND_{N(c)}(x) \left(E_{\ddot{x}_c \in \{0,1\}^{\ddot{N}(c)}} AND_{\ddot{N}(c)}(\ddot{x}_c)\right)\right) \\ &= 1 - \prod_{c \in C} \left(1 - AND_{N(c)}(x) 2^{-d_c}\right) \\ &= \text{shell}_{G,d}(x). \end{aligned}$$

Substituting in (8.4), we get

$$\widehat{G}_d(y, 0) = E_{x \in \{0,1\}^n} \text{ shell}_{G,d}(x) \mathcal{X}_y(x) = \widehat{\text{shell}}_{G,d}(y). \qquad \square$$

Proof of Lemma 8.4. Since, by construction, the entries of the random vector D are independent and the expected value of each is e^{-u}, we have

$$\begin{aligned} E_D \text{ shell}_{G,D}(x) &= 1 - E_D \prod_{c \in C} (1 - 2^{-D_c} AND_{N(c)}(x)) \\ &= 1 - \prod_{c \in C} (1 - (E_{D_c} 2^{-D_c}) AND_{N(c)}(x)) \\ &= 1 - \prod_{c \in C} (1 - e^{-u} AND_{N(c)}(x)). \end{aligned}$$

If $u = 0$, we get $E_D \text{ shell}_{G,D}(x) = 1 - \prod_{c \in C}(1 - AND_{N(c)}(x)) = G(x) = \text{skin}_{G,0}(x)$ by the definition of the extension of $\text{skin}_{G,u}$ to $u = 0$.

If $u > 0$, we have $1 - e^{-u} AND_{N(c)}(x) = (1 - e^{-u})^{AND_{N(c)}(x)}$ for all $c \in C$ and all $x \in \{0,1\}^n$ (if $AND_{N(c)}(x) = 0$, both terms are 1; if $AND_{N(c)}(x) = 1$, both terms are $1 - e^{-u}$). Thus

$$\begin{aligned} E_D \text{ shell}_{G,D}(x) &= 1 - \prod_{c \in C} (1 - e^{-u})^{AND_{N(c)}(x)} \\ &= 1 - (1 - e^{-u})^{\sum_{c \in C} AND_{N(c)}(x)} \\ &= \text{skin}_{G,u}(x). \end{aligned}$$

It follows that $E_D \text{ shell}_{G,D}(x) = \text{skin}_{G,u}(x)$ for all $u \geq 0$. $\square$

9. A sharper bound. We derived in section 5.11 an asymptotic version of Theorem 1.1 based on the LMN energy bound stated in Theorem 5.8. In this section, we drive the exact bound $16m^{2.2} 2^{-\sqrt{k}/10}$ of Theorem 1.1 using (1) another form of the LMN energy bound (Theorem 9.1 below), and (2) part (a) instead of part (b) of Lemma 7.4.

If we know that F is an s-DNF formula, we can extract from [13] the following bound which is tighter than that of Theorem 5.8 when s is not relatively large.

THEOREM 9.1 (LMN energy bound for s-DNF [13]). *Let G be an m-clause s-DNF formula and $t \geq 0$ be an integer; then energy$(G;t) \leq 2e^{-t/(10es)}$ if $t > 40es$.*

Proof. It follows from Lemmas 5 and 6 in [13] that if $f : \{0,1\}^n \to \{0,1\}$ and $pt > 8$, then

$$\text{energy}(f;t) \leq 2Pr_\rho[deg\,(f_\rho) > pt/2],$$

where ρ is a random p-restriction. Corollary 1 in [13] on Hastad's switching lemma asserts that if G is an s-DNF formula, then

$$Pr_\rho[deg\,(G_\rho) > k] < (5ps)^k,$$

where ρ is a random p-restriction. It follows that $\text{energy}(G;t) \leq 2(5ps)^{pt/2}$ if $pt > 8$. Setting $p = 1/(5es)$ to minimize $(5ps)^{pt/2}$, we get $\text{energy}(G;t) \leq 2e^{-t/(10es)}$ if $t > 40es$. □

First, we substitute the bound of Theorem 9.1 in Theorem 8.1.

COROLLARY 9.2 (energy of skin). *Let G be an m-clause s-DNF formula, $u \geq 0$, and let $t \geq 0$ be an integer. Then*

$$energy(skin_{G,u};t) \leq 2\exp\left(-\frac{t}{10e(s+\lceil u/\ln 2\rceil)}\right)$$

if $t > 40e(s + \lceil u/\ln 2\rceil)$.

Proof. It is enough to note that in the setting of Theorem 8.1, for each $d \in \{\lfloor v\rfloor, \lceil v\rceil\}^m$, G_d is by construction an $(s + \lceil v\rceil)$-DNF whose number of clauses equals that of G. □

Remark 9.3. In section 5.11, we obtained from Theorem 8.1 via Theorem 5.8 the bound $\text{energy}(\text{skin}_{G,u};t) \leq 2m2^{-\sqrt{t}/20}$. This bound does not depend on u. In the above corollary, we used Theorem 9.1 to derive a bound on $\text{energy}(\text{skin}_{G,u};t)$ which depends on u. If u is less than some value, this bound is better than $2m2^{-\sqrt{t}/20}$. However, the bound increases with u for s and t fixed, contradicting the experimental behavior of $\text{energy}(\text{skin}_{G,u};t)$. This stems from the fact that the special structure of G_d (the large number of new auxiliary variables) was not exploited when bounding $\text{energy}(G_d;t)$. Is it possible to exploit this structure to get a better bound? See also Remark 8.2 and Problem 7.5 for related open problems and improvement directions. Now we substitute the bound of Corollary 9.2 in part (a) of Lemma 7.4, which says that

$$\text{energy}(\text{cover}_G;t) \leq \left(\int_0^\infty \sqrt{\text{energy}(\text{skin}_{G,u};t)}e^{-u}du\right)^2$$

for each DNF formula G and each integer $t \geq 0$.

COROLLARY 9.4 (energy of cover). *Let G be an m-clause s-DNF formula and $t \geq 0$ be an integer. Then*

$$energy(cover_G;t) \leq 6 \times 2^{-\left(\sqrt{t/(5e\ln 2)+(s+1)^2}-(s+1)\right)}$$

if $\sqrt{t/(5e\ln 2)+(s+1)^2} - (s+1) > 4/\ln 2$.

Proof. To bound $\text{energy}(\text{cover}_G;t)$ in terms of s, we divide the integral into two parts $\int_0^{u_0}$ and $\int_{u_0}^\infty$, where $u_0 > 0$ is a parameter we optimize on. In the range

$0 < u \le u_0$, we use the bound of Corollary 9.2. For $u > u_0$, we use the trivial bound $\text{energy}(\text{skin}_{G,u}; t) \le \sum_y \widehat{\text{skin}}^2_{G,u}(y) = E[\text{skin}^2_{G,u}] \le 1$ (we can do better than that for $u > u_0$, but that will not significantly help). That is,

$$\begin{aligned}\sqrt{\text{energy}(\text{cover}_G; t)} &\le \int_0^{u_0} \sqrt{\text{energy}(\text{skin}_{G,u}; t)} e^{-u} du + \int_{u_0}^{\infty} \sqrt{\text{energy}(\text{skin}_{G,u}; t)} e^{-u} du \\ &\le \left(2 \exp\left(-\frac{t}{10e(s + \lceil u_0/\ln 2 \rceil)}\right)\right)^{1/2} \int_0^{u_0} e^{-u} du + \int_{u_0}^{\infty} e^{-u} du \\ &\le \sqrt{2} \exp\left(-\frac{t}{20e(s + u_0/\ln 2 + 1)}\right) + e^{-u_0},\end{aligned}$$

since $\int_0^{u_0} e^{-u} du \le 1$ and $\int_{u_0}^{\infty} e^{-u} du = e^{-u_0}$. This holds assuming that $t > 40e(s + \lceil u_0/\ln 2 \rceil)$, which is satisfied if $t > 40e(s + u_0/\ln 2 + 1)$. We use a suboptimal value of u_0 to simplify the bound. Set $u_0 \ge 0$ so that the two exponents are equal, i.e., $t = 20e(s + u_0/\ln 2 + 1)u_0$. Solving the quadratic equation, we get $2u_0/\ln 2 = \sqrt{t/(5e \ln 2) + (s+1)^2} - (s+1)$. Since $t = 20e(s + u_0/\ln 2 + 1)u_0$, the condition on t is equivalent to $u_0 > 2$, i.e., $2u_0/\ln 2 > 4/\ln 2$. Therefore $\text{energy}(\text{cover}_G; t) \le ((1+\sqrt{2})e^{-u_0})^2 < 6e^{-2u_0} = 6 \times 2^{-2u_0/\ln 2}$. ∎

Then we substitute the bound of Corollary 9.4 in Theorem 7.3.

COROLLARY 9.5 (zero-energy of s-DNF). *Let F be an m-clause s-DNF formula and $t \ge s$ be an integer. Then*

$$\textit{zeroEnergy}(F; t) \le 6m^2 2^{-\left(\sqrt{(t-s)/(5e \ln 2) + (2s+1)^2} - (2s+1)\right)} \tag{9.1}$$

if $m \ge 4$.

Proof. It is enough to note that, in the language of part (b) of Theorem 7.3, for each $c \in C_{main}$, F_c is a $2s$-DNF formula with at most $m-1$ clauses and at least one clause (by the definition of C_{main}). Moreover, $|C_{main}| \le |C| = m$. Theorem 7.3 implies (9.1) subject to the condition $\sqrt{(t-s)/(5e \ln 2) + (2s+1)^2} - (2s+1) > 4/\ln 2$. If this condition is not satisfied, then the upper bound in (9.1) is $\ge 6m^2 2^{-4/\ln 2} = 6m^2 e^{-4} > 1$ if $m \ge 4$. That is, under the assumption $m \ge 4$, the upper bound in (9.1) is trivial when the condition is not satisfied. ∎

Substituting the bound of Corollary 9.5 in Lemma 5.5 for $t = \lfloor \frac{k-s}{2} \rfloor$, we obtain the following corollary.

COROLLARY 9.6 (bias of s-DNF). *Let F be an m-clause s-DNF formula and $k \ge 3s$ be an integer; then*

$$\text{bias}(F; k) \le 6m^3 2^{-\left(\sqrt{(k-3s-1)/(10e \ln 2) + (2s+1)^2} - (2s+1)\right)}.$$

Proof. The condition $t = \lfloor \frac{k-s}{2} \rfloor \ge s$ is equivalent to $k \ge 3s$. To simplify the exponent, we used the bound $t = \lfloor \frac{k-s}{2} \rfloor \ge \frac{k-s}{2} - \frac{1}{2}$; hence $t - s \ge \frac{k-3s-1}{2}$. Finally, we drop the condition $m \ge 4$ since if $m < 4$, then F has at most $ms \le 3s$ variables, in which case the condition $k \ge 3s$ implies that $\text{bias}(F; k) = 0$. ∎

Finally, substituting the bound in Corollary 9.6 in Lemma 5.4 and optimizing on s, we conclude the proof of Theorem 1.1. We set below $s = \Theta(\sqrt{k})$ if $k = \Omega(\log^2 m)$.

COROLLARY 9.7 (bias of DNF). *Let F be an m-clause DNF formula and $k \ge 0$ be an integer; then*

$$\text{bias}(F; k) \le 16m^{2.2} 2^{-\sqrt{k}/10}.$$

Proof. Let $\epsilon = \mathrm{bias}(F;k)$. Substituting the bound of Corollary 9.6 in Lemma 5.4, we get that, for each integer $s \geq 1$ such that $k \geq 3s$, we have

$$\begin{aligned}\epsilon &\leq 6m^3 2^{-\left(\sqrt{(k-3s-1)/(10e\ln 2)+(2s+1)^2}-(2s+1)\right)} + m2^{-s} \\ &= m\left(2^{-\left(\sqrt{(k-3s-1)/(10e\ln 2)+(2s+1)^2}-(2s+1)-\log(6m^2)\right)} + 2^{-s}\right).\end{aligned}$$

Allowing s to take noninteger values, we obtain

$$\begin{aligned}\epsilon &\leq m\left(2^{-\left(\sqrt{(k-3\lfloor s\rfloor-1)/(10e\ln 2)+(2\lfloor s\rfloor+1)^2}-(2\lfloor s\rfloor+1)-\log(6m^2)\right)} + 2^{-\lfloor s\rfloor}\right) \\ &\leq m\left(2^{-\left(\sqrt{(k-3s-1)/(10e\ln 2)+(2s-1)^2}-(2s+1)-\log(6m^2)\right)} + 2^{-(s-1)}\right) \qquad (9.2)\end{aligned}$$

for each real number $s \geq 1$ such that $k \geq 3s$. We will equate the two exponents of (9.2) and solve for s. If s is a real number such that the two exponents are equal, i.e.,

$$\sqrt{(k-3s-1)/(10e\ln 2)+(2s-1)^2} - (2s+1) - \log(6m^2) = s-1, \qquad (9.3)$$

then $\epsilon \leq 2m2^{-(s-1)}$. Note that we ignored the conditions on s: $s \geq 1$ and $k \geq 3s$. We can do that since if $s < 1$, then $m2^{-(s-1)} > m \geq 1$, and hence the right-hand side of (9.2) is a trivial bound on ϵ since $\epsilon \leq 1$. Similarly, if $k < 3s$ and $s \geq 1$, then $m2^{-\left(\sqrt{(k-3s-1)/(10e\ln 2)+(2s-1)^2}-(2s+1)-\log(6m^2)\right)} > m \geq 1$, which again makes the right-hand side of (9.2) a trivial bound on ϵ. We verify below that

$$s = \sqrt{\frac{k}{M} + \alpha\log^2(6m^2) + \beta\log(6m) + \gamma} - a\log(6m^2) - b \qquad (9.4)$$

is a solution of (9.3), where $M \approx 94.208$, $\alpha = 0.16$, $\beta \approx 0.499$, $\gamma \approx 0.362$, $a = 2.2$, and $b \approx 0.416$. It follows that

$$\begin{aligned}\epsilon &\leq 2m2^{-(s-1)} = 2^2 m 2^{-\sqrt{k/M+\alpha\log^2(6m^2)+\beta\log(6m^2)+\gamma}+a\log(6m^2)+b} \\ &< 2^2 m 2^{-\sqrt{k/M}+a\log(6m^2)+b} = (2^{2+b}6^a)m^{1+2a}2^{-\sqrt{k/M}} \\ &< 16m^{2.2}2^{-\sqrt{k}/10}.\end{aligned}$$

To verify that (9.4) is a solution of (9.3), write (9.3) as

$$\begin{aligned}0 &= \left(3s+\log(6m^2)\right)^2 - (2s-1)^2 - \frac{k-3s-1}{10e\ln 2} \\ &= 5s^2 + \left(6\log(6m^2)+4+\frac{3}{10e\ln 2}\right)s + \log^2(6m^2) - 1 + \frac{1}{10e\ln 2} - \frac{k}{10e\ln 2} \\ &= 5\left(s^2 + 2(a\log(6m^2)+b)s + c\log^2(6m^2) - d - \frac{k}{M}\right),\end{aligned}$$

where $a = 0.6$, $b = 0.4 + \frac{3}{100e\ln 2} \approx 0.416$, $c = 0.2$, $d = 0.2 - \frac{1}{50e\ln 2} \approx 0.189$, and $M = 50e\ln 2 \approx 94.208$. The larger solution is

$$\begin{aligned}s &= \sqrt{\frac{k}{M} + (a\log(6m^2)+b)^2 - c\log^2(6m^2) + d} - a\log(6m^2) - b \\ &= \sqrt{\frac{k}{M} + \alpha\log^2(6m^2) + \beta\log(6m^2) + \gamma} - a\log(6m^2) - b,\end{aligned}$$

where $\alpha = a^2 - c = 0.16$, $\beta = 2ab \approx 0.499$, and $\gamma = b^2 + d \approx 0.362$. ∎

10. Optimal solution. The proof of Theorem 1.1 does not depend on this section since the former is based on the suboptimal solution constructed in section 7.

Let $F = (C, [n], N)$ be a DNF formula, and let $t \geq 0$. Recall that zeroEnergy$(F; t)$ is the minimum value of $E(F-f)^2$ over the choice of $f \in L(B_n)$ such that $deg(f) \leq t$, and f satisfies the *F-zeros-constraint* $f(x) = 0$ for each $x \in B_n$ such that $F(x) = 0$.

In this section, we derive a compact form of the optimal solution of the least square problem underlying the definition of zeroEnergy$(F; t)$. For simplicity, we restrict our attention to the case when F is a monotone DNF formula. The optimal solution can be characterized in terms of the zeta function of the dual order ideal P_F of B_n consisting of satisfying assignments of F. Unable to estimate the optimal solution, we leave the problem open.

Recall first the posets terminology in section 6.1. We need the following additional elementary poset notions. An *antichain* of a poset X is a subset A of X such that any two distinct elements of A are incomparable. A subset I of X is called a *dual order ideal* if $x \in I$ and $y \geq x$; then $y \in I$. We say that a dual order ideal is *generated by* a subset A of X if $I = \{x \in X : x \geq y \text{ for some } y \in A\}$. Any dual order ideal has a generating antichain.

Let $F = (C, [n], N)$ be a monotone DNF. We can associate with F the subposet P_F of B_n consisting of the satisfying assignments of F, i.e., $P_F = \{x \in B_n : F(x) = 1\}$. Let A_F be the set of clauses of F regarded as subsets of $[n]$, i.e., $A_F = \{N(c) : c \in C\} \subset B_n$. Equivalently, P_F is the dual order ideal of B_n generated by A_F. We call a dual order ideal of B_n *nontrivial* if it is not the empty ideal or B_n itself. Recall that we assumed in the definition of a DNF formula that it contains at least one clause and no empty clauses to avoid degenerate cases. Thus P_F is a nontrivial dual order ideal of B_n. Conversely, to each nontrivial dual order ideal P of B_n and to each set of generator A of P, we can associate a monotone DNF formula F such that $A_F = A$ and $P_F = P$. The formula F is unique up to duplicate clauses. Note also that A_F is an antichain if and only if no clause of F can be removed without changing the boolean function computed by F.

A key remark is the following.

LEMMA 10.1. *Let F be a monotone DNF formula on n variables. If $f \in L(B_n)$, then f satisfies the F-zeros-constraint if and only if f is a linear combination of $\{AND_z\}_{z \in P_F}$.*

Proof. Let $Z_F = B_n \backslash P_F = \{x \in B_n : F(x) = 0\}$. Thus the F-zeros-constraint on f is $f|_{Z_F} = 0$. The if part follows from the fact that, by the definitions of Z_F and P_F, $AND_z|_{Z_F} = 0$ for all $z \in P_F$. One way to demonstrate the only if part is to note that, since $\{AND_z\}_{z \in B_n}$ are linearly independent, dim $span\{AND_z\}_{z \in P_F} = |P_F| = \dim\{f \in L(P_F) : f|_{Z_F} = 0\}$. ∎

We cast the zero-energy problem in the language of zeta functions of dual order ideals.

DEFINITION 10.2. *Say that P is a nontrivial dual order of ideal of B_n, and let $t \geq 0$ be an integer. Let $P_t = \{z \in P : |z| \leq t\}$, and define the projection map $\pi_t : L(P) \to L(P_t)$, $f \mapsto f|_{P_t}$, and its transpose $\pi_t^T : L(P_t) \to L(P)$, the extension by zeros map. Consider the zeta function ζ_P of P as a linear transformation $L(P) \to L(P)$, and consider the linear transformation $\zeta_P \pi_t^T : L(P_t) \to L(P)$. Define*

$$\Delta_t(P) \stackrel{\text{def}}{=} \min_{g \in L(P_t)} \|1_P - \zeta_P \pi_t^T g\|_2^2,$$

where $1_P \in L(P)$ is the all ones function and $\|.\|_2$ is the L_2-norm on $L(P)$. Note that if $P_t = \emptyset$, by convention, $L(P_t)$ consists of the zero function.

That is, $\Delta_t(P)$ is the least square L_2-approximation error resulting from approximating the all ones function on P by $\zeta_P \pi_t^T g$ over the choice of $g \in L(P_t)$.

LEMMA 10.3. *Let F be a monotone DNF formula on n variables, and let $t \geq 0$ be an integer; then $2^n zeroEnergy(F;t) = \Delta_t(P_F)$.*

Proof. Let $P = P_F$, $Z = B_n \backslash P = \{x \in B_n : F(x) = 0\}$, and $V_t = \{f \in L(B_n) : f|_Z = 0 \text{ and } deg(f) \leq t\}$. Thus

$$2^n \text{zeroEnergy}(F;t) = \min_{f \in V_t} 2^n E(F-f)^2 = \min_{f \in V_t} \|1_P - f|_P\|_2^2,$$

since $F|_Z = f|_Z = 0$ and $F|_P = 1_P$. The lemma then follows from the key remark in Lemma 10.1, which says that if $f \in L(B_n)$, then $f|_Z = 0$ if and only if there exists $g \in L(P)$ such that $f = \sum_{z \in P} g(z) AND_z$. Note that (1) $deg(f) \leq t$ if and only if $g \in \pi_t^T L(P_t)$, and (2) $f = \sum_{z \in P} g(z) AND_z$ can be expressed as $f = \pi_P^T \zeta_P g$, where $\pi_P^T : L(P) \to L(B_n)$ is the extension by zeros map (the transpose of the projection map $\pi_P : L(B_n) \to L(P)$, $f \mapsto f|_P$). □

LEMMA 10.4. *Let P be a nontrivial dual order of ideal of B_n, and let $t \geq 0$ be an integer such that $P_t \neq \emptyset$. Let $v = \pi_t \zeta_P^T 1_P$, and let $M = \pi_t \zeta_P^T \zeta_P \pi_t^T$; i.e., M is the P_t-truncation of the matrix $\zeta_P^T \zeta_P$. Then M is invertible and*

$$\Delta_t(P) = |P| - v^T M^{-1} v. \tag{10.1}$$

Moreover,

$$v = 2^n (2^{-|x|})_{x \in P_t}, \tag{10.2}$$

$$M = 2^n (2^{-|x \cup y|})_{x,y \in P_t}. \tag{10.3}$$

We can also express $\Delta_t(P)$ as follows. Let

$$M^* = \begin{bmatrix} 1 & v^T \\ v & M \end{bmatrix},$$

and let D be the value which, when added to the $(\emptyset,\emptyset)$-entry of the matrix M^*, makes it singular; then

$$\Delta_t(P) = |P| - D - 1 \tag{10.4}$$

$$= |P| + \frac{\det(M^*)}{\det(M)} - 1. \tag{10.5}$$

Proof. We have a least square problem of the form $\min_g \|b - Ag\|_2^2$, where $b = 1_P$ and $A = \zeta_P \pi_t^T$. The matrix A has full column rank since ζ_P is nonsingular. The optimal solution is $\|b - Ag^*\|_2^2 = b^T b - (A^T b)^T g^*$, where $A^T A g^* = A^T b$. Since A has full column rank, the matrix $A^T A$ is invertible. In our case, we have $b^T b = 1_P^T 1_P = |P|$, $A^T b = \pi_t \zeta_P^T 1_p = v$, and $A^T A = \pi_t \zeta_P^T \zeta_P \pi_t^T = M$. This proves (10.1).

To verify (10.2), let $x \in P$. We have $(\zeta_P^T 1_P)(x) = \sum_{y \in P : x \leq y} 1 = \sum_{y \in B_n : x \leq y} 1$ since $x \in P$ and P is a dual order ideal of B_n. Thus $(\zeta_P^T 1_P)(x) = 2^{|[n] \backslash x|} = 2^n 2^{-|x|}$. Then (10.2) follows from restricting x to P_t. To verify (10.3), let $f \in L(P)$ and $x \in P$. We have

$$(\zeta_P^T \zeta_P f)(x) = \sum_{z \in P : z \geq x} \sum_{y \in P : y \leq z} f(y) = \sum_{y \in P} f(y) \sum_{z \in P : z \geq x,y} 1.$$

Since $x \in P$ and P is a dual order ideal, we have

$$\sum_{z \in P : z \geq x,y} 1 = \sum_{z \in B_n : z \geq x,y} 1 = \sum_{z \in B_n : z \geq x \cup y} 1 = 2^{|[n] \setminus (x \cup y)|} = 2^n 2^{-|x \cup y|}.$$

Hence $(\zeta_P^T \zeta_P f)(x) = 2^n \sum_{y \in P} f(y) 2^{-|x \cup y|}$. Then (10.3) follows by restricting f to $L(P_t)$ and x to P_t.

To verify (10.4), write (10.1) as $\Delta_t(P) = |P| - v^T g^*$, where $Mg^* = v$. In matrix form, we can express this system as

$$\begin{bmatrix} 1+D & v^T \\ v & M \end{bmatrix} \begin{bmatrix} 1 \\ -g^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where $D = |P| - 1 - \Delta_t(P)$. Then (10.4) follows from the fact that the $\emptyset$-entry of any vector in the null space of the perturbation of M^* by D must be nonzero since M is nonsingular.

Finally, (10.5) follows from (10.4). In general if M is a $p \times p$ matrix, $M^* = \left[\begin{smallmatrix} a & * \\ * & M \end{smallmatrix}\right]$ is a $(p+1) \times (p+1)$ augmentation of M, and $M^{*\prime} = \left[\begin{smallmatrix} a+D & * \\ * & M \end{smallmatrix}\right]$ is a perturbation of $M^{*\prime}$, then $\det(M^{*\prime}) = \det(M^*) + D \det(M)$. Thus when $M^{*\prime}$ is singular and M is nonsingular, we get $D = -\det(M^*)/\det(M)$. ∎

PROBLEM 10.5. *Let P be a nontrivial dual order ideal of B_n, generated by m elements of B_n, and let $t \geq 0$. Study and bound $\Delta_t(P)$ in terms of m and t starting from the characterization in Lemma* 10.4.

We leave this problem open. We can conclude the following bounds from Lemma 10.3 and Corollary 9.5.

COROLLARY 10.6. *Let P be a nontrivial dual order of ideal of B_n generated by m elements of B_n each of size at most s, and let $t \geq s$ be an integer. Then we have the following bound:*

$$2^{-n} \Delta_t(P) \leq 6m^2 2^{-\left(\sqrt{(t-s)/(5e \ln 2) + (2s+1)^2} - (2s+1)\right)}$$

if $m \geq 4$.

11. Concluding remarks. After the results of this paper were described in a preliminary form [6], Razborov [23] obtained a simpler construction of a function satisfying the zeros-constraint leading to a simpler proof of an asymptotic version of our main result in Theorem 1.1. The construction of Razborov is randomized and simplifies the second step of the proof, which reduces the s-DNF constrained L_2-approximation problem to the s-DNF L_2-approximation problem.

We conclude with some problems.

The bound in Theorem 1.1 probably can be improved by studying the problems in Remarks 8.2 and 9.3 and Problems 7.5 and 10.5.

Is it possible to somehow generalize the argument of Theorem 1.1 from depth-2 circuits to AC_0 circuits, i.e., to show that $\log^{O(d)} n$-wise independence $o(1)$-fools polynomial-size depth-d circuits? A different approach toward proving this is the low degree polynomial predictor approach in [5] (see section 5.7).

One of the basic questions motivating the work reported in this paper is the quadratic residue PRG introduced in [2]. Let p be an odd prime, and denote by $\mathbb{F}_p$ the finite field of size p. Fix a subset[6] $I \subset \mathbb{F}_p$ of size $n \geq 1$. The *quadratic residue*

[6] In [2], $I = \{0, 1, \ldots, n-1\}$, but the authors' analysis does not use this restriction.

PRG (QR-PRG) is given by $G_p^I : \mathbb{F}_p \to \{0,1\}^I$, where for each $t \in I$, $G_p^I(a)_t = 1$ if $a + t$ is a quadratic residue and 0 otherwise.

The irregularity of the quadratic residue distribution promises great derandomization capabilities and has intrigued mathematicians long before complexity theory existed. The following conjecture was the motivation behind the work reported in this paper.

CONJECTURE 11.1. *For all positive integers m, n and every $\epsilon > 0$, there is an integer $p_0 = poly(m, n, \frac{1}{\epsilon})$ such that if $p \geq p_0$ is a prime and $I \subset \mathbb{F}_p$ is of size n, then the QR-PRG G_p^I ϵ-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables.*

The QR-PRG was introduced in [2] as a $\frac{n}{\sqrt{p}}$-biased probability distribution. This follows from Weil's theorem on the analogue of the Riemann hypothesis for curves over finite fields. Using the $\frac{n}{\sqrt{p}}$-bias property of the QR-PRG, we obtain from Corollary 2.3 the following quasi-polynomial version.

COROLLARY 11.2. *For all positive integers m, n and every $\epsilon > 0$, there is an integer $p_0 = 2^{O(\log^2 \frac{m}{\epsilon} \log n)}$ such that if $p \geq p_0$ is a prime and $I \subset \mathbb{F}_p$ is of size n, then the QR-PRG G_p^I ϵ-fools any boolean function computable by an m-clause DNF (or CNF) formula on n variables.*

The conjecture would imply the first (unconditional) polynomial complexity PRG for depth-2 circuits. Note that there is no reason not to believe that the derandomization capabilities of the QR-PRG are far beyond the small bias property. Conjecture 11.1 is a natural starting point. At the other extreme, can one construct an infinite family of (unrestricted) circuits $\{C_n\}_n$, where C_n is a polynomial-size circuit on n variables, such that the prime cannot be made polynomially large enough in n and $\frac{1}{\epsilon}$ in order for the QR-PRG to ϵ-fool C_n?

Appendix A. LP duality calculations. In this appendix we show the LP duality calculations needed to characterize the class of functions that are fooled by the (δ, k)-bias property. The characterization is in Theorem A.1 below and is in terms of L_1-approximability by sandwiching polynomials of degree at most k and small L_1-norm in the Fourier domain.

Recall that we stated in Theorem 4.2 the special case of Theorem A.1 corresponding to the k-wise independence property, i.e., when $\delta = 0$.

Let μ be a probability distribution on $\{0,1\}^n$, $k \geq 0$ be an integer, and $\delta \geq 0$. By definition μ has *the (δ, k)-bias property* if μ δ-fools all parity functions on k or fewer of the n binary variables. In terms of the characters $\{\mathcal{X}_y\}_y$, this is equivalent to saying that $|E_\mu \mathcal{X}_y| \leq 2\delta$ for each nonzero y in $\{0,1\}^n$ whose weight is less than or equal to k.

THEOREM A.1. *Let $g : \{0,1\}^n \to \{0,1\}$, $k \geq 0$ be an integer, and $\delta, \epsilon \geq 0$. Then the (δ, k)-bias property ϵ-fools g if and only if there exist $g_l, g_u : \{0,1\}^n \to \mathbb{R}$ such that*

(i) *$deg(g_l) \leq k$ and $deg(g_u) \leq k$,*
(ii) *$g_l \leq g \leq g_u$,*
(iii) *$2\delta \sum_{y \neq 0} |\widehat{g_l}(y)| + E(g - g_l) \leq \epsilon$ and $2\delta \sum_{y \neq 0} |\widehat{g_u}(y)| + E(g_u - g) \leq \epsilon$, where the expectation is over the uniform probability distribution.*

Therefore, asymptotically and for $\delta > 0$, the (δ, k)-bias property $o(\epsilon)$-fools a boolean function $g : \{0,1\}^n \to \{0,1\}$ if and only if there exist $g_l, g_u : \{0,1\}^n \to \mathbb{R}$ such that

- (low degree) *$deg(g_l) \leq k$ and $deg(g_u) \leq k$,*
- (sandwiching polynomials) *$g_l \leq g \leq g_u$,*
- (small L_1-norm in the Fourier domain) *$\|\widehat{g_l}\|_1 = o\left(\frac{\epsilon}{\delta}\right)$ and $\|\widehat{g_u}\|_1 = o\left(\frac{\epsilon}{\delta}\right)$,*
- (small L_1-approximation error) *$E(g_u - g_l) = o(\epsilon)$.*

Proof. The proof is by LP duality. Let $M_k \subset \mathbb{R}^{\{0,1\}^n}$ be the convex polytope of (δ, k)-biased probability distributions μ on $\{0,1\}^n$.

If μ is a probability distribution μ on $\{0,1\}^n$, then by definition μ is (δ, k)-biased if $|E_\mu \mathcal{X}_y| \leq 2\delta$ for each nonzero y in $\{0,1\}^n$ whose weight is less than or equal to k.

Thus M_k consists of all $\mu : \{0,1\}^n \to \mathbb{R}$ such that $\mu \geq 0$, $\sum_x \mu(x) = 1$, and $-2\delta \leq \sum_x \mu(x)\mathcal{X}_y(x) \leq 2\delta$ for each $y \in N_k^*$, where $N_k^* = \{y \in \{0,1\}^n : y \neq 0 \text{ and } |y| \leq k\}$.

Fix $g : \{0,1\}^n \to \{0,1\}$, and note that if μ is a probability distribution on $\{0,1\}^n$, then $Pr_{x\sim\mu}[g(x) = 1] = E_\mu g$ since g takes binary values.

We have two feasible linear programs:

$$P_u = \max_{\mu \in M_k} E_\mu g - Eg \quad \text{and} \quad P_l = \max_{\mu \in M_k} Eg - E_\mu g.$$

It is enough to show that the dual linear programs are as follows:

(I) $P_u = \min_{g_u} E(g_u - g) + 2\delta \sum_{y\neq 0} |\widehat{g}_u(y)|$, where we are minimizing over all $g_u : \{0,1\}^n \to \mathbb{R}$ such that $deg(g_u) \leq k$ and $g_u(x) \geq g(x)$ for all $x \in \{0,1\}^n$.

(II) $P_l = \min_{g_l} E(g - g_l) + 2\delta \sum_{y\neq 0} |\widehat{g}_l(y)|$, where we are minimizing over all $g_l : \{0,1\}^n \to \mathbb{R}$ such that $deg(g_l) \leq k$ and $g_l(x) \leq g(x)$ for all $x \in \{0,1\}^n$.

Actually, we have to establish only (I) since (II) follows from (I) by replacing g with $1 - g$ and performing a change of variable from g_u to $1 - g_u$.

Explicitly, $P_u = \max_\mu \sum_x \mu(x)g(x) - Eg$, where $\mu : \{0,1\}^n \to \mathbb{R}$ is subject to the constraints

$$\begin{aligned} \sum_x \mu(x) &= 1, \\ \sum_x \mu(x)\mathcal{X}_y(x) &\leq 2\delta \text{ for all } y \in N_k^*, \\ -\sum_x \mu(x)\mathcal{X}_y(x) &\leq 2\delta \text{ for all } y \in N_k^*, \\ \mu(x) &\geq 0 \text{ for all } x \in \{0,1\}^n. \end{aligned}$$

Its dual is thus $P_u = \min \alpha_0 + 2\delta \sum_{y\in N_k^*}(\alpha'_y + \alpha''_y) - Eg$, where α_0, $\{\alpha'_y\}_{y\in N_k^*}$ and $\{\alpha''_y\}_{y\in N_k^*}$ are real coefficients subject to the constraints

$$\begin{aligned} \alpha_0 + \sum_{y\in N_k^*} (\alpha'_y - \alpha''_y)\mathcal{X}_y(x) &\geq g(x) \text{ for all } x \in \{0,1\}^n, \\ \alpha'_y, \alpha''_y &\geq 0 \text{ for all } y \in N_k^*. \end{aligned}$$

In general, if a is real number, then $\min\{a' + a'' : a', a'' \geq 0 \text{ such that } a' - a'' = a\} = |a|$. Applying this to $a' = \alpha'_y$, $a'' = \alpha''_y$, and $a = \alpha_y = \alpha'_y - \alpha''_y$, we get $P_u = \min \alpha_0 + 2\delta \sum_{y\in N_k^*} |\alpha_y| - Eg$, where α_0 and $\{\alpha_y\}_y$ are real coefficients subject to the constraints

$$\alpha_0 + \textstyle\sum_{y\in N_k^*} \alpha_y \mathcal{X}_y(x) \geq g(x) \quad \text{for all } x \in \{0,1\}^n.$$

Let $g_u = \alpha_0 + \sum_{y\in N_k^*} \alpha_y \mathcal{X}_y$. Noting that $\alpha_0 = Eg_u$ and $\alpha_y = \widehat{g}_u(y)$ for all $y \in N_k^*$, we get $P_u = \min E(g_u - g) + 2\delta \sum_{y\neq 0} |\widehat{g}_u(y)|$, where we are minimizing over all $g_u : \{0,1\}^n \to \mathbb{R}$ such that $deg(g_u) \leq k$ and $g_u(x) \geq g(x)$ for all $x \in \{0,1\}^n$. ∎

Appendix B. What will not work. To justify the move from L_1 to L_2 in section 5.5, it is appropriate to briefly mention two natural L_1-approaches which fall short of bounding the k-bias of s-DNF formulas.

Inclusion-exclusion. It is natural to try to construct the sandwiching polynomials of an s-DNF formula by inclusion-exclusion as explained in [5] (see section 5.5). This approach can be used to resolve the case of read-once DNF formulas (i.e., distinct clauses do not share variables), but we were unable to push it beyond the read-once case.

Lift and reduce to a linear program. Let F be an s-DNF formula on n variables, and let $A_1, \ldots, A_m$ be the clauses of F. Let μ be a k-wise independent probability distribution on $\{0,1\}^n$ such that $k > s$. Let μ_{unif} be the uniform probability distribution on $\{0,1\}^n$. Consider the map $L : \{0,1\}^n \to \{0,1\}^m$, $x \mapsto (A_c(x))_{c=1}^m$. Let μ^* (μ^*_{unif}, respectively) be the probability distribution induced via L on $\{0,1\}^m$ by μ (μ_{unif}, respectively). Thus $Pr_{\mu}[F(x) = 0] = \mu^*(0)$, $Pr_{\mu_{unif}}[F(x) = 0] = \mu^*_{unif}(0)$, and $E_{\mu^*}\mathcal{X}_y = E_{\mu^*_{unif}}\mathcal{X}_y$ for each $y \in \{0,1\}^m$ such that $|y| \leq \lfloor k/s \rfloor$.

This suggests relaxing the problem to the following linear program: $\max_{\mu_1,\mu_2} |\mu_1(0) - \mu_2(0)|$, where μ_1, μ_2 are probability distributions on $\{0,1\}^m$ such that $E_{\mu_1}\mathcal{X}_y = E_{\mu_2}\mathcal{X}_y$, for each $y \in \{0,1\}^m$ such that $|y| \leq t$, where $t = \lfloor k/s \rfloor$.

Unfortunately, that will not work. This follows from the approximate inclusion-exclusion lower bound of [14], which implies that the maximum of the above linear program cannot be made arbitrarily small unless $t = \Omega(\sqrt{m})$. One of the issues of this relaxation is that it ignores the actual values of the t-moments[7] of μ_1 and μ_2. It uses only the fact that the t-moments of μ_1 and μ_2 are equal. The values of those moments are simple and easy to derive from F, but taking them into consideration gives us an intriguing linear program, which is not clear how to bound.

Acknowledgments. The author would like to thank Sanjoy Mitter, Daniel Spielman, and Madhu Sudan for very helpful discussions on this material; Widad Machmouchi for valuable comments on the first and second drafts of the paper; and the anonymous referees for valuable comments which significantly improved the presentation of the paper.

REFERENCES

[1] J. Aspnes, R. Beigel, M. Furst, and S. Rudich, *The expressive power of voting polynomials*, Combinatorica, 14 (1994), pp. 135–148.

[2] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.

[3] N. Alon, O. Goldreich, and Y. Mansour, *Almost k-wise independence versus k-wise independence*, Inform. Process. Lett., 88 (2003), pp. 107–110.

[4] M. Ajtai and A. Wigderson, *Deterministic simulation of probabilistic constant depth circuits*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 11–19.

[5] L. Bazzi, *Minimum Distance of Error Correcting Codes versus Encoding Complexity, Symmetry, and Pseudorandomness*, Ph.D. dissertation, MIT, Cambridge, MA, 2003.

[6] L. Bazzi, *Polylogarithmic independence can fool DNF formulas*, in Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, 2007, pp. 63–73.

[7] If μ is a probability distribution on $\{0,1\}^m$ and $t \geq 0$ is an integer, define the *moments* vector of μ to be $c_\mu = (c_\mu(A) \stackrel{\text{def}}{=} E_{x\sim\mu} \wedge_{i\in A} x_i)_{A\subset[m]}$, and define the *t-moments* of μ to be the vector $(c_\mu(A))_{A\subset[m]:|A|\leq t}$. Note that $c_\mu = \zeta^T_{B_m}\mu$, where ζ_{B_m} is the zeta function of the poset B_m consisting of the set of subsets of $[m]$ ordered by inclusion.

[7] R. Beigel, N. Reingold, and D. Spielman, *The perceptron strikes back*, in Proceedings of the 6th Annual IEEE Conference on Structure in Complexity Theory, 1991, pp. 286–291.
[8] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.
[9] J. Hastad, *Computational Limitations for Small Depth Circuits*, Ph.D. dissertation, MIT, Cambridge, MA, 1986.
[10] R. Impagliazzo and A. Wigderson, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 220–229.
[11] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, 1988, pp. 68–80.
[12] R. J. Lechner, *Harmonic analysis of switching functions*, in Recent Development in Switching Theory, Academic Press, New York, 1971, pp. 121–228.
[13] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. ACM, 40 (1993), pp. 607–620.
[14] N. Linial and N. Nisan, *Approximate inclusion-exclusion*, Combinatorica, 10 (1990), pp. 349–365.
[15] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, in Proceedings of the 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 1–10.
[16] M. Luby and B. Velickovic, *On deterministic approximation of DNF*, Algorithmica, 16 (1996), pp. 415–433.
[17] M. Luby, B. Velickovic, and A. Wigderson, *Deterministic approximate counting of depth-2 circuits*, in Proceedings of the 2nd Israel Symposium on the Theory and Computing Systems, 1993, pp. 18–24.
[18] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.
[19] N. Nisan, *Pseudorandom bits for constant depth circuits*, Combinatorica, 12 (1991), pp. 63–70.
[20] N. Nisan and M. Szegey, *On the degree of Boolean functions as real polynomials*, Comput. Complexity, 4 (1994), pp. 301–313.
[21] N. Nisan and A. Wigderson, *Hardness vs. randomness*, in Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, 1988, pp. 2–11.
[22] A. Razborov, *Lower bounds on the size of bounded depth networks over a complete basis with logical addition*, Math. Zametki, 41 (1987), pp. 598–607.
[23] A. Razborov, *A Simple Proof of Bazzi's Theorem*, Report TR08-081, Electronic Colloquium on Computational Complexity, 2008.
[24] R. P. Stanely, *Enumerative Combinatorics*, Vol. I, Cambridge University Press, Cambridge, UK, 1997.
[25] L. Trevisan, *A note on deterministic approximate counting for k-DNF*, in Proceedings of the APPROX-RANDOM, 2004, pp. 417–426.
[26] U. Vazirani, *Randomness, Adversaries, and Computation*, Ph.D. dissertation, University of California, Berkeley, CA, 1986.
[27] A. C. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Science, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2273–2302

ON THE VALUE OF COORDINATION IN NETWORK DESIGN*

SUSANNE ALBERS†

Abstract. We study network design games where n self-interested agents have to form a network by purchasing links from a given set of edges. We consider Shapley cost sharing mechanisms that split the cost of an edge in a fair manner among the agents using the edge. It is well known that the price of anarchy of these games is as high as n. Another line of research has focused on evaluating the price of stability, i.e., the cost of the best Nash equilibrium relative to the social optimum. In this paper we investigate to which extent coordination among agents can improve the quality of solutions. We resort to the concept of *strong Nash equilibria*, which were introduced by Aumann and are resilient to deviations by coalitions of agents. We analyze the price of anarchy of strong Nash equilibria and develop lower and upper bounds for unweighted and weighted games in both directed and undirected graphs. These bounds are tight or nearly tight for many scenarios. It shows that, by using coordination, the price of anarchy drops from linear to logarithmic bounds. We complement these results by also proving the first superconstant lower bound on the price of stability of standard equilibria (without coordination) in undirected graphs. More specifically, we show a lower bound of $\Omega(\log W/\log\log W)$ for weighted games, where W is the total weight of all the agents. This almost matches the known upper bound of $O(\log W)$. Our results imply that, for most settings, the worst-case performance ratios of strong coordinated equilibria are essentially always as good as the performance ratios of the best equilibria achievable without coordination. These settings include unweighted games in directed graphs as well as weighted games in both directed and undirected graphs.

Key words. coalition, price of anarchy, price of stability, Shapley cost sharing, strong Nash equilibrium

AMS subject classifications. 68M10, 68Q25, 90B10, 90B18, 91A10, 91A43, 91A80, 91B32, 91B52

DOI. 10.1137/070701376

1. Introduction.

Communication networks are pervasive and critical to modern society. Nonetheless, the formation and evolution of large networks is not well understood, a major reason being that these networks typically are not built by a central authority but rather by many economic agents that have selfish interests. For this reason, research on network design has focused on game-theoretic approaches over the past years; see, e.g., [2, 3, 5, 6, 7, 8, 9, 11, 16, 21, 22].

We study network design games that have recently received a lot of attention [2, 3, 6, 7, 15, 18] and are simple yet powerful enough to capture the two most important objectives of agents: connection establishment and cost minimization. Consider a directed or undirected graph G where each edge e has a nonnegative cost $c(e)$. There are n agents, each of which has to connect a set of terminals. The agents form a network by selecting edges. A strategy S_i of an agent i is a set of edges connecting the desired terminals. The cost of the edges used by all the agents has to be covered. A fundamental cost sharing mechanism is *Shapley cost sharing*, which was proposed by Anshelevich et al. [3] for network design games and has been studied with respect to

*Received by the editors August 28, 2007; accepted for publication (in revised form) September 29, 2008; published electronically March 4, 2009. This work was supported by the German–Israeli Foundation for Scientific Research & Development, project G-783-61.6/2003, and by the German Research Foundation, project AL 464/4-2. This work was done while the author was on sabbatical leave at Carnegie Mellon University, Pittsburgh, and was supported by the ALADDIN project.
http://www.siam.org/journals/sicomp/38-6/70137.html

†Department of Computer Science, University of Freiburg, Georges Köhler Allee 79, 79110 Freiburg, Germany (salbers@informatik.uni-freiburg.de).

other networking problems as well [12, 17]. In Shapley cost sharing, the cost of an edge is shared in a fair manner among the agents using that edge. In an *unweighted game*, if k agents use an edge e in their strategies, then each of these agents pays a share of $c(e)/k$. In a *weighted game*, each agent i has a weight w_i and contributes a share of $c(e)w_i/W_e$, where W_e is the total weight of agents using e. We are interested in stable networks where no agent has the incentive to deviate from its strategy. Stability is modeled by considering Nash equilibria. A combination $\mathcal{S} = (S_1, \ldots, S_n)$ of strategies forms a Nash equilibrium if no agent has a better strategy with a strictly smaller cost if all the other agents adhere to their strategies. A widely accepted performance measure for evaluating the quality of Nash equilibria is the *price of anarchy* [19], which is the maximum ratio of the total cost incurred by any Nash equilibrium to the cost spent by the social optimum. Unfortunately, for our network design games, the price of anarchy is as high as n. An interesting alternative quality measure, proposed by Anshelevich et al. [2], is *price of stability* which is the ratio of the best Nash equilibrium relative to the social optimum. Anshelevich et al. [2] proved that the price of stability in unweighted network design games is $O(\log n)$.

The scenario described so far assumes that agents are completely noncooperative, isolated entities. However, for long-term decisions such as network design, given today's communication infrastructure, this assumption is not entirely realistic. It is more natural that agents will discuss possible strategies and, as in other economic markets, form coalitions taking strategic actions that are beneficial to all members of the group. In such cooperative environments we again seek stable solutions. In this context, in 1959 Aumann [4] introduced the concept of *strong Nash equilibria*, which ensure stability against deviations by every conceivable coalition of agents. More specifically, no coalition can cooperatively deviate in a way that benefits all its members, taking the actions of the agents outside the coalition as given. With respect to network design, an important question is whether coordination among agents yields strictly better solutions. Is it possible to achieve significant improvements? We prove that this is the case. When coordination is allowed, the price of anarchy of strong Nash equilibria drops from n to $O(\log n)$ in unweighted games. Similar improvements show in weighted games. Obviously, any strong Nash equilibrium is a standard Nash equilibrium, which is immune to deviations of single agents. Hence strong Nash equilibria cannot be better than the best standard Nash equilibria. A second natural question is how strong Nash equilibria rank relative to the best standard Nash equilibrium. When coordination is allowed, is the worst-case performance of stable states close to that of the best stable states achievable without cooperation? We answer this question in the affirmative in terms of anarchy and stability measures. For most settings, the price of anarchy of strong Nash equilibria is essentially always as good as the corresponding stability bounds of standard equilibria. These settings include unweighted games in directed graphs as well as weighted games in both directed and undirected graphs.

Previous results. Research on the network design games defined above was initiated by Anshelevich et al. [3]. In this paper the authors considered general cost sharing schemes that are not restricted to Shapley mechanisms. Anshelevich et al. studied undirected graphs and first addressed scenarios where each agent has to connect one terminal to a common destination. They designed Nash equilibria whose cost is equal to the cost of the optimum. Furthermore Anshelevich et al. [3] investigated the general scenario that each agent has to connect a set of terminals. In this case there are graphs that do not admit Nash equilibria. The authors therefore studied *α-approximate Nash equilibria* in which no agent can improve its cost by a factor of more than α, where $\alpha > 1$. Anshelevich et al. proved that there always exists a

3-approximate Nash equilibrium whose cost is equal to that of the optimum. Furthermore, they derived a polynomial time algorithm that gives a $(4.65 + \epsilon)$-approximate Nash equilibrium whose cost is twice the optimum.

In the following two paragraphs we describe the results known for network design games with Shapley cost sharing. The setting was introduced in a second paper by Anshelevich et al. [2] who first analyzed unweighted games. Using elegant potential function arguments based on a potential by Monderer and Shapley [20], the authors proved that every directed or undirected graph admits a Nash equilibrium and that the price of stability is upper bounded by $H(n)$. Here $H(n) = \sum_{i=1}^{n} 1/i$ is the nth harmonic number, which is closely approximated by the natural logarithm, i.e., $\ln(n+1) \leq H(n) \leq \ln n + 1$. The upper bound of $H(n)$ on the price of stability is tight for directed graphs. For undirected graphs Anshelevich et al. [2] showed a lower bound of 4/3 on the price of stability; the lower bound construction uses two agents that have to establish a connection to a common destination. Additionally, Anshelevich et al. [2] considered weighted games and showed the existence of Nash equilibria in two-agent games. For directed graphs they gave a lower bound of $\Omega(\max\{n, \log W\})$ on the price of stability, where W is the total weight of all the agents.

Chen and Roughgarden [7] further investigated weighted games in directed graphs. They showed that there are graphs that do not admit Nash equilibria. Chen and Roughgarden then demonstrated that, for any $\alpha = \Omega(\log w_{\max})$, α-approximate Nash equilibria do exist and that the price of stability is $O((\log W)/\alpha)$. Here $w_{\max}$ is the maximum weight of any agent. These trade-offs are nearly tight. Further work on unweighted games was presented by Fiat et al. [15] and Chekuri et al. [6].

We became aware that, independent of our work, very recently Epstein, Feldman, and Mansour [10] studied strong Nash equilibria for unweighted network design games in directed graphs. They assumed that each agent has to connect a pair of terminals and considered Shapley as well as general cost sharing mechanisms. Epstein, Feldman, and Mansour first observed that there are directed graphs that do not admit strong Nash equilibria. As their main contribution they presented topological characterizations for equilibrium existence. More specifically, they showed that if each agent has to connect a terminal to a common destination, each series parallel graph has a strong Nash equilibrium. If arbitrary terminal pairs are allowed, every extension parallel graph admits a strong Nash equilibrium when Shapley cost sharing is adopted. Furthermore, they analyzed the quality of strong Nash equilibria, showing a bound of $\Theta(\log n)$ on the price of anarchy for Shapley cost sharing and a bound of 1 for general cost sharing schemes when each agent has to connect to a common destination.

Finally, the authors of [1, 13, 14] studied the strong Nash equilibria in scheduling and load balancing games.

Our contribution. This paper presents an in-depth study of network design games with Shapley cost sharing when coordination among agents is allowed. We present upper and lower bounds on the price of anarchy achieved by strong Nash equilibria. We study scenarios with unrestricted coordination, i.e., coalitions of any size (or weight) may be formed, and we also consider settings where the size (or weight) of a coalition is limited.

The first part of the paper addresses unweighted network design games. We first observe that there are directed as well as undirected graphs that do not admit strong Nash equilibria and then give a sufficient existence condition. More specifically, we show that α-approximate strong Nash equilibria exist in any directed or undirected graph, for any $\alpha \geq H(c)$, if coalitions of size up to c, $1 \leq c \leq n$, are allowed. Again, $H(c)$ is the cth harmonic number. An α-approximate strong Nash equilibrium, for

$\alpha \geq 1$, is one where no coalition (of prescribed size or weight) can deviate such that every member of the coalition improves its cost by a factor of more than α.

We next prove that the price of anarchy of strong Nash equilibria is upper bounded by $H(n) \approx \ln n$, allowing coalitions of any size. This upper bound holds for any directed or undirected graph that admits a strong Nash equilibrium. Hence, using coordination, we achieve an exponential improvement in terms of the price of anarchy, compared to noncooperative environments. We show that the upper bound of $H(n)$ is tight in directed graphs. For undirected graphs we develop a lower bound of $\Omega(\sqrt{\log n})$ on the price of anarchy. These results can be generalized to α-approximate strong Nash equilibria for any $\alpha \geq 1$. In this case all the upper and lower bounds multiply by a factor of α. For the generalized setting that coalitions of size up to c, $1 \leq c \leq n$, are allowed, we prove an upper bound of $\alpha \frac{n}{c} H(c)$ on the price of anarchy of α-approximate strong Nash equilibria. Again, this bound holds for any directed or undirected graph that admits an α-approximate strong Nash equilibrium for some $\alpha \geq 1$, and not just for the range $\alpha \geq H(c)$. Suppose that $\alpha = 1$. If $c = 1$, we obtain the anarchy ratio of n achieved by standard equilibria. If $c = n$, we obtain the best ratio of $H(n)$. Since $H(n)$ is a lower bound on the price of stability of (standard) Nash equilibria in directed graphs [2], we conclude that in directed graphs the worst-case performance ratios of strong Nash equilibria are essentially always as good as the performance ratios achievable by the best standard Nash equilibria.

In the second part of the paper we extend the above results to weighted network design games. We first give a sufficient condition for the existence of α-approximate strong Nash equilibria. We then prove that in directed and undirected graphs the price of anarchy of strong Nash equilibria is at most $1 + \ln W$ if the formation of coalitions is not restricted. Here W is the sum of the weights of all agents. For directed graphs we show a matching lower bound of $\Omega(\log W)$. For undirected graphs we prove a lower bound of $\Omega(\sqrt{\log W})$. Again, for any $\alpha \geq 1$, the results extend to α-approximate strong Nash equilibria, where the lower and upper bounds simply multiply by α. When coordination among agents is limited, we consider two scenarios: (1) As usual, the number of agents in a coalition might be limited. (2) The sum of the weights of the agents forming a coalition may be limited so that agents of high weight cannot leave agents of low weight in costly configurations. For this general setting we present bounds trading the price of anarchy vs. the coalition size or weight. Furthermore, we prove a lower bound on the price of stability of standard Nash equilibria in undirected graphs. We construct a family of graphs in which the price of stability is $\Omega(\log W / \log \log W)$. No superconstant lower bound was known for undirected graphs, for neither weighted nor unweighted games. Our lower bound holds even if every agent has to connect only a pair of terminals. However, individual terminal pairs are allowed. Together with the known lower bound of $\Omega(\log W)$ for directed graphs [2], we conclude that, in undirected as well as directed graphs, anarchy bounds of strong Nash equilibria essentially match the stability bounds of standard Nash equilibria.

We remark that our set of results is disjoint from that of Epstein, Feldman, and Mansour [10], except for the observation that graphs do not necessarily admit strong Nash equilibria, the upper bound of $\frac{n}{c} H(c)$, and the lower bound of $H(n)$ on the price of anarchy in unweighted games. While the upper bound proof by Epstein, Feldman, and Mansour is based on the potential function by Monderer and Shapley, in our paper we use new combinatorial arguments to establish the result. Generally speaking, our study here is more comprehensive in that we allow each agent to connect a set of terminals and consider directed and undirected graphs as well as unweighted and weighted games.

Analysis techniques. As mentioned above, our upper bounds on the price of anarchy are achieved using new combinatorial arguments that do not rely on potential functions: Starting from a strong Nash equilibrium, we perform a sequence of specific strategy changes for varying size coalitions. For each strategy change there exists one unsatisfied agent whose original cost can be bounded relative to the optimum. From a technical point of view, our strongest contribution is the lower bounds for undirected graphs. We present a new recursive framework for constructing lower bounds in network design games. Applying the recursive construction for varying parameters, we are able to obtain anarchy as well as stability bounds in both unweighted and weighted games. The protocol could also be applied to derive bounds for directed graphs, but simpler constructions work in the directed case. While the same recursive framework can be applied to construct graphs for anarchy and stability bounds, the analyses of the graphs differ. To establish anarchy bounds we have to prove that no coalition can deviate, which turns out to be a nontrivial task because all possible coalitions and strategy changes over the recursive levels must be examined. To establish a stability bound, we have to show that no better Nash equilibria exist. In fact, we will prove that our graphs admit only one Nash equilibrium.

2. Problem statement and definitions. We formally introduce the network design problems and game-theoretic concepts studied in this paper.

Network design games. Consider a graph $G = (V, E, c)$ with a nonnegative cost function $c : E \mapsto \mathbb{R}_+^0$ defined on the edges. Graph G may be directed or undirected as we will study network design in both directed and undirected graphs. Associated with G are n selfish agents, each of which has certain connectivity requirements. More specifically, let $T_i \subseteq V$ be the set of terminals that agent i wishes to connect. If G is a directed graph, then for (selected) terminal pairs $t, t' \in T_i$ we additionally have to specify which direction between the pair should be established. A strategy of an agent i consists of a set $S_i \subseteq E$ of edges satisfying the connection requirements. We assume that strategies S_i are minimal; i.e., dropping any edge of S_i leads to a configuration in which the connectivity requirements of agent i are not satisfied anymore. A combination $\mathcal{S}$ of strategies is the vector $\mathcal{S} = (S_1, \ldots, S_n)$ of individual agent strategies. Edges used by the agents have to be paid for. We consider Shapley cost sharing mechanisms that split the cost $c(e)$ of an edge e in a fair manner among the agents using that edge. In an *unweighted game*, if k agents use an edge e, then each of the k agents pays a share of $c(e)/k$ for that edge. Thus, for a combination $\mathcal{S}$ of strategies, the total cost of agent i is equal to $cost_i(\mathcal{S}) = \sum_{e \in S_i} c(e)/|\{j : e \in S_j\}|$. In a *weighted game* each agent i has a weight w_i and pays a share proportional to its weight. For any edge $e \in S_i$, agent i pays a share of $c(e)w_i/W_e$, where $W_e = \sum_{j:e \in S_j} w_j$ is the total weight of the agents j using e in their strategies. Formally, the cost of agent i in a weighted game is $cost_i(\mathcal{S}) = \sum_{e \in S_i} c(e)w_i/W_e$.

Strong Nash equilibria. We are interested in stable solutions where agents have no incentive to deviate from their strategies. Previous work has considered Nash equilibria that are resilient to deviations of single agents. A weakness of Nash equilibria is their vulnerability to deviations by coalitions of agents. To overcome this problem, Aumann [4] defined the notion of *strong Nash equilibria*. A strong Nash equilibrium is resilient to deviations of coalitions; i.e., there exists no coalition of agents that can jointly change strategies such that every agent in the coalition has a strictly smaller cost. Formally, let I be a nonempty coalition of agents. For a combination $\mathcal{S}$ of strategies, let $\mathcal{S}_I$ be the projection of $\mathcal{S}$ on I; i.e., $\mathcal{S}_I$ are the strategies of agents $i \in I$. Similarly, $\mathcal{S}_{-I}$ represents the strategies of agents $i \notin I$.

For coalition I, let $\mathcal{S}'_I$ be another choice of strategies. A combination $\mathcal{S}$ of strategies forms a strong Nash equilibrium if, for no nonempty coalition I, there exists a strategy change $\mathcal{S}'_I$ such that $cost_i(\mathcal{S}'_I, \mathcal{S}_{-I}) < cost_i(\mathcal{S})$ for all agents $i \in I$. Note that a standard Nash equilibrium is a strong Nash equilibrium where only coalitions of size one are allowed. In this spirit one can consider generalized settings in which coalitions of size at most c are permitted, $1 \leq c \leq n$. As for weighted games we will also be interested in scenarios where the total weight of agents forming a coalition is limited. This ensures that agents of high weight cannot impose too much control on agents outside a coalition.

As we shall see, strong Nash equilibria do not always exist. For this reason we relax the notion of stability, calling a combination of strategies stable if agents cannot improve their cost by a factor of more than α. More specifically, for a real value $\alpha \geq 1$, a combination $\mathcal{S}$ of strategies forms an α-approximate strong Nash equilibrium if, for no nonempty coalition I, there exists a strategy change $\mathcal{S}'_I$ such that $cost_i(\mathcal{S}'_I, \mathcal{S}_{-I}) < cost_i(\mathcal{S})/\alpha$ for all agents $i \in I$. Similarly, we can define α-approximate Nash equilibria when the size or weight of a coalition is limited. We remark that in the context of α-approximate strong equilibria another definition seems reasonable. We could call a combination of strategies an α-approximate strong Nash equilibrium if no coalition can improve its *total* cost by a factor of more than α, while still requiring that every agent of the coalition performs strictly better than before. Obviously, an α-approximate Nash equilibrium according to this second definition is an α-approximate equilibrium under the former definition but not vice versa. Thus, our original definition allows for more configurations representing equilibrium states. For this reason and because our first definition requires a sufficiently high benefit for *each* agent of a coalition to perform a strategy change, we adopt the original definition in this paper. However, all the results that we will present in the following sections also hold for the second definition as well.

Performance measures. We are interested in the performance of strong Nash equilibria relative to the social optimum. Let $cost(\mathcal{S}) = \sum_{i=1}^{n} cost_i(\mathcal{S})$ be the total cost of all the agents, and let $cost(OPT)$ be the cost of the globally optimal solution. We say that strong Nash equilibria *achieve a price of anarchy of* c if $\max_{\mathcal{S}} \frac{cost(\mathcal{S})}{cost(OPT)} \leq c$, where the maximum is taken over all strong Nash equilibria. The notion can be extended to (α-approximate) strong Nash equilibria with coalitions of limited size or weight. In this paper we will also be interested in the *price of stability* of standard Nash equilibria where coordination among agents is not allowed. The price of stability is $\min_{\mathcal{S}} \frac{cost(\mathcal{S})}{cost(OPT)}$, where the minimum is taken over all Nash equilibria.

3. Upper bounds for unweighted games. We study the existence of strong Nash equilibria and then develop upper bounds on the price of anarchy. The proof of the following proposition is presented in the appendix.

PROPOSITION 3.1. *There exist directed and undirected graphs that do not admit strong Nash equilibria.*

THEOREM 3.2. *In any directed or undirected graph, α-approximate strong Nash equilibria exist, for any $\alpha \geq H(c)$, if coalitions of size up to c are allowed.*

Proof. We use a classical potential function by Monderer and Shapley [20] to show the existence of α-approximate strong Nash equilibria. Given a graph $G = (V, E, c)$ and a combination $\mathcal{S} = (S_1, \ldots, S_n)$ of strategies, let n_e be the number of agents currently using edge $e \in E$ in their strategies, i.e., $n_e = |\{i : e \in S_i\}|$. The potential is defined as $\Phi(\mathcal{S}) = \sum_{e \in E} c(e) H(n_e)$. We will show that, while $\mathcal{S}$ does not form an α-approximate strong Nash equilibrium, when allowing coalitions of size up to c,

any α-*improvement move* strictly decreases the potential. An α-improvement move, for a coalition I with $|I| \leq c$, is a strategy change $\mathcal{S}'_I$ such that $cost_i(\mathcal{S}'_I, \mathcal{S}_{-I}) < cost_i(\mathcal{S})/\alpha$ for any agent $i \in I$. Suppose that we perform a sequence of such α-improvement moves starting from the social optimum. As the potential is upper bounded by $H(n)cost(OPT)$ and lower bounded by 0, the sequence of improvement moves must converge to an α-approximate strong Nash equilibrium.

We analyze an α-improvement move, performed by a coalition I with $|I| \leq c$. The strategy change $\mathcal{S}'_I$ of I can be viewed as being executed in two steps: (1) In a first step agents $i \in I$ drop all the edges used in strategies S_i. At this point no agent $i \in I$ shares the cost of any edge. (2) In a second step agents $i \in I$ join the edges they want to use in their new strategies $\mathcal{S}'_I$. Let E_1 be the set of edges dropped in step (1), and let E_2 be the set of edges added in step (2). These edge sets need not be disjoint. For any $e \in E$, let n_e^1 be the number of agents sharing e just after step (1), and let n_e^2 be the number of agents sharing e after step (2). The absolute value of the cost reduction experienced by I due to step (1) is

$$cost^- = \sum_{e \in E_1} c(e) \frac{n_e - n_e^1}{n_e},$$

because $e \in E_1$ is dropped by $n_e - n_e^1$ agents that each paid a share of $c(e)/n_e$. The value of this cost reduction is equal to the cost of I in the original configuration, i.e., $cost^- = \sum_{i \in I} cost_i(\mathcal{S})$, because after step (1) the cost of I is 0. The cost increase of I due to step (2) is

$$cost^+ = \sum_{e \in E_2} c(e) \frac{n_e^2 - n_e^1}{n_e^2},$$

since $e \in E_2$ is bought by $n_e^2 - n_e^1$ agents $i \in I$ who pay $c(e)/n_e^2$ each. This cost increase is equal to the cost of I in the new configuration, i.e., $cost^+ = \sum_{i \in I} cost_i(\mathcal{S}'_I, \mathcal{S}_{-I})$, because the cost of I was 0 before step (2) and the strategy change is complete after step (2). By the definition of an α-improvement move, $\sum_{i \in I} cost_i(\mathcal{S}'_I, \mathcal{S}_{-I}) < \sum_{i \in I} cost_i(\mathcal{S})/\alpha$ and hence

$$\alpha cost^+ - cost^- < 0. \tag{3.1}$$

Next we consider the potential change $\Delta\Phi$. The potential change stems from edges $e \in E_1 \cup E_2$ where cost shares change. Let Φ^- be the absolute value of the potential drop due to step (1) of the improvement move, and let Φ^+ be the potential increase due to step (2). We will show $-\Phi^- \leq -cost^-$ and $\Phi^+ \leq \alpha cost^+$. This implies $\Delta\Phi = -\Phi^- + \Phi^+ \leq -cost^- + \alpha cost^+$, and using (3.1) we obtain $\Delta\Phi < 0$, which is to be proved.

To verify $-\Phi^- \leq -cost^-$ we observe

$$\Phi^- = \sum_{e \in E_1} c(e)(H(n_e) - H(n_e^1)) \geq \sum_{e \in E_1} c(e) \frac{n_e - n_e^1}{n_e} = cost^-.$$

The inequality holds because $H(n_e) - H(n_e^1) = 1/(n_e^1 + 1) + 1/(n_e^1 + 2) + \cdots + 1/n_e \geq (n_e - n_e^1)/n_e$. It remains to prove $\Phi^+ \leq \alpha cost^+$. The potential increase is given by $\Phi^+ = \sum_{e \in E_2} c(e)(H(n_e^2) - H(n_e^1))$. We show that, for any $e \in E_2$,

$$H(n_e^2) - H(n_e^1) \leq H(n_e^2 - n_e^1) \frac{n_e^2 - n_e^1}{n_e^2}. \tag{3.2}$$

The desired inequality for the potential increase then follows because $n_e^2 - n_e^1 \leq c$ as at most c agents can join any edge in step (2) and $H(c) \leq \alpha$. The expressions in (3.2) are

$$H(n_e^2) - H(n_e^1) = \frac{1}{n_e^1+1} + \frac{1}{n_e^1+2} + \cdots + \frac{1}{n_e^2},$$
$$H(n_e^2 - n_e^1)\frac{n_e^2 - n_e^1}{n_e^2} = \left(1 + \frac{1}{2} + \cdots + \frac{1}{n_e^2 - n_e^1}\right)\frac{n_e^2 - n_e^1}{n_e^2}.$$

We compare the kth terms of these expressions, for $k = 1, \ldots, n_e^2 - n_e^1$, and establish (3.2) by proving $\frac{1}{n_e^1+k} \leq \frac{1}{k} \cdot \frac{n_e^2 - n_e^1}{n_e^2}$. This is equivalent to showing $0 \leq n_e^1(n_2^2 - n_e^1) - kn_e^1$, and this holds because $f(k) = n_e^1(n_2^2 - n_e^1) - kn_e^1$ is decreasing in k and $f(n_e^2 - n_e^1) = 0$. □

THEOREM 3.3. *In any directed or undirected graph and for any $\alpha \geq 1$, the price of anarchy of α-approximate strong Nash equilibria is upper bounded by $\frac{\alpha n}{c} H(c)$ if coalitions of size up to c are allowed.*

If there are no restrictions on the coalition size and we are interested in true strong Nash equilibria (i.e., $\alpha = 1$), we obtain the following corollary.

COROLLARY 3.4. *In any directed or undirected graph the price of anarchy of strong Nash equilibria is upper bounded by $H(n)$.*

Proof of Theorem 3.3. Let G be a graph that admits α-approximate strong Nash equilibria for some $\alpha \geq 1$, and let $\mathcal{S} = (S_1, \ldots, S_n)$ be such an equilibrium state. The basic idea of the proof is to consider all coalitions of size exactly c. For each coalition I we perform a process consisting of exactly c steps in which the agents of I try to buy the edges of the social optimum. At the end of each step exactly one agent will leave the process. Making use of the fact that in $\mathcal{S}$ no coalition of size up to c can improve its cost by a factor of more than α, we will be able to upper bound $cost_i(\mathcal{S})$ of the agent i leaving the process relative to the cost of the social optimum. More specifically, we will prove that, for any coalition I of size exactly c,

$$\sum_{i \in I} cost_i(\mathcal{S}) \leq \alpha H(c) cost(OPT). \tag{3.3}$$

Let $\mathcal{I}$ be the set of all coalitions of size exactly c. Summing (3.3) over all the $\binom{n}{c}$ coalitions $I \in \mathcal{I}$, we obtain $\sum_{I \in \mathcal{I}} \sum_{i \in I} cost_i(\mathcal{S}) \leq \alpha \binom{n}{c} H(c) cost(OPT)$. Any fixed agent i, $1 \leq i \leq n$, occurs in exactly $\binom{n-1}{c-1}$ coalitions $I \in \mathcal{I}$. Hence $\sum_{I \in \mathcal{I}} \sum_{i \in I} cost_i(\mathcal{S}) = \binom{n-1}{c-1} cost(\mathcal{S})$. We conclude that

$$cost(\mathcal{S}) \leq \binom{n}{c} / \binom{n-1}{c-1} \cdot \alpha H(c) cost(OPT) = \frac{n}{c} \cdot \alpha H(c) cost(OPT),$$

which establishes the stated price of anarchy.

Fix an arbitrary coalition I of size exactly c. We will prove (3.3). Let E^{OPT} be the set of edges bought by the social optimum and, for any $i \in I$, let E_i^{OPT} be a minimal set of edges necessary to connect the terminals of agent i within the optimal solution. We now start the process mentioned above. Let $I_1 := I$ be the initial coalition consisting of c agents. Suppose that we have already performed $k-1$ steps of the process, where initially $k = 1$, and let I_k be the coalition given at the beginning of the kth step, where $1 \leq k \leq c$. The kth step proceeds as follows. Starting from initial configuration $\mathcal{S}$, the agents of I_k perform a strategy change $\mathcal{S}_{I_k}^k$ in which $i \in I_k$

buys set E_i^{OPT}. Let $\mathcal{S}^k = (\mathcal{S}^k_{I_k}, \mathcal{S}_{-I_k})$ be the resulting configuration. The new cost of agent $i \in I_k$ is

$$cost_i(\mathcal{S}^k) = \sum_{e \in E_i^{OPT}} \frac{c(e)}{|\{j \in I_k : e \in E_j^{OPT}\} \cup \{j \notin I_k : e \in S_j\}|} \leq \sum_{e \in E_i^{OPT}} \frac{c(e)}{|\{j \in I_k : e \in E_j^{OPT}\}|}.$$

Since the original configuration $\mathcal{S}$ forms an α-approximate strong Nash equilibrium, the strategy change cannot improve the cost of every agent $i \in I_k$ by a factor of more than α. Thus there must exist an agent i_k with $cost_{i_k}(\mathcal{S}^k) \geq cost_{i_k}(\mathcal{S})/\alpha$ and hence

$$cost_{i_k}(\mathcal{S}) \leq \alpha \sum_{e \in E_{i_k}^{OPT}} \frac{c(e)}{|\{j \in I_k : e \in E_j^{OPT}\}|}. \tag{3.4}$$

This agent i_k leaves the coalition I_k. If there is more than one agent satisfying the above cost inequality, we select an arbitrary one of them. The new coalition at the end of the step is $I_{k+1} := I_k \setminus \{i_k\}$. The process ends after exactly c steps when the coalition is empty. Summing (3.4) over all of the c steps and taking into account that the sequence of agents leaving the process forms I, we find

$$\sum_{i \in I} cost_i(\mathcal{S}) \leq \alpha \sum_{k=1}^{c} \sum_{e \in E_{i_k}^{OPT}} \frac{c(e)}{|\{j \in I_k : e \in E_j^{OPT}\}|}. \tag{3.5}$$

We analyze the right-hand side of the above inequality, which sums edge costs $c(e)$ over edges $e \in E^{OPT}$. Consider any fixed edge $e \in E^{OPT}$, and let $n_e = |\{i \in I : e \in E_i^{OPT}\}|$ be the number of agents in I using e in the described strategy changes. The cost of e contributes to the right-hand side of (3.5) whenever one of the n_e agents leaves the process. The ℓth time this happens, the contribution is $c(e)/(n_e - \ell + 1)$ for $\ell = 1, \ldots, n_e$. Thus, the cost contribution is $c(e)H(n_e) \leq c(e)H(c)$, and we conclude that $\sum_{i \in I} cost_i(\mathcal{S}) \leq \alpha \sum_{e \in E^{OPT}} c(e)H(c) = \alpha H(c) cost(OPT)$. ∎

4. Lower bounds for unweighted games. We first present a lower bound for directed graphs. This lower bound implies that if there is no restriction on the coalition size, our upper bound of Corollary 3.4 is optimal.

THEOREM 4.1. *In directed graphs and for any $\alpha \geq 1$, the price of anarchy of α-approximate strong Nash equilibria is at least $\alpha \max\{n/c, H(n)\}$ if coalitions of size at most c are allowed.*

Proof. We modify lower bound graphs that were presented previously in the literature [2]. For the bound of $\alpha n/c$, consider a simple graph consisting of two vertices s and t that are connected by two parallel edges of cost αn and $c + \epsilon$, respectively; see Figure 4.1(a). Associated with the graph are n agents, all of which have to connect terminals s and t. An optimal solution will buy the edge of cost $c + \epsilon$. On the other hand, the configuration in which all of the n agents share the expensive edge of cost αn, each one paying a cost of α, represents an α-approximate strong Nash equilibrium: Any coalition of size up to c, when performing a strategy change and buying the edge of cost $c + \epsilon$, incurs a cost of at least $1 + \epsilon/c$ per agent. Hence the agents of the coalition do not save a factor of more than α in cost.

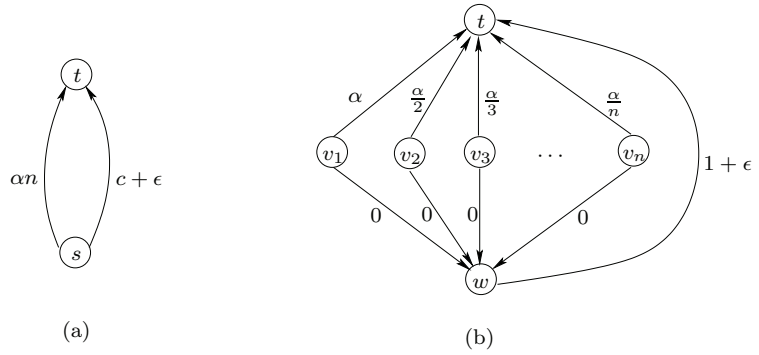


FIG. 4.1. *Directed graphs enforcing a high price of anarchy.*

In order to establish the lower bound of $\alpha H(n)$, we use the graph depicted in Figure 4.1(b). There are n vertices $v_1, \ldots, v_n$, where v_i is connected to a vertex t via a directed edge (v_i, t) of cost α/i and to a vertex w via a directed edge (v_i, w) of cost 0. Additionally, there is a directed edge (w, t) of cost $1 + \epsilon$. Associated with the graph are n agents, where agent i has to connect v_i to t. An optimal solution satisfies the connection requirements by buying the edges of cost 0 and the edge (w, t) of cost $1+\epsilon$. The configuration in which agent i connects v_i to t using its private edge (v_i, t) of cost α/i forms an α-approximate strong Nash equilibrium. Any coalition of size, say c, that performs a strategy change and purchases edge (w, t) incurs a cost of $(1 + \epsilon)/c$ per agent. However, there is at least one agent in the coalition whose original cost was at most α/c and to whom the incentive of changing is not sufficiently high. □

We next develop a lower bound for undirected networks. Our lower bound construction is quite involved, and we therefore concentrate on the most general scenario where there is no limit on the coalition size.

THEOREM 4.2. *For any $\alpha \geq 1$, there exists a family of undirected graphs, each admitting an α-approximate strong Nash equilibrium whose cost is $\Omega(\alpha\sqrt{\log n})$ times that of the social optimum.*

Proof. For ease of exposition we first prove the theorem for $\alpha = 1$ and then show how to adapt the proof for any $\alpha > 1$. In the following we first define our family of graphs. We then identify the social optimum and prove that there exists an expensive strong Nash equilibrium.

The graphs G are defined recursively. Let n be a positive integer such that $\lfloor\sqrt{\log n}\rfloor \geq 2$. In this proof logarithms are taken to the base 3. Let $d_{\max} = \lfloor\sqrt{\log n}\rfloor - 1$. The recursive construction proceeds in $d_{\max} + 1$ steps. At the bottom level of the recursion, i.e., at maximum depth $d_{\max}$, G consists of graphs $G^{d_{\max}}$ *of order* $d_{\max}$; cf. Figure 4.2(a). A graph $G^{d_{\max}}$ is composed of a *stem edge* $\{v, w\}$ of cost $s_{d_{\max}} = 1/3^{d_{\max}}$ and a *bridge* $\{u, v\}$ *of order* $d_{\max}$ having cost $b_{d_{\max}} = 2/3^{2d_{\max}}$. The bridge and the stem are joined at vertex v. Vertices u and w are connected via an *arc* $\{u, w\}$ *of order* $d_{\max}$ having cost $a_{d_{\max}} = 1/3^{d_{\max}}$. We call u the *base* and w the *tip* of $G^{d_{\max}}$. Associated with $G^{d_{\max}}$ are $n_{d_{\max}} = \lceil n/3^{d_{\max}(d_{\max}+1)}\rceil$ *agents of order* $d_{\max}$, each having to connect terminals u and w. By the choice of $d_{\max}$ we have $n_{d_{\max}} \geq 1$.

Assume that graphs of order $d_{\max}, d_{\max} - 1, \ldots, d + 1$ are defined. Then a graph G^d *of order* d, which resides at depth d of the recursion, is constructed as follows;

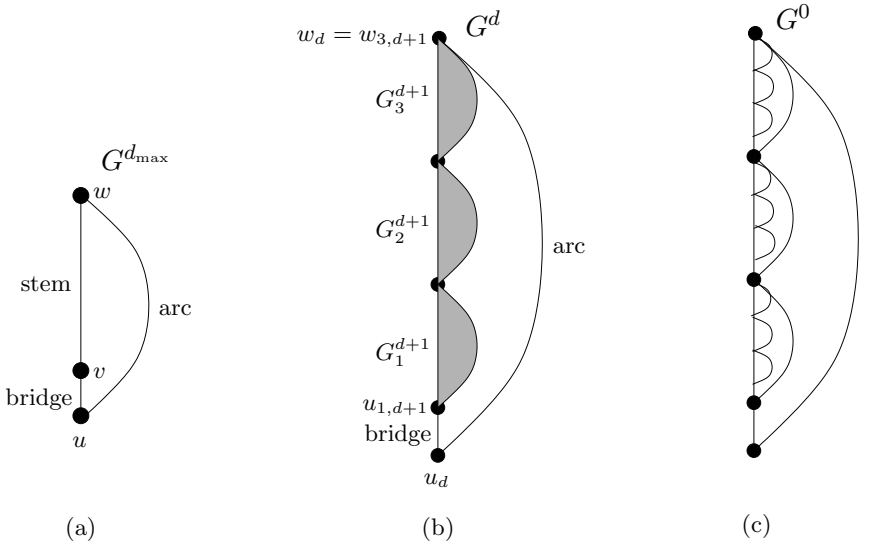


FIG. 4.2. *The recursive construction of graphs G^d.*

see Figure 4.2(b). Graph G^d consists of three graphs G_1^{d+1}, G_2^{d+1}, and G_3^{d+1} of order $d+1$ that are attached to each other. More specifically, the tip of G_1^{d+1} and the base of G_2^{d+1} are merged, i.e., the two vertices are united, and the tip of G_2^{d+1} is merged with the base of G_3^{d+1}. Let $u_{1,d+1}$ be the base of G_1^{d+1}. Attached to this vertex is a bridge $\{u_d, u_{1,d+1}\}$ of order d having a cost of $b_d = 2/3^{2d}$. Let $w_{3,d+1}$ be the tip of G_3^{d+1} and set $w_d := w_{3,d+1}$. We call u_d the *base* and w_d the tip of G^d. Additionally, G^d contains an arc $\{u_d, w_d\}$ of order d connecting the base and the tip. This arc has cost $a_d = 1/3^d$. Associated with G^d are $n_d = \lceil n/3^{d(d+1)} \rceil - 3\lceil n/3^{(d+1)(d+2)} \rceil$ agents of order d, all of which have to connect u_d to w_d. As we shall see, these agents will govern the connection decisions within G^d. The bridge will have the effect that, in a strong Nash equilibrium, the order-d agents will establish their connections using the arc of order d instead of routing through the graphs G_k^{d+1}, $1 \le k \le 3$.

The construction proceeds down to a depth $d = 0$. Associated with graph G^0 are $n_0 = \lceil n/3^0 \rceil - 3\lceil n/3^2 \rceil = n - 3\lceil n/3^2 \rceil$ agents of order 0 that have to connect the outermost vertices of G^0. Graph $G := G^0$ is the graph we will work with. A high level sketch of $G = G^0$ is given in Figure 4.2(c).

We start with some observations on G. First, all the vertices and terminals of the graph are located on a *backbone* consisting of all the stem edges and bridges. The nested structure of G contains 3^d subgraphs of order d for any $0 \le d \le d_{\max}$.

PROPOSITION 4.3. *The least expensive path connecting the base and the tip of a graph G^d using only edges of G^d has a total edge cost of exactly $1/3^d$ for any $0 \le d \le d_{\max}$.*

Proof. The statement holds for $d = d_{\max}$ as the arc of $G^{d_{\max}}$ has cost $a_{d_{\max}} = 1/3^{d_{\max}}$, while the path crossing the bridge has cost $b_{d_{\max}} + s_{d_{\max}} = 2/3^{2d_{\max}} + 1/3^{d_{\max}}$. Suppose that the statement holds for depths $d_{\max}, \ldots, d+1$. In G^d the arc of order d has cost $a_d = 1/3^d$, while, using the inductive hypothesis, any path using the bridge of order d has a cost of at least $b_d + 3 \cdot 1/3^{d+1} = 2/3^{2d} + 1/3^d > 1/3^d$. ∎

The total number of agents associated with G and all of its subgraphs is equal to

$$\begin{aligned} N_0 &= \sum_{d=0}^{d_{\max}-1} 3^d(\lceil n/3^{d(d+1)}\rceil - 3\lceil n/3^{(d+1)(d+2)}\rceil) + 3^{d_{\max}}\lceil n/3^{d_{\max}(d_{\max}+1)}\rceil \\ &= n - 3^{d_{\max}}\lceil n/3^{d_{\max}(d_{\max}+1)}\rceil + 3^{d_{\max}}\lceil n/3^{d_{\max}(d_{\max}+1)}\rceil \\ &= n. \end{aligned}$$

More generally, in G the total number of agents associated with all the order-d graphs G^d and the subgraphs therein is, for any d with $0 \le d \le d_{\max}$,

$$\begin{aligned} N_d &= \sum_{i=d}^{d_{\max}-1} 3^i(\lceil n/3^{i(i+1)}\rceil - 3\lceil n/3^{(i+1)(i+2)}\rceil) + 3^{d_{\max}}\lceil n/3^{d_{\max}(d_{\max}+1)}\rceil \\ &= 3^d\lceil n/3^{d(d+1)}\rceil, \end{aligned}$$

which is equal to $n/3^{d^2}$ when ignoring ceilings.

The social optimum in G buys the backbone of the graph. As there are $3^{d_{\max}}$ graphs of order $d_{\max}$, the total cost of the stem edges is $3^{d_{\max}}s_{d_{\max}} = 3^{d_{\max}} \cdot 1/3^{d_{\max}} = 1$. There are 3^d graphs of order d, $0 \le d \le d_{\max}$, and hence the total cost of order-d bridges is $3^d b_d = 3^d \cdot 2/3^{2d} = 2/3^d$. Summing over all d we find that the total cost of the bridges is $\sum_{d=0}^{d_{\max}} 2/3^d \le 3$. We conclude that the cost of the social optimum is bounded by 4.

Consider the configuration $\mathcal{S}$ in which, for any graph G^d within G, any order-d agent associated with this graph G^d establishes its required connection via the corresponding arc of order d. That is, $\mathcal{S}$ buys all the arcs but no bridges or stem edges. As we will show in the remainder of this proof, $\mathcal{S}$ forms a strong Nash equilibrium. We evaluate the cost of $\mathcal{S}$. As there are 3^d graphs of order d, the total cost of order-d arcs is $3^d a_d = 3^d \cdot 1/3^d = 1$ for any fixed d with $0 \le d \le d_{\max}$. Summing over all d, we obtain $cost(\mathcal{S}) = d_{\max} + 1 \ge \lfloor\sqrt{\log n}\rfloor$, and this establishes the desired performance ratio.

It remains to show that $\mathcal{S}$ is indeed a strong Nash equilibrium. To this end we have to show that no coalition I of agents has an *improvement move*. We will always consider nonempty coalitions. An improvement move, for a coalition I, is a strategy change $\mathcal{S}'_I$ such that $cost_i(\mathcal{S}'_I, \mathcal{S}_{-I}) < cost_i(\mathcal{S})$ for any agent $i \in I$. In our graph G, as all the agents have to connect pairs of terminals, a strategy of an agent is a simple path connecting the desired vertices. The property that there does not exist an improvement move follows from Lemma 4.4, which we prove below.

LEMMA 4.4. *For $d = 0, \ldots, d_{\max}$, no coalition involving agents of order d has an improvement move.*

For the proof of Lemma 4.4 we need Lemma 4.5, which we prove first.

LEMMA 4.5. *Consider a fixed d, $0 \le d \le d_{\max}$, and suppose that no coalition involving agents of order smaller than d has an improvement move. Furthermore, assume that no coalition I involving agents of order d has an improvement move in which an order-d agent $i \in I$ associated with a graph $G^d(i)$ chooses a path containing edges outside $G^d(i)$. Then no coalition involving agents of order d has an improvement move.*

Proof of Lemma 4.5. Let I be a coalition that involves agents of order d. We have to show that I has no improvement move. Based on the assumptions of the lemma, we can restrict ourselves to coalitions I that do not contain agents of order smaller

than d. Furthermore, based on the assumptions, we only have to consider strategy changes where each order-d agent $i \in I$ establishes the required connection within its graph $G^d(i)$. Let $I' \subseteq I$ be any maximal subcoalition of order-d agents that are associated with the same graph $G^d(I')$. We will show that any strategy change $\mathcal{S}'_{I'}$ that consists of choosing connection paths within $G^d(I')$ leads to a strictly higher cost for that subcoalition; i.e., at least one agent $i \in I'$ has a strictly higher cost, and the strategy change is no improvement move.

Graph $G^d(I')$ has n_d order-d agents associated with it. Let f be the fraction defecting, i.e., $f = |I'|/n_d$. In the original configuration $\mathcal{S}$, when routing through the arc of order d, subcoalition I' paid a cost of $fa_d = f/3^d$. When changing strategy and choosing a different connection route within $G^d(I')$, each $i \in I'$ selects a path P_i that crosses the bridge of order d and then, if $d = d_{\max}$, traverses the stem edge of $G^{d_{\max}}(I')$ (see Figure 4.2(a)). If $d < d_{\max}$, path P_i then traverses the order-$(d+1)$ graphs $G_k^{d+1}(I')$, $1 \le k \le 3$, located within $G^d(I')$ (see Figure 4.2(b)). If $d = d_{\max}$, then the total cost of edges on P_i is $b_{d_{\max}} + s_{d_{\max}} \ge (1 + 2/3^d)/3^d$. If $d < d_{\max}$, then the total cost is at least $b_d + 3 \cdot 1/3^{d+1} \ge (1 + 2/3^d)/3^d$ because, by Proposition 4.3, the least expensive path traversing an order-$(d+1)$ graph has cost $1/3^{d+1}$. In both cases we have the same lower bound on the cost, expressed in terms of d. The cost of P_i is not shared by agents of order smaller than d, as they are not part of the original coalition I, nor is the cost shared by order-d agents associated with other graphs $G^d \neq G^d(I')$. The cost of P_i can be shared only by agents of order larger than d, and there exist N_{d+1} such agents if $d < d_{\max}$. If $d = d_{\max}$, the cost is not shared by other agents.

If $d = d_{\max}$, we are done because the new cost of I' is $(1 + 2/3^d)/3^d$, while the original cost was $f/3^d \le 1/3^d$. If $d < d_{\max}$, then at best all the N_{d+1} agents of order larger than d support the edges traversed by I', and the new cost of I' is at least $cost'_{I'} \ge \frac{fn_d}{fn_d + N_{d+1}}(1 + \frac{2}{3^d})\frac{1}{3^d}$. We need to show that $cost'_{I'}$ is higher than the original cost of $f/3^d$, which is equivalent to proving $\frac{n_d}{fn_d + N_{d+1}}(1 + \frac{2}{3^d}) > 1$. Since $0 < f \le 1$ it suffices to show that

$$\frac{n_d}{n_d + N_{d+1}}\left(1 + \frac{2}{3^d}\right) > 1.$$

Using the definition of n_d, eliminating ceilings, we find

$$\begin{aligned} n_d &> n/3^{d(d+1)} - n/3^{(d+1)^2+d} - 3 = \frac{n3^d}{3^{(d+1)^2}}\left(3 - \frac{1}{3^{2d}} - \frac{3^{d^2+d+2}}{n}\right) \\ &\ge \frac{n3^d}{3^{(d+1)^2}}\left(2 - \frac{3^{(d+1)(d+2)}}{n}\right) \\ &\ge \frac{n3^d}{3^{(d+1)^2}}. \end{aligned}$$

The second inequality holds because $1/3^{2d} \le 1$. For the third inequality note that $d < d_{\max}$ and $d_{\max} = \lfloor\sqrt{\log n}\rfloor - 1$ imply $(d+1)(d+2) \le \log n$ and hence $3^{(d+1)(d+2)} \le n$. Moreover, we have $N_{d+1} < 2n/3^{(d+1)^2}$. We conclude that

$$\frac{n_d}{n_d + N_{d+1}}\left(1 + \frac{2}{3^d}\right) > \frac{3^d}{3^d + 2}\left(1 + \frac{2}{3^d}\right) = \frac{3^d}{3^d + 2} \cdot \frac{3^d + 2}{3^d} = 1. \qquad \square$$

Proof of Lemma 4.4. We prove the lemma inductively for increasing values of d. For $d = 0$, the statement follows immediately from Lemma 4.5 as the assumptions of

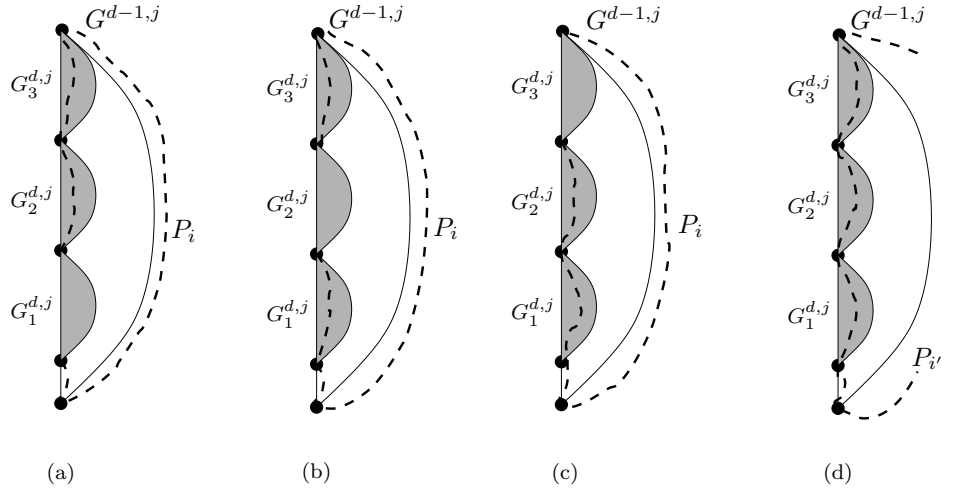


FIG. 4.3. *The paths traversed after strategy change.*

that lemma are trivially satisfied: There are no agents of order smaller than 0, and an agent of order 0 cannot connect its terminals using edges outside G^0. Suppose that the statement of Lemma 4.4 holds for depth $0, \ldots, d-1$. We prove that no coalition I involving order-d agents has an improvement move in which an order-d agent $i \in I$ associated with a graph $G^d(i)$ chooses a path using edges outside $G^d(i)$. The inductive step then follows from Lemma 4.5.

So consider a coalition I involving agents of order d and a corresponding strategy change $\mathcal{S}'_I$ in which at least one order-d agent chooses edges outside its order-d graph to connect the desired terminals. We show that the strategy change is not an improvement move. By the inductive hypothesis, we can restrict ourselves to coalitions I not involving agents of order smaller than d. Thus I contains only agents of order d or larger. Let $I' \subseteq I$ be the maximum subcoalition of order-d agents i choosing connection paths outside their graph $G^d(i)$. As $d \geq 1$, each such graph belongs to a graph G^{d-1} in the nested structure of G^0. Consider all graphs of order $d-1$ containing at least one $G^d(i)$, $i \in I'$, and number these order-$(d-1)$ graphs in an arbitrary way. Let J be the resulting index set. Each graph $G^{d-1,j}$, $j \in J$, contains three graphs $G_1^{d,j}, G_2^{d,j}, G_3^{d,j}$ of order d. For $k = 1, 2, 3$, let f_k^j be the fraction of the order-d agents associated with $G_k^{d,j}$ that are members of I'; i.e., $f_k^j = |\{i \in I' : i \text{ is order-}d \text{ agent associated with } G_k^{d,j}\}|/n_d$. Recall that n_d is the number of agents associated with an order-d graph. We have $0 \leq f_k^j \leq 1$ and $f_1^j + f_2^j + f_3^j > 0$.

In the original configuration $\mathcal{S}$, coalition I' buys the order-d arcs and hence pays a total cost of $cost_{I'} = \sum_{j \in J} \sum_{k=1}^{3} f_k^j a_d$. This expression holds even if some of the f_k^j are zero. We show in the following that the new cost $cost'_{I'}$ of I' is strictly higher than $cost_{I'}$. Hence at least one agent in I' does not improve its cost, and the strategy change is not an improvement move.

In order to estimate $cost'_{I'}$ consider an agent $i \in I'$, and let $G^{d-1,j}$, with $j \in J$, be the graph where i's graph $G^d(i)$ is located. First suppose that $G^d(i) = G_1^{d,j}$; cf. Figure 4.3(a). After the strategy change, i connects the base and the tip of $G_1^{d,j}$ on a path P_i that uses edges outside $G_1^{d,j}$. Since strategies are simple paths, all the edges of P_i are

outside $G_1^{d,j}$. Starting at the base of $G_1^{d,j}$, path P_i has to traverse the bridge of order $d-1$ in $G^{d-1,j}$. To reach the tip of $G_1^{d,j}$, path P_i has to travel to the tip of $G^{d-1,j}$. This can be done using the arc of order $d-1$ or using another subpath outside $G^{d-1,j}$. After having reached the tip of $G^{d-1,j}$, path P_i traverses $G_3^{d,j}$ and $G_2^{d,j}$, reaching the desired terminal. Ignoring edges visited between the base and the tip of $G^{d-1,j}$, agent i has to pay a share for the order-$(d-1)$ bridge and for the subpaths of P_i within $G_2^{d,j}$ and $G_3^{d,j}$. The total cost of edges traversed within graph $G_k^{d,j}$, $k \in \{2,3\}$, is at least $1/3^d$ by Proposition 4.3. The cost may be shared with other agents.

If $G^d(i) = G_2^{d,j}$ or $G^d(i) = G_3^{d,j}$, the situation is similar; see Figures 4.3(b) and (c), respectively. In the first case, P_i has to traverse $G_1^{d,j}$ and the order-$(d-1)$ bridge. From there it has to travel to the tip of $G^{d-1,j}$ and pass through $G_3^{d,j}$. Edges used within a graph $G_k^{d,j}$, $k \in \{1,3\}$, have a total cost of at least $1/3^d$; cost sharing may occur. If $G^d(i) = G_3^{d,j}$, path P_i passes through $G_2^{d,j}$ and $G_1^{d,j}$. After traversing the bridge of order $d-1$ it connects to the tip of $G^{d-1,j}$, which is also the tip of $G_3^{d,j}$ representing the desired terminal.

Next let C^j be the total number of agents $i' \in I'$ that are *not* associated with $G_1^{d,j}$, $G_2^{d,j}$, or $G_3^{d,j}$, but choose edges of these graphs when performing the strategy change. In order to use such edges, the new path $P_{i'}$ of i' must pass through the base and the tip of $G^{d-1,j}$; see Figure 4.3(d). The path between these two vertices crosses the bridge of order $d-1$ and must consist of subpaths within $G_k^{d,j}$, $k = 1,2,3$.

We are ready to lower bound the new cost $cost'_{I'}$ of I'. To this end we will consider only the cost spent in graphs $G_k^{d,j}$, $1 \le k \le 3$ and $j \in J$, and on order-$(d-1)$ bridges in $G^{d-1,j}$. Fix a $j \in J$. Graph $G_1^{d,j}$ is traversed by exactly $(f_2^j + f_3^j)n_d + C^j$ agents from I', each using edges of cost at least $1/3^d$. The cost is shared by at most $(f_2^j + f_3^j)n_d + C^j + (1 - f_1^j)n_d + N_{d+1}$ agents. Here $(1 - f_1^j)n_d$ is the number of order-d agents associated with $G_1^{d,j}$ that establish their connection within this graph, and N_{d+1} is the total number of agents of order larger than d that may participate in the strategy change and reside in the entire coalition I. If $d = d_{\max}$, then we set $N_{d+1} = 0$. Thus subcoalition I' spends a cost of at least

$$\frac{1}{3^d} \cdot \frac{(f_2^j + f_3^j)n_d + C^j}{(1 - f_1^j + f_2^j + f_3^j)n_d + N_{d+1} + C^j}$$

in graph $G_1^{d,j}$. Similarly, in $G_2^{d,j}$ and $G_3^{d,j}$ the costs are

$$\frac{1}{3^d} \cdot \frac{(f_1^j + f_3^j)n_d + C^j}{(1 - f_2^j + f_1^j + f_3^j)n_d + N_{d+1} + C^j} \quad \text{and} \quad \frac{1}{3^d} \cdot \frac{(f_1^j + f_2^j)n_d + C^j}{(1 - f_3^j + f_1^j + f_2^j)n_d + N_{d+1} + C^j}.$$

Finally the bridge of order $d-1$ has cost $b_{d-1} = 2/3^{2(d-1)}$, which is shared by $(f_1^j + f_2^j + f_3^j)n_d + N_{d+1} + C^j$ agents, and I' incurs a cost of $\frac{2}{3^{2(d-1)}}((f_1^j + f_2^j + f_3^j)n_d + C^j)/((f_1^j + f_2^j + f_3^j)n_d + N_{d+1} + C^j)$.

Note that for any $k \in \{1,2,3\}$ the other two indices from that set can be expressed as $k' = k \bmod 3 + 1$ and $k'' = (k+1) \bmod 3 + 1$. We conclude that

$$cost'_{I'} \ge \sum_{j \in J} \left(\sum_{k=1}^{3} \frac{(f_{k'}^j + f_{k''}^j)n_d + C^j}{(1 - f_k^j + f_{k'}^j + f_{k''}^j)n_d + N_{d+1} + C^j} \cdot \frac{1}{3^d} + \frac{(f_1^j + f_2^j + f_3^j)n_d + C^j}{(f_1^j + f_2^j + f_3^j)n_d + N_{d+1} + C^j} \cdot \frac{2}{3^{2(d-1)}} \right).$$

Ratios of the form $(x+c)/(y+c)$ are increasing in c if $x \le y$. Hence we can drop the terms C^j and obtain

$$cost'_{I'} \ge \sum_{j\in J}\left(\sum_{k=1}^{3}\frac{(f^j_{k'}+f^j_{k''})n_d}{(1-f^j_k+f^j_{k'}+f^j_{k''})n_d+N_{d+1}}\cdot\frac{1}{3^d} + \frac{(f^j_1+f^j_2+f^j_3)n_d}{(f^j_1+f^j_2+f^j_3)n_d+N_{d+1}}\cdot\frac{2}{3^{2(d-1)}}\right).$$

Reordering the expression in the brackets, by focusing on one particular f^j_k in the numerators, we find

$$cost'_{I'} \ge \sum_{j\in J}\sum_{k=1}^{3}\left(\left(\frac{f^j_k n_d}{(1+f^j_k-f^j_{k'}+f^j_{k''})n_d+N_{d+1}} + \frac{f^j_k n_d}{(1+f^j_k+f^j_{k'}-f^j_{k''})n_d+N_{d+1}}\right)\cdot\frac{1}{3^d} + \frac{f^j_k n_d}{(f^j_1+f^j_2+f^j_3)n_d+N_{d+1}}\cdot\frac{2}{3^{2(d-1)}}\right).$$

To simplify the last expression we observe that for any real values x, y, and c inequality $\frac{1}{x-y+c}+\frac{1}{x+y+c}\ge\frac{2}{x+c}$ holds, which we apply for $x=f^j_k n_d$ and $y=(f^j_{k'}-f^j_{k''})n_d$ as well as $c=n_d+N_{d+1}$. Thus the two terms in the inner bracket sum to at least $\frac{2f^j_k n_d}{(1+f^j_k)n_d+N_{d+1}}$. Furthermore, as for the last term in the above expression,

$$\frac{f^j_k n_d}{(f^j_1+f^j_2+f^j_3)n_d+N_{d+1}} \ge \frac{f^j_k n_d}{(f^j_k+2)n_d+N_{d+1}} \ge \frac{1}{2}\cdot\frac{f^j_k n_d}{(1+f^j_k)(n_d+N_{d+1})}.$$

Hence

$$cost'_{I'} \ge \sum_{j\in J}\sum_{k=1}^{3}\left(\frac{2f^j_k n_d}{(1+f^j_k)n_d+N_{d+1}}\cdot\frac{1}{3^d}+\frac{f^j_k n_d}{(1+f^j_k)(n_d+N_{d+1})}\cdot\frac{1}{3^{2(d-1)}}\right) > \sum_{j\in J}\sum_{k=1}^{3}\frac{2f^j_k n_d}{(1+f^j_k)(n_d+N_{d+1})}\left(1+\frac{2}{3^d}\right)\frac{1}{3^d}.$$

The last line follows because $\frac{1}{3^{2(d-1)}}\ge 2\frac{2}{3^d}\frac{1}{3^d}$.

As shown at the end of the proof of Lemma 4.5, $\frac{n_d}{n_d+N_{d+1}}(1+\frac{2}{3^d})>1$ if $d<d_{\max}$. If $d=d_{\max}$, then $N_{d+1}=0$ and the inequality is also satisfied. In each case

$$cost'_{I'} > \sum_{j\in J}\sum_{k=1}^{3}\frac{2f^j_k}{1+f^j_k}\frac{1}{3^d} \ge \sum_{j\in J}\sum_{k=1}^{3} f^j_k\frac{1}{3^d},$$

and the new cost of I' is strictly larger than the original cost of I' in configuration $\mathcal{S}$. □

This completes the proof of Theorem 4.2 for $\alpha=1$. Finally, we show how to adapt the proof for any $\alpha>1$. In the construction of the graphs G^d only the costs of

the arcs change. An arc of order d now has cost αa_d. All other costs remain the same. This increases the cost of configuration $\mathcal{S}$ by a factor of α, i.e., $cost(\mathcal{S}) \geq \alpha\lfloor\sqrt{\log n}\rfloor$, while the cost of the social optimum remains the same. This establishes a performance ratio of $\Omega(\alpha\sqrt{\log n})$.

In the statements of Lemmas 4.4 and 4.5, the term "improvement move" has to be replaced by "α-improvement move." An α-improvement move, for a coalition I, is a strategy change $\mathcal{S}'_I$ such that $cost_i(\mathcal{S}'_I, \mathcal{S}_{-I}) < cost_i(\mathcal{S})/\alpha$ for any agent $i \in I$. In the proof of Lemma 4.5 we considered any coalition I involving agents of order d or larger and investigated strategy changes where order-d agents establish connections with their respective graph of order d. We identified a subcoalition I', with $f = |I'|/n_d$, incurring a new cost of $cost'_{I'} > f/3^d$. This cost inequality still holds in our modified graph as edge costs did not decrease. Since $cost'_{I'} > f/3^d = (\alpha f/3^d)/\alpha$ and $\alpha f/3^d$ is the original cost of I' in the scaled graph, the strategy change is not an α-improvement move.

In the proof of Lemma 4.4 we studied coalitions I involving agents of order d or larger. We analyzed strategy changes in which order-d agents buy edges outside their graph of order d and identified a subcoalition I' of order-d agents incurring a new total cost of $cost'_{I'} > \sum_{j\in J}\sum_{k=1}^{3} f_k^j/3^d$, where $G_k^{d,j}$ were the graphs the agents $i \in I'$ are associated with. Again, when arcs are scaled by a factor of α, this cost inequality still holds. As the original cost of I' in the scaled graph is $\sum_{j\in J}\sum_{k=1}^{3} \alpha f_k^j/3^d$, the strategy change is not an α-improvement move. ∎

5. Weighted games. In this section we study weighted network design games where each agent i has a positive weight w_i. We scale the weights such that the minimum weight is equal to 1, and hence $w_i \geq 1$ for all agents. Let $W = \sum_{i=1}^{n} w_i$ be the total weight of all the agents.

If agents are allowed to coordinate their strategies, two scenarios are of interest. In a first setting we assume that coalitions of size up to c are allowed for any $1 \leq c \leq n$. In this case let W^c be the maximum total weight of any coalition having size at most c. In a second setting we assume that the total weight of a coalition is upper bounded so that agents of high weight cannot impose too much control on agents of low weight. In this case let $W_{\max}$ be the maximum total weight any coalition may have.

We extend our results shown for unweighted games.

5.1. Upper bounds. We first give a sufficient condition for the existence of strong Nash equilibria in weighted games and evaluate their performance in terms of the price of anarchy.

THEOREM 5.1. *In any directed or undirected graph α-approximate strong Nash equilibria exist for any $\alpha \geq 1 + \ln(1 + \overline{W})$. Here $\overline{W} = W^c$ if coalitions of size up to c are allowed, and $\overline{W} = W_{\max}$ if coalitions of weight up to $W_{\max}$ are allowed.*

Proof. We again use potential function arguments to show the existence of α-approximate strong Nash equilibria but have to work with a more general potential function, compared to that used in unweighted games. Given a graph $G = (V, E, c)$ and a configuration $\mathcal{S} = (S_1, \ldots, S_n)$, let $E_{\mathcal{S}} = \cup_{i=1}^{n} S_i$ be the union of all edges used by the agents. For any $e \in E_{\mathcal{S}}$, let $W_e = \sum_{i:e\in S_i} w_i$ be the total weight of the agents currently using e in their strategies. Define

$$\Phi(\mathcal{S}) = \sum_{e\in E_{\mathcal{S}}} c(e)(1 + \ln W_e).$$

We show that while $\mathcal{S}$ does not form an α-approximate strong Nash equilibrium, an α-improvement move of a coalition I strictly decreases the potential. This ensures

that a sequence of improvement moves starting from the social optimum will converge because, at any time, $0 \leq \Phi \leq (1 + \ln W)cost(OPT)$.

Consider an α-improvement move of a coalition I of agents. Again, we view the move as being performed in two steps: (1) Agents $i \in I$ first drop all the edges of their strategies S_i. Let E_1 be this set of edges. (2) Agents $i \in I$ buy the edges they wish to have in their new strategies. Let E_2 be the set of edges involved. In the following, let $cost^-$ be the absolute value of the cost reduction experienced by I due to step (1). Note that $cost^-$ is equal to the cost of I in configuration $\mathcal{S}$. Let Φ^- be the absolute value of the potential drop. Similarly, let $cost^+$ be the value of the cost increase of I in step (2), and let Φ^+ be the corresponding potential increase. The value of $cost^+$ is equal to the cost of I in the new configuration after their strategy change. Using the definition of an α-improvement move, we find $\alpha cost^+ - cost^- < 0$. It remains to show that $cost^- \leq \Phi^-$ and $\Phi^+ \leq \alpha cost^+$, which implies $\Delta\Phi = -\Phi^- + \Phi^+ < 0$.

For any edge $e \in E$, let W_e^1 be the total weight of agents sharing e after step (1). The cost reduction experienced by I due to edge $e \in E_1$ is $cost_e^- = c(e)(W_e - W_e^1)/W_e$. For any $e \in E_1$, let Φ_e^- denote the potential drop caused by this edge. If $W_e^1 = 0$, then $cost_e^- = c(e) \leq c(e)(1 + \ln W_e) = \Phi_e^-$. If $W_e^1 > 0$, then $W_e^1 \geq 1$ and

$$cost_e^- = c(e)\frac{W_e - W_e^1}{W_e} \leq c(e)\int_{W_e^1}^{W_e} \frac{1}{z}dz = c(e)(\ln W_e - \ln W_e^1) = \Phi_e^-.$$

We conclude that $cost^- = \sum_{e \in E_1} cost_e^- \leq \sum_{e \in E_1} \Phi_e^- = \Phi^-$.

For any $e \in E_2$ let W_e^2 be the total weight of agents sharing e after step (2). The cost increase experienced by I due to edge $e \in E_2$ is $cost_e^+ = c(e)(W_e^2 - W_e^1)/W_e^2$ because agents in I purchasing e have a total weight of $W_e^2 - W_e^1$. Let Φ_e^+ be the potential increase caused by $e \in E_2$. If $W_e^1 = 0$, then $\Phi_e^+ = c(e)(1 + \ln W_e^2) \leq c(e)(1 + \ln(1 + \overline{W})) \leq \alpha cost_e^+$. If $W_e^1 > 0$, then $\Phi_e^+ = c(e)(\ln W_e^2 - \ln W_e^1) = c(e)\ln(W_e^2/W_e^1)$. To establish $\Phi_e^+ \leq \alpha cost_e^+$, we prove that

$$f(W_e^2) = \ln(W_e^2/W_e^1) - (1 + \ln(1 + \overline{W}))\frac{W_e^2 - W_e^1}{W_e^2}$$

is upper bounded by 0 for all $W_e^2 \geq W_e^1$. This implies, as desired, $\Phi^+ = \sum_{e \in E_2} \Phi_e^+ \leq \sum_{e \in E_2} \alpha cost_e^+ = \alpha cost^+$, because $\alpha \geq 1 + \ln(1 + \overline{W})$. Computing the first derivative of f, we find that f is decreasing for values of W_e^2 between W_e^1 and $(1+\ln(1+\overline{W}))W_e^1$ and increasing for larger values. Since $f(W_e^1) = 0$, we obtain that f is upper bounded by 0 for any W_e^2 with $W_e^1 \leq W_e^2 \leq (1 + \ln(1 + \overline{W}))W_e^1$. If $W_e^2 > (1 + \ln(1 + \overline{W}))W_e^1$, then $W_e^1 < W_e^2/(1 + \ln(1 + \overline{W}))$ and

$$(1 + \ln(1 + \overline{W}))(W_e^2 - W_e^1)/W_e^2 \geq \ln(1 + \overline{W}).$$

Hence

$$\begin{aligned} f(W_e^2) &\leq \ln(W_e^2/W_e^1) - \ln(1 + \overline{W}) \\ &= \ln\left(1 + \frac{W_e^2 - W_e^1}{W_e^1}\right) - \ln(1 + \overline{W}) \\ &\leq \ln(1 + \overline{W}) - \ln(1 + \overline{W}) = 0. \end{aligned}$$

The last inequality follows because $\frac{W_e^2 - W_e^1}{W_e^1} \leq W_e^2 - W_e^1$, since $W_e^1 \geq 1$, and $W_e^2 - W_e^1 \leq \overline{W}$, since agents of total weight at most $\overline{W}$ can join any edge e. ∎

THEOREM 5.2. *In any directed or undirected graph and for any $\alpha \geq 1$, the price of anarchy of α-approximate strong Nash equilibria is upper bounded by $\frac{\alpha n}{c}(1+\ln W^c)$ if coalitions of size up to c are allowed. The price of anarchy is upper bounded by $\frac{2\alpha W}{W_{\max}}(1+\ln W_{\max})$ if coalitions of weight up to $W_{\max}$ are allowed.*

If there are no restrictions on the coalitions being formed and for $\alpha = 1$, we obtain the following corollary.

COROLLARY 5.3. *In any directed or undirected graph the price of anarchy of strong Nash equilibria is upper bounded by $1+\ln W$.*

Proof of Theorem 5.2. We generalize the proof of Theorem 3.3. Given an α-approximate strong Nash equilibrium $\mathcal{S} = (S_1, \ldots, S_n)$, we consider a coalition I of legal size or weight and show

$$\sum_{i \in I} cost_i(\mathcal{S}) \leq \alpha(1+\ln W_I) cost_i(OPT), \tag{5.1}$$

where W_I it the total weight of agents $i \in I$. If coalitions of size up to c are allowed, inequality (5.1) gives $\sum_{i\in I} cost_i(\mathcal{S}) \leq \alpha(1+\ln W^c) cost(OPT)$. Summing this inequality over all the $\binom{n}{c}$ coalitions of size exactly c, we obtain $cost(\mathcal{S}) \leq \frac{\alpha n}{c}(1+\ln W^c) cost(OPT)$. If coalitions of weight up to $W_{\max}$ are allowed, we partition the n agents into maximal possible coalitions of admissible weight. This partitioning consists of at most $2W/W_{\max}$ coalitions because only one of these coalitions can have a total weight of at most $W_{\max}/2$ and the total weight of any two coalitions is larger than $W_{\max}$. For each coalition of the partitioning we sum up (5.1) and obtain $cost(\mathcal{S}) \leq \frac{2\alpha W}{W_{\max}}(1+\ln W_{\max}) cost(OPT)$.

In order to establish (5.1), for any fixed coalition I, we perform the same process as in the proof of Theorem 3.3, where subcoalitions of I change strategy and purchase the edge set E^{OPT} of the social optimum. For any $i \in I$, let $E_i^{OPT} \subseteq E^{OPT}$ be a minimal edge set necessary to connect the terminals of agent i in the optimal solution. The process starts with $I_1 := I$. In the kth step, for $k = 1, \ldots, |I|$, agents i in the remaining subcoalition I_k change strategies and connect their terminals using E_i^{OPT}. Since the original configuration $\mathcal{S}$ is an α-approximate strong Nash equilibrium, there must exist one agent i_k whose cost in the original configuration $\mathcal{S}$ is bounded by

$$cost_{i_k}(\mathcal{S}) \leq \alpha \sum_{e \in E_{i_k}^{OPT}} c(e) \frac{w_{i_k}}{W_{I_k}^e}, \tag{5.2}$$

where $W_{I_k}^e$ is the total weight of agents sharing e, i.e., $W_{I_k}^e = \sum_{i \in I_k^e} w_i$ with $I_k^e = \{i : i \in I_k \text{ and } e \in E_i^{OPT}\}$. This agent i_k leaves the process, and $I_{k+1} := I_k \setminus \{i_k\}$. Summing (5.2) over all the $|I|$ steps, we obtain

$$\sum_{i \in I} cost_i(\mathcal{S}) \leq \alpha \sum_{k=1}^{|I|} \sum_{e \in E_{i_k}^{OPT}} c(e) \frac{w_{i_k}}{W_{I_k}^e}. \tag{5.3}$$

We estimate the contribution of $c(e)$, for a fixed edge $e \in E^{OPT}$, in the right-hand side expression of (5.3). A contribution to the sum occurs whenever an agent $i \in I$ with $e \in E_i^{OPT}$ leaves the process. Let $i_1, \ldots, i_\ell \in I$ be the agents using e, i.e., $e \in E_{i_j}^{OPT}$ for $j = 1, \ldots, \ell$, and assume that these agents are numbered according to the time when they leave the process of strategy changes. For $j = 1, \ldots, \ell$, let $s_j = w_{i_j} + \cdots + w_{i_\ell}$

be the suffix sum of these agents' weights. Then edge e contributes a total of

$$\begin{aligned}
&c(e)\left(\frac{w_{i_1}}{w_{i_1}+\cdots+w_{i_\ell}}+\frac{w_{i_2}}{w_{i_2}+\cdots+w_{i_\ell}}+\cdots+\frac{w_{i_\ell}}{w_{i_\ell}}\right)\\
&\le c(e)\left(\sum_{j=1}^{\ell-1}\int_{s_{j+1}}^{s_j}\frac{1}{z}dz+1\right)\le c(e)\left(1+\int_{s_\ell}^{s_1}\frac{1}{z}dz\right)\\
&\le c(e)\left(1+\int_{1}^{s_1}\frac{1}{z}dz\right)=c(e)(1+\ln s_1)\\
&\le c(e)(1+\ln W_I).
\end{aligned}$$

The third inequality holds because $s_\ell = w_\ell \ge 1$. Summing this cost estimate over all edges $e \in E^{OPT}$, we obtain the desired bound on $\sum_{i\in I} cost_I(\mathcal{S})$. ∎

5.2. Lower bounds. We develop lower bounds on the performance of strong Nash equilibria in directed and undirected graphs.

THEOREM 5.4. *In directed graphs the price of anarchy of α-approximate strong Nash equilibria is at least $\Omega(\alpha \max\{n/c, \log W\})$ if coalitions of size at most c are allowed, and at least $\Omega(\alpha \max\{W/W_{\max}, \log W\})$ if coalitions of weight up to $W_{\max}$ are allowed.*

Proof. In the setting where coalitions of size up to c are permitted, a lower bound of $\alpha n/c$ was already shown for unweighted games in Theorem 4.1. We first prove the lower bound of $\alpha W/W_{\max}$ if coalitions of weight up to $W_{\max}$ are feasible. Consider n agents with arbitrary weights w_i, $1 \le i \le n$. We use the simple network depicted in Figure 4.1(a) but change the costs of the two parallel edges. The expensive edge now has cost αW, whereas the inexpensive one costs $W_{\max}+\epsilon$. Recall that all the n agents have to connect terminals s and t. The state in which all the agents establish their connection using the expensive edge forms an α-approximate strong Nash equilibrium: Any legal coalition incurs a cost of at most $\alpha W \frac{W_{\max}}{W} = \alpha W_{\max}$ on the expensive edge. Switching to the inexpensive edge results in a cost of $W_{\max}+\epsilon$ for the coalition, which is not attractive enough. Obviously, the social optimum routes connections via the inexpensive edge.

We next show a lower bound of $\Omega(\alpha \log W)$ for both scenarios, where either the size or the weight of a coalition is limited. Without loss of generality let W be a power of 2 and let $n = \log_2 W + 1$. We use the graph of Figure 4.1(b) but change the costs of the edges. Each edge (v_i, t) now has cost α, $1 \le i \le n$, and edge (w, t) has cost $2+\epsilon$. The edges (v_i, w) still have a cost of 0. Agent i, $1 \le i < n$, has a weight of $W/2^i$ and wishes to connect terminals v_i and t. The last agent n has a weight of 1 and has to connect v_n to t. The total weight of all of the n agents is exactly W. The state in which every agent i, $1 \le i \le n$, establishes its connection using edge (v_i, t) represents an α-approximate strong Nash equilibrium: In any coalition I of legal size or weight, the agent $i_0 \in I$ of maximum weight in I dominates the other agents in I; i.e., the weight of i_0 is at least as large as the total weight of all the other agents in I. Hence, when I changes strategy and purchases edge (w, t), agent i_0 has to pay at least $1+\epsilon/2$, and this is not smaller than an α-fraction of the cost incurred for the private edge (v_{i_0}, t). The cost of the strong Nash equilibrium is $\alpha(1+\log_2 W)$ while the social optimum incurs a cost of $2+\epsilon$. ∎

THEOREM 5.5. *For any $\alpha \ge 1$, there exists a family of undirected graphs, each admitting an α-approximate strong Nash equilibrium whose cost is $\Omega(\alpha\sqrt{\log W})$ times that of the social optimum.*

Proof. We extend the proof of Theorem 4.2 and first concentrate on $\alpha = 1$. Let W be a real weight with $\lfloor \sqrt{\log W} \rfloor \geq 2$. Again logarithms are taken to the base 3. As before we construct a graph $G = G^0$ in a recursive manner, choosing $d_{\max} = \lfloor \sqrt{\log W} \rfloor - 1$ in the case of weighted games. In any graph G^d, $0 \leq d \leq d_{\max}$, the edge costs are the same as those defined in the proof of Theorem 4.2. However, the number of associated agents changes. Associated with a graph $G^{d_{\max}}$ is *one* agent of order $d_{\max}$ having a weight of $w_{d_{\max}} = W/3^{d_{\max}(d_{\max}+1)}$. Associated with a graph G^d, $0 \leq d < d_{\max}$, is *one* agent of order d having a weight of $w_d = W/3^{d(d+1)} - 3W/3^{(d+1)(d+2)}$. The total weight of all the agents is exactly W. The total weight of all the agents associated with order-d graphs G^d and the subgraphs therein is $W_d = W/3^{d^2}$ for any $0 \leq d \leq d_{\max}$.

As edge costs have not changed, the social optimum is still constant. As usual, let $\mathcal{S}$ be the configuration in which an order-d agent purchases the arc of order-d within its graph. Then $cost(\mathcal{S}) \geq d_{\max} + 1 \geq \lfloor \sqrt{\log W} \rfloor$.

To show that $\mathcal{S}$ forms a strong Nash equilibrium, we can extend Lemmas 4.4 and 4.5 in a straightforward way. In the arguments agent numbers such as n_d and N_{d+1} are to be replaced by weights w_d and W_{d+1}. Some of the arguments and calculations in the proofs simplify because ceilings can be ignored and fractions f and f_k^j, reflecting portions of order-d agents that defect from routing through their order-d arcs, are now equal to either 0 or 1. Finally, for $\alpha > 1$, we again scale the arc costs by α. □

6. The price of stability in undirected graphs. In this section we address the price of stability of standard Nash equilibria in weighted games. Anshelevich et al. [2] showed a lower bound of $\Omega(\log W)$ for directed graphs. Again, $W = \sum_{i=1}^{n} w_i$ is the total weight of all the agents. We prove a lower bound for undirected graphs. No superconstant lower bound was known for undirected graphs, for neither unweighted nor weighted games.

THEOREM 6.1. *In undirected graphs the price of stability is* $\Omega(\log W / \log\log W)$. *This lower bound holds even if each agent has to connect only a pair of terminals. Individual terminal pairs are allowed.*

Proof. We construct a family of graphs, each admitting only one Nash equilibrium. The cost of this equilibrium will be $\Omega(\log W / \log\log W)$ times that of the social optimum. The basic structure of the graphs is the same as those constructed in the proof of Theorem 4.2. However, the parameters are chosen differently here. Let W be a positive integer with $\log W \geq 3$. Again, logarithms are taken to the base 3. Let $d_{\max} = \lfloor \log W / (\log\log W + 1) \rfloor$. Inequality $\log W \geq 3$ implies $d_{\max} \geq 1$.

In the basic graphs $G^{d_{\max}}$ a stem edge has cost $s_{d_{\max}} = 1/3^{d_{\max}}$, and the bridge of order $d_{\max}$ has cost $b_{d_{\max}} = 3/(3^{d_{\max}} \log W)$. The arc of order $d_{\max}$ costs $a_{d_{\max}} = 1/3^{d_{\max}}$. Associated with any $G^{d_{\max}}$ is one order-$d_{\max}$ agent of weight $w_{d_{\max}} = W/(3\log W)^{d_{\max}}$ wishing to connect the base and the tip of $G^{d_{\max}}$.

For any d, $0 \leq d < d_{\max}$, in a graph G^d of order d, the bridge of order d has cost $b_d = 3/(3^d \log W)$, and the arc of order d costs $a_d = 1/3^d$. Associated with G^d is one order-d agent of weight $w_d = W/(3\log W)^d - 3W/(3\log W)^{d+1}$ having to connect the base and the tip of G^d. The outermost graph $G = G^0$ is the graph we will work with.

The total weight of agents associated with one order-d graph G^d and all the subgraphs therein is $W_d = W/(3\log W)^d$. This holds for $d = d_{\max}$. Suppose that the property holds for orders $d_{\max}, d_{\max} - 1, \ldots, d + 1$. Since a graph of order d is composed of three graphs of order $d + 1$, the total weight of agents in G^d is equal to

$$W_d = w_d + 3W/(3\log W)^{d+1} = W/(3\log W)^d.$$

In particular, we obtain that the total weight of agents in $G = G^0$ is exactly W.

PROPOSITION 6.2. *The least expensive path connecting the base and the tip of a graph G^d using only edges of G^d has a total edge cost of exactly $1/3^d$ for any $0 \le d \le d_{\max}$.*

Proof. The statement of the proposition holds for $d = d_{\max}$ because the arc of $G^{d_{\max}}$ has cost $a_{d_{\max}} = 1/3^{d_{\max}}$, while the path crossing the bridge has cost $b_{d_{\max}} + s_{d_{\max}} = 3/(3^{d_{\max}} \log W) + 1/3^{d_{\max}}$. Assume that the statement of the proposition holds for depths $d_{\max}, \ldots, d+1$. In G^d the arc of order d has cost $a_d = 1/3^d$, while, by the induction hypothesis, any path using the bridge of order d has a cost of at least $b_d + 3 \cdot 1/3^{d+1} = 3/(3^d \log W) + 1/3^d > 1/3^d$. ∎

The social optimum in G buys the backbone consisting of stem edges and bridges. There exist exactly $3^{d_{\max}}$ subgraphs of order $d_{\max}$, and hence the total cost of stem edges is $3^{d_{\max}} s_{d_{\max}} = 1$. For any fixed d, $0 \le d \le d_{\max}$, graph G contains 3^d graphs of order d, each being equipped with an order-d bridge of cost $b_d = 3/(3^d \log W)$. Thus the total cost of order-d bridges is $3^d b_d = 3^d \cdot 3/(3^d \log W) = 3/\log W$. Summing over all d we find that the total cost of bridges is upper bounded by $(d_{\max} + 1) \cdot 3/\log W \le (2 \log W/\log\log W)(3/\log W) \le 6/\log\log W \le 6$. Hence the cost of the social optimum is constant.

Consider configuration $\mathcal{S}$ in which, for any graph G^d within G, the order-d agent associated with G^d purchases the order-d arc in this graph. We will prove in the following that $\mathcal{S}$ is a Nash equilibrium and that it is the only Nash equilibrium in G. As there are 3^d graphs of order d, the total cost of order-d arcs is $3^d a_d = 3^d \cdot 1/3^d = 1$, and summing over all d we obtain $cost(\mathcal{S}) = d_{\max} + 1 \ge \log W/(\log\log W + 1)$, which gives the stated lower bound on the price of stability.

In the remainder of this proof we show, in a first step, that $\mathcal{S}$ forms a Nash equilibrium and then, in a second step, that $\mathcal{S}$ is the only equilibrium in G.

We proceed with the proof that $\mathcal{S}$ forms an equilibrium state. Let i be an order-d agent associated with a graph G^d. We show that any strategy change performed by i yields a strictly higher cost. If i deviates from its original strategy in $\mathcal{S}$, it can establish the required connection either (1) by using a path within G^d or (2) by using a path of edges outside G^d.

In case (1), the path P_i used by agent i to connect its terminal pair has to traverse the order-d bridge in G^d, which has a cost of $b_d = 3/(3^d \log W)$. If $d = d_{\max}$, path P_i continues on the stem edge of cost $s_{d_{\max}} = 1/3^{d_{\max}}$. If $d < d_{\max}$, then P_i has to traverse three graphs of order $d+1$, the total cost of which is at least $3 \cdot 1/3^{d+1} = 1/3^d$. The total weight of agents that can share the cost of P_i is upper bounded by W_d. Thus agent i incurs a cost of at least

$$\frac{w_d}{W_d} \cdot \frac{1}{3^d}\left(1 + \frac{3}{\log W}\right).$$

If $d = d_{\max}$, then $w_d = W_d$ and the latter expression is larger than the cost of $1/3^{d_{\max}}$ incurred for buying the order-$d_{\max}$ arc in $G_{d_{\max}}$. If $d < d_{\max}$, then we have

$$\begin{aligned}
\frac{w_d}{W_d} \cdot \frac{1}{3^d}\left(1 + \frac{3}{\log W}\right) &= \frac{W/(3\log W)^d - 3W/(3\log W)^{d+1}}{W/(3\log W)^d} \cdot \frac{1}{3^d}\left(1 + \frac{3}{\log W}\right) \\
&= \left(1 - \frac{1}{\log W}\right)\left(1 + \frac{3}{\log W}\right)\frac{1}{3^d} \\
&= \frac{\log^2 W + 2\log W - 3}{\log^2 W} \cdot \frac{1}{3^d} \\
&> 1/3^d,
\end{aligned}$$

because $\log W \geq 3$. Again, buying the order-d arc of cost $a_d = 1/3^d$ is a strictly better strategy.

In case (2), we have $d > 0$, and the path P_i used by agent i crosses the bridge of order $d-1$ and visits the base of graph G^{d-1} containing G^d. The structure of P_i is depicted in Figures 4.3(a)–(c), which we used in the proof of Theorem 4.2; the situation here is the very same. To reach the tip of G^d, path P_i must visit the tip of G^{d-1}, from where it can continue. Path P_i must fully traverse two subgraphs of order d within G^{d-1}. Such a subgraph can be traversed on an arc of order d, having cost $a_d = 1/3^d$, where the weight of the agent that bought this arc in $\mathcal{S}$ is w_d. Thus the cost of a_d can be shared among two order-d agents. If P_i does not use the order-d arc, the total cost of edges traversing an order-d subgraph is at least $1/3^d$, and the total weight of agents sharing the edge cost is $W_d - w_d < w_d$. Thus, for the traversal of the two order-d subgraphs, agent i pays at least

$$2\frac{w_d}{2w_d} \cdot \frac{1}{3^d} = \frac{1}{3^d}.$$

Since the traversal of the order-$(d-1)$ bridge has positive cost, path P_i incurs a cost strictly higher than that of the original strategy of i in $\mathcal{S}$.

It remains to show that $\mathcal{S}$ is the only Nash equilibrium. To this end we will prove that in any Nash equilibrium, an order-d agent associated with a given graph G^d must buy the corresponding order-d arc in G^d. In other words, an equilibrium state must be equal to $\mathcal{S}$. The desired statement that in any Nash equilibrium an order-d agent purchases the corresponding order-d arc in its graph G^d follows from the next lemma. Loosely speaking, this lemma says that in a Nash equilibrium connections in G are established locally. We first state the lemma and then explain its implications.

LEMMA 6.3. *Consider a fixed order-d graph G^d in G and assume that in any Nash equilibrium all the agents associated with G^d and its subgraphs establish their connections using only edges of G^d. Furthermore, assume that all agents* not *associated with G^d or its subgraphs do* not *use any edges of G^d when routing their connections. Then in any Nash equilibrium the following two properties hold.*

(a) *The order-d agent associated with G^d buys the arc of order d in G^d.*

(b) *If $d < d_{\max}$, then for any of the three order-$(d+1)$ subgraphs G_k^{d+1}, $1 \leq k \leq 3$, within G^d, the agents associated with G_k^{d+1} and its subgraphs establish their connections using only edges of G_k^{d+1}.*

Using this lemma we can finish the proof of our theorem: For $d = 0$, trivially, all agents associated with $G = G^0$ and its subgraphs must establish connections within G^0, and there exist no agents outside G^0 that could use edges of G^0. Thus the conditions of Lemma 6.3 are met, and we obtain that the order-0 agent buys the arc of order 0 (part (a)) and that, for any of the three subgraphs G_k^1, $1 \leq k \leq 3$, agents associated with any G_k^1 and its subgraphs establish connections using only edges of this graph G_k^1 (part (b)). Inductively, Lemma 6.3 yields that, for any d, (a) any order-d agent purchases the order-d arc within its graph and (b) for any subgraph G^{d+1} of order $d+1$, all agents associated with G^{d+1} at its subgraphs establish the required connections locally within G^{d+1}.

Proof of Lemma 6.3. Part (a): Suppose that in a Nash equilibrium, an order-d agent associated with a graph G^d does not purchase the arc of order d. Let P be the path used by the agent to connect its terminal pair. Since, by assumption of the lemma, the agent establishes its connection within G^d, path P must cross the order-d bridge; see Figures 4.2(a) and (b). If $d = d_{\max}$, then the path traverses the stem edge

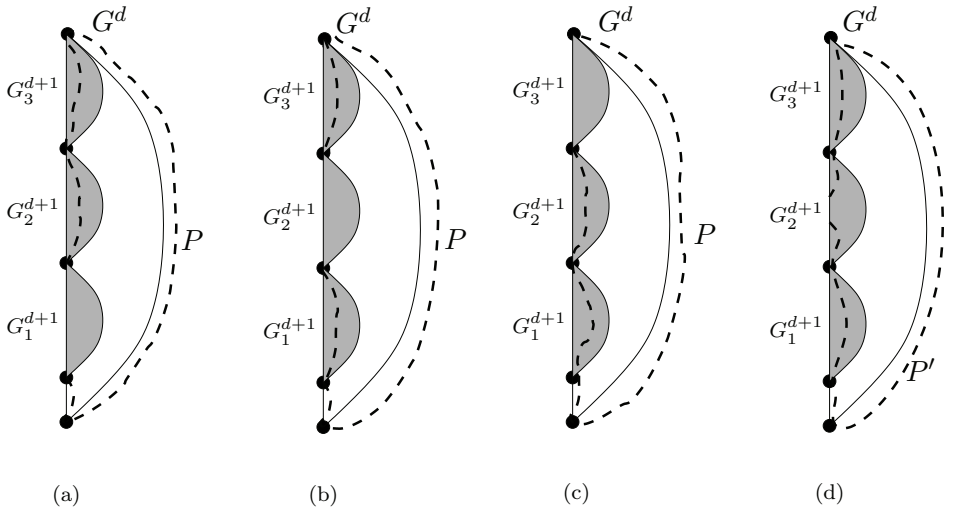


FIG. 6.1. *The paths taken by an order-$(d+1)$ agent within G_k^d, $1 \le k \le 3$.*

of cost $s_{d_{\max}} = 1/3^{d_{\max}}$ in $G^d = G^{d_{\max}}$ to reach the tip of the graph. If $d < d_{\max}$, path P traverses the three subgraphs G_k^{d+1}, $1 \le k \le 3$, to reach the tip of G^d. When traversing the subgraphs, then by Proposition 6.2 path P visits edges of total cost at least $3 \cdot 1/3^{d+1} = 1/3^d$. Hence, in any case, the total cost of edges traversed by P is at least $b_d + 1/3^d = 3/(3^d \log W) + 1/3^d = (1 + 3/\log W)/3^d$. By assumption of the lemma, agents not associated with G^d or its subgraphs do not use edges of G^d. Hence the cost of P is shared by agents of total weight at most $W_d = W/(3 \log W)^d$ that are associated with G^d and its subgraphs. Hence the total cost of the order-d agent is at least $\frac{w_d}{W_d} \frac{1}{3^d}(1 + \frac{3}{\log W})$. We argue that this expression is strictly larger than $a_d = 1/3^d$, which is the cost of purchasing the arc of order d. If $d = d_{\max}$, then $w_d = W_d$ and we are done. If $d < d_{\max}$, then as shown on the previous page we have $\frac{w_d}{W_d} \frac{1}{3^d}(1 + \frac{3}{\log W}) > \frac{1}{3^d}$.

Part (b): We first prove that any order-$(d+1)$ agent associated with a graph G_k^{d+1}, $1 \le k \le 3$, establishes its connection within G_k^{d+1}. We then show that agents of order larger than $d+1$ associated with subgraphs of G_k^{d+1}, if such subgraphs exist, also route their connections within G_k^{d+1}.

In a first step we lower bound the cost incurred by order-$(d+1)$ agents if they buy edges outside their graph. In the following, if $d+1 = d_{\max}$, we set $W_{d+2} = 0$.

CLAIM 1. *If the agent of order $d+1$ associated with a graph G_k^{d+1} establishes its connection using edges outside G_k^{d+1}, then its total cost is at least $C = \frac{2w_{d+1}}{2w_{d+1}+9W_{d+2}} \frac{1}{3^{d+1}}(1 + \frac{3}{\log W})$.*

Proof. We first show that if the order-$(d+1)$ agent associated with G_k^{d+1} implements its connection using edges outside G_k^{d+1}, then the path P used by this agent must traverse the bridge of order d as well as the other two graphs of order $d+1$ within G^d. We consider all possible values of k and refer the reader to Figures 6.1(a)–(c) for the structure of P. The situation is the same as that described in Figures 4.3(a)–(c); the only difference is that here the outer graph has order d instead of $d-1$. Recall that strategies are simple paths connecting the desired terminals. If $k = 1$ (cf. Figure 6.1(a)), then the order-$(d+1)$ agent associated with G_1^{d+1} must traverse the order-d

bridge of G^d. In order to reach the tip of G_1^{d+1}, path P must visit the tip of G^d from where P traverses G_3^{d+1} and G_2^{d+1}. Similarly, if $k = 2$ (cf. Figure 6.1(b)), then path P traverses G_1^{d+1}, crosses the bridge of order d, travels to the tip of G^d, and visits G_3^{d+1} to reach the tip of G_2^{d+1}. Finally, if $k = 3$ (cf. Figure 6.1(c)), path P must traverse G_2^{d+1} and G_1^{d+1}. The path then crosses the bridge of order d and travels to the tip of G^d, which is also the tip of G_3^{d+1}.

We lower bound the cost incurred by the order-$(d+1)$ agent associated with G_k^{d+1} for path P. For brevity, we will denote this agent by i_k. By the assumptions of the lemma to be proved, agents not associated with G^d or its subgraphs do not use edges of G^d. Graph G^d contains three order-$(d+1)$ agents of total weight $3w_{d+1}$. If $d \le d_{\max} - 2$, then G^d also contains nine graphs of order $d+2$, each hosting agents of total weight W_{d+2}. We set $W_{d+2} = 0$ if $d = d_{\max} - 1$. We argued in the last paragraph that path P must cross the bridge of order d, which has a cost of $b_d = 3/(3^d \log W)$. This cost is split among agents of total weight at most $3w_{d+1} + 9W_{d+2}$. Hence, for the bridge of order d, agent i_k pays at least

$$\frac{w_{d+1}}{3w_{d+1} + 9W_{d+2}} \cdot \frac{3}{3^d \log W} \ge \frac{2w_{d+1}}{2w_{d+1} + 9W_{d+2}} \cdot \frac{3}{3^{d+1} \log W}. \tag{6.1}$$

Also, as argued in the last paragraph, path P connecting the terminals of i_k has to traverse the other two subgraphs of order $d+1$ in G^d, which are indexed $k' = k \bmod 3 + 1$ and $k'' = (k+1) \bmod 3 + 1$. To traverse one such subgraph, path P traverses edges of total cost at least $1/3^{d+1}$. We distinguish cases depending on whether the order-$(d+1)$ agents associated with $G_{k'}^{d+1}$ and $G_{k''}^{d+1}$ implement their connections using edges inside or outside their respective graphs. First, assume that the order-$(d+1)$ agents associated with $G_{k'}^{d+1}$ and $G_{k''}^{d+1}$ establish connections within their respective subgraphs. In this case path P encounters other agents of total weight at most $w_{d+1} + 9W_{d+2}$ in each of these subgraphs. Thus, the traversal cost of P is shared among agents of total weight at most $2w_{d+1} + 9W_{d+2}$. We obtain that agent i_k incurs a cost of at least

$$2\frac{w_{d+1}}{2w_{d+1} + 9W_{d+2}} \cdot \frac{1}{3^{d+1}} \tag{6.2}$$

for the traversal of $G_{k'}^{d+1}$ and $G_{k''}^{d+1}$. Next, assume that the order-$(d+1)$ agents associated with $G_{k'}^{d+1}$ and $G_{k''}^{d+1}$ establish connections using edges outside their graphs. In this case, again, path P encounters other agents of total weight at most $w_{d+1} + 9W_{d+2}$ when traversing any of these subgraphs (in $G_{k'}^{d+1}$ the associated order-$(d+1)$ agent is not present; the analogous statement holds for $G_{k''}^{d+1}$). Hence the cost incurred in traversing any of the two subgraphs $G_{k'}^{d+1}$ and $G_{k''}^{d+1}$ is shared among agents of total weight at most $2w_{d+1} + 9W_{d+2}$, and we obtain the same cost bound as that given in (6.2). Finally, assume that in exactly one of the subgraphs among $G_{k'}^{d+1}$ and $G_{k''}^{d+1}$ the associated order-$(d+1)$ agent establishes its connection within its subgraph. As for the other of the two subgraphs, the associated order-$(d+1)$ agent uses edges outside its graph. Without loss of generality let $G_{k'}^{d+1}$ be the graph where connections are made inside, and let $G_{k''}^{d+1}$ be the graph where connections are established using edges outside. The other case is symmetric. When P traverses $G_{k'}^{d+1}$, agents of total weight at most $2w_{d+1} + 9W_{d+2}$ are present, and cost sharing on edges can be done among agents of weight at most $3w_{d+1} + 9W_{d+2}$. When P visits $G_{k''}^{d+1}$, agents of total weight at most $9W_{d+2}$ are present: The order-$(d+1)$ agent associated with $G_{k'}^{d+1}$ is

not present because it uses connections inside its graph, and the order-$(d+1)$ agent associated with $G_{k''}^{d+1}$ is not present because it uses a strategy outside its graph. Hence cost sharing on edges can be done among agents of total weight at most $w_{d+1}+9W_{d+2}$ and the cost incurred by i_k in traversing the two other order-$(d+1)$ graphs is at least

$$\left(\frac{w_{d+1}}{3w_{d+1}+9W_{d+2}}+\frac{w_{d+1}}{w_{d+1}+9W_{d+2}}\right)\frac{1}{3^{d+1}} \geq 2\frac{w_{d+1}}{2w_{d+1}+9W_{d+2}}\cdot\frac{1}{3^{d+1}},$$

which is the same expression as (6.2). Summing the costs incurred for crossing the order-d bridge and for traversing other order-$(d+1)$ graphs (see (6.1) and (6.2)), we conclude that agent i_k pays at least the cost of C stated in the claim. □

If the order-$(d+1)$ agent associated with G_k^{d+1} purchases the arc of order $d+1$ within G_k^{d+1}, its cost is at most $a_{d+1}=1/3^{d+1}$. A strategy using edges outside the graph incurs a cost of at least C as stated in the above claim. We show that $C > a_{d+1}$, which proves that in a Nash equilibrium the order-$(d+1)$ agent associated with G_k^{d+1} establishes the required connection via the order-$(d+1)$ arc. If $d = d_{\max}-1$, then $W_{d+2}=0$, and we are done because $C=\frac{1}{3^{d+1}}(1+\frac{3}{\log W})>\frac{1}{3^{d+1}}$.

If $d \leq d_{\max}-2$, then

$$C=\frac{2w_{d+1}}{2w_{d+1}+9W_{d+2}}\frac{1}{3^{d+1}}\left(1+\frac{3}{\log W}\right)=\frac{1}{1+(9W_{d+2}/2w_{d+1})}\frac{1}{3^{d+1}}\left(1+\frac{3}{\log W}\right).$$

It remains to show $9W_{d+2}/(2w_{d+1}) < 3/\log W$, which proves the desired inequality $C > 1/3^{d+1}$. We have

$$\frac{9W_{d+2}}{2w_{d+1}}=\frac{9W/(3\log W)^{d+2}}{2W/(3\log W)^{d+1}-6W/(3\log W)^{d+2}}=\frac{3}{2\log W-2}<\frac{3}{\log W}.$$

The last inequality holds because $\log W \geq 3$.

To finish the proof of part (b) of the lemma we have to show that if $d \leq d_{\max}-2$, then any agent i of order $d+2$ or larger that is associated with a subgraph of G_k^{d+1}, $1 \leq k \leq 3$, establishes its connection within G_k^{d+1}. Suppose this were not the case. Then agent i chooses a path P that leaves G_k^{d+1} through its base. Figure 6.1(d) shows a sample path for $k=2$. To connect to the desired terminal, path P must visit the tip of G_k^{d+1} from where it can continue on edges inside G_k^{d+1}. Since P uses edges outside G_k^{d+1}, it does not use the arc of order $d+1$ in G_k^{d+1} and hence must traverse the bridge of order d in G^d. The cost of this bridge is shared by agents of total weight at most $9W_{d+2}$ because we have shown that all the agents of order $d+1$ in G^d establish connections within their respective subgraphs and the order-d agent associated with G^d purchases the arc of order d (see part (a)). Let P' be the subpath of P connecting the base and the tip of G_k^{d+1}. On P' agent i incurs a cost of at least

$$cost(P') \geq \frac{w(i)}{9W_{d+2}}\frac{3}{3^d\log W},$$

where $w(i)$ is the weight of agent i. In the given Nash equilibrium, consider the strategy used by the order-$(d+1)$ agent associated with G_k^{d+1}. The cost of this agent is at most $1/3^{d+1}$ because this is the cost incurred when buying the order-$(d+1)$ arc in G_k^{d+1}, which is always an option. This order-$(d+1)$ agent has to connect the base and the tip of G_k^{d+1} and, as shown in the previous paragraphs, use edges within G_k^{d+1}. Now, agent i can replace P' by the strategy used by the order-$(d+1)$ agent of G_k^{d+1},

incurring a cost of at most $w(i)/(w(i)+w_{d+1})\cdot 1/3^{d+1} \le w(i)/w_{d+1}\cdot 1/3^{d+1}$. We show that the latter expression is strictly smaller than the cost $cost(P')$, contradicting the fact that the configuration in which i used edges outside G_k^{d+1} was a Nash equilibrium. Inequality $cost(P') > w(i)/w_{d+1}\cdot 1/3^{d+1}$ is equivalent to showing $9W_{d+2}/w_{d+1} < 9/\log W$. We have

$$\frac{9W_{d+2}}{w_{d+1}} = \frac{9W/(3\log W)^{d+2}}{W/(3\log W)^{d+1} - 3W/(3\log W)^{d+2}} \le \frac{3}{\log W - 1} < \frac{9}{\log W},$$

since $\log W \ge 3$. □

This completes the proof of the theorem. □

The lower bound of Theorem 6.1 is nearly tight. First, the potential function arguments of the proof of Theorem 5.1 imply that there exists an α-approximate Nash equilibrium whose cost is at most $1+\ln W$ times that of the social optimum if $\alpha \ge 1+\ln(1+w_{\max})$. Here $w_{\max}$ is the maximum weight of any agent. Second, Chen and Roughgarden [7] showed that in directed graphs, for any $\alpha = \Omega(\log w_{\max})$, the price of stability of α-approximate Nash equlibria is $O((\log W)/\alpha)$. This result can be extended to undirected graphs.

7. Conclusions. In this paper we have investigated the value of coordination in network design games. We have developed lower and upper bounds on the price of anarchy attained by strong Nash equilibria in unweighted and weighted games, considering both undirected and directed graphs. It shows that strong Nash equilibria achieve much better performance ratios than standard Nash equilibria and that these ratios are often as good as those of the best standard equilibrium states. There is still room for improvements. For undirected graphs we have developed an upper bound of $H_n \approx \ln n$ and a lower bound of $\Omega(\sqrt{\log n})$ on the price of anarchy in unweighted games. In weighted games the bounds are $1+\ln W$ and $\Omega(\sqrt{\log W})$, respectively. An interesting open problem is to determine the true ratios for undirected graphs.

Furthermore, in this paper we have also devised the first superconstant lower bound on the price of stability in undirected graphs. More specifically, we proved a lower bound of $\Omega(\log W/\log\log W)$ for weighted network design games. A challenging open problem is to determine the price of stability in unweighted games.

Appendix.

Proof of Proposition 3.1. We prove the result for undirected graphs and then show how to direct edges to obtain the desired statement for directed networks as well. Consider the graph given in Figure A.1. We have a vertex set $V = \{v_1, v_2, v_3, w_1, w_2, w_3, t\}$, where vertex w_i is connected to t via a *main edge* $\{w_i, t\}$ of cost 1, $1 \le i \le 3$. Furthermore, there are *auxiliary edges* $\{v_i, w_i\}$ of cost $1/2$ and *auxiliary edges* $\{v_i, w_{i \bmod 3+1}\}$ of cost $1/2+\epsilon$, $1 \le i \le 3$. Here $\epsilon > 0$ is an arbitrarily small value. Associated with the graph are three agents, where agent i has to connect terminals v_i and t, $1 \le i \le 3$. We will consider all possible states and show that none represents a strong Nash equilibrium. Any state in which all of the three main edges are purchased does not form a strong Nash equilibrium because two agents i and $i' = i \bmod 3 + 1$ could team up, sharing the main edge $\{w_{i'}, t\}$. As the original cost of each of the two agents was at least $1+1/6 = 7/6$ and the new cost is at most $1+\epsilon$, this yields a cost reduction for each member of the coalition.

Next suppose that there exists a strong Nash equilibrium in which two agents share a main edge $\{w_i, t\}$, while the third agent buys a second main edge $\{w_j, t\}$, $j \ne i$. Then agent i is one of the agents sharing $\{w_i, t\}$ and connects to w_i using

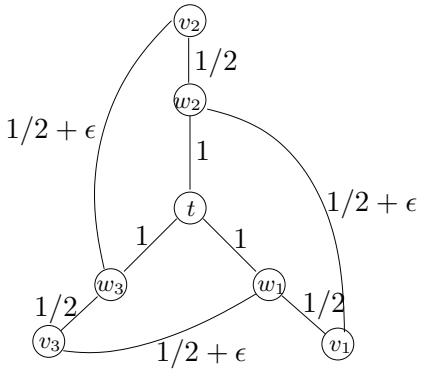


FIG. A.1. *A graph without a strong Nash equilibrium.*

edge $\{v_i, w_i\}$ since otherwise agent i could strictly improve its cost by purchasing a third of $\{w_i, t\}$. We now distinguish cases depending on whether $j = i \bmod 3 + 1$ or $j = (i + 1) \bmod 3 + 1$. Note that our graph is symmetric in vertex index and agent number i. Thus we may assume without loss of generality that $i = 1$; i.e., agent 1 and some other agent share $\{w_1, t\}$. A third agent buys $\{w_j, t\}$, where $j = 2$ or $j = 3$. We distinguish cases depending on whether $j = 2$ or $j = 3$.

If $j = 2$, then agent 2 must be the one buying $\{w_2, t\}$ as connecting v_2 to w_1 requires the traversal of three auxiliary edges, the cost of which is at least $1/2 + \epsilon/3$. This cost is higher than that of buying $\{v_2, w_2\}$, and hence agent 2 would prefer to share $\{w_2, t\}$ instead of $\{w_1, t\}$. Thus agent 3 shares main edge $\{w_1, t\}$ and connects to w_1 using edge $\{v_3, w_1\}$ of cost $1/2+\epsilon$ because any other path of auxiliary edges has a strictly higher cost. We conclude that agent 2 pays a cost of $3/2$ and agent 3 a cost of $1/2 + \epsilon + 1/2 = 1 + \epsilon$. Now agents 2 and 3 can form a coalition, sharing main edge $\{w_3, t\}$. The new cost of agent 2 is $1/2+\epsilon+1/2 < 3/2$ and the new cost of agent 3 is $1/2+1/2 < 1+\epsilon$, contradicting the assumption that the original configuration was a strong Nash equilibrium.

If $j = 3$, then again agent 3 buys main edge $\{w_3, t\}$: If agent 3 shared $\{w_1, t\}$ and agent 2 bought $\{w_3, t\}$, agent 2 would connect to w_3 using edge $\{v_2, w_3\}$ and agent 3 would connect to w_1 using edge $\{v_3, w_1\}$ as other paths of auxiliary edges are strictly more expensive. Both agents pay a cost of $1/2 + \epsilon$ for these connections. In this situation agent 3 could strictly improve its cost by connecting to w_3 and sharing $\{w_3, t\}$ instead of $\{w_1, t\}$. We conclude that agent 2 shares main edge $\{w_1, t\}$ and connects to w_1 at a cost of $1 + \epsilon + 1/4$. Now agent 2 can improve its cost by buying edge $\{v_2, w_3\}$ at a cost of $1/2 + \epsilon$ and sharing edge $\{w_3, t\}$ instead of $\{w_1, t\}$. We obtain a contradiction to the fact that the original configuration was a strong Nash equilibrium.

We finally have to investigate the case that a configuration buys only one main edge $\{w_i, t\}$, the cost of which is shared among the three agents. Again we may assume without loss of generality that $i = 1$. Then agent 3 connects to w_1 using edge $\{v_3, w_1\}$ and agent 2 connects to w_1 using a path of auxiliary edges that results in a cost of at least $1 + \epsilon + 1/4$. Hence the total cost of agent 2 is at least $1 + 1/4 + 1/3 + \epsilon > 3/2$, and agent 2 can improve its strategy by buying edges $\{v_2, w_2\}$ and $\{w_2, t\}$.

We note that the graph can be extended to any agent number n by inserting nodes $v_4, \ldots, v_n$ affiliated with agents numbered 4 to n, where agent i wishes to connect v_i to t, $4 \le i \le n$. Each such v_i is connected to t via a private edge.

This concludes the analysis of undirected graphs. To obtain the result for directed graphs we simply direct edges toward the destination t. We have *main edge* (w_i, t) as well as *auxiliary edges* (v_i, w_i) and $(v_i, w_{i \bmod 3+1})$, $1 \leq i \leq 3$. Directing the edges only restricts the set of possible states, while all strategy changes proposed above can still be performed. □

Acknowledgments. We thank Avrim Blum, Anupam Gupta, and Katrina Ligett for many interesting technical discussions. Furthermore, we thank two anonymous referees for helpful comments improving the presentation of the paper.

REFERENCES

[1] N. Andelman, M. Feldman, and Y. Mansour, *Strong price of anarchy*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2007, pp. 189–198.

[2] E. Anshelevich, A. Dasgupta, J. M. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden, *The price of stability for network design with fair cost allocation*, in Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 2004, pp. 295–304.

[3] E. Anshelevich, A. Dasgupta, E. Tardos, and T. Wexler, *Near optimal network design with selfish agents*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2003, pp. 511–520.

[4] R. J. Aumann, *Acceptable points in general cooperative n-person games*, in Contributions to the Theory of Games, Vol. IV, Ann. of Math. Stud. 40, A. W. Tucker and R. D. Luce, eds., Princeton University Press, Princeton, NJ, 1959, pp. 287–324.

[5] V. Bala and S. Goyal, *A non-cooperative model of network formation*, Econometrica, 68 (2000), pp. 1181–1229.

[6] C. Chekuri, J. Chuzhoy, L. Lewin-Eytan, J. Naor, and A. Orda, *Non-cooperative multicast and facility location games*, in Proceedings of the 7th ACM Conference on Electronic Commerce (EC), ACM, New York, 2006, pp. 72–81.

[7] H.-L. Chen and T. Roughgarden, *Network design with weighted players*, in Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM, New York, 2006, pp. 29–38.

[8] J. Corbo and D. Parkes, *The price of selfish behavior in bilateral network formation*, in Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 2005, pp. 99–107.

[9] N. R. Devanur, M. Mihail, and V. V. Vazirani, *Strategyproof cost-sharing mechanisms for set cover and facility location games*, Decis. Support Syst., 39 (2005), pp. 11–22.

[10] A. Epstein, M. Feldman, and Y. Mansour, *Strong equilibrium in cost sharing connection games*, in Proceedings of the 8th ACM Conference on Electronic Commerce (EC), ACM, New York, 2007, pp. 84–92.

[11] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker, *On a network creation game*, in Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 2003, pp. 347–351.

[12] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker, *Sharing the cost of multicast transmissions*, J. Comput. System Sci., 63 (2001), pp. 21–41.

[13] M. Feldman and T. Tamir, *Approximate strong equilibrium for job scheduling games*, in Proceedings of the First International Symposium on Algorithmic Game Theory (SAGT), Lecture Notes in Comput. Sci. 4997, Springer, Berlin, 2008, pp. 58–69.

[14] A. Fiat, H. Kaplan, M. Levy, and S. Olonetsky, *Strong price of anarchy for machine load balancing*, in Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 4596, Springer, Berlin, 2007, pp. 583–594.

[15] A. Fiat, H. Kaplan, M. Levy, S. Olonetsky, and R. Shabo, *On the price of stability for designing undirected networks with fair cost allocations*, in Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 4051, Springer, Berlin, 2006, pp. 608–618.

[16] H. Haller and S. Sarangi, *Nash networks with heterogeneous links*, Math. Social Sci., 50 (2005), pp. 181–201.

[17] S. Herzog, S. Shenker, and D. Estrin, *Sharing the "cost" of multicast trees: An axiomatic analysis*, IEEE/ACM Trans. Netw., 5 (1997), pp. 847–860.

[18] M. Hoefer, *Non-cooperative tree creation*, in Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 4162, Springer, Berlin, 2006, pp. 517–527.

[19] E. Koutsoupias and C. Papadimitriou, *Worst-case equilibria*, in Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 1563, Springer, Berlin, 1999, pp. 404–413.

[20] D. Monderer and L. Shapley, *Potential games*, Games Econom. Behav., 14 (1996), pp. 124–143.

[21] M. Pal and E. Tardos, *Group strategyproof mechanisms via primal-dual algorithms*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 2003, pp. 584–593.

[22] A. Vetta, *Nash equilibria in competitive societies with applications to facility location, traffic routing and auctions*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, 2002, pp. 416–425.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2303–2329

METRIC EMBEDDINGS WITH RELAXED GUARANTEES*

T.-H. HUBERT CHAN†, KEDAR DHAMDHERE‡, ANUPAM GUPTA§, JON KLEINBERG¶, AND ALEKSANDRS SLIVKINS‖

Abstract. We consider the problem of embedding finite metrics with *slack*: We seek to produce embeddings with small dimension and distortion while allowing a (small) constant fraction of all distances to be arbitrarily distorted. This definition is motivated by recent research in the networking community, which achieved striking empirical success at embedding Internet latencies with low distortion into low-dimensional Euclidean space, provided that some small slack is allowed. Answering an open question of Kleinberg, Slivkins, and Wexler [in *Proceedings of the* 45*th IEEE Symposium on Foundations of Computer Science*, 2004], we show that provable guarantees of this type can in fact be achieved in general: Any finite metric space can be embedded, with constant slack and constant distortion, into constant-dimensional Euclidean space. We then show that there exist stronger embeddings into ℓ_1 which exhibit *gracefully degrading* distortion: There is a single embedding into ℓ_1 that achieves distortion at most $O(\log \frac{1}{\epsilon})$ on all but at most an ϵ fraction of distances *simultaneously* for all $\epsilon > 0$. We extend this with distortion $O(\log \frac{1}{\epsilon})^{1/p}$ to maps into general ℓ_p, $p \geq 1$, for several classes of metrics, including those with bounded doubling dimension and those arising from the shortest-path metric of a graph with an excluded minor. Finally, we show that many of our constructions are tight and give a general technique to obtain lower bounds for ϵ-slack embeddings from lower bounds for low-distortion embeddings.

Key words. metric embeddings, low-distortion embeddings, metric spaces, metric decompositions, randomized algorithms

AMS subject classifications. 68Q25, 68W40, 68W20, 51F99, 54C25

DOI. 10.1137/060670511

1. Introduction.

Over the past decade, the field of metric embeddings has gained much importance in algorithm design. The central genre of problem in this area is the mapping of a given metric space into a "simpler" one, in such a way that the distances between points do not change too much. More formally, an *embedding* of a finite metric space (V, d) into a *target* metric space (V', d') is a map $\varphi : V \to V'$. Recent work on embeddings has used *distortion* as the fundamental measure of quality; the distortion of an embedding is the worst multiplicative factor by which distances

*Received by the editors September 25, 2006; accepted for publication (in revised form) October 6, 2008; published electronically March 4, 2009. The conference version of this paper has appeared in *IEEE Foundations of Computer Science* (FOCS), Pittsburgh, PA, 2005.
http://www.siam.org/journals/sicomp/38-6/67051.html

†Algorithms and Complexity, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany (hchan@mpi-inf.mpg.de). This work was done when this author was a graduate student at Carnegie Mellon University and was supported in part by NSF ITR CCR-0122581 (The ALADDIN project) and also by the NSF CAREER award and an Alfred P. Sloan Fellowship of Anupam Gupta.

‡Google Inc., Mountain View, CA 94043 (kedar.dhamdhere@gmail.com). This work was done when this author was a graduate student at Carnegie Mellon University and was supported in part by NSF ITR CCR-0122581 (The ALADDIN project).

§Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213 (anupamg@cs.cmu.edu). This author's research was supported by an NSF CAREER award CCF-0448095 and by an Alfred P. Sloan Fellowship.

¶Department of Computer Science, Cornell University, Ithaca, NY 14853 (kleinber@cs.cornell.edu). This author's research was supported by a David and Lucile Packard Foundation Fellowship and NSF grants CCF-0325453, IIS-0329064, and CCR-0122381; this work was done in part while on sabbatical leave at Carnegie Mellon University.

‖Microsoft Research, Mountain View, CA 94043 (slivkins@microsoft.com). This work was done when this author was a graduate student at Cornell University and was supported by the Packard Fellowship of Jon Kleinberg.

are increased by the embedding[1]. The popularity of distortion has been driven by its applicability to approximation algorithms: If the embedding $\varphi : V \to V'$ has a distortion of D, then the cost of solutions to some optimization problems on (V, d) and on $(\varphi(V), d')$ can differ only by some function of D; this idea has led to numerous approximation algorithms [24].

In parallel with theoretical work on embeddings, there has been a surge of interest in the networking community on *network embedding* problems closely related to the framework above (see, e.g., [13, 37, 42]). This work is motivated by different applications: One takes the point-to-point latencies among nodes in a network such as the Internet, treats this as a distance matrix,[2] and embeds the nodes into a low-dimensional space so as to approximately preserve the distances. In this way, each node is assigned a short sequence of virtual "coordinates," and distances between nodes can be approximated simply by looking up their coordinates and computing the distance, rather than having to interact with the relevant nodes themselves. As location-aware applications in networks become increasingly prevalent—for example, finding the nearest server in a distributed application with replicated services or finding the nearest copy of a file or resource in a peer-to-peer system—having such distance information in a compact and easily usable form is an issue of growing importance (see, e.g., the discussion in [13]).

In the context of these networking applications, however, distortion as defined above has turned out to be too demanding an objective function—many metrics cannot be embedded into Euclidean space with constant distortion; many of those that can be so embedded require a very large number of dimensions; and the algorithms to achieve these guarantees require a type of centralized coordination (and extensive measurement of distances) that is generally not feasible in Internet settings. Instead, the recent networking work has provided *empirical* guarantees of the following form: If we allow a small fraction of all distances to be *arbitrarily* distorted, we can embed the remainder with (apparently) constant distortion in constant-dimensional Euclidean space. Such guarantees are natural for the underlying networking applications; essentially, a very small fraction of the location-based lookups may yield poor performance (due to the arbitrary distortion), but for the rest the quality of the embedding will be very good.

These types of results form a suggestive contrast with the theoretical work on embeddings. In particular, are the strong empirical guarantees for Internet latencies the result of fortuitous artifacts of this particular set of distances, or is something more general going on? To address this, Kleinberg, Slivkins, and Wexler [28] defined the notion of embeddings with *slack*: In addition to the metrics (V, d) and (V', d') in the initial formulation above, we are also given a *slack parameter* ϵ, and we want to find a map φ whose distortion is bounded by some quantity $D(\epsilon)$ on all but an ϵ fraction of the pairs of points in $V \times V$. (Note that we allow the distortion on the remaining ϵn^2 pairs of points to be arbitrarily large.) Roughly, Kleinberg, Slivkins, and Wexler [28] showed that any metric of bounded doubling dimension—in which every ball can be covered by a constant number of balls of half the radius—can be embedded with

[1]Formally, for an embedding $\varphi : V \to V'$, the *distortion* is the smallest D so that $\exists\, \alpha, \beta \geq 1$ with $\alpha \cdot \beta \leq D$ such that $\frac{1}{\alpha}\, d(x, y) \leq d'(\varphi(x), \varphi(y)) \leq \beta\, d(x, y)$ for all pairs $x, y \in V \times V$. Note that this definition of distortion is slightly nonstandard—since $\alpha, \beta \geq 1$, it is no longer invariant under arbitrary scaling; however, this is merely for notational convenience, and all of our results can be cast in the usual definitions of distortion.

[2]While the triangle inequality can be violated by network latencies, empirical evidence suggests that these violations are small and/or infrequent enough to make metric methods a useful approach.

constant distortion into constant-dimensional Euclidean space, allowing a constant slack ϵ. Such metrics, which have been extensively studied in their own right, have also been proposed on several occasions as candidates for tractable abstractions of the set of Internet latencies (see, e.g., [16, 26, 37, 39]).

There were two main open questions posed in [28].

(1) There was no evidence that the main embedding result of [28] needed to be restricted to metrics of bounded doubling dimension. Could it be the case that for *every* finite metric space, and every $\epsilon > 0$, there is an embedding of the metric with distortion $f(\epsilon)$ into Euclidean space?
(2) Rather than have the embedding depend on the given slack parameter ϵ, a much more flexible and powerful alternative would be to have a *single* embedding of the metric with the property that, for some (slowly growing) function $D(\cdot)$, it achieved distortion $D(\epsilon)$ on all but an ϵ fraction of distance pairs for all $\epsilon > 0$. We call such an embedding *gracefully degrading* [28] and ask whether such an embedding (with a polylogarithmic function $D(\cdot)$) could exist for all metrics.

In this paper we resolve the first of these questions in the affirmative, showing constant distortion with constant slack for all metrics. Moreover, the embedding we design to achieve this guarantee is *beacon-based*, requiring only the measurement of distances involving a small set of distinguished "beacon nodes"; see section 2. Approaches that measure only a small number of distances are crucial in networking applications, where the full set of distances can be enormous; see, e.g., [21, 17, 29, 37, 38, 43] for beacon-based approaches and further discussions. We then resolve the second question in the affirmative for metrics that admit an $O(1)$-padded decomposition (a notion from previous work on embeddings that we specify precisely in section 1.1); this includes several well-studied classes of metrics including those with bounded doubling dimension and those arising from the shortest-path metric of a graph with an excluded minor. We further show that gracefully degrading distortion can be achieved in the ℓ_1 norm for all metrics. The second question has been subsequently solved in full in [2] (see also the bibliographic notes in what follows), providing an embeddings with gracefully degrading distortion for all metrics in ℓ_p for every $p \geq 1$. Finally, we show that many of our constructions are tight and give a general technique to obtain lower bounds for ϵ-slack embeddings from lower bounds for low-distortion embeddings.

Basic definitions. Before we formally present our results, let us present some of the notions that will be used throughout the paper. We will assume that the metric space (V, d) is also represented as a graph on the nodes V, with the length of edge uv being $d(u, v) = d_{uv}$. We imagine this graph as having n^2 edges, one for each pair $u, v \in V \times V$; this makes the exposition cleaner and does not change the results in any significant way. For a map $\varphi : V \to V'$ let us define the notion of the *distortion of a set S of edges* under embedding φ as the smallest $D \geq 1$ such that for some positive constant K and all edges $(u, v) \in S$ we have

$$d(u, v) \leq d'(\varphi(u), \varphi(v))/K \leq D \cdot d(u, v).$$

Note that the distortion of φ (as given in Footnote 1) is the same as the distortion of the set of all edges.

DEFINITION 1.1 (ϵ-slack distortion). *Given ϵ, an embedding $\varphi : V \to V'$ has* distortion D with ϵ-slack *if a set of all but an ϵ-fraction of edges has distortion at most D under φ.*

We will also consider a stronger notion of slack, for which we need the following definition. Let $\rho_u(\epsilon)$ be the radius of the smallest ball around u that contains at least ϵn nodes. Call an edge uv *ϵ-long* if $d_{uv} \geq \min(\rho_u(\epsilon), \rho_v(\epsilon))$. Then there are at least $(1-\epsilon)\,n^2$ edges that are ϵ-long. For any such edge uv, at least one end point u is at least as far from the other end point v as the (ϵn)th closest neighbor of v.

DEFINITION 1.2 (ϵ-uniform slack distortion). *Given ϵ, an embedding $\varphi : V \to V'$ has* distortion D with ϵ-uniform slack *if the set of all ϵ-long edges has distortion at most D.*

Note that for an ϵ-uniform slack embedding, the number of ignored edges incident on any node is at most ϵn.

While the above notions of embeddings with slack allow the map φ to depend on the slack ϵ, the following notion asks for a single map that is *good for all ϵ simultaneously.*

DEFINITION 1.3 (gracefully degrading distortion). *An embedding $\psi : V \to V'$ has a* gracefully degrading distortion $D(e)$ *if, for each $\epsilon > 0$, the distortion of the set of all ϵ-long edges is at most $D(\epsilon)$.*

Our results. We now make precise the main results described above and also describe some further results in the paper. Our first result shows that if we are allowed constant slack, we can indeed embed *any* metric space into constant dimensions with constant distortion.

THEOREM 1.4. *For any source metric space (V, d), any target metric space ℓ_p, $p \geq 1$, and any parameter $\epsilon > 0$, we give the following two $O(\log \frac{1}{\epsilon})$-distortion embeddings:*

(a) *with ϵ-slack into $O(\log^2 \frac{1}{\epsilon})$ dimensions, and*

(b) *with ϵ-uniform slack into $O(\log n \log \frac{1}{\epsilon})$ dimensions.*

Both embeddings can be computed with high probability by randomized beacon-based algorithms.

These results extend Bourgain's theorem on embedding arbitrary metrics into ℓ_p, $p \geq 1$, with distortion $O(\log^2 n)$ [10] and are proved in a similar manner.

Note that the bounds on both the distortion as well as the dimension in Theorem 1.4(a) are independent of the number of nodes n, which suggests that they could be extended to infinite metrics; this is further discussed in section 2. In part (b), the dimension is proportional to $\log n$; we show that, for arbitrary metrics, this dependence on n is indeed inevitable. As an aside, let us mention that metrics of bounded doubling dimension do not need such a dependence on n: In Slivkins [43], these metrics are embedded into any ℓ_p, $p \geq 1$, with ϵ-uniform slack, distortion $O(\log \frac{1}{\epsilon} \log\log \frac{1}{\epsilon})$, and dimension $(\log \frac{1}{\epsilon})^{O(\log \frac{1}{\epsilon})}$.

We then study *embeddings into trees.* We extend the known results of probabilistic embedding into trees [5, 6, 14, 7] to obtain embeddings with slack. In particular, we use the technique of Fakcharoenphol, Rao, and Talwar [14] to obtain the following two results.

THEOREM 1.5. *For any input metric space (V, d) and any parameter $\epsilon > 0$ there exists an embedding into a tree metric with ϵ-uniform slack and distortion $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.*

In fact, the tree metric in Theorem 1.5 is induced by a *hierarchically separated tree* (*HST*) [5], which is a rooted tree with edge weights w_e such that $w_e < w_{e'}/2$ whenever edge e' is on the path from the root to edge e.

THEOREM 1.6. *For any input metric space (V, d), the randomized embedding of* [14] *into tree metrics has expected gracefully degrading distortion $D(\epsilon) = O(\log \frac{1}{\epsilon})$.*[3]

[3]More formally, we show that if an edge uv is ϵ-long, then $d_{uv} \leq E_T[d_T(u,v)] \leq O(\log \frac{1}{\epsilon})\, d_{uv}$, where d_T is the tree metric generated by the randomized algorithm in [14].

Since tree metrics are isometrically embeddable into L_1, this immediately implies that we can embed any metric into L_1 with gracefully degrading distortion $D(\epsilon) = O(\log \frac{1}{\epsilon})$.

However, the dimension of the above embedding into L_1 may be prohibitively large. To overcome this hurdle and to extend this embedding to ℓ_p, $p > 1$, we explore a different approach.

THEOREM 1.7. *Consider a metric space (V, d) which admits β-padded decompositions. Then it can be embedded into ℓ_p, $p \geq 1$, with $O(\log^2 n)$ dimensions and gracefully degrading distortion $D(\epsilon) = O(\beta)(\log \frac{1}{\epsilon})^{1/p}$.*

For the reader unfamiliar with padded decompositions, let us mention that $\beta \leq O(\dim_V)$, the doubling dimension of the metric, which in turn is always bounded above by $O(\log n)$. Moreover, doubling metrics and metrics induced by planar graphs have $\beta = O(1)$; hence Theorem 1.7 implies that such metrics admit embeddings into ℓ_p, $p \geq 1$, with gracefully degrading distortion $O(\log \frac{1}{\epsilon})^{1/p}$. Note that for $p > 1$ this result can be seen as a strengthening of Theorem 1.4(b) on embeddings with ϵ-uniform slack.

The proof of Theorem 1.7 is technically the most involved part of the paper; at a high level, we develop a set of scale-based embeddings which are then combined together (as in most previous embeddings)—however, since the existing ways to perform this do not seem to guarantee gracefully degrading distortion, we construct new ways of defining distance scales.

Finally, we prove *lower bounds* on embeddings with slack: We give a very general theorem that allows us to convert lower bounds on the distortion and dimension of embeddings that depend only on $n = |V|$ into lower bounds in terms of the slack parameter ϵ. This result works under very mild conditions and allows us to prove matching or nearly matching lower bounds for all of our results on ϵ-slack embeddings. These lower bounds are summarized in Table 5.1 on page 2322.

Related work. This work is closely related to the large body of work on metric embeddings in theoretical computer science; see the surveys [24, 25] for a general overview of the area. Our results build on much of the previous work on embeddings into ℓ_p, including [10, 33, 41, 34, 19, 30, 31], and on embeddings of metrics into distributions of trees [3, 5, 6, 20, 14, 7]. Among the special classes of metrics we consider are *doubling metrics* [4, 19, 44, 22]; the book by Heinonen [23] gives more background on the analysis of metric spaces.

All of these papers consider low-distortion embeddings *without slack*. Note that an embedding with $(\epsilon = 1/2n^2)$-slack or $(\epsilon = 1/2n)$-uniform-slack is the same as an embedding with no slack; for many of our results, plugging in these values of ϵ gives us the best known slackless results—hence our results can be viewed as extensions of these previous results.

The notion of embedding with slack can be viewed as a natural variant of metric Ramsey theory. The first work on metric Ramsey-type problems was by Bourgain, Figiel, and Milman [11], and a comprehensive study was more recently developed by Bartal and coworkers [8, 9]. In the original metric Ramsey problem we seek a large subset of the points in the metric space which admit a low-distortion embedding, whereas an embedding with slack provides low distortion for a subset of the pairs of points.

Bibliographic note. The results in this paper have been obtained independently by Abraham, Bartal, and Neiman, which led to a merged publication [1]. The results on lower bounds (section 5) and on embedding into distributions of trees (Theorem 1.6) were proved similarly by both groups. For the rest of the results, the

techniques are quite different. The two groups of authors have agreed to write up the full versions of their results separately.

Extensions and further directions. The main question left open by this work is whether every metric admits a low-dimensional embedding into ℓ_p, $p \geq 1$, with gracefully degrading distortion $D(\epsilon)$. This has been answered affirmatively in Abraham, Bartal, and Neiman [2], with $D(\epsilon) = O(\log \frac{1}{\epsilon})$ and dimension $O(\log n)$, using a new type of more advanced metric decomposition. They also show a tight result of $O(1/\sqrt{\epsilon})$ distortion for ϵ-slack embedding into a tree metric and improve the distortion in Theorem 1.7 by a factor of $\beta^{1/p}$.

For specific families of metrics it is still interesting to provide embeddings into ℓ_p with gracefully degrading distortion $D(\epsilon) = o(\log \frac{1}{\epsilon})$; recall that Theorem 4.1 gives such embeddings for decomposable metrics. In particular, we would like to ask this question for embedding arbitrary subsets of ℓ_1 into ℓ_2.

1.1. Notation and preliminaries. Throughout the paper (V, d) is the metric space to be embedded, and $d_{uv} = d(u, v)$ is the distance between nodes $u, v \in V$. Define the closed ball $\mathbf{B}_u(r) = \{v \in V \mid d_{uv} \leq r\}$. The distance between a node u and set $S \subseteq V$ is denoted $d(u, S) = \min_{v \in S} d_{uv}$, and hence $d(u, V \setminus \mathbf{B}_u(r)) > r$. We will assume that the smallest distance in the metric is 1 and the largest distance (or the diameter) is Φ_d.

A *coordinate map* f is a function from V to $\mathbb{R}$; for an edge uv define $f(uv) = |f(u) - f(v)|$. Call such map 1-*Lipschitz* if for every edge $f(uv) \leq d_{uv}$. For $k \in \mathbb{N}$ define $[k]$ as the set $\{0, 1, \ldots, k-1\}$.

Doubling metrics and measures. A metric space (V, d) is *s-doubling* if every set $S \subseteq V$ of diameter Δ can be covered by s sets of diameter $\Delta/2$; the *doubling dimension* of such a metric is $\lceil \log s \rceil$ [23, 19]. A doubling metric is one whose doubling dimension is bounded. A *measure* is *s-doubling* if the measure of any ball $B_u(r)$ is at most s times larger than the measure of $B_u(r/2)$. It is known that for any s-doubling metric there exists an s-doubling measure; moreover, such measure can be efficiently computed [23, 22].

Padded decompositions. Let us recall the definition of a *padded decomposition* (see, e.g., [19, 30]). Given a finite metric space (V, d), a positive parameter $\Delta > 0$, and $\beta : V \to \mathbb{R}$, a Δ-bounded *β-padded decomposition* is a distribution Π over partitions of V such that the following conditions hold.

(a) For each partition P in the support of Π, the diameter of every cluster in P is at most Δ.

(b) If P is sampled from Π, then each ball $\mathbf{B}_x(\frac{\Delta}{\beta(x)})$ is partitioned by P with probability at most $\frac{1}{2}$.

For simplicity say that a metric *admits β-padded decompositions* (where β is a number) if for every $\Delta > 0$ it admits a Δ-bounded β-padded decomposition. It is known that any finite metric space admits $O(\log n)$-padded decomposition [5]. Moreover, metrics of doubling dimension $\dim_V$ admit $O(\dim_V)$-padded decompositions [19]; furthermore, if a graph G excludes a K_r-minor (e.g., if it has treewidth $\leq r$), then its shortest-path metric admits $O(r^2)$-padded decompositions [27, 41, 15].

2. Embeddings with slack into ℓ_p. In this section we show that for any $\epsilon > 0$ any metric can be embedded into ℓ_p for $p \geq 1$ with ϵ-slack and distortion $O(\log \frac{1}{\epsilon})$, thus resolving one of the two main questions left open by [28].

Let us fix $\epsilon > 0$ and write $\rho_u = \rho_u(\epsilon)$. Recall that an edge uv is *ϵ-long* if $d_{uv} \geq \min(\rho_u, \rho_v)$; call it *$\epsilon$-good* if $d_{uv} \geq 4 \min(\rho_u, \rho_v)$. We partition all of the ϵ-long

edges into two groups, namely, those which are ϵ-good and those which are not, and use a separate embedding (i.e., a separate block of coordinates) to handle each of the groups. Specifically, we handle ϵ-good edges using a Bourgain-style embedding from [28], and for the rest of the ϵ-long edges we use an auxiliary embedding such that for any edge uv the embedded uv-distance is $\Theta(\rho_u+\rho_v)$. The combined embedding has dimension $O(\log^2 \frac{1}{\epsilon})$ and achieves distortion $O(\log \frac{1}{\epsilon})$ on a set of all but an ϵ-fraction of edges.

There are several ways in which this result can be refined. First, we can ask for low ϵ-*uniform-slack* distortion and require distortion $O(\log \frac{1}{\epsilon})$ on the set of all ϵ-*long* edges; we can indeed get this, but we have to boost the number of dimensions to $O(\log n \log \frac{1}{\epsilon})$. As Theorem 2.2 shows, this increase is indeed required. We note that this logarithmic increase in the number of dimensions is not the case for doubling metrics: Slivkins [43] shows how these metrics are embedded into any ℓ_p, $p \geq 1$, with ϵ-uniform slack, distortion $O(\log \frac{1}{\epsilon} \log\log \frac{1}{\epsilon})$, and dimension $(\log \frac{1}{\epsilon})^{O(\log \frac{1}{\epsilon})}$.

Second, this embedding can be computed in a distributed *beacon-based* framework. Here a small number of nodes are selected independently and uniformly at random and designated as *beacons*. Then the coordinates of each node are computed as a (possibly randomized) function of its distances to the beacons.

Third, note that, for the ϵ-slack result, the target dimension is independent of n, which suggests that this result can be extended to infinite metrics. To state such an extension, let us modify the notion of slack accordingly. Following [43], let us assume that an infinite metric space is equipped with a probability measure μ on nodes. This measure induces a *product measure* $\mu \times \mu$ on edges. We say that a given embedding ϕ has *distortion D with (ϵ, μ)-slack* if some set of edges of product measure at least $1-\epsilon$ incurs distortion at most D under ϕ. Note that, in the finite case, ϵ-slack coincides with (ϵ, μ)-slack when μ is the counting measure, i.e., when all nodes are weighted equally.

In the embedding algorithm, instead of selecting beacons uniformly at random (i.e., with respect to the counting measure) we select them with respect to measure μ. The proof carries over without much modification; we omit it from this version of the paper.

THEOREM 2.1. *For any source metric space (V, d), any target metric space ℓ_p, $p \geq 1$, and any parameter $\epsilon > 0$, we give the following two $O(\log \frac{1}{\epsilon})$-distortion embeddings:*

(a) *with ϵ-slack into $O(\log^2 \frac{1}{\epsilon})$ dimensions, and*

(b) *with ϵ-uniform slack into $O(\log n \log \frac{1}{\epsilon})$ dimensions.*

These embeddings can be computed with high probability by randomized beacon-based algorithms that use, respectively, only $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ and $O(\frac{1}{\epsilon} \log n)$ beacons.

Proof. Let $\delta > 0$ be the desired total failure probability. The embedding algorithm is essentially the same for both parts, with one difference: We let $k = O(\log \frac{1}{\delta} + \log \frac{1}{\epsilon})$ for part (a) and $k = O(\log \frac{1}{\delta} + \log n)$ for part (b). We describe a centralized algorithm first and prove that it indeed constructs the desired embedding. Then we show how to make this algorithm beacon-based.

We use two blocks of coordinates of size kt and k, respectively, where $t = \lceil \log \frac{1}{\epsilon} \rceil$. The first block comes from a Bourgain-style embedding without the smaller distance scales. For each $i \in [t]$ choose k independent random subsets of V of size 2^i each; call them S_{ij}, $j \in [k]$. The first-block coordinates of a given node u are

$$f_{ij}(u) = (kt)^{-1/p}\, d(u, S_{ij}), \text{ where } i \in [t],\ j \in [k]. \tag{2.1}$$

For every node u and every $j \in [k]$, choose a number $\beta_{ui} \in \{-1, 1\}$ independently and

uniformly at random. The second-block coordinates of u are $g_j(u) = k^{-1/p}\,\rho_u\,\beta_{uj}$, where $j \in [k]$. This completes the embedding.

For an edge uv, let $f(uv)$ and $g(uv)$ denote the ℓ_p-distance between u and v in the first and the second block of coordinates, respectively. By construction, $f(uv) \le d_{uv}$ and $g(uv) \le \rho_u + \rho_v$. Moreover,

(2.2) for every ϵ-good edge uv, $f(uv) \ge \Omega(d_{uv}/t)$ with probability $\ge 1 - t/2^{\Omega(k)}$.

Indeed, fix an ϵ-good edge uv and let $d = d_{uv}$. Let α_i be the minimum of the following three quantities: $\rho_u(2^{-i})$, $\rho_v(2^{-i})$, and $d/2$. The numbers α_i are nonincreasing; $\alpha_0 = d/2$. Moreover, since uv is ϵ-good, we have $\alpha_t \le \min(\rho_u, \rho_v, d/2) \le d/4$. By a standard Bourgain-style argument it follows that for each i the event

$$\sum_j |d(u, S_{ij}) - d(v, S_{ij})| \ge \Omega(k)(\alpha_i - \alpha_{i+1})$$

happens with failure probability at most $1/2^{\Omega(k)}$. (We omit the details from this version of the paper.) Therefore, with failure probability at most $t/2^{\Omega(k)}$, this event happens for all $i \in [t]$ simultaneously, in which case

$$\sum_{ij} |d(u, S_{ij}) - d(v, S_{ij})| \ge \sum_{i \in [t]} \Omega(k)(\alpha_i - \alpha_{i+1}) = \Omega(k)(\alpha_0 - \alpha_t) \ge \Omega(kd),$$

so $f(uv) \ge \Omega(d/t)$ for the case $p = 1$. It is easy to extend this to $p > 1$ using standard inequalities. This proves the claim (2.2).

Furthermore, we claim that for each edge uv, $g(uv) = \Omega(\rho_u + \rho_v)$ with failure probability at most $1/2^{\Omega(k)}$. Indeed, let N_j be the indicator random variable for the event $\beta_{uj} \ne \beta_{vj}$. Since N_j's are independent and their sum N has expectation $k/2$, by Chernoff bounds (Lemma A.1(a)) $N \ge k/4$ with the desired failure probability. This completes the proof of the claim.

Now fix an ϵ-long edge uv and let $d = d_{uv}$. Without loss of generality assume $\rho_u \le \rho_v$; note that $\rho_u \le d$. Since $B_u(\rho_u) \subset B_v(\rho_u + d)$, the cardinality of the latter ball is at least ϵn. It follows that $\rho_v \le \rho_u + d$, so $g(uv) \le \rho_u + \rho_v \le 3d$. Since $f(uv) \le d$, the embedded uv-distance is $O(d)$.

To lower-bound the embedded uv-distance, note that with failure probability at most $t/2^{\Omega(k)}$ the following happens: If edge uv is ϵ-good, then this distance is $\Omega(d/t)$ due to $f(uv)$; else it is $\Omega(d)$ due to $g(uv)$. For part (a) we use the Markov inequality to show that with failure probability at most δ this happens for all but an ϵ-fraction of ϵ-long edges. For part (b) we take a union bound to show that with failure probability at most δ this happens for *all* ϵ-long edges. This completes the proof of correctness for the centralized embedding.

It remains to provide the beacon-based version of the algorithm. Let S be the union of all sets S_{ij}. The Bourgain-style part of the algorithm depends only on distances to the $\Theta(k/\epsilon)$ nodes in S, so it can be seen as beacon-based, with all nodes in S acting as beacons. To define the second block of coordinates we need to know the ρ_u's, which we do not. However, we will estimate them using the same set S of beacons.

Fix a node u. Let B be the open ball around u of radius ρ_u, i.e., the set of all nodes v such that $d_{uv} < \rho_u$. Let B' be the smallest ball around u that contains at least $4\epsilon n$ nodes. Note that S is a set of ck/ϵ beacons chosen independently and uniformly at random for some constant c.

In expectation at most ck beacons land in B and at least $4ck$ beacons land in B'. By Chernoff bounds (Lemma A.1(a) and (b) with failure probability at most $1/2^{\Omega(k)}$ the following event E_u happens: At most $2ck$ beacons land in B and at least $2ck$ beacons land in B'. Rank the beacons according to their distance from u, and let w be the $(2ck)$th closest beacon. Define our estimate of ρ_u as $\rho'_u = d_{uw}$. Note that if event E_u happens, then ρ'_u lies between ρ_u and $\rho_u(4\epsilon)$.

Consider a 4ϵ-good edge uv such that both E_u and E_v happen. Then (as in the non-beacon-based proof) we can upper-bound the embedded uv-distance by $O(d_{uv})$ and lower-bound it by $\Omega(d_{uv}/t)$ with high probability. For part (a) we use the Markov inequality to show that with failure probability at most δ event E_u happens for all but an ϵ-fraction of nodes. For part (b) we take a union bound to show that with failure probability at most δ this event happens for all nodes. □

The following theorem lower-bounds the target dimension required for ϵ-uniform slack, essentially showing that in part (b) of the above theorem the dependence on $\log n$ is indeed necessary.

THEOREM 2.2. *For any $\epsilon < \frac{1}{2}$ there is a metric space (V, d) such that any ϵ-uniform slack embedding into l_p, $p \geq 1$, with distortion D requires $\Omega(\log_D n)$ dimensions.*

Proof. Take a clique on $\epsilon\, n$ red and $(1-\epsilon)n$ blue nodes, assign length two to each of the blue-blue edges, and assign unit lengths to all of the remaining edges. Consider the metric induced by this graph. Now all of the blue-blue edges are ϵ-long, and thus any distortion-D ϵ-uniform-slack embedding must maintain all of the distances between the blue vertices. But this is just a uniform metric on $(1-\epsilon)n$ nodes, and the lower bound follows by a simple volume argument. □

3. Embeddings into trees. Probabilistic embedding of finite metric space into trees was introduced in [5]. Fakcharoenphol, Rao, and Talwar [14] proved that finite metric space embeds into a distribution of dominating trees with distortion $O(\log n)$ (slightly improving the result of [6]); other proofs can be found in [7]. In this section we exploit the technique of [14] to obtain embeddings with slack. First we show that it gives a probabilistic embedding of arbitrary metrics into tree metrics with *expected* gracefully degrading distortion $D(\epsilon) = O(\log 1/\epsilon)$. For technical convenience, we will treat n-point metrics as functions from $[n] \times [n]$ to reals. Note that all metrics d_T generated by the algorithm in [14] are *dominating*; i.e., for any edge uv we have $d(u, v) \leq d_T(u, v)$.

THEOREM 3.1. *For any input metric space (V, d), let d_T be the dominating HST metric on V constructed by the randomized algorithm in Fakcharoenphol, Rao, and Talwar* [14]. *Then the embedding from (V, d) to (V, d_T) has* expected *gracefully degrading distortion $D(\epsilon) = O(\log 1/\epsilon)$. Specifically, for any parameter $\epsilon > 0$ and any ϵ-long edge uv we have*

$$d_{uv} \leq E_\varphi[d_T(u, v)] \leq O(\log 1/\epsilon)\, d_{uv}. \tag{3.1}$$

Since tree metrics are isometrically embeddable into L_1, it follows that we can embed any metric into L_1 with gracefully degrading distortion $D(\epsilon) = O(\log \frac{1}{\epsilon})$.

Proof. For simplicity let us assume that all distances in (V, d) are distinct; otherwise we can perturb them a little bit and make them distinct, without violating the triangle inequality; see the full version of this paper for details. In what follows we will assume a working knowledge of the decomposition scheme in [14].

Let us fix the parameter $\epsilon > 0$ and an ϵ-long edge uv, and let $d = d(u, v)$. Let us assume without loss of generality that $\rho_u(\epsilon) \leq \rho_v(\epsilon)$. Then $\rho_u(\epsilon) \leq d$, so $|\mathbf{B}_u(d)| \leq \epsilon n$.

Run the randomized algorithm of [14] to build a tree T and the associated tree metric d_T. The decomposition scheme will separate u and v at some distance scale $2^i \geq d/2$. Let Δ be the maximum distance in the input metric. Under the distribution over tree metrics d_T that is induced by the algorithm, the expected distance $E[d_T(u,v)]$ between u and v in tree T is equal to the sum

$$\sum_{i \geq \log d - 1}^{\log \Delta} 4 \cdot 2^i \times \Pr[(u,v) \text{ first separated at level } 2^i].$$

Look at the sum for i such that $d/2 \leq 2^i < 4d$; this is at most $48d$. By the analysis of [14], the rest of the sum, i.e., the sum for $i \geq \log 4d$, is at most

$$\sum_{i \geq \log 4d}^{\log \Delta} 4 \cdot 2^i \times \frac{2d}{2^i} \log \frac{|\mathbf{B}_u, 2^i)|}{|\mathbf{B}_u, 2^{i-2})|}.$$

Since the above sum telescopes, it is at most

$$8d \cdot 2 \log (n/|\mathbf{B}_u(d)|) \leq O(d \log 1/\epsilon),$$

which proves the second inequality in (3.1). The first inequality in (3.1) holds trivially because all metrics d_T generated by the algorithm in [14] are dominating. ∎

The above embedding into ℓ_1 can be made algorithmic by sampling from the distribution and embedding each sampled tree into ℓ_1 using a fresh set of coordinates; however, the number of trees now needed to give a small distortion may be as large as $\Omega(n \log n)$. We will see how to obtain gracefully degrading distortion with a smaller number of dimensions in the next section.

A slightly modified analysis yields an embedding into a *single tree*.

THEOREM 3.2. *For any source metric space (V,d) and any parameter $\epsilon > 0$ there exists an embedding into a dominating HST metric with ϵ-uniform slack and distortion $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.*

4. Low-dimensional embeddings with gracefully degrading distortion. In this section we prove our result on embeddings into ℓ_p, $p \geq 1$, with gracefully degrading distortion.

THEOREM 4.1. *Consider a metric space (V,d) which admits β-padded decompositions. Then it can be embedded into ℓ_p, $p \geq 1$, with $O(\log^2 n)$ dimensions and gracefully degrading distortion $D(\epsilon) = O(\beta)(\log \frac{1}{\epsilon})^{1/p}$. The embedding procedure is given as a randomized algorithm which succeeds with high probability.*

The proof of this theorem builds on the well-known embedding algorithms of Bourgain [10] and Linial, London, and Rabinovich [33] and combines ideas given in [41, 19, 28, 43, 30] with some novel ones. To the best of our understanding, the embeddings given in the previous papers do not directly give us gracefully degrading distortion, and hence the additional machinery indeed seems to be required.

Let us fix $k = O(\log n)$, where the constant will be specified later. We will construct an embedding $\varphi : V \to \ell_p$ with $7k^2$ dimensions; the coordinates of φ will be indexed by triples $(i,j,l) \in [k] \times [k] \times [7]$.

We will show how to construct the map φ in the rest of this section, which has the following conceptual steps. We first define a concrete notion of "distance scales" in section 4.1, in terms in which we can cast many previous embeddings, and specify the desired properties for the distance scales in our embedding. We then show how to construct the distance scales as well as the claimed embedding φ in section 4.2 and show that it has gracefully degrading distortion in section 4.3.

4.1. Distance scales and scale bundles. Our algorithm, just like the algorithms in [10, 33, 41, 19, 28, 30, 31], operates on distance scales that start around the diameter of the metric and go all the way down to the smallest distance in the metric. Informally, the embedding φ has a block of coordinates for each distance scale such that if the true uv-distance for some edge uv is within this scale, then the uv-distance in these coordinates of φ is roughly equal to the true distance. These blocks of coordinates are then combined into an embedding that works for all scales simultaneously.

Different embeddings use very different notions of distance scales; in cases like the Rao-style embeddings [41, 19], there are clear coordinates for each distance that is a power of 2—but in Bourgain-style embeddings, this is not the case. To be able to give a unified picture, let us formally define a *distance scale* f to be a coordinate map $f : V \to \mathbb{R}$. A *scale bundle* $\{f_{ij}\}$ is then a collection of coordinate maps f_{ij} such that, for every fixed index j and node u, the values $f_{ij}(u)$ are *decreasing* with i.

We can now cast and interpret previous embeddings in this language: In the Bourgain-style embeddings [10, 33], $f_{ij}(u)$ is the radius of the smallest ball around u containing 2^{n-i} nodes, and hence the cardinality of $\mathbf{B}_u(f_{ij}(u))$ halves as we increase i. In the Rao-style embeddings, the scales are $f_{ij}(u) = \text{diameter}(V)/2^i$, and hence the distance scales halve as we increase i. The measured descent embedding in [30] essentially ensures a judicious mixture of the above two properties: As we increase i, the ball $\mathbf{B}_u(f_{ij}(u))$ either halves in radius or halves in cardinality, whichever comes first.

For our embedding, we need *both* the radius and the cardinality of $\mathbf{B}_u(f_{ij}(u))$ to halve—and hence we have to define the scale bundles accordingly. This would be easy to achieve by itself; however, to give good upper bounds on the embedded distance, we also need each distance scale to be sufficiently smooth, by which we mean that all of the distance scales f_{ij} must themselves be 1-Lipschitz. In other words, we want that $|f_{ij}(u) - f_{ij}(v)| \le d(u, v)$. The construction of the scale bundle $\{f_{ij}\}$ with both halving and smoothness properties turns out to be a bit nontrivial, the details of which are given in the next section.

4.2. The embedding algorithm. Let us construct the embedding for Theorem 4.1. We have not attempted to optimize the multiplicative constant for distortion, having chosen the constants for ease of exposition while ensuring that the proofs work.

First we will construct a *scale bundle* $\{f_{ij} : i, j \in [k]\}$. For a fixed j, the maps f_{ij} are constructed by an independent random process, inductively from $i = 0$ to $i = k - 1$. We start with $f_{(0,j)}(\cdot)$ equal to the diameter Φ_d of the metric. Given f_{ij}, we construct $f_{(i+1,j)}$ as follows. Let U_{ij} be a random set such that each node u is included independently with probability $1/|\mathbf{B}_u(4f_{ij}(u))|$. Assuming U_{ij} is nonempty, define $f_{(i+1,j)}(u)$ as the minimum of $d(u, U_{ij})$ and $f_{ij}(u)/2$. If U_{ij} is empty, set $f_{(i+1,j)}(u) = f_{ij}(u)/2$. This completes the construction of the scale bundle.

To proceed, let us state a lemma that captures, for our purposes, the structure of the source metric space: This is the only place in the proof of Theorem 4.1 where we use padded decomposition.

LEMMA 4.2. *Consider a metric space (V, d) which admits β-padded decompositions. Then for any 1-Lipschitz coordinate map f and any $p \ge 1$ there is a randomized embedding g into ℓ_p with $t = 6$ dimensions so that*

(a) *each coordinate of g is positive, 1-Lipschitz, and upper-bounded by f; and*
(b) *if $f(u)/d_{uv} \in [\frac{1}{4}; 4]$ for some edge uv, then, with probability $\Omega(1)$,*

$$\|g(u) - g(v)\|_p \ge \Omega(d_{uv}\, t^{1/p}/\beta). \tag{4.1}$$

In short, this lemma transforms a "smooth" distance scale f into a "smooth" low-dimensional embedding g which approximately preserves distances along the "relevant" edges. Here "smooth" means "1-Lipschitz," an edge (u, v) is relevant to f if $d_{uv} \approx f(u)$ or $d_{uv} \approx f(v)$, and distances are preserved in the sense of (4.1). Once the relevant edges are taken care of, we want the coordinates of g to be as small as possible in order to upper-bound the embedded distance on larger distance scales.

Sections 4.4 and 4.6 contain two different proofs of this lemma; the first one uses padded decomposition techniques from [19, 30], and the other uses some Bourgain-style ideas [10, 33] which we believe are novel and possibly of independent interest.[4]

Fix a pair $i, j \in [k]$. Apply Lemma 4.2 to the map f_{ij}, and obtain a 6-dimensional embedding; denote these 6 coordinates as $g_{(i,j,l)}$, $1 \le l \le 6$. Let W_{ij} be a random set such that each node u is included independently with probability $1/|\mathbf{B}_u(f_{ij}(u)/2)|$. Define $g_{(i,j,0)}(u)$ as the minimum of $f_{ij}(u)$ and $d(u, W_{ij})$. Finally, we set

$$\varphi_{(i,j,l)} = k^{-1/p}\, g_{(i,j,l)}. \tag{4.2}$$

LEMMA 4.3. *The maps f_{ij}, g_{ij}, and $\varphi_{(i,j,l)}$ are 1-Lipschitz.*

Proof. Indeed, $f_{(0,j)}$ is 1-Lipschitz by definition, and the inductive step follows since the min of two 1-Lipschitz maps is 1-Lipschitz. For the same reason, the maps $g_{(i,j,l)}$ are 1-Lipschitz as well, and therefore so are the maps $\varphi_{(i,j,l)}$. □

Since $k = O(\log n)$, it immediately follows that the embedded distance is at most $O(\log n)$ times the true distance. In the next section we will prove a sharper upper bound of $O(d_{uv})(\log \frac{1}{\epsilon})^{1/p}$ for any ϵ-long edge uv and a lower bound $\Omega(d_{uv}/\beta)$ for any edge.

4.3. Analysis. In this section we complete the proof of Theorem 4.1 by giving bounds on the stretch and contraction of the embedding φ. The following definition will be useful: For a node u, an interval $[a, b]$ is *u-broad* if a or b is equal to d_{uv} for some v, $a \le b/4$ and $|\mathbf{B}_u(a)| \le \frac{1}{32}|\mathbf{B}_u(b)|$.

Let us state two lemmas that capture the useful properties of the maps f_{ij} and $g_{(i,j,0)}$, respectively; note that these properties are independent of the doubling dimension of the underlying metric. The proofs are deferred to section 4.5.

LEMMA 4.4. *With high probability it is the case that*

(a) *for any 1-Lipschitz maps $f'_{ij} \le f_{ij}$ and any ϵ-long edge uv it is the case that $\sum_{ij} f'_{ij}(uv) \le O(k d_{uv} \log \frac{1}{\epsilon})$.*

(b) *for each node u each u-broad interval contains values f_{ij} for $\Omega(k)$ different values of j.*

LEMMA 4.5. *Fix edge uv and indices ij; let $R = f_{ij}(u)$ and $d = d_{uv}$. Given that $R \ge 4d$ and $|\mathbf{B}_u(d/4)| = c\,|\mathbf{B}_u(R)|$, the event $g_{(i,j,0)}(uv) \ge \Omega(d)$ happens with conditional probability $\Omega(c)$.*

Proof of Theorem 4.1. Recall that the final embedding φ is defined by (4.2). Fix an ϵ-long edge uv, and let $d = d_{uv}$. Since $g_{(i,j,l)} \le f_{ij}$ for each l, by Lemma 4.4(a) the embedded uv-distance is upper-bounded by $O(d \log \frac{1}{\epsilon})$ for $p = 1$; the same argument gives an upper bound of $O(d)(\log \frac{1}{\epsilon})^{1/p}$ for $p > 1$.

It remains to lower-bound the embedded uv-distance by $\Omega(d/\beta)$, where β is the parameter in Theorem 4.1 and Lemma 4.2. Denote by $g_{ij}(uv)$ the total ℓ_p-distance between u and v in the coordinates $g_{(i,j,l)}$, $l \ge 1$. Denote by $\mathcal{E}_{ij}$ the event that

[4] More precisely, the second proof is for the important special case when β is the doubling dimension. In this proof the target dimension becomes $t = O(\beta \log \beta)$, which results in target dimension $O(\log^2 n)(\beta \log \beta)$ in Theorem 4.1.

$g_{(i,j,0)}(uv)$ or $g_{ij}(uv)$ is at least $\Omega(d/\beta)$. It suffices to prove that with high probability events $\mathcal{E}_{ij}$ happen for at least $\Omega(k)$ (i,j)-pairs. We consider two cases, depending on whether $\rho_u(\epsilon/32) \geq d/4$.

Case a. If $\rho_u(\epsilon/32) \geq d/4$, then the interval $I = [d/4; d]$ is u-broad, so by Lemma 4.4(b) there are $\Omega(k)$ different j's such that $f_{ij}(u) \in I$ for some i. By Lemma 4.2 and Chernoff bounds (Lemma A.1(a)) for $\Omega(k)$ of these ij pairs, we have $g_{ij}(uv) \geq \Omega(d/\beta)$, Case a complete.

Case b. Assume $\rho_u(\epsilon/32) < d/4$; consider interval $I = [d; \max[4d, \rho_u(32\epsilon)]]$. We claim that

$$\Pr[\mathcal{E}_{ij} \mid f_{ij}(u) \in I] \geq \Omega(1) \text{ for each } (i,j)\text{-pair}. \tag{4.3}$$

Indeed, fix ij and suppose $f = f_{ij}(u) \in I$. There are two cases $f \in [d; 4d]$ and $f \in (4d; \rho_u(32\epsilon)]$. In the first case by Lemma 4.2 $g_{ij}(uv) \geq \Omega(d/\beta)$ with conditional probability at least $\Omega(1)$. In the second case

$$|\mathbf{B}_u(d/4)| \geq \epsilon n/32 \geq 2^{-10}\,(32\epsilon n) \geq 2^{-10}\,|\mathbf{B}_u(f)|,$$

so by Lemma 4.5 $g_{(i,j,0)}(uv) \geq \Omega(d)$ with conditional probability $\Omega(1)$. This completes the proof of (4.3).

Let X_j be the indicator variable of the following random event: $\mathcal{E}_{ij}$ and $f_{ij}(u) \in I$ for some i. Since the interval I is u-broad, by Lemma 4.4(b) there are $\Omega(k)$ different j's such that $f_{ij}(u) \in I$ for some i. Let J be the set of all such j's. Then conditional on J, $\{X_j, j \in J\}$ are $\Omega(k)$ independent 0-1 random variables of expectation $\Omega(1)$. By Chernoff bounds (Lemma A.1(a)) their sum is $\Omega(1)$ with high probability, completing the proof for Case b. □

4.4. Analysis: Proof of Lemma 4.2. In this section we use padded decomposition techniques from [19, 30] to prove Lemma 4.2. Let us recall the definitions of a *padded decomposition* and a *decomposition bundle* [19, 30].

DEFINITION 4.6. *Given a finite metric space (V,d), a positive parameter $\Delta > 0$, and a mapping $\beta : V \to \mathbb{R}$, a* Δ*-bounded* β-padded decomposition *is a distribution* Π *over partitions of V such that the following conditions hold.*

(a) *For each partition P in the support of* Π*, the diameter of every cluster in P is at most Δ.*

(b) *If P is sampled from* Π*, then each ball $\mathbf{B}_x(\Delta/\beta(x))$ is partitioned by P with probability $< \frac{1}{2}$.*

Given a function $\beta : V \times \mathbb{Z} \to \mathbb{R}$, a β-padded decomposition bundle *on V is a set of padded decompositions $\{\eta(i) : i \in \mathbb{Z}\}$ such that each $\eta(i)$ is a 2^i-bounded $\beta(\cdot, i)$-padded decomposition of V.*

If a metric admits a β-padded decomposition bundle such that β is constant, we simply say that this metric *admits β-padded decompositions.*

The randomized construction. Let η be a β-padded decomposition bundle. For each $s \in \mathbb{Z}$, let the decomposition P_s be chosen according to the distribution $\eta(s)$. We denote $P_s(x)$ to be the unique cluster in P_s containing x.

Moreover, for $s \in \mathbb{Z}$, let $\{\sigma_s(C) : C \subseteq V\}$ be independently and identically distributed (i.i.d.) unbiased $\{0,1\}$-random variables. Let $T = \{0, 1, \ldots, 5\}$. Let $s(x) := \lceil \log_2 f(x) \rceil$. For each $t \in T$, we define a (random) subset

$$W^t := \{x \in V : \sigma_{s(x)-t}(P_{s(x)-t}(x)) = 0\}, \tag{4.4}$$

from which we obtain $g_t(\cdot) = \min\{d(\cdot, W^t), f(\cdot)\}$.

Bounding the contraction of the embedding. We fix vertices $x, y \in V$ and let $d = d(x, y)$. Consider the embedded distance between them. The aim is to show that, under some condition, there exists t such that $|g_t(x) - g_t(y)| \geq \rho d$ happens with constant probability, where ρ depends on the β-padded decomposition bundle.

LEMMA 4.7. *Suppose $f(x) \in [\frac{d}{4}, 4d]$ and $t \in T$ is the integer such that $\hat{s} := s(x) - t$ satisfies $2^{\hat{s}} \in [d/8, d/4)$. Let $J := \{-1, 0, 1\}$ and $\rho := \min\{\frac{1}{32\beta(x,s)} : s \in \hat{s} + J\}$. Then the event $|g_t(x) - g_t(y)| \geq \rho d$ happens with probability at least $1/64$.*

Proof. Consider the random process that determines the coordinate g_t. We like to show that the union of the following two disjoint events happens with constant probability, which implies our goal. There are two cases.

Case 1. The set W^t contains x but is disjoint with $B_y(\rho d)$.

Case 2. The set W^t contains no points from $B_x(2\rho d)$ but at least one point from $B_y(\rho d)$.

Let us define the following auxiliary events.

- Event $\mathcal{E}_1$ occurs when x is contained in W^t.
- Event $\mathcal{E}_2$ occurs when W^t is disjoint with $B_y(\rho d)$.
- Event $\mathcal{E}_3$ occurs when, for all $z \in B_x(2\rho d)$ and $s \in \hat{s} + J$, x and z are in the same cluster in $\eta(s)$.
- Event $\mathcal{E}_4$ occurs if, for all $s \in \hat{s} + J$, $\sigma_s(P_s(x)) = 1$.

Observe that the event $\mathcal{E}_1 \cap \mathcal{E}_2$ implies the event in Case 1. Note that, given a decomposition $\eta(\hat{s})$, the point x lies in a cluster different from those intersecting $B_y(\rho d)$, because $2^{\hat{s}} < \frac{d}{4} < (1 - \rho)d$. Hence the events $\mathcal{E}_1$ and $\mathcal{E}_2$ are conditionally independent, given $\eta(\hat{s})$; this in turn implies that

$$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 | \eta(\hat{s})] = \Pr[\mathcal{E}_1 | \eta(\hat{s})] \Pr[\mathcal{E}_2 | \eta(\hat{s})] = \tfrac{1}{2} \Pr[\mathcal{E}_2 | \eta(\hat{s})].$$

Since this fact holds for all decompositions $\eta(\hat{s})$, it follows that $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2] = \frac{1}{2} \Pr[\mathcal{E}_2]$.

Observe that the event $\mathcal{E}_3 \cap \mathcal{E}_4 \cap \overline{\mathcal{E}_2}$ implies the event in Case 2. This follows from the fact that $|s(x) - s(z)| \in J$. Since $f(x) \geq \frac{d}{4}$, f is 1-Lipschitz, and $d(x, z) \leq 2\rho d \leq \frac{d}{8}$, it follows that $f(x)$ and $f(z)$ are within a multiplicative factor of 2 from each other. Hence $s(x)$ and $s(z)$ differ by at most one. Again, given the decompositions $\eta(s)$, $s \in \hat{s} + J$, the event $\mathcal{E}_4$ is independent of the event $\mathcal{E}_3 \cap \overline{\mathcal{E}_2}$. Hence

$$\Pr\left[\mathcal{E}_3 \cap \mathcal{E}_4 \cap \overline{\mathcal{E}_2}\right] = \Pr[\mathcal{E}_4] \Pr\left[\mathcal{E}_3 \cap \overline{\mathcal{E}_2}\right] = \tfrac{1}{8} \Pr\left[\mathcal{E}_3 \cap \overline{\mathcal{E}_2}\right].$$

Finally, it follows that the union of the events in Cases 1 and 2 happens with probability at least

$$\tfrac{1}{2} \Pr[\mathcal{E}_2] + \tfrac{1}{8} \Pr[\mathcal{E}_3 \cap \overline{\mathcal{E}_2}] \geq \tfrac{1}{8} \Pr[\mathcal{E}_3 \cap \mathcal{E}_2] + \tfrac{1}{8} \Pr[\mathcal{E}_3 \cap \overline{\mathcal{E}_2}] = \tfrac{1}{8} \Pr[\mathcal{E}_3].$$

In order to show that $\mathcal{E}_3$ happens with constant probability, we make use of the properties of β-padded decomposition bundle. Since for all $s \in \hat{s} + J$ we have

$$2\rho d \leq 2/32\beta(x, s) \cdot d \leq 2^s/\beta(x, s),$$

it follows that $\mathcal{E}_3$ happens with probability at least $1/8$. Therefore, it follows that the desired event happens with probability at least $1/64$. ∎

4.5. Analysis: Maps f_{ij} and $g_{(i,j,0)}$. In this subsection we prove Lemmas 4.4 and 4.5. First we prove part (a) of Lemma 4.4, which is essentially the upper bound on the embedded distance for the case $p = 1$. We start with a local smoothness property of the sets U_{ij}.

CLAIM 4.8. *Fix $i, j \in [k]$ and an edge uv. Condition on the map f_{ij}; i.e., pause our embedding algorithm right after f_{ij} is constructed; let $r = f_{ij}(u)$. If $d_{uv} \le r/4$, then*

$$\Pr[v \in U_{ij}] \le 1/|\mathbf{B}_u(r)| \le \Pr\left[v \in U_{(i+3,j)}\right].$$

Proof. Let $B = \mathbf{B}_u(r)$. For the right-hand side inequality, letting $r' = f_{(i+3,j)}(v)$ we have

$$4r' \le f_{ij}(v)/2 \le (r + d_{uv})/2 \le 5r/8,$$

so $d_{uv} + 4r' < r$. It follows that $\mathbf{B}_v(r') \subset B$, so $v \in U_{(i+3,j)}$ with probability $1/|\mathbf{B}_v(4r')| \ge 1/|B|$.

For the left-hand side inequality, letting $r' = f_{ij}(v)$ we have

$$4r' \ge 4(r - d_{uv}) \ge r + d_{uv},$$

so $B \subset \mathbf{B}_v(4r')$. Therefore, $v \in U_{ij}$ with probability $1/|\mathbf{B}_v(4r')| \le 1/|B|$. ∎

Fix a node u; for simplicity assume $k = 4k_0 + 1$ for some $k_0 \in \mathbb{N}$. Let $B_{ij} = \mathbf{B}_u(f_{ij})$, and let X_{ij} be the indicator random variable for the event that $|B_{(4i+4,\,j)}| \le |B_{(4i,\,j)}|/2$. Note that, for a fixed j, the random variables X_{ij} are not independent. However, we can show that, given all previous history, the ijth event happens with at least a constant probability.

CLAIM 4.9. *For each $i \in [k_0]$, $j \in [k]$, and $q = 1 - e^{-1/2}$ we have*

$$\Pr[X_{ij} = 1 \mid f_{lj}, l < i] \ge q.$$

Proof. Indeed, fix ij, and let $f = f_{(4i,j)}(u)$ and $f' = f_{(4i+4,j)}(u)$. Let r be the radius of the smallest ball around u that contains at least $|B_{(4i,\,j)}|/2$ nodes, and let $B = \mathbf{B}_u(r)$.

Clearly, $X_{ij} = 1$ if and only if $f' \le r$. By definition of f_{ij}'s we have $f' \le f/16$, so we are done if $r \ge f/16$. Else by Claim 4.8 any node $v \in B$ included into the set $U_{(4i+3,j)}$ with probability at least $1/2|B|$, so the probability of including at least one node in B into this set (in which case $f' \le r$) is at least $1 - (1 - 1/2|B|)^{|B|} \ge q$. ∎

For a random variable X define the *distribution function* $F_X(t) = \Pr[X < t]$. For two random variables X and Y, say Y *stochastically dominates* X (written as $Y \succeq X$ or $X \preceq Y$) if $F_Y(t) \le F_X(t)$ for all $t \in \mathbb{R}$. Note that if $X \ge Y$, then $X \succeq Y$. Consider a sequence of i.i.d. Bernoulli random variables $\{Y_i\}$ with success probability q. By Claim 4.9 and Lemma A.3 (proved in Appendix A) we have the following:

$$\sum_{i=0}^{t} X_{ij} \succeq \sum_{i=0}^{t} Y_i \text{ for any } t \in [k_0] \text{ and each } j \in [k]. \tag{4.5}$$

We'll use (4.5) to prove the following crucial claim.

CLAIM 4.10. *Fix node u and $\epsilon > 0$; for each j let T_j be the smallest i such that $f_{ij}(u) \le \rho_u(\epsilon)$ or k if no such i exists. Then $\sum_j T_j = O(k \log \frac{1}{\epsilon})$ with high probability.*

Proof. Let $\alpha = \lceil \log \frac{1}{\epsilon} \rceil$. Let L_j be the smallest t such that $\sum_{i=0}^{t} X_{ij} \ge \alpha$ or k_0 if such t does not exist; note that $T_j \le 4L_j$. For the sequence $\{Y_i\}$, let Z_r be the number of trials between the $(r-1)$th success and the rth success. Let $A_j = \sum_{r=(j-1)\alpha+1}^{j\alpha} Z_r$ and $Z = \sum_{r=1}^{k\alpha} Z_r$. By (4.5) for any integer $t \in [k_0]$

$$\Pr[L_j > t] = \Pr\left[\sum_{i=0}^{t} X_{ij} < \alpha\right] \le \Pr\left[\sum_{i=0}^{t} Y_i < \alpha\right] = \Pr\left[\sum_{r=1}^{\alpha} Z_r > t\right] = \Pr[A_1 > t]. \tag{4.6}$$

Since $\{A_j\}$ are i.i.d., by (4.6) and Lemma A.2 it follows that $\sum_j L_j \succeq \sum_j A_j = Z$. Therefore, by Lemma A.4

$$\Pr\left[\sum T_j > 8k\alpha/q\right] \leq \Pr\left[\sum L_j > 2k\alpha/q\right] \leq \Pr[Z > 2k\alpha/q] < (0.782)^{k\alpha},$$

which is at most $1/n^3$ when $k = O(\log n)$ with large enough constant. □

Now we have all tools to prove Lemma 4.4(a).

Proof of Lemma 4.4(a). Use $T_j = T_j(u)$ from Claim 4.10. Fix some ϵ-long edge uv, and let $d = d_{uv}$. Let $t_j = \max(T_j(u), T_j(v))$. Then by the 1-Lipschitz property $f'_{ij}(uv) \leq d$ for all ij; moreover, for any ij such that $i \geq t_j$ both $f_{ij}(u)$ and $f_{ij}(v)$ are at most $d/2^{i-t_j}$. Then $f'_{ij}(uv)$ is at most twice that much (since $f'_{ij} \leq f_{ij}$), so taking the sum of the geometric series we see that

$$\sum_{ij} f'_{ij}(uv) \leq \sum_j \left(dt_j + \sum_{i \geq t_j} \tfrac{d}{2}^{i-t_j}\right) \leq \sum_j O(dt_j) = O\left(kd \log \tfrac{1}{\epsilon}\right),$$

where the last inequality follows by Claim 4.10. □

To prove part (b) of Lemma 4.4, let us recall the definition of a u-broad interval: For a node u, an interval $[a, b]$ is *u-broad* if a or b is equal to d_{uv} for some v, $a \leq b/4$ and $|\mathbf{B}_u(a)| \leq \frac{1}{32}|\mathbf{B}_u(b)|$.

Proof of Lemma 4.4(b). It suffices to consider the u-broad intervals $[a, b]$ such that one of the end points is equal to d_{uv} for some v and the other is the largest a or the smallest b, respectively, such that the interval is u-broad. Call these intervals *u-interesting*; note that there are at most $2n$ such intervals for each u.

Fix node u and a u-broad interval $I = [a, b]$, fix j, and let $r_i = f_{ij}(u)$. It suffices to show that with constant probability some r_i lands in I. Indeed, then we can use Chernoff bounds (Lemma A.1(a)), and then we can take the union bound over all nodes u and all u-interesting intervals.

Denote by $\mathcal{E}_i$ the event that $r_i > b$ and $r_{i+1} < a$; note that these events are disjoint. Since some r_i lands in I if and only if none of the $\mathcal{E}_i$'s happen, we need to bound the probability of $\cup\mathcal{E}_i$ away from 1.

For each integer $l \geq 0$ define the interval

$$I_l = \left[\rho_u\left(\epsilon\, 2^l\right),\ \rho_u\left(\epsilon\, 2^{l+1}\right)\right), \text{ where } \epsilon n = |\mathbf{B}_u(b)|.$$

For each $\alpha \in \{0, 1, 2, 3\}$ let $N_{(l,\alpha)}$ be the number of i's such that $r_{4i+\alpha} \in I_l$. We claim that $E[N_{(l,\alpha)}] \leq 1/q$.

Consider the case $\alpha = 0$; other cases are similar. Let $N_l = N_{(l,\alpha)}$, and suppose $N_l \geq 1$. Let i_0 be the smallest i such that $r_{4i} \leq I_l$. Then $N_l \geq t$ implies $X_{ij} = 0$ for each $i \in [i_0; i_0 + t - 2]$. Recall that the construction of the maps f_{ij} starts with $f_{(0,j)}$. Given the specific map $f = f_{(i_0,j)}$, the construction of the maps f_{ij}, $i > i_0$, is equivalent to a similarly defined construction that starts with $f_{(i_0,j)} = f$. Therefore, by (4.5) (applied to this modified construction) we have

$$\Pr[N_l \geq t] \leq \Pr\left[\sum_{\beta=0}^{t-2} X_{(i_0+\beta,\,j)} = 0\right] \leq \Pr\left[\sum_{\beta=0}^{t-2} Y_\beta = 0\right] = (1-q)^{t-1},$$

$$E[N_l] = \sum_{t=1}^{\infty} \Pr\left[N_l \geq t\right] \leq \sum_{t=1}^{\infty} (1-q)^{t-1} = \frac{1}{q},$$

claim proved. For simplicity assume $k = 4k_0 + 1$; it follows that

$$\sum_{i=0}^{k-1} \Pr[r_i \in I_l] = \sum_{\alpha=0}^{3} \sum_{i=0}^{k_0-1} \Pr[r_{4i+\alpha} \in I_l] = \sum_{\alpha=0}^{3} E\left[N_{(l,\alpha)}\right] \le 4/q. \tag{4.7}$$

By Claim 4.8 if $r_i \in I_l$, then $r_{i+1} \le a$ with conditional probability at most $|\mathbf{B}_u(a)|/|\mathbf{B}_u(r_u)| \le 2^{-l}/32$. Therefore, $\Pr[\mathcal{E}_i \,|\, r_i \in I_l] \le 2^{-l}/32$. By (4.7) it follows that

$$\begin{aligned}\Pr[\cup\mathcal{E}_i] &= \sum_{i=0}^{k-1} \Pr[\mathcal{E}_i] = \sum_{i=0}^{k-1}\sum_{l=0}^{\infty} \Pr\left[r_i \in I_l \text{ and } \mathcal{E}_i\right] \le \sum_{i=0}^{k-1}\sum_{l=0}^{\infty} \Pr[r_i \in I_l] \times \frac{2^{-l}}{32} \\ &= \frac{1}{32}\sum_{l=0}^{\infty} 2^{-l} \sum_{i=0}^{k-1} \Pr[r_i \in I_l] \le \frac{1}{8q}\sum_{l=0}^{\infty} 2^{-l} = \frac{1}{4q} < 1,\end{aligned}$$

so some r_i lands in I with at least a constant probability. □

It remains to prove Lemma 4.5 about the maps $g_{(i,j,0)}$.

Proof of Lemma 4.5. Let's pause our embedding algorithm right after the map f_{ij} is chosen and consider the probability space induced by the forthcoming random choices. Let $X_w = f_{ij}(w)$. First we claim that

$$\Pr\left[g_{(i,j,0)}(u) \le r \,|\, r \le X_u/8\right] \ge \Omega(\beta_r), \tag{4.8}$$

where $\beta_r = |\mathbf{B}_u(r)|/|\mathbf{B}_u(X_u)|$. Indeed, suppose $r \le X_u/8$, let $B = \mathbf{B}_u(r)$, and consider any $w \in B$. Then by (4.11)

$$\begin{aligned}\Pr[w \in W_{ij}] &= 1/|\mathbf{B}_w(X_w/2)| \ge 1/|\mathbf{B}_u(X)| \ge \beta_r|B|, \\ \Pr\left[\,g_{(i,j,0)}(u) \le r\,\right] &= \Pr[W_{ij} \text{ hits } B] \ge 1 - (1-\beta_r|B|)^{|B|} \ge 1 - e^{-\beta_r} \ge \Omega(\beta_r),\end{aligned}$$

proving (4.8). Now let $B = \mathbf{B}_v(X_v/8)$; then by (4.11) any $w \in B$ is included into the set W_{ij} with probability at most $1/B$, so

$$\Pr\left[\,g_{(i,j,0)}(v) \ge X_v/8\right] = \Pr[W_{ij} \text{ misses } B] \ge (1 - 1/|B|)^{|B|} \ge 1/4. \tag{4.9}$$

Finally, let's combine (4.8) and (4.9) to prove the claim. Let $r = d/4$, and suppose $X \ge 4d$. Since $X_v \ge X - d_{uv} \ge 3d$, by (4.9) event $g_{(i,j,0)}(v) \ge 3d/8$ happens with probability at least $1/4$. This event and the one in (4.8) are independent since they depend only on what happens in the balls $\mathbf{B}_u(d/4)$ and $\mathbf{B}_v(3d/8)$, respectively, which are disjoint. Therefore, with probability at least $\Omega(\beta_r)$ both events happen, in which case $g_{(i,j,0)}(uv) \ge d/8$. □

4.6. A Bourgain-style proof of Lemma 4.2 for doubling metrics. In this section we use the ideas of [10, 33] to derive an alternative proof of Lemma 4.2 for the important special case when β is the doubling dimension. In this proof the target dimension becomes $t = O(\beta \log \beta)$, which results in target dimension $O(\log^2 n)(\beta \log \beta)$ in Theorem 4.1.

Let us note that in the well-known embedding algorithms of Bourgain [10] and Linial, London, and Rabinovich [33] any two nodes are sampled with the same probability, i.e., with respect to the counting measure. Here use a nontrivial extension of Bourgain's technique where we sample with respect to a doubling measure transformed with respect to a given 1-Lipschitz map.

We state our result as follows.

LEMMA 4.11. *Consider a finite metric space (V, d) equipped with a nondegenerate measure μ and a 1-Lipschitz coordinate map f; write $f_u = f(u)$. For every node u let*

$$\beta_\mu(u) = 2\mu[\,\mathbf{B}_u(f_u)\,] \,/\, \mu[\,\mathbf{B}_u(f_u/16)\,].$$

Then for any $k, t \in \mathbb{N}$ there is a randomized embedding g into ℓ_p, $p \geq 1$, with dimension kt so that

(a) *each coordinate map of g is 1-Lipschitz and upper-bounded by f; and*

(b) $\|g(u) - g(v)\|_p \geq \Omega(d_{uv}/t)(kt)^{1/p}$ *with failure probability at most $< t/2^{\Omega(k)}$ for any edge uv such that*

$$f(u)/d_{uv} \in [1/4;\, 4] \text{ and } \max_{w \in \{u,v\}} \beta_\mu(w) \leq 2^t. \tag{4.10}$$

To prove Lemma 4.2 for a metric of doubling dimension β, recall that for any such metric there exists a 2^β-doubling measure μ. Plug this measure in Lemma 4.11, with $t = 4\beta + 1$ and $k = O(\log \beta)$; note that $\beta_\mu(u) \leq 2^t$ for every node u. We get the embedding in ℓ_p with $O(\beta \log \beta)$ dimensions that satisfies the conditions in Lemma 4.2.

We'll need the following simple fact:

(4.11)
If $d_{uv} \leq f(u)/8$ for some edge uv, then $\mathbf{B}_u(f(u)/8) \subset \mathbf{B}_v(f(v)/2) \subset \mathbf{B}_u(f(u))$.

Indeed, letting $f_u = f(u)$ the first inclusion follows since $f_v/2 \geq (f_u - d_{uv})/2 \geq f_u/8 + d_{uv}$, and the second one holds since $d_{uv} + f_v/2 \leq d_{uv} + (f_u + d_{uv})/2 < f_u$.

Proof of Lemma 4.11. Define the transformation of μ with respect to f as $\mu_f(u) = \mu(u)/2\mu(B)$, where $B = \mathbf{B}_u(f_u/2)$. The coordinates are indexed by ij, where $i \in [t]$ and $j \in [k]$. For each (i, j)-pair construct a random set U_{ij} by selecting $\lceil 2^i \mu_f(V) \rceil$ nodes independently according to the probability distribution $\mu_f(\cdot)/\mu_f(V)$. Let us define the ijth coordinate of u as $g_{ij}(u) = \min(f_u, d(u, U_{ij}))$.

Note that each map g_{ij} is 1-Lipschitz as the minimum of two 1-Lipschitz maps. Therefore, part (a) holds trivially. The hard part is part (b). Fix an edge uv; let $d = d_{uv}$. For any node w let $\alpha_w(\epsilon)$ be the smallest radius r such that $\mu_f[\mathbf{B}_w(r)] \geq \epsilon$, and let

$$\rho_i = \max[\psi_u(2^{-i}), \psi_v(2^{-i})], \text{ where } \psi_w(\epsilon) = \min[\alpha_w(\epsilon), d/2, f_w].$$

CLAIM 4.12. *For each $i \geq 1$ and each $j \in [k]$ with probability $\Omega(1)$ we have*

$$g_{ij}(uv) := |g_{ij}(u) - g_{ij}(v)| \geq \rho_i - \rho_{i+1}.$$

Then by Chernoff bounds (Lemma A.1(a)) with probability at least $1 - 2^{-\Omega(k)}$

$$\sum_{ij} g_{ij}(uv) \geq \sum_{i=1}^{t} \Omega(k)(\rho_i - \rho_{i+1}) = \Omega(k)(\rho_1 - \rho_t). \tag{4.12}$$

Proof of Claim 4.12. Fix $i \geq 1$ and j, and note that if $\rho_{i+1} = d/2$, then $\rho_i = d/2$, in which case the claim is trivial. So let's assume $\rho_{i+1} < d/2$ and without loss of generality suppose $\psi_u(2^{-i}) \geq \psi_v(2^{-i})$. Consider the open ball B of radius ρ_i around u. Since $\rho_i = \psi_u(2^{-i}) \leq \alpha_u(2^{-i})$, it follows that $\mu_f(B) \leq 2^{-i}$. Now there are two cases:

- If $\rho_{i+1} = f_v$, then the desired event $g_{ij}(uv) \geq \rho_i - \rho_{i+1}$ happens whenever U_{ij} misses B, which happens with at least a constant probability since $\mu_f(B) \leq 2^{-i}$.
- If $\rho_{i+1} < f_v$, then the desired event happens whenever U_{ij} misses B and hits $B' = \mathbf{B}_v(\rho_{i+1})$. This happens with at least a constant probability by Claim 4.14 since $\rho_{i+1} \geq \psi_v(1/2^{i+1}) \geq \alpha_v(1/2^{i+1})$ and therefore $\mu_f(B') \geq 1/2^{i+1}$, and the two balls B and B' are disjoint.

This completes the proof of the claim. ∎

CLAIM 4.13. *For any node w we have $\alpha_w(\frac{1}{2}) \geq f_w/8$ and $\alpha_w(1/\beta_\mu(w)) \leq f_w/16$.*

Proof. Let $B = \mathbf{B}_w(f_w/8)$. By (4.11) for any $w' \in B$

$$\mu(w) \,/\, 2\mu[\,\mathbf{B}_w(f_w)\,] \;\leq\; \mu_f(w') \;\leq\; \mu(w)/2\mu(B),$$

so $\mu_f(B) \leq \frac{1}{2}$ and $\mu_f[\,\mathbf{B}_w(f_w/16)\,] \geq 1/\beta_\mu(w)$. ∎

Suppose (4.10) holds; let $x = \max(f_u, f_v)$. Then by Claim 4.13 and the definitions of ρ_i and ψ_w we have

$$\rho_1 \geq \max_{w\in\{u,v\}} \min(f_w/8, d/2) \geq \min(x/8, d/2),$$

$$\rho_t \leq \max_{w\in\{u,v\}} \alpha_w(2^{-t}) \leq \max_{w\in\{u,v\}} \alpha_w\left(1/\beta_\mu(w)\right) \leq \max_{w\in\{u,v\}} f_w/16 \leq x/16.$$

By (4.12) for $p = 1$ it remains to show that $\rho_1 - \rho_t \geq \Omega(d)$. There are two cases:

- If $f_v \leq 4d$, then $\rho_1 \geq x/8$, so $\rho_1 - \rho_t \geq x/16 \geq \Omega(d)$.
- If $f_v > 4d$, then $\rho_1 \geq d/2$ and (since f is 1-Lipschitz)

$$\rho_t \;\leq\; f_v/16 \;\leq\; (f_u + d)/16 \;\leq\; 5d/16,$$

so $\rho_1 - \rho_t \geq 3d/16$.

This completes the proof for the case $p = 1$. To extend it to $p > 1$, note that the embedded uv-distance is

$$\left(\sum_{ij} g_{ij}(uv)^p\right)^{1/p} = (kt)^{1/p}\left(\frac{1}{kt}\sum_{ij} g_{ij}(uv)^p\right)^{1/p}$$

$$\geq (kt)^{1/p}\left(\frac{1}{kt}\sum_{ij} g_{ij}(uv)\right) \geq \Omega\left(\frac{d}{t}\right)(kt)^{1/p}.$$

This completes the proof of the lemma. ∎

In the above proof we used the following claim which is implicit in [33] and also stated in [28]; we prove it here for the sake of completeness.

CLAIM 4.14. *Let μ be a probability measure on a finite set V. Consider disjoint events $E, E' \subset V$ such that $\mu(E) \geq q$ and $\mu(E') \leq 2q < 1/2$ for some number $q > 0$. Let S be a set of $\lceil 1/q \rceil$ points sampled independently from V according to μ. Then S hits E and misses E' with at least a constant probability.*

Proof. Obviously, the probability that S hits E and misses E' can decrease only if we set $\Pr[E] = q$ and $\Pr[E'] = 2q$. Treat sampling a given point as three independent random choices (which result in exactly the same selection probabilities): First we choose, with probability $1-2q$, whether this point misses E'; then (if it indeed misses) we choose, with probability $q' = \frac{q}{1-2q} \leq 2q$, whether it hits E; and finally the specific point is chosen from, respectively, E, E', or $V \setminus (E \cup E')$. Without loss of generality rearrange the order of events: First we choose whether all points miss E' and then

upon success choose whether at least one point hits E. These two events happen independently with probabilities, respectively, $(1-2q)^{1/q} \geq 2^{-1/2}$ and

$$1-(1-q')^{1/q} \geq 1-(1-2q)^{1/q} \geq 1-e^{-2}.$$

So the total success probability is at least $c = (1-e^{-2})/\sqrt{2}$, which is an absolute constant as required. □

5. Lower bounds on embeddings with slack. In this section we describe a general technique to derive lower bounds for ϵ-slack embeddings from lower bounds for ordinary embeddings. For simplicity of exposition, we will first give a concrete example proving lower bounds for ϵ-slack embeddings into ℓ_p (which will follow from a lower bound for embedding expanders into ℓ_p [34]). Then we provide the general technique; the bounds obtained by this technique are given in Table 5.1. Let us mention that allowing arbitrary *expansions* is crucial to our results: If we insisted that *none of the pairwise distances should increase*, the lower bound of $\Omega(\frac{1}{p}\log n)$ distortion [34] for embeddings into ℓ_p holds even with ϵ-slack (see section 5.2 for more details).

THEOREM 5.1. *For an arbitrarily small positive ϵ there exists a finite metric space on arbitrarily many nodes that requires distortion $\Omega(\frac{1}{p}\log\frac{1}{\epsilon})$ to embed into ℓ_p, $p \geq 1$, with ϵ-slack.*

Proof. Given an ϵ such that $0 < \epsilon \leq 1/12$, let $k = 1/(3\sqrt{\epsilon})$. Fix n, the number of nodes in our counterexample.

We now construct a graph G on n vertices. Consider a constant degree expander graph H on k vertices. Let (H,d) be the shortest path metric defined by H. For each vertex $s \in H$, let L_s be a path containing n/k vertices. Attach the path L_s to s at one of its end points. The length of each edge of L_s is small enough so that if δ is the length of path L_s, then $\delta \cdot D \leq 1/2$. Let the new graph be G and the shortest path metric defined on it be (G,d). We now prove that if (G,d) can be embedded into ℓ_p with distortion D and ϵ-slack, then H can be embedded into ℓ_p with distortion $4D$ *without any slack*.

Let $\varphi : G \to \ell_p$ be the embedding of (G,d) into ℓ_p with distortion D and ϵ-slack. Let E denote the set of *ignored* pairs; i.e., let us assume that the complement of E incurs distortion at most D. Note that ϵ-slack means that $|E| \leq \epsilon n^2 2$. We delete all of the vertices that participate in more than $\sqrt{\epsilon}n$ pairs in E. By a simple counting argument, at most $\sqrt{\epsilon}n$ vertices of G can be deleted. Therefore, at least one point from each path survives. For each $s \in H$, let v_s denote a survived vertex from the path L_s. We define an embedding ψ of H into ℓ_p as $\psi(s) = \varphi(v_s)$.

We now bound the distortion of the embedding ψ by $4D$. Let x, y be two vertices in H. Then v_x and v_y are the survivors in L_x and L_y, respectively. Note that v_x and

TABLE 5.1

Embeddings with slack ϵ. Lower bounds on distortion. Here $\mathcal{F}$ is the family of doubling metrics that are induced by planar graphs. Bounds for ϵ-uniform slack can be obtained by replacing $\sqrt{\epsilon}$ by ϵ.

Type of embedding	Our lower bound	Original example
All metrics into ℓ_p, $p \geq 1$	$\Omega(\frac{1}{p})(\log\frac{1}{\epsilon})$	Constant-degree expanders [34]
$\mathcal{F}$ into ℓ_p, $p \in (1,2]$	$\Omega(p-1)\sqrt{\log 1/\epsilon}$	Laakso fractal [32]
Growth-constrained ℓ_1-metrics into ℓ_1^d	$\Omega(\sqrt{\log_d 1/\epsilon})$	Laakso fractal [32]
$\mathcal{F}$ into distributions of dominating trees	$\Omega(\log\frac{1}{\epsilon})$	$n \times n$ grid [3]
All metrics into tree metrics	$\Omega(1/\sqrt{\epsilon})$	n-cycle [40, 18]
ℓ_2^{2m+1} into ℓ_2^{2m}	$\Omega(1/\sqrt{\epsilon})^{1/m}$	[36]

v_y participate in at most $\sqrt{\epsilon}n$ pairs in E. Since $|L_y| = 3\sqrt{\epsilon}n$, it follows that there is another survivor $t \in L_y$ such that neither $\{t, v_x\}$ nor $\{t, v_y\}$ is in E. Since the distortion of the map φ is D, we can assume that for edge $(u, v) \notin E$,

$$d(u, v) \le \|\varphi(u) - \varphi(v)\|_p \le D \cdot d(u, v).$$

Now we can bound $\psi(xy) := \|\psi(x) - \psi(y)\|_p$ as follows:

$$\begin{aligned}\psi(xy) &= \|\varphi(v_x) - \varphi(v_y)\|_p \\ &\le \|\varphi(v_x) - \varphi(t)\| + \|\varphi(t) - \varphi(v_y)\| \\ &\le D\,(d(v_x, t) + d(t, v_y)) \\ &\le D\,(1 + 3\delta)\,d(x, y) \le 2D\,d(x, y).\end{aligned}$$

Similarly,

$$\begin{aligned}\psi(xy) &\ge \|\varphi(v_x) - \varphi(t)\|_p - \|\varphi(t) - \varphi(v_y)\|_p \\ &\ge d(v_x, t) - Dd(t, v_y) \ge (1 - D\delta)d(x, y) \\ &\ge d(x, y)/2.\end{aligned}$$

Hence $\frac{1}{2}d(u, v) \le \psi(uv) \le 2D \cdot d(u, v)$, and so ψ is a map from H to ℓ_p with distortion $4D$.

To finish the proof of the theorem, we note that a constant-degree expander on k vertices requires $\Omega(\log k/p)$ distortion to embed into ℓ_p [34]. □

5.1. General lower-bounding technique. The technique used in Theorem 5.1 of starting with a $O(1)$-degree expander H_k on k vertices, replacing each vertex with a path on n/k vertices to get G, and for suitable $k \approx O(1/\sqrt{\epsilon})$ arguing that ϵ-slack embeddings of G_n give us slackless embeddings of H_k with (roughly) the same distortion is quite general. In fact, we use it to obtain lower bounds on *both the distortion and dimensions of embeddings* into ℓ_p from similar lower bounds for slackless embeddings; similar results can be obtained for embeddings into trees or distributions of trees. We summarize these results in Table 5.1.

THEOREM 5.2. *Suppose for each k there exists a k-node metric H_k such that any embedding of H_k into ℓ_p with $L(k)$ dimensions has distortion at least $D(k)$. Then for an arbitrarily small positive ϵ there exist finite metrics M, M^* on an arbitrarily large number of nodes such that any embedding of*

(a) *M into ℓ_p with $L(\frac{1}{3\sqrt{\epsilon}})$ dimensions has ϵ-slack distortion $\Omega(D(\frac{1}{3\sqrt{\epsilon}}))$.*

(b) *M^* into ℓ_p with $L(\frac{1}{3\epsilon})$ dimensions has ϵ-uniform slack distortion $\Omega(D(\frac{1}{3\epsilon}))$.*

Moreover, if metrics $\{H_k\}$ are planar (resp., K_r-minor-free, doubling, ℓ_p^d), then so are M and M^*.

Note that this result can be used to translate, e.g., the Brinkman and Charikar [12] lower bound for dimensionality reduction in ℓ_1 into the realm of ϵ-slack as well.

Similarly, we provide a lower bound theorem for (probabilistic) embeddings into trees.

THEOREM 5.3. *Suppose for each k there exists a k-node metric H_k such that any (probabilistic) embedding of H_k into trees has distortion at least $D(k)$. Then for an arbitrarily small positive ϵ there exist finite metrics M, M^* on an arbitrarily large number of nodes such that any (probabilistic) embedding of*

(a) *M into trees has ϵ-slack distortion $\Omega(D(\frac{1}{3\sqrt{\epsilon}}))$.*

(b) *M^* into trees has ϵ-uniform slack distortion $\Omega(D(\frac{1}{3\epsilon}))$.*

Moreover, if metrics $\{H_k\}$ are planar (resp., K_r-minor-free, doubling, ℓ_p^d), then so are M and M^*.

For instance, we can now derive a lower bound of $\Omega(1/\sqrt{\epsilon})$ on the distortion incurred when embedding the n-cycle into a single tree.

The proofs of the two above theorems are based on the following lemma.

LEMMA 5.4 (master lemma). *Suppose H is a metric on k points and $\mathcal{T}$ is a collection of metrics on k points such that any embedding of H into $\mathcal{T}$ incurs a distortion at least D. Suppose $\mathcal{S}$ is a collection of metrics such that every subset of k points in each metric in $\mathcal{S}$ embeds into $\mathcal{T}$ with distortion at most ρ. Setting $\epsilon = 1/9k^2$, there exist arbitrarily large metrics that embed into $\mathcal{S}$ with ϵ-slack distortion $\Omega(\frac{D}{\rho})$.*

Remark. In order to obtain lower bounds for *ϵ-uniform slack* embeddings instead of ϵ-slack embeddings, we need to set $\epsilon = 1/3k$ instead of $\epsilon = 1/9k^2$; the rest of the proof remains essentially unchanged.

Before we prove Lemma 5.4, let us show how to derive the above results from it.

Proof of Theorem 5.2. Suppose $\{H_k\}$ is the given family of metrics. Let us fix a large enough k such that $\epsilon = 1/9k^2$ is small enough. Now in Lemma 5.4, let us set H to be H_k and $\mathcal{T}$ to be the collection of metrics with k points in ℓ_p with at most $L(k)$ dimensions. Hence H embeds into $\mathcal{T}$ with distortion at least $D(k) = D(\frac{1}{3\sqrt{\epsilon}})$. We set $\mathcal{S}$ to be the family of metrics in ℓ_p with at most $L(k) = L(\frac{1}{3\sqrt{\epsilon}})$ dimensions. It follows that any subset of k points in any metric in $\mathcal{S}$ embeds into $\mathcal{T}$ with distortion 1. Hence we conclude that there exists a family of metrics, each of which embeds into ℓ_p with at most $L(\frac{1}{3\sqrt{\epsilon}})$ dimensions with ϵ-slack distortion at least $\Omega(D(\frac{1}{3\sqrt{\epsilon}}))$. □

The application of Lemma 5.4 to prove the lower bounds for embeddings into trees is very similar; we sketch it here to emphasize the general patterns, as well as the slight changes required.

Proof of Theorem 5.3. Again, we fix a large enough k, and set $\epsilon = 1/9k^2$. As before, H is set to be H_k. We set $\mathcal{T}$ to be the family of tree metrics on k points (or distribution of tree metrics on k points). Again, H embeds into $\mathcal{T}$ with distortion at least $D(k) = D(\frac{1}{3\sqrt{\epsilon}})$. We set $\mathcal{S}$ to be the family of tree metrics (or distribution of tree metrics). Note that, by a result of Gupta [18], any subset of k points in any metric in $\mathcal{S}$ embeds into $\mathcal{T}$ with distortion at most 8. Now the result of Theorem 5.3 follows from Lemma 5.4 as before. □

Let us now prove the Lemma 5.4: First we show how to construct a family of metrics with the desired properties. Suppose $H = (S, d)$ is a metric such that $|S| = k$. Moreover, H embeds into $\mathcal{T}$ with distortion at least D. Without loss of generality, assume that the pairwise distance in H is at least 1. For each n that is a multiple of $3k$, we define a metric $\hat{H}$ with n points in the following way. These would be the family of metrics that exhibits the lower bound for slack embeddings.

Consider a uniform line metric with point set L of size $\frac{n}{k}$ such that the two terminal points are at distance δ away from each other, where δ is small and whose value will be specified later. For each $s \in S$, we identify s with a terminal point of a copy L_s of the line metric L. We call the augmented metric $\hat{H} = (V, d)$ with point set $V = \cup_{s \in S} L_s$. If H is already in some host space X, we just need the condition that, for each $s \in S$, we can embed a copy of L of length δ isomorphically into X that identifies one end point with s. Common metric spaces like ℓ_p certainly satisfy this condition. (Note that to avoid too many symbols, we use d for the various metrics.) Hence, for $u \in L_x$ and $v \in L_y$, $|d(u, v) - d(x, y)| \leq 2\delta$.

PROPOSITION 5.5. *Let H and $\hat{H}$ be metrics defined as above. Then (a) if H is a metric induced by a K_r-minor-free graph, then so is $\hat{H}$, and (b) if H is a doubling metric, then so is $\hat{H}$.*

The next lemma states a crucial property of the edges that are ignored by any ϵ-slack embedding.

LEMMA 5.6. *Suppose an ϵ-slack embedding of some metric space (V, d) ignores the set of edges E. Then there exists a subset $T \subseteq V$ of size at least $(1 - \sqrt{\epsilon})n$ such that each vertex in T intersects with at most $\sqrt{\epsilon}n$ edges in E.*

Proof. It suffices to show that it is impossible to have a subset $S \subseteq V$ of size greater than $\sqrt{\epsilon}n$ such that each vertex in S intersects more than $\sqrt{\epsilon}n$ edges in E. Otherwise, the total number of edges ignored would be greater than $(\sqrt{\epsilon}n)^2/2 > \epsilon n^2/2 > \epsilon\binom{n}{2}$. ∎

Note that, for an *ϵ-uniform slack* embedding, the number of ignored edges incident on any node is at most ϵn by definition; this is one place in the proof which changes when considering uniform slack.

The following lemma implies Lemma 5.4.

LEMMA 5.7. *Let $H = (S, d)$ be a metric space on k points. Suppose $\mathcal{T}$ and $\mathcal{S}$ are families of metrics such that H embeds into $\mathcal{T}$ with distortion at least D, and every subset of k points in each metric in $\mathcal{S}$ embeds into $\mathcal{T}$ with distortion at most ρ.*

Suppose δ is small enough such that $(\frac{D}{4\rho} + 2)\delta \leq \frac{1}{2}$. Let $\hat{H} = (V, d)$ be the metric space defined as above. Let $\epsilon := 1/9k^2$. Then, $\hat{H}$ embeds into $\mathcal{S}$ with ϵ-slack distortion at least $D/4\rho$.

Proof. Suppose, on the contrary, φ is an embedding of $\hat{H}$ into $\mathcal{S}$ with ϵ-slack distortion $R < D/4\rho$ that ignores the set E of edges. Then, by Lemma 5.6, there exists a subset T of V such that $|T| \geq (1 - \sqrt{\epsilon})n$ and, for all $v \in T$, v intersects at most $\sqrt{\epsilon}n$ edges in E.

For each $s \in S$, the set L_s contains $\frac{n}{k} = 3\sqrt{\epsilon}n$ points, and hence there exists some point in $T \cap L_s$, which we call v_s. We define an embedding ψ of H into $\mathcal{S}$ given by $\psi(s) := \varphi(v_s)$. We next bound the distortion of the embedding ψ. Let $x, y \in S$. Since v_x and v_y are in T, each of them has at most $\sqrt{\epsilon}n$ neighbors. Observing that $|L_y| = 3\sqrt{\epsilon}n$, it follows that there exists a point $t \in L_y$ such that neither $\{v_x, t\}$ nor $\{v_y, t\}$ is contained in E. We can assume that for $\{u, v\} \notin E$, $d(u, v) \leq ||\varphi(u) - \varphi(v)|| \leq Rd(u, v)$.

Hence it follows that

$$\begin{aligned}\|\psi(x) - \psi(y)\| =& \|\varphi(v_x) - \varphi(v_y)\| \\ \leq& \|\varphi(v_x) - \varphi(t)\| + \|\varphi(t) - \varphi(v_y)\| \\ \leq& R(d(v_x, t) + d(t, v_y)) \leq R(d(x, y) + 3\delta) \\ \leq& R(1 + 3\delta)d(x, y) \leq 2Rd(x, y),\end{aligned}$$

and, similarly,

$$\begin{aligned}\|\psi(x) - \psi(y)\| \geq& \|\varphi(v_x) - \varphi(t)\| - \|\varphi(t) - \varphi(v_y)\| \\ \geq& d(v_x, t) - Rd(t, v_y) \geq d(x, y) - 2\delta - R\delta \\ \geq& (1 - (R + 2)\delta)d(x, y) \geq d(x, y)/2,\end{aligned}$$

where the last inequality follows from the fact that $(R + 2)\delta \leq 1/2$. It then follows that ψ embeds H into $\mathcal{S}$ with distortion at most $4R$. However, since any metric in $\mathcal{S}$ embeds into $\mathcal{T}$ with distortion at most ρ, it follows that H embeds into $\mathcal{T}$ with distortion at most $4\rho R < D$, from which we obtain the desired contradiction. ∎

5.2. Lower bounds for contracting embeddings. Let us consider contracting embeddings with slack. Formally, a contracting embedding has distortion D with ϵ-slack if no pairwise distance expands and all but ϵ-fraction of the pairs contract by no more than D. We show that such embeddings incur an $\Omega(\log n)$ distortion in order to embed constant-degree expander graphs into ℓ_p, $p \geq 1$.

THEOREM 5.8. *For the shortest-paths metric of a bounded-degree expander on n vertices, distortion of any contracting embedding into ℓ_p, $p \geq 1$, is $\Omega(\frac{1}{p}\log n)$ even if we allow slack $\epsilon < \frac{1}{2}$.*

Proof. Let $G = (V, E)$ be a bounded-degree expander on n vertices, and let ρ denote its shortest-path metric. Let φ be a contracting embedding of this metric to ℓ_p, $p \geq 1$, with distortion D and slack $\epsilon < \frac{1}{2}$. Let σ denote the metric on ℓ_p; to simplify the notation, we will denote $\varphi(V) \subseteq \ell_p$ by V. Define

$$R(\sigma) = \sqrt{\sigma^2(V \times V)/\sigma^2(E)}\,, \text{ where}$$
$$\sigma^2(S) = \textstyle\sum_{(x,y)\in S} \sigma(x,y)^2 \text{ for any set } S \subseteq V \times V.$$

First we show that $R(\sigma) \leq O(\sqrt{n})$. The proof is exactly the same as that of Theorem 15.5.1 in Matŏusek [35] and works despite the fact that we allow $\varepsilon \cdot n^2$ pairwise distances to be as low as 0. Note that

$$\sigma^2(E) = \sum_{(x,y)\in E} \sigma(x,y)^2 \leq \sum_{(x,y)\in E} \rho(x,y)^2 = O(n).$$

Now we bound $\sigma^2(V \times V)$ from below. If all n^2 pairs were contracted by at most D, then we would get

$$\sigma^2(V \times V) \geq \sum_{(u,v)} \left(\frac{\rho(u,v)}{D}\right)^2 \geq \frac{n^2 \log^2 n}{D^2}.$$

However, we need to take into account the fact that $\varepsilon \cdot n^2$ pairs of vertices could have distance 0 between them. Therefore, $\sigma^2(V \times V)$ is at least $(n/D)^2(\log^2 n)$ minus the loss due to the slack. To upper-bound this loss, consider a pair (x, y) of nodes for which the distortion is bigger than D. The pair will contribute 0 instead of $\rho(x, y)/D$. Thus the loss due to the pair (x, y) is at most $(\log n)/D$. Therefore, the total loss due to the slack is at most $\epsilon(n/D)^2(\log^2 n)$. Therefore, since $R(\sigma) \leq O(\sqrt{n})$, it follows that $D = \Omega(\log n)$. ∎

Appendix A. Tools from probability theory. Here we state some tools from probability theory that we used in section 4.

LEMMA A.1 (Chernoff bounds). *Consider the sum X of n independent random variables on $[0, \Delta]$.*

(a) *For any $\mu \leq E(X)$ and any $\epsilon \in (0, 1)$ we have*

$$\Pr[X < (1-\epsilon)\mu] \leq \exp(-\epsilon^2\mu/2\Delta).$$

(b) *For any $\mu \geq E(X)$ and any $\beta \geq 1$ we have* $\Pr[X > \beta\mu] \leq \left[\frac{1}{e}(e/\beta)^\beta\right]^{\mu/\Delta}$.

For a random variable X define the *distribution function* $F_X(t) = \Pr[X < t]$. For two random variables X and Y, say Y *stochastically dominates* X (written as $Y \succeq X$ or $X \preceq Y$) if $F_Y(t) \leq F_X(t)$ for all $t \in \mathbb{R}$.

LEMMA A.2. *Consider two sequences of independent random variables $\{X_i\}$ and $\{Y_i\}$ such that all X_i and Y_i have finite domains and $X_i \preceq Y_i$ for each i. Then for each k we have $\sum_{i=1}^k X_i \preceq \sum_{i=1}^k Y_i$.*

LEMMA A.3. *Consider two sequences of Bernoulli random variables* $\{X_i\}$ *and* $\{Y_i\}$ *such that variables* $\{Y_i\}$ *are independent and*

$$\Pr[X_i = 1 \mid X_j, j < i] \geq \Pr[Y_i = 1]$$

for each i. *Then* $\sum_{i=1}^{k} X_i \succeq \sum_{i=1}^{k} Y_i$ *for each* k.

Proof. We first show that for all $t \in [T]$,

$$\Pr\left[\sum_{r=1}^{t} X_r + \sum_{r=t+1}^{T} Y_r \leq m\right] \leq \Pr\left[\sum_{r=1}^{t-1} X_r + \sum_{r=t}^{T} Y_r \leq m\right], \tag{A.1}$$

which would immediately imply the lemma. Observe that for any fixed number a (or in general any random variable that is measurable in the σ-field generated by the random variables $\{X_r : r < t\}$), we have

$$\Pr\left[X_t \leq a | X_r, r < t\right] \leq Pr[Y_t \leq a] = \Pr\left[Y_t \leq a | X_r, r < t\right].$$

Note that the interesting case is when $a \in [0, 1)$. The inequality comes from the assumption concerning the conditional probabilities of the sequence $\{X_r\}$, and the equality comes from the fact that Y_t is independent of the sequence $\{X_r\}$.

Since both X_t and Y_t are independent of $\{Y_r : r > t\}$, the above inequality would still hold if we further condition on the random variables $\{Y_r : r > t\}$. Finally, setting $a = m - \sum_{i<t} X_r - \sum_{i>t} Y_r$, which is measurable in the σ-field generated by $J := \{X_r : r < t\} \cup \{Y_r : r > t\}$, we obtain

$$\Pr\left[\sum_{r=1}^{t} X_r + \sum_{r=t+1}^{T} Y_r \leq m \,|\, J\right] \leq \Pr\left[\sum_{r=1}^{t-1} X_r + \sum_{r=t}^{T} Y_r \leq m \,|\, J\right].$$

Taking the expectation on both sides gives (A.1). ∎

LEMMA A.4. *Consider a sequence of i.i.d. Bernoulli random variables* $\{Y_i\}$ *with success probability* q. *Let* Z_r *be the number of trials between the* $(r-1)$*th success and the* r*th success. Then*

$$\Pr\left[\sum_{r=1}^{k} Z_r > 2k/q\right] \leq (0.782)^k. \tag{A.2}$$

Proof. Each Z_r has a geometric distribution with parameter q, so its moment generating function is

$$E\left[e^{tZ_r}\right] = \frac{qe^t}{q - (1-q)e^t}.$$

Let $Z = \sum_{r=1}^{k} Z_r$. Since Z_r's are i.i.d., it follows that

$$E\left[e^{tZ}\right] = E\left[\prod_r e^{tZ_r}\right] = \left(E\left[e^{tZ_1}\right]\right)^k.$$

By the Markov inequality for any $t > 0$ we have

$$Pr\left[Z > 2\frac{k}{q}\right] = Pr\left[e^{tZ} > e^{2tk/q}\right] \leq E\left[e^{tZ}\right] e^{-2tk/q} \leq \left(\frac{qe^t}{(1-(1-q)e^t)e^{2t/q}}\right)^k.$$

Plugging in $q = 1 - 1/\sqrt{e}$ and $t = q$ we have (A.2). ∎

REFERENCES

[1] I. Abraham, Y. Bartal, T.-H. H. Chan, K. Dhamdhere, A. Gupta, J. Kleinberg, O. Neiman, and A. Slivkins, *Metric embeddings with relaxed guarantees*, in Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS), Pittsburgh, PA, 2005, pp. 83–100.

[2] I. Abraham, Y. Bartal, and O. Neiman, *Advances in metric embedding theory*, in Proceedings of the 38th ACM Symposium on Theory of Computing (STOC), Seattle, WA, 2006.

[3] N. Alon, R. M. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput., 24 (1995), pp. 78–100.

[4] P. Assouad, *Plongements lipschitziens dans R^n*, Bull. Soc. Math. France, 111 (1983), pp. 429–448.

[5] Y. Bartal, *Probabilistic approximations of metric spaces and its algorithmic applications*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS), Burlington, VT, 1996, pp. 184–193.

[6] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th ACM Symposium on Theory of Computing (STOC), Dallas, TX, 1998, pp. 161–168.

[7] Y. Bartal, *Graph decomposition lemmas and their role in metric embedding methods*, in Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), Bergen, Norway, 2004, pp. 89–97.

[8] Y. Bartal, B. Bollobás, and M. Mendel, *Ramsey-type theorems for metric spaces with applications to online problems*, J. Comput. System Sci., 72 (2002), pp. 890–921. Preliminary version in 42*nd IEEE FOCS*, 2001.

[9] Y. Bartal, N. Linial, M. Mendel, and A. Naor, *On metric Ramsey-type phenomena*, Ann. of Math., 162 (2005), pp. 643–709. Preliminary version in 35*th ACM STOC*, 2003.

[10] J. Bourgain, *On Lipschitz embeddings of finite metric spaces in Hilbert space*, Israel J. Math., 52 (1985), pp. 46–52.

[11] J. Bourgain, T. Figiel, and V. Milman, *On Hilbertian subsets of finite metric spaces*, Israel J. Math., 55 (1986), pp. 147–152.

[12] B. Brinkman and M. S. Charikar, *On the impossibility of dimension reduction in l_1*, J. ACM, 52 (2005), pp. 766–788. Preliminary version in 44th IEEE FOCS, 2003

[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, *Vivaldi: A decentralized network coordinate system*, in Proceedings of the ACM Special Interest Group on Data Communications (SIGCOMM), Portland, OR, 2004.

[14] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, J. Comput. System Sci., 69 (2004), pp. 485–497. Preliminary version in 35*th ACM STOC*, 2003.

[15] J. Fakcharoenphol and K. Talwar, *An improved decomposition theorem for graphs excluding a fixed minor*, in Proceedings of the RANDOM-APPROX, Princeton, NJ, Springer-Verlag, 2003, pp. 36–46.

[16] M. Fomenkov, L. Claffy, B. Huffaker, and D. Moore, *Macroscopic internet topology and performance measurements from the DNS root name servers*, in Proceedings of the Usenix Large Installation System Administration (LISA) Conference, San Diego, CA, 2001.

[17] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, *IDMaps: A global Internet host distance estimation service*, IEEE/ACM Trans. Networking, 9 (2001), pp. 525–540. Preliminary version in IEEE INFOCOM 1999.

[18] A. Gupta, *Steiner points in tree metrics don't (really) help*, in Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), Philadelphia, 2001, pp. 220–227.

[19] A. Gupta, R. Krauthgamer, and J. R. Lee, *Bounded geometries, fractals, and low–distortion embeddings*, in Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS), Cambridge, MA, 2003, pp. 534–543.

[20] A. Gupta, I. Newman, Y. Rabinovich, and A. Sinclair, *Cuts, trees and ℓ_1-embeddings of graphs*, Combinatorica, 24 (2004), pp. 233–269. Preliminary version in 40th FOCS, 1999.

[21] J. D. Guyton and M, F. Schwartz, *Locating nearby copies of replicated Internet servers*, in Proceedings of the ACM Special Interest Group on Data Communications (SIGCOMM), Cambridge, MA, 1995.

[22] S. Har-Peled and M. Mendel, *Fast construction of nets in low dimensional metrics and their applications*, SIAM J. Comput., 35 (2006), pp. 1148–1184. Preliminary version in 21st ACM SoCG, 2005.

[23] J. Heinonen, *Lectures on Analysis on Metric Spaces*, Universitext, Springer-Verlag, New York, 2001.

[24] P. Indyk, *Algorithmic applications of low-distortion geometric embeddings*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), Los Vegas, NV, 2001, pp. 10–33.

[25] P. Indyk and J. Matoušek, *Low-distortion embeddings of finite metric spaces*, in Handbook of Discrete and Computational Geometry, 2nd ed., J. E. Goodman and J. O'Rourke, eds., Discrete Math. Appl. (Boca Raton), Chapman & Hall/CRC, Boca Raton, FL, 2004, pp. 177–196.

[26] D. R. Karger and M. Ruhl, *Finding nearest neighbors in growth-restricted metrics*, in Proceedings of the 34th ACM Symposium on Theory of Computing (STOC), Montreal, Canada, 2002, pp. 63–66.

[27] P. Klein, S. A. Plotkin, and S. B. Rao, *Excluded minors, network decomposition, and multicommodity flow*, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC), 1993, San Diego, CA, pp. 682–690.

[28] J. Kleinberg, A. Slivkins, and T. Wexler, *Triangulation and embedding using small sets of beacons*, in Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS), Rome, Italy, 2004, pp. 444–453.

[29] C. Kommareddy, N. Shankar, and B. Bhattacharjee, *Finding close friends on the Internet*, in Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP), Berlin, Germany, 2001.

[30] R. Krauthgamer, J. Lee, M. Mendel, and A. Naor, *Measured descent: A new embedding method for finite metrics*, Geom. Funct. Anal., 15 (2005), pp. 839–858. Preliminary version in 45*th IEEE FOCS*, 2004.

[31] J. R. Lee, *On distance scales, embeddings, and efficient relaxations of the cut cone*, in Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), Vancouver, Canada, 2005, pp. 92–101.

[32] J. R. Lee, M. Mendel, and A. Naor, *Metric structures in L1: Dimension, snowflakes, and average distortion*, European J. Combin., 6 (2005), pp. 1180–1190. Preliminary version in LATIN, 2004.

[33] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245. Preliminary version in 35*th IEEE FOCS*, 1994.

[34] J. Matoušek, *On embedding expanders into l_p spaces*, Israel J. Math., 102 (1997), pp. 189–197.

[35] J. Matoušek, *Lectures on Discrete Geometry*, Grad. Texts in Math. 212, Springer-Verlag, New York, 2002.

[36] J. Matoušek, *Bi-Lipschitz embeddings into low dimensional Euclidean spaces*, Comment. Math. Univ. Carolin., 31 (1990), pp. 589–600.

[37] T. S. E. Ng and H. Zhang, *Predicting Internet network distance with coordinates-based approaches*, in Proceedings of the 21st Joint Conference of the IEEE Computer and Communication (INFOCOM), New York, 2002.

[38] T. S. E. Ng and H. Zhang, *A network positioning system for the Internet*, in Proceedings of the USENIX, Boston, 2004.

[39] C. G. Plaxton, R. Rajaraman, and A. W. Richa, *Accessing nearby copies of replicated objects in a distributed environment*, Theory Comput. Syst., 32 (1999), pp. 241–280. Preliminary version in 9*th ACM SPAA*, 1997.

[40] Y. Rabinovich and R. Raz, *Lower bounds on the distortion of embedding finite metric spaces in graphs*, Discrete Comput. Geom., 19 (1998), pp. 79–94.

[41] S. B. Rao, *Small distortion and volume preserving embeddings for planar and Euclidean metrics*, in Proceedings of the 15th ACM Symposium on Computational Geometry (SoCG), Miami, FL, 1999, pp. 300–306.

[42] Y. Shavitt and T. Tankel, *Big-bang simulation for embedding network distances in Euclidean space*, IEEE/ACM Trans. Netw., 12 (2004), pp. 993–1006. Preliminary version in 22nd IEEE INFOCOM, 2003.

[43] A. Slivkins, *Distributed approaches to triangulation and embedding*, in Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), Vancouver, Canada, 2005, pp. 640–649.

[44] K. Talwar, *Bypassing the embedding: Algorithms for low-dimensional metrics*, in Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), Chicago, IL, 2004, pp. 281–290.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2330–2355

THE CONNECTIVITY OF BOOLEAN SATISFIABILITY: COMPUTATIONAL AND STRUCTURAL DICHOTOMIES*

PARIKSHIT GOPALAN†, PHOKION G. KOLAITIS‡, ELITZA MANEVA‡, AND CHRISTOS H. PAPADIMITRIOU§

Abstract. Boolean satisfiability problems are an important benchmark for questions about complexity, algorithms, heuristics, and threshold phenomena. Recent work on heuristics and the satisfiability threshold has centered around the structure and connectivity of the solution space. Motivated by this work, we study structural and connectivity-related properties of the space of solutions of Boolean satisfiability problems and establish various dichotomies in Schaefer's framework. On the structural side, we obtain dichotomies for the kinds of subgraphs of the hypercube that can be induced by the solutions of Boolean formulas, as well as for the diameter of the connected components of the solution space. On the computational side, we establish dichotomy theorems for the complexity of the connectivity and *st*-connectivity questions for the graph of solutions of Boolean formulas. Our results assert that the intractable side of the computational dichotomies is PSPACE-complete, while the tractable side—which includes but is not limited to all problems with polynomial-time algorithms for satisfiability—is in P for the *st*-connectivity question, and in coNP for the connectivity question. The diameter of components can be exponential for the PSPACE-complete cases, whereas in all other cases it is linear; thus, diameter and complexity of the connectivity problems are remarkably aligned. The crux of our results is an expressibility theorem showing that in the tractable cases, the subgraphs induced by the solution space possess certain good structural properties, whereas in the intractable cases, the subgraphs can be arbitrary.

Key words. Boolean satisfiability, computational complexity, PSPACE, PSPACE-completeness, dichotomy theorems, graph connectivity

AMS subject classifications. 03D15, 68Q15, 68Q17, 68Q25, 05C40

DOI. 10.1137/07070440X

1. Introduction.

In 1978, Schaefer [31] introduced a rich framework for expressing variants of Boolean satisfiability and proved a remarkable *dichotomy theorem*: the satisfiability problem is in P for certain classes of Boolean formulas, while it is NP-complete for all other classes in the framework. This result pinpoints the computational complexity of numerous well-known variants of Boolean SAT, such as 3-SAT, HORN 3-SAT, NOT-ALL-EQUAL 3-SAT, and 1-IN-3 SAT. Schaefer's dichotomy theorem yields a classification of the computational complexity of constraint satisfaction problems (CSPs) over the Boolean domain. Feder and Vardi [15] conjectured that a dichotomy theorem holds for the complexity of CSPs over arbitrary finite domains. This conjecture remains open to date, in spite of concerted efforts and some partial progress, such as the case of domains of size 3 [8].

*Received by the editors October 4, 2007; accepted for publication (in revised form) November 25, 2008; published electronically March 4, 2009. A preliminary version of this article appeared in ICALP'06 [16].

http://www.siam.org/journals/sicomp/38-6/70440.html

†Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 (parik@cs.washington.edu). This work was done in part while the first author was a summer intern at IBM Almaden.

‡Department of Computer Science Principles and Methodologies, IBM Almaden Research Center, San Jose, CA 95120 (kolaitis@almaden.ibm.com, enmaneva@us.ibm.com). This work was done while the second author was on leave from UC Santa Cruz and while the third author was an intern at IBM Almaden.

§Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720 (christos@cs.berkeley.edu). The fourth author's research was supported by NSF grant CCF0635319, a gift from Yahoo!, a MICRO grant, and a grant from the France-Berkeley Fund.

In this article we concentrate on the Boolean CSPs, but we ask a new set of questions, which was motivated from the study of random instances of satisfiability. In recent years, the structure of the space of solutions for random instances has been the main consideration at the basis of both algorithms for and mathematical analysis of the satisfiability problem [3, 26, 28, 25]. It has been conjectured for 3-SAT [28, 4] and proved for 8-SAT [27, 1] that the solution space fractures as one approaches the *critical region* from below. This apparently leads to performance deterioration of the standard satisfiability algorithms, such as WalkSAT [32] and DPLL [2]. It is also the main consideration behind the design of the survey propagation algorithm, which has far superior performance on random instances of satisfiability [28]. This body of work has served as a motivation for us to pursue the investigation reported here.

Our aim in this article is to carry out a comprehensive exploration of the *connectivity properties* of the space of solutions of Boolean formulas from a worst-case point of view. The solutions (satisfying assignments) of a given n-variable Boolean formula φ induce a subgraph $G(\varphi)$ of the n-dimensional hypercube. Thus, the following two decision problems, called the *connectivity problem* and the *st-connectivity problem*, arise naturally: (i) Given a Boolean formula φ, is $G(\varphi)$ connected? (ii) Given a Boolean formula φ and two solutions $\mathbf{s}$ and $\mathbf{t}$ of φ, is there a path from $\mathbf{s}$ to $\mathbf{t}$ in $G(\varphi)$?

While there has been an intensive study of the structure of the solution space of Boolean satisfiability problems for random instances, our work seems to be the first to explore this issue from a worst-case viewpoint. A priori, it is not clear what to expect. Is the hardness of the satisfiability question for a given CSP at all correlated with the properties of the graphs realizable as $G(\varphi)$ of formulas in the given class? What kinds of graphs are realizable? Are there Boolean CSPs whose connectivity graphs can have connected components with exponential diameter and, if yes, can we characterize these CSPs? What is the complexity of the *st*-connectivity and the connectivity problem, and is it correlated with the answer to the previous question?

In this article, we investigate the above questions for Boolean CSPs and obtain answers for all of them. Our first complexity-theoretic result is a dichotomy theorem for the *st*-connectivity problem. This result reveals that the tractable side is much more generous than the tractable side for satisfiability, while the intractable side is PSPACE-complete. Specifically, Schaefer showed that the satisfiability problem is solvable in polynomial time precisely for formulas built from Boolean relations all of which are bijunctive, or all of which are Horn, or all of which are dual Horn, or all of which are affine. We identify new classes of Boolean relations, called *tight* relations, that properly contain the classes of bijunctive, Horn, dual Horn, and affine relations. We show that *st*-connectivity is solvable in linear time for formulas built from tight relations, and PSPACE-complete in all other cases. Our second main result is a dichotomy theorem for the connectivity problem: it is in coNP for formulas built from tight relations, and PSPACE-complete in all other cases.

In addition to these two complexity-theoretic dichotomies, we establish a structural dichotomy theorem for the diameter of the connected components of the solution space of Boolean formulas. This result asserts that, in the PSPACE-complete cases, the diameter of the connected components can be exponential, but in all other cases it is linear. Thus, small diameter and tractability of the *st*-connectivity problem are remarkably aligned.

To establish our results, the main challenge is to show that for noneasy relations, both the connectivity problem and the *st*-connectivity problem are PSPACE-hard. In Schaefer's dichotomy theorem, NP-hardness of satisfiability was a consequence of

an *expressibility* theorem, which asserted that every Boolean relation can be obtained as a projection over a formula built from clauses in the "hard" relations. Schaefer's notion of expressibility is inadequate for our problem. Instead, we introduce and work with a delicate and stricter notion of expressibility, which we call *structural expressibility*. Intuitively, structural expressibility means that, in addition to definability via a projection, the space of witnesses of the existential quantifiers in the projection has certain strong connectivity properties that allow us to capture the graph structure of the relation that is being defined. It should be noted that Schaefer's dichotomy theorem can also be proved using a Galois connection and Post's celebrated classification of the lattice of Boolean clones (see [5]). This method, however, does not appear to apply to connectivity, as the boundaries discovered here cut across Boolean clones. Thus, the use of structural expressibility or some other refined definability technique seems unavoidable.

The first step towards proving PSPACE-completeness is to show that both connectivity and *st*-connectivity are hard for 3-CNF-formulas; this is proved by a reduction from a generic PSPACE computation. Next, we identify the simplest relations that are not tight: these are ternary relations whose graph is a path of length 4 between assignments at Hamming distance 2. We show that these paths can structurally express all 3-CNF clauses. The crux of our hardness result is an *expressibility* theorem to the effect that one can structurally express such a path from any set of relations which is not tight.

Finally, we show that all *tight* relations have "good" structural properties. Specifically, in a tight relation every component has a unique minimum element, or every component has a unique maximum element, or the Hamming distance coincides with the shortest-path distance in the relation. These properties are inherited by every formula built from tight relations and yield both small diameter and linear algorithms for *st*-connectivity.

Related work. In parallel and independently of our work, a similar dichotomy was found for the k-colorability problem by Bonsma and Cereceda [6] and Cereceda, van den Heuvel, and Johnson [9]. Specifically, Cereceda, van den Heuvel, and Johnson [9] showed that the case of 3-colorability has properties similar to our tight problems—its structure implies (by a proof that is much more intricate than ours for tight problems) that the diameter of connected components of the graph of 3-colorings is at most quadratic in the number of vertices. This implies that the *st*-connectivity question for 3-colorability is in coNP. Furthermore, Bonsma and Cereceda [6] showed that for $k \geq 4$, *st*-connectivity is PSPACE-complete. This result shows an interesting complexity-theoretic difference between 3-colorability and 4-colorability. It also indicates that an extension of our result to larger domains will be quite challenging, since one would have to identify a set of "tight" problems that includes 3-colorability. It is conceivable that characterizing the complexity of the connectivity question is easier than characterizing the complexity of the satisfiability question, but at present there is no conjecture that generalizes both the Boolean case and the case of colorings.

In an earlier piece of related work, Brightwell and Winkler [7] considered the connectivity of the graph of solutions of the graph-homomorphism problems (which are a subclass of CSPs). They characterized those graphs H for which the graph-homomorphism problem with template H, also known as the H-coloring problem, has a graph of solutions that is always connected. Their motivation is the study of uniqueness of Gibbs measures, which is related to the performance of standard Markov chain algorithms for sampling and counting solutions. We note that the state-space

of many of the Markov chains studied in this area is the set of solutions, and steps of the chain are single-variable flips, so that the chain is just a random walk on the graph of solutions considered here.

In a different direction, there has been a substantial body of work on dichotomy theorems for various aspects of constraint satisfaction on the Boolean domain, including optimization [10, 14, 21], counting [12], inverse satisfiability [20], minimal satisfiability [22], lexicographically minimal satisfiability [30], and propositional abduction [13]. Our results contribute to this body of work.

Finally, it is worth mentioning that satisfiability in the context of random instances has also been considered not only for specific CSPs such as k-SAT and k-colorability, but also for general CSPs. Several models of general random CSPs have been studied by Creignou and Daudé [11], and independently by Molloy [29].

2. Basic concepts and statements of results. A CNF-formula is a Boolean formula of the form $C_1 \wedge \cdots \wedge C_n$, where each C_i is a clause, i.e., a disjunction of literals. If k is a positive integer, then a k-CNF-formula is a CNF-formula $C_1 \wedge \cdots \wedge C_n$ in which each clause C_i is a disjunction of at most k literals.

A *logical relation* R is a nonempty subset of $\{0,1\}^k$ for some $k \geq 1$; k is the *arity* of R. Let $\mathcal{S}$ be a finite set of logical relations. A CNF($\mathcal{S}$)*-formula* over a set of variables $V = \{x_1, \ldots, x_n\}$ is a finite conjunction $C_1 \wedge \cdots \wedge C_n$ of clauses built using relations from $\mathcal{S}$, variables from V, and the constants 0 and 1; this means that each C_i is an expression of the form $R(\xi_1, \ldots, \xi_k)$, where $R \in \mathcal{S}$ is a relation of arity k, and each ξ_j is a variable in V or one of the constants 0, 1. Note that the constants 0 and 1 are allowed in CNF($\mathcal{S}$)-formulas; this is equivalent to assuming that the set $\mathcal{S}$ contains the singleton logical relations $\{0\}$ and $\{1\}$. One could also consider CNF($\mathcal{S}$)-formulas without constants. In fact, this class of formulas has already been considered by Schaefer [31], as well as by other researchers in subsequent investigations. In particular, Schaefer [31] also established a dichotomy theorem for the complexity of the satisfiability problem for CNF($\mathcal{S}$)-formulas without constants. A *solution* of a CNF($\mathcal{S}$)-formula φ is an assignment $s = (a_1, \ldots, a_n)$ of Boolean values to the variables that makes every clause of φ true. A CNF($\mathcal{S}$)-formula is *satisfiable* if it has at least one solution.

The *satisfiability problem* SAT($\mathcal{S}$) associated with a finite set $\mathcal{S}$ of logical relations asks the following: Given a CNF($\mathcal{S}$)-formula φ, is it satisfiable? Many well-known restrictions of Boolean satisfiability, such as 3-SAT, NOT-ALL-EQUAL 3-SAT, and POSITIVE 1-IN-3 SAT, can be cast as SAT($\mathcal{S}$) problems, for a suitable choice of $\mathcal{S}$. For instance, let $R_0 = \{0,1\}^3 \backslash \{000\}$, $R_1 = \{0,1\}^3 \backslash \{100\}$, $R_2 = \{0,1\}^3 \backslash \{110\}$, $R_3 = \{0,1\}^3 \backslash \{111\}$. Then 3-SAT is the problem SAT($\{R_0, R_1, R_2, R_3\}$). Similarly, POSITIVE 1-IN-3SAT is SAT($\{R_{1/3}\}$), where $R_{1/3} = \{100, 010, 001\}$.

Schaefer [31] identified the complexity of *every* satisfiability problem SAT($\mathcal{S}$), where $\mathcal{S}$ ranges over all finite sets of logical relations. To state Schaefer's main result, we need to define some basic concepts.

DEFINITION 2.1. *Let R be a logical relation.*

1. *R is* bijunctive *if it is the set of solutions of a* 2-CNF*-formula.*
2. *R is* Horn *if it is the set of solutions of a Horn formula, where a Horn formula is a* CNF*-formula such that each conjunct has at most one positive literal.*
3. *R is* dual Horn *if it is the set of solutions of a dual Horn formula, where a dual Horn formula is a* CNF*-formula such that each conjunct has at most one negative literal.*
4. *R is* affine *if it is the set of solutions of a system of linear equations over $\mathbb{Z}_2$.*

Each of these types of logical relations can be characterized in terms of *closure* properties [31]. In what follows, we use boldface letters to denote vectors of Boolean values or vectors of variables. A relation R is bijunctive if and only if it is closed under the *majority* operation; this means that if $\mathbf{a}, \mathbf{b}, \mathbf{c} \in R$, then $\text{maj}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \in R$, where $\text{maj}(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is the vector whose ith bit is the majority of a_i, b_i, c_i. A relation R is Horn if and only if it is closed under $\wedge$; this means that if $\mathbf{a}, \mathbf{b} \in R$, then $\mathbf{a} \wedge \mathbf{b} \in R$, where $\mathbf{a} \wedge \mathbf{b}$ is the vector whose ith bit is $a_i \wedge b_i$. Similarly, R is dual Horn if and only if it is closed under $\vee$. Finally, R is affine if and only if it is closed under $\mathbf{a} \oplus \mathbf{b} \oplus \mathbf{c}$.

DEFINITION 2.2. *A set $\mathcal{S}$ of logical relations is* Schaefer *if at least one of the following conditions holds:*

1. *Every relation in $\mathcal{S}$ is bijunctive.*
2. *Every relation in $\mathcal{S}$ is Horn.*
3. *Every relation in $\mathcal{S}$ is dual Horn.*
4. *Every relation in $\mathcal{S}$ is affine.*

A logical relation R is Schaefer *if the singleton $\{R\}$ is Schaefer.*

Since the property of a set being Schaefer is characterized in terms of closure under operations that are binary or ternary, it follows that there is a cubic algorithm to determine whether or not a given a finite set $\mathcal{S}$ of logical relations is Schaefer (assuming that each relation in $\mathcal{S}$ is given as a list of Boolean vectors).

THEOREM 2.3 (Schaefer's dichotomy theorem [31]). *Let $\mathcal{S}$ be a finite set of logical relations. If $\mathcal{S}$ is Schaefer, then* SAT($\mathcal{S}$) *is in* P*; otherwise,* SAT($\mathcal{S}$) *is* NP*-complete.*

Theorem 2.3 is called a dichotomy theorem because Ladner [23] has shown that if P $\neq$ NP, then there are problems in NP that are neither in P nor NP-complete. Thus, Theorem 2.3 asserts that no SAT($\mathcal{S}$) problem is a problem of the kind discovered by Ladner. Note that the aforementioned characterization of Schaefer sets in terms of closure properties yields a cubic algorithm for determining, given a finite set $\mathcal{S}$ of logical relations, whether SAT($\mathcal{S}$) is in P or is NP-complete (here, the input size is the sum of the sizes of the relations in $\mathcal{S}$).

The more difficult part of the original proof of Schaefer's dichotomy theorem is to show that if $\mathcal{S}$ is not Schaefer, then SAT($\mathcal{S}$) is NP-complete. This is a consequence of a powerful result about the expressibility of logical relations. We say that a relation R is *expressible from* a set $\mathcal{S}$ of relations if there is a CNF($\mathcal{S}$)-formula $\varphi(\mathbf{x}, \mathbf{y})$ such that $R = \{\mathbf{a} |\ \exists \mathbf{y} \varphi(\mathbf{a}, \mathbf{y})\}$.

THEOREM 2.4 (Schaefer's expressibility theorem [31]). *Let $\mathcal{S}$ be a finite set of logical relations. If $\mathcal{S}$ is not Schaefer, then every logical relation is expressible from $\mathcal{S}$.*

In this paper, we are interested in the connectivity properties of the space of solutions of CNF($\mathcal{S}$)-formulas. If φ is a CNF($\mathcal{S}$)-formula with n variables, then the *solution graph* $G(\varphi)$ of φ denotes the subgraph of the n-dimensional hypercube induced by the solutions of φ. This means that the vertices of $G(\varphi)$ are the solutions of φ, and there is an edge between two solutions of $G(\varphi)$ precisely when they differ in exactly one variable.

We consider the following two algorithmic problems for CNF($\mathcal{S}$)-formulas.

Problem 1. *The Connectivity Problem* CONN($\mathcal{S}$). Given a CNF($\mathcal{S}$)-formula φ, is $G(\varphi)$ connected? (If φ is unsatisfiable, then the answer to this problem is "yes.")

Problem 2. *The st-Connectivity Problem* ST-CONN($\mathcal{S}$). Given a CNF($\mathcal{S}$)-formula φ and two solutions $\mathbf{s}$ and $\mathbf{t}$ of φ, is there a path from $\mathbf{s}$ to $\mathbf{t}$ in $G(\varphi)$?

To pinpoint the computational complexity of CONN($\mathcal{S}$) and ST-CONN($\mathcal{S}$), we need to introduce certain new types of relations.

DEFINITION 2.5. *Let $R \subseteq \{0,1\}^k$ be a logical relation.*

1. *R is* componentwise bijunctive *if every connected component of the graph $G(R)$ is a bijunctive relation.*
2. *R is* OR-free *if the relation* $\mathrm{OR} = \{01, 10, 11\}$ *cannot be obtained from R by setting $k-2$ of the coordinates of R to a constant $\mathbf{c} \in \{0,1\}^{k-2}$. In other words, R is* OR*-free if $(x_1 \vee x_2)$ is not definable from R by fixing $k-2$ variables.*
3. *R is* NAND-free *if the relation* $\mathrm{NAND} = \{00, 01, 10\}$ *cannot be obtained from R by setting $k-2$ of the coordinates of R to a constant $\mathbf{c} \in \{0,1\}^{k-2}$. In other words, R is* NAND*-free if $(\bar{x}_1 \vee \bar{x}_2)$ is not definable from R by fixing $k-2$ variables.*

We are now ready to introduce the key concept of a *tight* set of relations.

DEFINITION 2.6. *A set $\mathcal{S}$ of logical relations is* tight *if at least one of the following three conditions holds:*

1. *Every relation in $\mathcal{S}$ is componentwise bijunctive.*
2. *Every relation in $\mathcal{S}$ is* OR*-free.*
3. *Every relation in $\mathcal{S}$ is* NAND*-free.*

A logical relation R is tight *if the singleton $\{R\}$ is tight.*

In section 4, we show that if $\mathcal{S}$ is Schaefer, then it is tight. Moreover, we show that the converse does not hold. It is also easy to see that there is a polynomial-time algorithm (in fact, a cubic algorithm) for testing whether a given relation is tight.

Just as Schaefer's dichotomy theorem follows from an expressibility statement, our dichotomy theorems are derived from the following theorem, which we will call the structural expressibility theorem. The precise definition of the concept of *structural expressibility* is given in section 3. Intuitively, this concept strengthens the concept of expressibility with the requirement that the space of the witnesses to the existentially quantified variables has certain strong connectivity properties.

THEOREM 2.7 (structural expressibility theorem). *Let $\mathcal{S}$ be a finite set of logical relations. If $\mathcal{S}$ is not tight, then every logical relation is structurally expressible from $\mathcal{S}$.*

Using the structural expressibility theorem, we obtain the following dichotomy theorems for the computational complexity of $\textsc{Conn}(\mathcal{S})$ and $\textsc{st-Conn}(\mathcal{S})$.

THEOREM 2.8. *Let $\mathcal{S}$ be a finite set of logical relations. If $\mathcal{S}$ is tight, then* $\textsc{Conn}(\mathcal{S})$ *is in* coNP*; otherwise, it is* PSPACE*-complete.*

THEOREM 2.9. *Let $\mathcal{S}$ be a finite set of logical relations. If $\mathcal{S}$ is tight, then* $\textsc{st-Conn}(\mathcal{S})$ *is in* P*; otherwise,* $\textsc{st-Conn}(\mathcal{S})$ *is* PSPACE*-complete.*

We also show that if $\mathcal{S}$ is tight, but not Schaefer, then $\textsc{Conn}(\mathcal{S})$ is coNP-complete.

Example 1. The set $\mathcal{S} = \{R_{1/3}\}$, where $R_{1/3} = \{100, 010, 001\}$, is tight (actually, it is componentwise bijunctive), but not Schaefer. It follows that $\textsc{Sat}(\mathcal{S})$ is NP-complete (recall that this problem is POSITIVE 1-IN-3 SAT), $\textsc{st-Conn}(\mathcal{S})$ is in P, and $\textsc{Conn}(\mathcal{S})$ is coNP-complete.

Example 2. The set $\mathcal{S} = \{R_{\mathrm{NAE}}\}$, where $R_{\mathrm{NAE}} = \{0,1\}^3 \setminus \{000, 111\}$, is not tight; hence $\textsc{Sat}(\mathcal{S})$ is NP-complete (this problem is POSITIVE NOT-ALL-EQUAL 3-SAT), while both $\textsc{st-Conn}(\mathcal{S})$ and $\textsc{Conn}(\mathcal{S})$ are PSPACE-complete.

Example 3. The set $\mathcal{S} = \{R_V\}$, where $R_V = \{110, 100, 000, 001, 011\}$, is not tight. It is OR-free, but not NAND-free or componentwise bijunctive, and it is not Schaefer. Hence $\textsc{Sat}(\mathcal{S})$ is NP-complete, $\textsc{st-Conn}(\mathcal{S})$ is in P, and $\textsc{Conn}(\mathcal{S})$ is in coNP.

Remark. The last example illustrates that OR-free (NAND-free) relations cannot be thought of as componentwise Horn (dual Horn).

The dichotomy in the computational complexity of CONN($\mathcal{S}$) and ST-CONN($\mathcal{S}$) is accompanied by a parallel structural dichotomy in the size of the diameter of $G(\varphi)$ (where, for a CNF($\mathcal{S}$)-formula φ, the *diameter of* $G(\varphi)$ is the maximum of the diameters of the components of $G(\varphi)$).

THEOREM 2.10. *Let $\mathcal{S}$ be a finite set of logical relations. If $\mathcal{S}$ is tight, then for every* CNF($\mathcal{S}$)*-formula φ, the diameter of $G(\varphi)$ is linear in the number of variables of φ; otherwise, there are* CNF($\mathcal{S}$)*-formulas φ such that the diameter of $G(\varphi)$ is exponential in the number of variables of φ.*

Our results and their comparison to Schaefer's dichotomy theorem are summarized in the table below.

$\mathcal{S}$	SAT($\mathcal{S}$)	ST-CONN($\mathcal{S}$)	CONN($\mathcal{S}$)	Diameter
Schaefer	P	P	coNP	$O(n)$
Tight, non-Schaefer	NP-complete	P	coNP-complete	$O(n)$
Nontight	NP-complete	PSPACE-complete	PSPACE-complete	$2^{\Omega(\sqrt{n})}$

We conjecture that the complexity of CONN($\mathcal{S}$) exhibits a *trichotomy*, that is, for every finite set $\mathcal{S}$ of logical relations, one of the following holds:

1. CONN($\mathcal{S}$) is in P.
2. CONN($\mathcal{S}$) is coNP-complete.
3. CONN($\mathcal{S}$) is PSPACE-complete.

As mentioned above, we will show that if $\mathcal{S}$ is tight but not Schaefer, then CONN($\mathcal{S}$) is coNP-complete. We will also show that if $\mathcal{S}$ is bijunctive or affine, then CONN($\mathcal{S}$) is in P. Hence, to settle the above conjecture, it remains to pinpoint the complexity of CONN($\mathcal{S}$) whenever $\mathcal{S}$ is Horn and whenever $\mathcal{S}$ is dual Horn. In the conference version [16] of the present paper, we further conjectured that if $\mathcal{S}$ is Horn or dual Horn, then CONN($\mathcal{S}$) is in P. In other words, we conjectured that if $\mathcal{S}$ is Schaefer, then CONN($\mathcal{S}$) is in P. This second conjecture, however, was subsequently disproved by Makino, Tamaki, and Yamamoto [24], who discovered a particular Horn set $\mathcal{S}$ such that CONN($\mathcal{S}$) is coNP-complete. Here, we go beyond the results obtained in the conference version of the present paper and identify additional conditions on a Horn set $\mathcal{S}$ implying that CONN($\mathcal{S}$) is in P. These new results suggest a natural dichotomy within Schaefer sets of relations and, thus, provide evidence for the trichotomy conjecture.

The remainder of this paper is organized as follows. In section 3, we prove the structural expressibility theorem, establish the hard side of the dichotomies for CONN($\mathcal{S}$) and for ST-CONN($\mathcal{S}$), and contrast our result to Schaefer's expressibility and dichotomy theorems. In section 4, we describe the easy side of the dichotomy—the polynomial-time algorithms and the structural properties for tight sets of relations. In addition, we obtain partial results towards the trichotomy conjecture for CONN($\mathcal{S}$).

3. The hard case of the dichotomy: Nontight sets of relations. In this section, we address the *hard* side of the dichotomy, where we deal with the more computationally intractable cases. This is also the harder part of our proof. We define the notion of structural expressibility in section 3.1 and prove the structural expressibility theorem in section 3.2. This theorem implies that for all nontight sets $\mathcal{S}$ and $\mathcal{S}'$, the connectivity problems CONN($\mathcal{S}$) and CONN($\mathcal{S}'$) are polynomial-time equivalent; moreover, the same holds for the connectivity problems ST-CONN($\mathcal{S}$) and ST-CONN($\mathcal{S}'$). In addition, the diameters of the solution graphs of CNF($\mathcal{S}$)-formulas and CNF($\mathcal{S}'$)-formulas are also related polynomially. In section 3.3, we prove that for 3-CNF-formulas the connectivity problems are PSPACE-complete, and the diameter can be exponential. This fact combined with the structural expressibility

theorem yields the hard side of all of our dichotomy results, as well as the exponential size of the diameter.

We will use $\mathbf{a}, \mathbf{b}, \ldots$ to denote Boolean vectors, and $\mathbf{x}$ and $\mathbf{y}$ to denote vectors of variables. We write $|\mathbf{a}|$ to denote the Hamming weight (number of 1's) of a Boolean vector $\mathbf{a}$. Given two Boolean vectors $\mathbf{a}$ and $\mathbf{b}$, we write $|\mathbf{a}-\mathbf{b}|$ to denote the Hamming distance between $\mathbf{a}$ and $\mathbf{b}$. Finally, if $\mathbf{a}$ and $\mathbf{b}$ are solutions of a Boolean formula φ and lie in the same component of $G(\varphi)$, then we write $d_\varphi(\mathbf{a}, \mathbf{b})$ to denote the shortest-path distance between $\mathbf{a}$ and $\mathbf{b}$ in $G(\varphi)$.

3.1. Structural expressibility. As stated in the previous section, in his dichotomy theorem, Schaefer [31] used the following notion of expressibility: a relation R is *expressible from* a set $\mathcal{S}$ of relations if there is a CNF($\mathcal{S}$)-formula φ so that $R = \{\mathbf{a}|\ \exists \mathbf{y} \varphi(\mathbf{a}, \mathbf{y})\}$. This notion is not sufficient for our purposes. Instead, we introduce a more delicate notion which we call *structural expressibility*. Intuitively, we view the relation R as a subgraph of the hypercube, rather than just a subset, and require that this graph structure also be captured by the formula φ.

DEFINITION 3.1. *A relation R is* structurally expressible *from a set of relations $\mathcal{S}$ if there is a* CNF($\mathcal{S}$)*-formula φ such that the following conditions hold:*

1. $R = \{\mathbf{a}|\ \exists \mathbf{y} \varphi(\mathbf{a}, \mathbf{y})\}$.
2. *For every $\mathbf{a} \in R$, the graph $G(\varphi(\mathbf{a}, \mathbf{y}))$ is connected.*
3. *For $\mathbf{a}, \mathbf{b} \in R$ with $|\mathbf{a} - \mathbf{b}| = 1$, there exists $\mathbf{w}$ such that $(\mathbf{a}, \mathbf{w})$ and $(\mathbf{b}, \mathbf{w})$ are solutions of φ.*

For $\mathbf{a} \in R$, the *witnesses* of $\mathbf{a}$ are the $\mathbf{y}$'s such that $\varphi(\mathbf{a}, \mathbf{y})$ is true. The last two conditions say that the witnesses of $\mathbf{a} \in R$ are connected, and that neighboring $\mathbf{a}, \mathbf{b} \in R$ have a common witness. This allows us to simulate an edge $(\mathbf{a}, \mathbf{b})$ in $G(R)$ by a path in $G(\varphi)$, and thus relate the connectivity properties of the solution spaces. There is, however, a price to pay: it is much harder to come up with formulas that structurally express a relation R. An example is when $\mathcal{S}$ is the set of all paths of length 4 in $\{0,1\}^3$, a set that plays a crucial role in our proof. While 3-SAT relations are easily expressible from $\mathcal{S}$ in Schaefer's sense, the CNF($\mathcal{S}$)-formulas that structurally express 3-SAT relations are fairly complicated and have a large witness space.

An example of the difference between a structural and a nonstructural expression is shown in Figure 3.1. Consider the logical relation given by the formula $(x_1 \vee x_2 \vee x_3)$; the graph of this logical relation is depicted in Figure 3.1(a). Consider also the NOT-ALL-EQUAL relation $R_{\text{NAE}} = \{0,1\}^3 \setminus \{000, 111\}$. Figure 3.1(b) depicts the graph of the expression $\varphi(x_1, x_2, x_3, y_1, y_2) = R_{\text{NAE}}(x_1, x_2, y_1) \wedge R_{\text{NAE}}(x_2, x_3, y_2) \wedge R_{\text{NAE}}(y_1, y_2, 1)$. This is a structural expression because $(x_1 \vee x_2 \vee x_3) \equiv \exists y_1, y_2 \varphi(x_1, x_2, x_3, y_1, y_2)$ and connectivity is preserved. Finally, Figure 3.1(c) depicts the graph of the expression $\psi(x_1, x_2, x_3, y_1) = R_{\text{NAE}}(x_1, x_2, y_1) \wedge R_{\text{NAE}}(\bar{y}_1, x_3, 0) \wedge R_{\text{NAE}}(y_1, x_2, 1)$. Even though $(x_1 \vee x_2 \vee x_3) \equiv \exists y_1 \psi(x_1, x_2, x_3, y_1)$, this is *not* a structural expression because connectivity is not preserved.

LEMMA 3.2. *Let $\mathcal{S}$ and $\mathcal{S}'$ be sets of relations such that every $R \in \mathcal{S}'$ is structurally expressible from $\mathcal{S}$, and, moreover, there is a polynomial-time algorithm that produces a structural expression from $\mathcal{S}$ for every $R \in \mathcal{S}'$. Given a* CNF($\mathcal{S}'$)*-formula $\psi(\mathbf{x})$, one can efficiently construct a* CNF($\mathcal{S}$)*-formula $\varphi(\mathbf{x}, \mathbf{y})$ such that*

1. *$\psi(\mathbf{x}) \equiv \exists \mathbf{y}\ \varphi(\mathbf{x}, \mathbf{y})$;*
2. *if $(\mathbf{s}, \mathbf{w^s}), (\mathbf{t}, \mathbf{w^t}) \in \varphi$ are connected in $G(\varphi)$ by a path of length d, then there is a path from $\mathbf{s}$ to $\mathbf{t}$ in $G(\psi)$ of length at most d;*
3. *if $\mathbf{s}, \mathbf{t} \in \psi$ are connected in $G(\psi)$, then for every witness $\mathbf{w^s}$ of $\mathbf{s}$, and every witness $\mathbf{w^t}$ of $\mathbf{t}$, there is a path from $(\mathbf{s}, \mathbf{w^s})$ to $(\mathbf{t}, \mathbf{w^t})$ in $G(\varphi)$.*

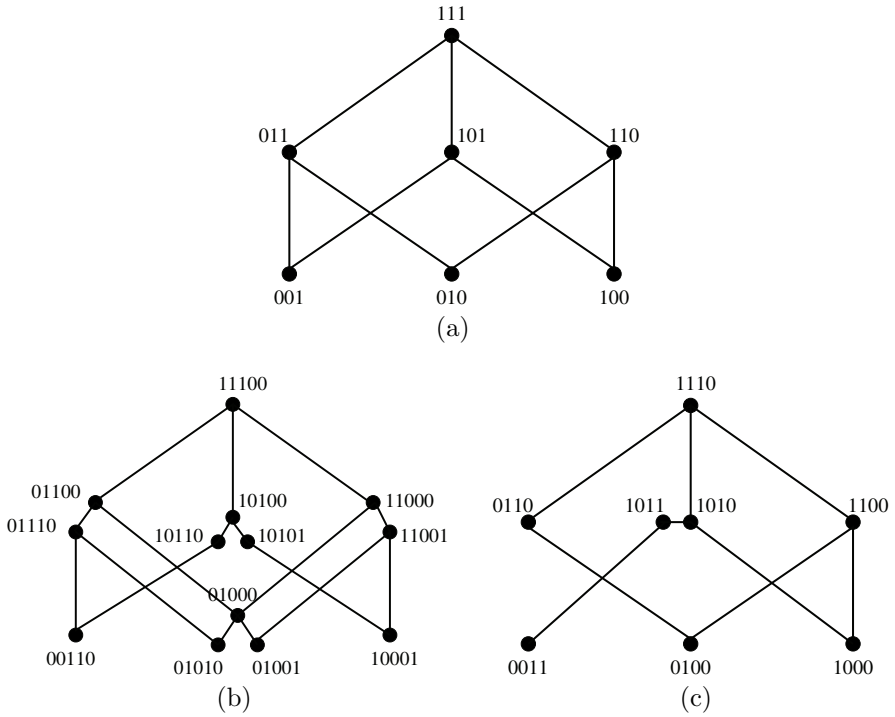


FIG. 3.1. *Expressing the relation* $(x_1 \vee x_2 \vee x_3)$ *from the* R_{NAE} *relation.*

Proof. Suppose ψ is a formula on n variables that consists of m clauses $C_1, \ldots, C_m$. For clause C_j, assume that the set of variables is $V_j \subseteq [n] = \{1, \ldots, n\}$, and that it involves relation $R_j \in \mathcal{S}$. Thus, $\psi(\mathbf{x})$ is $\wedge_{j=1}^{m} R_j(\mathbf{x}_{V_j})$. Let φ_j be the structural expression for R_j from $\mathcal{S}'$, so that $R_j(\mathbf{x}_{V_j}) \equiv \exists \mathbf{y}_j \ \varphi_j(\mathbf{x}_{V_j}, \mathbf{y}_j)$. Let $\mathbf{y}$ be the vector $(\mathbf{y}_1, \ldots, \mathbf{y}_m)$ and let $\varphi(\mathbf{x}, \mathbf{y})$ be the formula $\wedge_{j=1}^{m} \varphi_j(\mathbf{x}_{V_j}, \mathbf{y}_j)$. Then $\psi(\mathbf{x}) \equiv \exists \mathbf{y}\ \varphi(\mathbf{x}, \mathbf{y})$.

Statement 2 follows from 1 by projection of the path on the coordinates of $\mathbf{x}$. For statement 3, consider $\mathbf{s}, \mathbf{t} \in \psi$ that are connected in $G(\psi)$ via a path $\mathbf{s} = \mathbf{u^0} \to \mathbf{u^1} \to \cdots \to \mathbf{u^r} = \mathbf{t}$. For every $\mathbf{u^i}, \mathbf{u^{i+1}}$, and clause C_j, there exists an assignment $\mathbf{w^i}_j$ to $\mathbf{y}_j$ such that both $(\mathbf{u^i}_{V_j}, \mathbf{w^i}_j)$ and $(\mathbf{u^{i+1}}_{V_j}, \mathbf{w^i}_j)$ are solutions of φ_j, by condition 2 of structural expressibility. Thus $(\mathbf{u^i}, \mathbf{w^i})$ and $(\mathbf{u^{i+1}}, \mathbf{w^i})$ are both solutions of φ, where $\mathbf{w^i} = (\mathbf{w^i}_1, \ldots, \mathbf{w^i}_m)$. Further, for every $\mathbf{u^i}$, the space of solutions of $\varphi(\mathbf{u^i}, \mathbf{y})$ is the product space of the solutions of $\varphi_j(\mathbf{u^i}_{V_j}, \mathbf{y}_j)$ over $j = 1, \ldots, m$. Since these are all connected by condition 3 of structural expressibility, $G(\varphi(\mathbf{u^i}, \mathbf{y}))$ is connected. The following describes a path from $(\mathbf{s}, \mathbf{w^s})$ to $(\mathbf{t}, \mathbf{w^t})$ in $G(\varphi)$: $(\mathbf{s}, \mathbf{w^s}) \rightsquigarrow (\mathbf{s}, \mathbf{w^0}) \to (\mathbf{u^1}, \mathbf{w^0}) \rightsquigarrow (\mathbf{u^1}, \mathbf{w^1}) \to \cdots \rightsquigarrow (\mathbf{u^{r-1}}, \mathbf{w^{r-1}}) \to (\mathbf{t}, \mathbf{w^{r-1}}) \rightsquigarrow (\mathbf{t}, \mathbf{w^t})$. Here $\rightsquigarrow$ indicates a path in $G(\varphi(\mathbf{u^i}, \mathbf{y}))$. □

COROLLARY 3.3. *Suppose* $\mathcal{S}$ *and* $\mathcal{S}'$ *are sets of relations such that every* $R \in \mathcal{S}'$ *is structurally expressible from* $\mathcal{S}$*, and, moreover, there is a polynomial-time algorithm that produces a structural expression from* $\mathcal{S}$ *for every* $R \in \mathcal{S}'$*.*

1. *There are polynomial-time reductions from* CONN($\mathcal{S}'$) *to* CONN($\mathcal{S}$)*, and from* ST-CONN($\mathcal{S}'$) *to* ST-CONN($\mathcal{S}$)*.*
2. *If there exists a* CNF($\mathcal{S}'$)*-formula* $\psi(\mathbf{x})$ *with* n *variables,* m *clauses, and diameter* d*, then there exists a* CNF($\mathcal{S}$)*-formula* $\varphi(\mathbf{x}, \mathbf{y})$*, where* $\mathbf{y}$ *is a vector of* $O(m)$ *variables, such that the diameter of* $G(\phi)$ *is at least* d*.*

3.2. The structural expressibility theorem. In this subsection, we prove the structural expressibility theorem. The main step in the proof is Lemma 3.4, which shows that if $\mathcal{S}$ is not tight, then we can structurally express the 3-clause relations from the relations in $\mathcal{S}$. If $k \geq 2$, then a *k-clause* is a disjunction of k variables or negated variables. For $0 \leq i \leq k$, let D_i be the set of all satisfying truth assignments of the k-clause whose first i literals are negated, and let $\mathcal{S}_k = \{D_0, D_1, \ldots, D_k\}$. Thus, $\mathrm{CNF}(\mathcal{S}_k)$ is the collection of k-CNF-formulas.

LEMMA 3.4. *If set $\mathcal{S}$ of relations is not tight, $\mathcal{S}_3$ is structurally expressible from $\mathcal{S}$.*

Proof. First, observe that all 2-clauses are structurally expressible from $\mathcal{S}$. There exists $R \in \mathcal{S}$ which is not OR-free, so we can express $(x_1 \vee x_2)$ by substituting constants in R. Similarly, we can express $(\bar{x}_1 \vee \bar{x}_2)$ using a relation that is not NAND-free. The last 2-clause $(x_1 \vee \bar{x}_2)$ can be obtained from OR and NAND by a technique that corresponds to reverse resolution. $(x_1 \vee \bar{x}_2) = \exists y\ (x_1 \vee y) \wedge (\bar{y} \vee \bar{x}_2)$. It is easy to see that this gives a structural expression. From here onwards we assume that $\mathcal{S}$ contains all 2-clauses. The proof now proceeds in four steps. First, we will express a relation in which there exist two elements that are at graph distance larger than their Hamming distance. Second, we will express a relation that is just a single path between such elements. Third, we will express a relation which is a path of length 4 between elements at Hamming distance 2. Finally, we will express the 3-clauses.

Step 1. *Structurally expressing a relation in which some distance expands.* For a relation R, we say that the distance between $\mathbf{a}$ and $\mathbf{b}$ *expands* if $\mathbf{a}$ and $\mathbf{b}$ are connected in $G(R)$, but $d_R(\mathbf{a}, \mathbf{b}) > |\mathbf{a} - \mathbf{b}|$. In section 4.2, Lemma 4.3, we will show that no distance expands in componentwise bijunctive relations. The same also holds true for the relation $R_{\mathrm{NAE}} = \{0,1\}^3 \setminus \{000, 111\}$, which is not componentwise bijunctive. Nonetheless, we show here that if R is not componentwise bijunctive, then, by adding 2-clauses, we can structurally express a relation Q in which some distance expands. For instance, when $R = R_{\mathrm{NAE}}$, then we can take $Q(x_1, x_2, x_3) = R_{\mathrm{NAE}}(x_1, x_2, x_3) \wedge (\bar{x_1} \vee \bar{x}_3)$, as shown in Figure 3.2. The distance between $\mathbf{a} = 100$ and $\mathbf{b} = 001$ in Q expands. Similarly, in the general construction, we identify $\mathbf{a}$ and $\mathbf{b}$ on a cycle, and add 2-clauses that eliminate all the vertices along the shorter arc between $\mathbf{a}$ and $\mathbf{b}$.

Since $\mathcal{S}$ is not tight, it contains a relation R which is not componentwise bijunctive. If R contains $\mathbf{a}, \mathbf{b}$ where the distance between them expands, we are done. So assume that for all $\mathbf{a}, \mathbf{b} \in G(R)$, $d_R(\mathbf{a}, \mathbf{b}) = |\mathbf{a} - \mathbf{b}|$. Since R is not componentwise bijunctive, there exists a triple of assignments $\mathbf{a}, \mathbf{b}, \mathbf{c}$ lying in the same component such that $\mathrm{maj}(\mathbf{a}, \mathbf{b}, \mathbf{c})$ is not in that component. Choose the triple such that the sum of pairwise distances $d_R(\mathbf{a}, \mathbf{b}) + d_R(\mathbf{b}, \mathbf{c}) + d_R(\mathbf{c}, \mathbf{a})$ is minimized. Let $U = \{i | a_i \neq b_i\}$, $V = \{i | b_i \neq c_i\}$, and $W = \{i | c_i \neq a_i\}$. Since $d_R(\mathbf{a}, \mathbf{b}) = |\mathbf{a} - \mathbf{b}|$, a shortest path does not flip variables outside of U, and each variable in U is flipped exactly once. The same holds for V and W. We note some useful properties of the sets U, V, W:

1. *Every index $i \in U \cup V \cup W$ occurs in exactly two of U, V, W.* Consider going by a shortest path from $\mathbf{a}$ to $\mathbf{b}$ to $\mathbf{c}$ and back to $\mathbf{a}$. Every $i \in U \cup V \cup W$ is seen an even number of times along this path since we return to $\mathbf{a}$. It is seen at least once, and at most thrice, so in fact it occurs twice.
2. *Every pairwise intersection $U \cap V, V \cap W$, and $W \cap U$ is nonempty.* Suppose the sets U and V are disjoint. From property 1, we must have $W = U \cup V$. But then it is easy to see that $\mathrm{maj}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \mathbf{b}$. This contradicts the choice of $\mathbf{a}, \mathbf{b}, \mathbf{c}$.

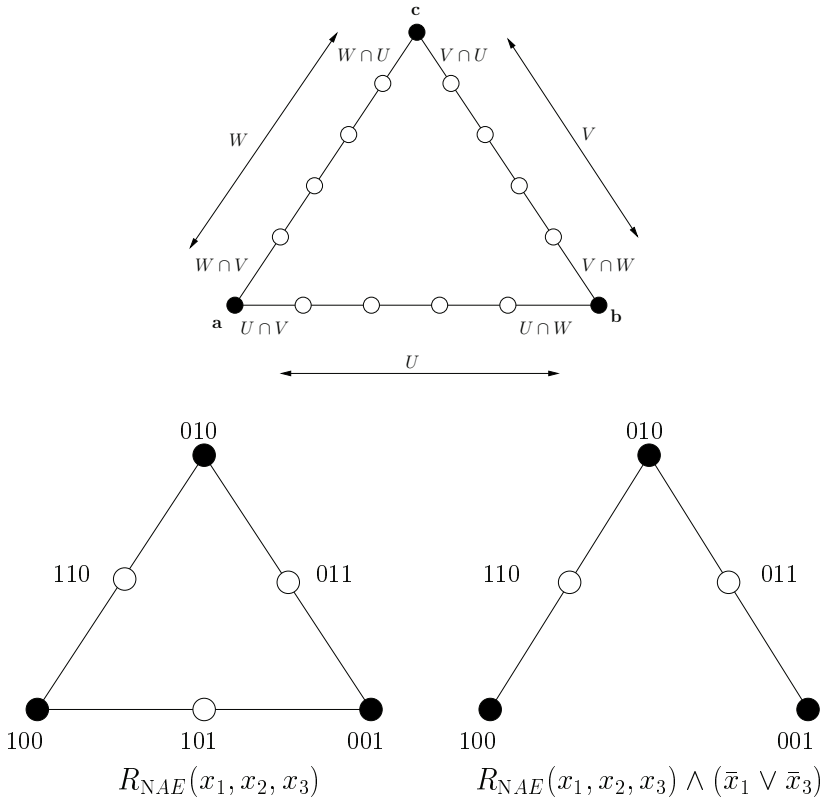


FIG. 3.2. *Step* 1 *of the proof of Lemma* 3.4*, and an example.*

3. *The sets* $U \cap V$ *and* $U \cap W$ *partition the set* U. By property 1, each index of U occurs in one of V and W as well. Also since no index occurs in all three sets U, V, W this is in fact a disjoint partition.
4. *For each index* $i \in U \cap W$*, it holds that* $\mathbf{a} \oplus \mathbf{e}_i \notin R$. Assume for the sake of contradiction that $\mathbf{a}' = \mathbf{a} \oplus \mathbf{e}_i \in R$. Since $i \in U \cap W$ we have simultaneously moved closer to both $\mathbf{b}$ and $\mathbf{c}$. Hence we have $d_R(\mathbf{a}', \mathbf{b}) + d_R(\mathbf{b}, \mathbf{c}) + d_R(\mathbf{c}, \mathbf{a}') < d_R(\mathbf{a}, \mathbf{b}) + d_R(\mathbf{b}, \mathbf{c}) + d_R(\mathbf{c}, \mathbf{a})$. Also $\mathrm{maj}(\mathbf{a}', \mathbf{b}, \mathbf{c}) = \mathrm{maj}(\mathbf{a}, \mathbf{b}, \mathbf{c})$. But this contradicts our choice of $\mathbf{a}, \mathbf{b}, \mathbf{c}$.

Property 4 implies that the shortest paths to $\mathbf{b}$ and $\mathbf{c}$ diverge at $\mathbf{a}$, since for any shortest path to $\mathbf{b}$ the first variable flipped is from $U \cap V$, whereas for a shortest path to $\mathbf{c}$ it is from $W \cap V$. Similar statements hold for the vertices $\mathbf{b}$ and $\mathbf{c}$. Thus along the shortest path from $\mathbf{a}$ to $\mathbf{b}$ the first bit flipped is from $U \cap V$ and the last bit flipped is from $U \cap W$. On the other hand, if we go from $\mathbf{a}$ to $\mathbf{c}$ and then to $\mathbf{b}$, all the bits from $U \cap W$ are flipped before the bits from $U \cap V$. We use this crucially to define Q. We will add a set of 2-clauses that enforce the following rule on paths starting at $\mathbf{a}$: *Flip variables from* $U \cap W$ *before variables from* $U \cap V$. This will eliminate all shortest paths from $\mathbf{a}$ to $\mathbf{b}$ since they begin by flipping a variable in $U \cap V$ and end with $U \cap W$. The paths from $\mathbf{a}$ to $\mathbf{b}$ via $\mathbf{c}$ survive since they flip $U \cap W$ while going from $\mathbf{a}$ to $\mathbf{c}$ and $U \cap V$ while going from $\mathbf{c}$ to $\mathbf{b}$. However, all remaining paths have length at least $|\mathbf{a} - \mathbf{b}| + 2$ since they flip twice some variables not in U.

Take all pairs of indices $\{(i, j) | i \in U \cap W, j \in U \cap V\}$. The following conditions

hold from the definition of U, V, W: $a_i = \bar{c}_i = \bar{b}_i$ and $a_j = c_j = \bar{b}_j$. Add the 2-clause C_{ij} asserting that the pair of variables $x_i x_j$ must take values in $\{a_i a_j, c_i c_j, b_i b_j\} = \{a_i a_j, \bar{a}_i a_j, \bar{a}_i \bar{a}_j\}$. The new relation is $Q = R \wedge_{i,j} C_{ij}$. Note that $Q \subset R$. We verify that the distance between $\mathbf{a}$ and $\mathbf{b}$ in Q expands. It is easy to see that for any $j \in U$, the assignment $\mathbf{a} \oplus \mathbf{e}_j \notin Q$. Hence there are no shortest paths left from $\mathbf{a}$ to $\mathbf{b}$. On the other hand, it is easy to see that $\mathbf{a}$ and $\mathbf{b}$ are still connected, since the vertex $\mathbf{c}$ is still reachable from both.

Step 2. *Isolating a pair of assignments whose distance expands.* The relation Q obtained in Step 1 may have several disconnected components. This *cleanup* step isolates a single pair of assignments whose distance expands. By adding 2-clauses, we show that one can express a path of length $r+2$ between assignments at distance r.

Take $\mathbf{a}, \mathbf{b} \in Q$ whose distance expands in Q and $d_Q(\mathbf{a}, \mathbf{b})$ is minimized. Let $U = \{i | a_i \neq b_i\}$ and $|U| = r$. Shortest paths between $\mathbf{a}$ and $\mathbf{b}$ have certain useful properties:

1. *Each shortest path flips every variable from U exactly once.* Observe that each index $j \in U$ is flipped an odd number of times along any path from $\mathbf{a}$ to $\mathbf{b}$. Suppose it is flipped thrice along a shortest path. Starting at $\mathbf{a}$ and going along this path, let $\mathbf{b}'$ be the assignment reached after flipping j twice. Then the distance between $\mathbf{a}$ and $\mathbf{b}'$ expands, since j is flipped twice along a shortest path between them in Q. Also $d_Q(\mathbf{a}, \mathbf{b}') < d_Q(\mathbf{a}, \mathbf{b})$, contradicting the choice of $\mathbf{a}$ and $\mathbf{b}$.
2. *Every shortest path flips exactly one variable $i \notin U$.* Since the distance between $\mathbf{a}$ and $\mathbf{b}$ expands, every shortest path must flip some variable $i \notin U$. Suppose it flips more than one such variable. Since $\mathbf{a}$ and $\mathbf{b}$ agree on these variables, each of them is flipped an even number of times. Let i be the first variable to be flipped twice. Let $\mathbf{b}'$ be the assignment reached after flipping i the second time. It is easy to verify that the distance between $\mathbf{a}$ and $\mathbf{b}'$ also expands, but $d_Q(\mathbf{a}, \mathbf{b}') < d_Q(\mathbf{a}, \mathbf{b})$.
3. *The variable $i \notin U$ is the first and last variable to be flipped along the path.* Assume the first variable flipped is not i. Let $\mathbf{a}'$ be the assignment reached along the path before we flip i the first time. Then $d_Q(\mathbf{a}', \mathbf{b}) < d_Q(\mathbf{a}, \mathbf{b})$. The distance between $\mathbf{a}'$ and $\mathbf{b}$ expands since the shortest path between them flips the variables i twice. This contradicts the choice of $\mathbf{a}$ and $\mathbf{b}$. Assume $j \in U$ is flipped twice. Then as before we get a pair $\mathbf{a}', \mathbf{b}'$ that contradict the choice of $\mathbf{a}, \mathbf{b}$.

Every shortest path between $\mathbf{a}$ and $\mathbf{b}$ has the following structure: first a variable $i \notin U$ is flipped to $\bar{a}_i$, then the variables from U are flipped in some order, and finally the variable i is flipped back to a_i.

Different shortest paths may vary in the choice of $i \notin U$ in the first step and in the order in which the variables from U are flipped. Fix one such path $T \subseteq Q$. Assume that $U = \{1, \ldots, r\}$ and the variables are flipped in this order, and the additional variable flipped twice is $r+1$. Denote the path by $\mathbf{a} \to \mathbf{u^0} \to \mathbf{u^1} \to \cdots \to \mathbf{u^r} \to \mathbf{b}$. Next we prove that we cannot flip the $r+1$th variable at an intermediate vertex along the path.

4. *For $1 \leq j \leq r-1$ the assignment $\mathbf{u^j} \oplus \mathbf{e_{r+1}} \notin Q$.* Suppose that for some j we have $\mathbf{c} = \mathbf{u^j} \oplus \mathbf{e_{r+1}} \in Q$. Then $\mathbf{c}$ differs from $\mathbf{a}$ on $\{1, \ldots, i\}$ and from $\mathbf{b}$ on $\{i+1, \ldots, r\}$. The distance from $\mathbf{c}$ to at least one of $\mathbf{a}$ or $\mathbf{b}$ must expand, or else we get a path from $\mathbf{a}$ to $\mathbf{b}$ through $\mathbf{c}$ of length $|\mathbf{a}-\mathbf{b}|$, which contradicts the fact that this distance expands. However, $d_Q(\mathbf{a}, \mathbf{c})$ and $d_Q(\mathbf{b}, \mathbf{c})$ are strictly less than $d_Q(\mathbf{a}, \mathbf{b})$, so we get a contradiction to the choice of $\mathbf{a}, \mathbf{b}$.

We now construct the path of length $r+2$. For all $i \geq r+2$ we set $x_i = a_i$ to get a relation on $r+1$ variables. Note that $\mathbf{b} = \bar{a}_1 \dots \bar{a}_r a_{r+1}$. Take $i < j \in U$. Along the path T the variable i is flipped before j so the variables $x_i x_j$ take one of three values $\{a_i a_j, \bar{a}_i a_j, \bar{a}_i \bar{a}_j\}$. So we add a 2-clause C_{ij} that requires $x_i x_j$ to take one of these values and take $T = Q \wedge_{i,j} C_{ij}$. Clearly, every assignment along the path lies in T. We claim that these are the only solutions. To show this, take an arbitrary assignment $\mathbf{c}$ satisfying the added constraints. If for some $i < j \leq r$ we have $c_i = a_i$ but $c_j = \bar{a}_j$, this would violate C_{ij}. Hence the first r variables of $\mathbf{c}$ are of the form $\bar{a}_1 \dots \bar{a}_i a_{i+1} \dots a_r$ for $0 \leq i \leq r$. If $c_{r+1} = \bar{a}_{r+1}$, then $\mathbf{c} = \mathbf{u}^i$. If $c_{r+1} = a_{r+1}$, then $\mathbf{c} = \mathbf{u}^i \oplus \mathbf{e}_{r+1}$. By property 4 above, such a vector satisfies Q if and only if $i = 0$ or $i = r$, which correspond to $\mathbf{c} = \mathbf{a}$ and $\mathbf{c} = \mathbf{b}$, respectively.

Step 3. *Structurally expressing paths of length* 4. Let $\mathcal{P}$ denote the set of all ternary relations whose graph is a path of length 4 between two assignments at Hamming distance 2. Up to permutations of coordinates, there are 6 such relations. Each of them is the conjunction of a 3-clause and a 2-clause. For instance, the relation $M = \{100, 110, 010, 011, 001\}$ can be written as $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3)$. (It is named so because its graph looks like the letter M on the cube.) These relations are "minimal" examples of relations that are not componentwise bijunctive. By projecting out intermediate variables from the path T obtained in Step 2, we structurally express one of the relations in $\mathcal{P}$. We structurally express other relations in $\mathcal{P}$ using this relation.

We will write all relations in $\mathcal{P}$ in terms of $M(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3)$ by negating variables. For example, $M(\bar{x}_1, x_2, x_3) = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_3) = \{000, 010, 110, 111, 101\}$.

Define the relation $P(x_1, x_{r+1}, x_2) = \exists x_3 \dots x_r \; T(x_1, \dots, x_{r+1})$. The table below, listing all tuples in P and their witnesses, shows that the conditions for structural expressibility are satisfied, and $P \in \mathcal{P}$.

x_1, x_2, x_{r+1}	$x_3, \dots, x_r$
$a_1 a_2 a_{r+1}$	$a_3 \dots a_r$
$a_1 a_2 \bar{a}_{r+1}$	$a_3 \dots a_r$
$\bar{a}_1 a_2 \bar{a}_{r+1}$	$a_3 \dots a_r$
$\bar{a}_1 \bar{a}_2 \bar{a}_{r+1}$	$a_3 \dots a_k,\ \bar{a}_3 a_4 \dots a_r,\ \bar{a}_3 \bar{a}_4 a_5 \dots a_r,\ \dots,\ \bar{a}_3 \bar{a}_4 \dots \bar{a}_r$
$\bar{a}_1 \bar{a}_2 a_{r+1}$	$\bar{a}_3 \bar{a}_4 \dots \bar{a}_r$

Let $P(x_1, x_2, x_3) = M(l_1, l_2, l_3)$, where l_i is one of $\{x_i, \bar{x}_i\}$. We can now use P and 2-clauses to express every other relation in $\mathcal{P}$. Given $M(l_1, l_2, l_3)$, every relation in $\mathcal{P}$ can be obtained by negating some subset of the variables. Hence it suffices to show that we can express structurally $M(\bar{l}_1, l_2, l_3)$ and $M(l_1, \bar{l}_2, l_3)$ (M is symmetric in x_1 and x_3). In the following let λ denote one of the literals $\{y, \bar{y}\}$, such that it is $\bar{y}$ if and only if l_1 is $\bar{x}_1$.

$$\begin{aligned} M(\bar{l}_1, l_2, l_3) &= (\bar{l}_1 \vee l_2 \vee l_3) \wedge (l_1 \vee \bar{l}_3) \\ &= \exists y \; (\bar{l}_1 \vee \bar{\lambda}) \wedge (\lambda \vee l_2 \vee l_3) \wedge (l_1 \vee \bar{l}_3) \\ &= \exists y \; (\bar{l}_1 \vee \bar{\lambda}) \wedge (\lambda \vee l_2 \vee l_3) \wedge (l_1 \vee \bar{l}_3) \wedge (\bar{\lambda} \vee \bar{l}_3) \\ &= \exists y \; (\bar{l}_1 \vee \bar{\lambda}) \wedge (l_1 \vee \bar{l}_3) \wedge M(\lambda, l_2, l_3) \\ &= \exists y \; (\bar{l}_1 \vee \bar{\lambda}) \wedge (l_1 \vee \bar{l}_3) \wedge P(y, x_2, x_3). \end{aligned}$$

In the second step the clause $(\bar{\lambda} \vee \bar{l}_3)$ is implied by the resolution of the clauses $(\bar{l}_1 \vee \bar{\lambda}) \wedge (l_1 \vee \bar{l}_3)$.

For the next expression let λ denote one of the literals $\{y, \bar{y}\}$, such that it is negated if and only if l_2 is $\bar{x}_2$.

$$\begin{aligned} M(l_1, \bar{l}_2, l_3) &= (l_1 \vee \bar{l}_2 \vee l_3) \wedge (\bar{l}_1 \vee \bar{l}_3) \\ &= \exists y\ (l_1 \vee l_3 \vee \lambda) \wedge (\bar{\lambda} \vee \bar{l}_2) \wedge (\bar{l}_1 \vee \bar{l}_3) \\ &= \exists y\ (\bar{\lambda} \vee \bar{l}_2) \wedge M(l_1, \lambda, l_3) \\ &= \exists y\ (\bar{\lambda} \vee \bar{l}_2) \wedge P(x_1, y, x_3). \end{aligned}$$

The above expressions are both based on resolution, and it is easy to check that they satisfy the properties of structural expressibility.

Step 4. *Structurally expressing* $\mathcal{S}_3$. We structurally express $(x_1 \vee x_2 \vee x_3)$ from M using a formula derived from a gadget in [17]. This gadget expresses $(x_1 \vee x_2 \vee x_3)$ in terms of "Protected OR," which corresponds to our relation M.

$$\begin{aligned} (x_1 \vee x_2 \vee x_3) = \exists y_1 \dots y_5\ &(x_1 \vee \bar{y}_1) \wedge (x_2 \vee \bar{y}_2) \wedge (x_3 \vee \bar{y}_3) \wedge (x_3 \vee \bar{y}_4) \\ &\wedge M(y_1, y_5, y_3) \wedge M(y_2, \bar{y}_5, y_4). \end{aligned} \tag{3.1}$$

The table below, listing the witnesses of each assignment for (x_1, x_2, x_3), shows that the conditions for structural expressibility are satisfied.

x_1, x_2, x_3	$y_1 \dots y_5$											
111	00011	00111	00110	00100	01100	01101	01001	11001	11000	10000	10010	10011
110							01001	11001	11000	10000		
100										10000		
101	00011	00111	00110	00100						10000	10010	10011
001	00011	00111	00110	00100								
011	00011	00111	00110	00100	01100	01101	01001					
010							01001					

From the relation $(x_1 \vee x_2 \vee x_3)$ we derive the other 3-clauses by reverse resolution, for instance,

$$(\bar{x}_1 \vee x_2 \vee x_3) = \exists y\ (\bar{x}_1 \vee \bar{y}) \wedge (y \vee x_2 \vee x_3). \qquad \square$$

To complete the proof of the structural expressibility theorem, we show that an arbitrary relation can be expressed structurally from $\mathcal{S}_3$.

LEMMA 3.5. *Let* $R \subseteq \{0,1\}^k$ *be any relation of arity* $k \geq 1$. R *is structurally expressible from* $\mathcal{S}_3$.

Proof. If $k \leq 3$, then R can be expressed as a formula in $\mathrm{CNF}(\mathcal{S}_3)$ with constants, without introducing witness variables. This kind of expression is always structural.

If $k \geq 4$, then R can be expressed as a formula in $\mathrm{CNF}(\mathcal{S}_k)$, without witnesses (i.e., structurally). We will show that every k-clause can be expressed structurally from $\mathcal{S}_{k-1}$. Then, by induction, it can be expressed structurally from $\mathcal{S}_3$. For simplicity we express a k-clause corresponding to the relation D_0. The remaining relations are expressed equivalently. We express D_0 in a way that is standard in other complexity reductions and turns out to be structural:

$$(x_1 \vee x_2 \vee \dots \vee x_k) = \exists y\ (x_1 \vee x_2 \vee y) \wedge (\bar{y} \vee x_3 \vee \dots \vee x_k).$$

This is the reverse operation of resolution. For any satisfying assignment for $\mathbf{x}$, its witness space is either $\{0\}$, $\{1\}$, or $\{0,1\}$, so in all cases it is connected. Furthermore, the only way two neighboring satisfying assignments for x can have no common witness

is if one of them has witness set $\{0\}$ and the other one has witness set $\{1\}$. This implies that the first one has $(x_3, \ldots, x_k) = (0, \ldots, 0)$, and the other one has $(x_1, x_2) = (0, 0)$, and thus they differ in the assignments of at least two variables: one from $\{x_1, x_2\}$ and one from $\{x_3, \ldots, x_k\}$. In that case they cannot be neighboring assignments. Therefore all requirements of structural expressibility are satisfied. □

3.3. Hardness results for 3-CNF-formulas. From Lemma 3.4 and Corollary 3.3, it follows that, to prove the hard side of our dichotomy theorems, it suffices to focus on 3-CNF-formulas.

The proof that CONN($\mathcal{S}_3$) and ST-CONN($\mathcal{S}_3$) are PSPACE-complete is fairly intricate; it entails a direct reduction from the computation of a space-bounded Turing machine. The result for ST-CONN can also be proved easily using results of Hearne and Demaine on nondeterministic constraint logic [17]. However, it does not appear that completeness for CONN follows from their results.

LEMMA 3.6. ST-CONN($\mathcal{S}_3$) *and* CONN($\mathcal{S}_3$) *are* PSPACE*-complete.*

Proof. Given a CNF($\mathcal{S}_3$)-formula φ and satisfying assignments $\mathbf{s}, \mathbf{t}$, we can guess a path of length at most 2^n between them and verify that each vertex along the path is indeed a solution to φ. Hence ST-CONN($\mathcal{S}_3$) is in NPSPACE, which equals PSPACE by Savitch's theorem. Similarly for CONN($\mathcal{S}_3$), by reusing space we can check for all pairs of assignments whether they are satisfying and, if they both are, whether they are connected in $G(\varphi)$. It follows that both problems are in PSPACE.

Next we show that CONN($\mathcal{S}_3$) and ST-CONN($\mathcal{S}_3$) are PSPACE-hard. Let A be a language decided by a deterministic Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ in space n^k for some constant $k \geq 1$, where n is the length of the input. We give a polynomial-time reduction from A to ST-CONN($\mathcal{S}_3$) and CONN($\mathcal{S}_3$).

The reduction maps a string w (with $|w| = n$) to a 3-CNF-formula φ and two satisfying assignments for the formula, which are connected in $G(\varphi)$ if and only if M accepts w. Furthermore, all satisfying assignments of φ are connected to one of these two assignments, so that $G(\varphi)$ is connected if and only if M accepts w.

Before we show how to construct φ, we modify M in several ways to obtain a new machine M' which depends on w:

1. We define the tape of M' to be cyclical of length $n^k + 1$ with a special symbol # written in cell 0 which cannot be overwritten. The input w is placed after this symbol. Notice that the machine M accepts or rejects w within n^k space, and therefore the # symbol is never read when the machine is initialized in a legal way (at the state q_0, input from Σ^*, and the head at the initial position). The # symbol may be read if the machine is initialized at a different configuration. We modify the transitions of M so that if the # symbol is read, the machine M' goes into the state q_{reject}.
2. We add to M' a clock that counts from 0 to $n^k \times |Q| \times |\Gamma|^{n^k} = 2^{O(n^{k+1})}$, which is the total number of possible distinct configurations of M when it uses only n^k space. For this, M' uses a separate tape of length $O(n^{k+1})$ with the alphabet $\{0, 1\}$. Before a transition happens, control is passed on to the clock, its counter is incremented, and finally the transition is completed.
3. We define a standard accepting configuration. Whenever q_{accept} is reached, the clock is stopped and set to zero, the original tape is erased (except for #) and the head is placed in the initial position, always in state q_{accept}.
4. Whenever q_{reject} is reached the machine goes into its initial configuration. First w is written back on the input tape after the # symbol. This step requires adding n states to the machine in order to write the n letters of w.

This increases the number of states of M' to $O(n)$. Next, the rest of the tape is erased, the clock is set to zero, the head is placed in the initial position, and the state is set to q_0 (and thus the computation resumes).

5. Whenever the clock overflows, the machine goes into q_{reject}.

The new machine M' runs forever if w is not in A, and it accepts if w is in A. It also has the property that every configuration having # in position 0 leads either to the accepting configuration or to the initial configuration with input w. Therefore the space of such configurations is connected if and only if $w \in A$. Let's denote by Q' the states of M' and by δ' its transitions. As mentioned earlier, $|Q'| = O(n)$, and M' runs on two tapes, one of size $N = n^k$ and the other (for the clock) of size $N_c = O(n^{k+1})$. The alphabet of M' on one tape is Γ, and on the other $\{0,1\}$. For simplicity we can also assume that at each transition the machine uses only one of the two tapes and moves its head either left or right.

Next, we construct an intermediate CNF-formula ψ whose solutions are the configurations of M'. However, the space of solutions of ψ is disconnected.

For each $i \in [N]$ and $a \in \Gamma$, we have a variable $x(i,a)$. If $x(i,a) = 1$, this means that the ith tape cell contains symbol a. For every $i \in [N]$ there is a variable $y(i)$ which is 1 if the head is at position i. For every $q \in Q'$, there is a variable $z(q)$ which is 1 if the current state is q. Similarly for every $j \in [N_c]$ and $a \in \{0,1\}$ we have variables $x_c(j,a)$ and a variable $y_c(j)$ which is 1 if the head of the clock tape is at position j.

We enforce the following conditions:

1. Every cell contains some symbol and cell 0 contains #:

$$\psi_1 = \bigwedge_{i\in[N]} \left(\vee_{a\in\Gamma} \;\; x(i,a)\right) \bigwedge_{j\in[N_c]} \left(\vee_{a\in\{0,1\}} \;\; x_c(j,a)\right) \wedge x(0,\#).$$

2. No cell contains two symbols:

$$\psi_2 = \bigwedge_{i\in[N]} \bigwedge_{a\neq a'\in\Gamma} \left(\overline{x(i,a)} \vee \overline{x(i,a')}\right) \bigwedge_{j\in[N_c]} \left(\overline{x_c(j,0)} \vee \overline{x_c(j,1)}\right).$$

3. The head is in some position, the clock head is in some position, and the machine is in some state:

$$\psi_3 = \left(\vee_{i\in[N]} \;\; y(i)\right) \bigwedge \left(\vee_{j\in[N_c]} \;\; y_c(j)\right) \bigwedge \left(\vee_{q\in Q'} \;\; z(q)\right).$$

4. The main tape head is in a unique position, the clock head is in a unique position, and the machine is in a unique state:

$$\psi_4 = \bigwedge_{i\neq i'\in[N]} \left(\overline{y(i)} \vee \overline{y(i')}\right) \bigwedge_{j\neq j'\in[N_c]} \left(\overline{y_c(j)} \vee \overline{y_c(j')}\right) \bigwedge_{q\neq q'\in Q'} \left(\overline{z(q)} \vee \overline{z(q')}\right).$$

Solutions of $\psi = \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4$ are in 1-1 correspondence with configurations of M'. Furthermore, the assignments corresponding to any two distinct configurations differ in at least two variables (hence the space of solutions is totally disconnected).

Next, to connect the solution space along valid transitions of M', we relax conditions 2 and 4 by introducing new transition variables, which allow the head to have two states or a cell to have two symbols at the same time. This allows us to go from one configuration to the next.

Consider a transition $\delta'(q,a) = (q', b, R)$, which operates on the first tape, for example. Fix the position of the head of the first tape to be i. The variables that are

changed by the transition are $x(i,a)$, $y(i)$, $z(q)$, $x(i,b)$, $y(i+1)$, $z(q')$ (some of these may be the same variable, such as if $a = b$, but at least two are guaranteed to be different since the head has to move left or right). Before the transition the first three are set to 1, the second three are set to 0, and after the transition they are all flipped. Corresponding to this transition (which is specified by i, q, and a), we introduce a transition variable $t(i,q,a)$. We now relax conditions 2 and 4 as follows:

- Replace $\big(\overline{x(i,a)} \vee \overline{x(i,b)}\big)$ by $\big(\overline{x(i,a)} \vee \overline{x(i,b)} \vee t(i,q,a)\big)$.
- Replace $\big(\overline{y(i)} \vee \overline{y(i+1)}\big)$ by $\big(\overline{y(i)} \vee \overline{y(i+1)} \vee t(i,q,a)\big)$.
- Replace $\big(\overline{z(q)} \vee \overline{z(q')}\big)$ by $\big(\overline{z(q)} \vee \overline{z(q')} \vee t(i,q,a)\big)$.

This is done for every value of q, a, and i (and also for transitions acting on the clock tape). We add the transition variables to the corresponding clauses so that, for example, the clause $\big(\overline{x(i,a)} \vee \overline{x(i,b)}\big)$ could potentially become very long, such as

$$\Big(\overline{x(i,a)} \vee \overline{x(i,b)} \vee t(i,q_1,a) \vee t(i,q_2,a) \vee \cdots\Big).$$

However, the total number of transition variables is only polynomial in n. We also add a constraint for every pair of transition variables $t(i,q,a)$, $t(i',q',a')$, saying they cannot be 1 simultaneously: $(\overline{t(i,q,a)} \vee \overline{t(i',q',a')})$. This ensures that only one transition can be happening at any time. The effect of adding the transition variables to the clauses of ψ_2 and ψ_4 is that by setting $t(i,q,a)$ to 1, we can simultaneously set $x(i,a)$ and $x(i,b)$ to 1, and so on. This gives a path from the initial configuration to the final configuration as follows: Set $t(i,q,a) = 1$, set $x(i,b) = 1$, $y(i+1) = 1$, $z(q') = 1$, $x(i,a) = 0$, $y(i) = 0$, $z(q) = 0$, then set $t(i,q,a) = 0$. Thus consecutive configurations are now connected. To avoid connecting to other configurations, we also add an expression to ensure that these are the only assignments the 6 variables can take when $t(i,q,a) = 1$:

$$\psi_{i,q,a} = \overline{t(i,q,a)} \vee ((x(i,a), y(i), z(q), x(i,b)), y(i+1), z(q')) \in \{111000, 111100, 111110, 111111, 011111, 001111, 000111\}.$$

This expression can of course be written in conjunctive normal form.

Call the resulting CNF-formula $\varphi(\mathbf{x}, \mathbf{x_c}, \mathbf{y}, \mathbf{y_c}, \mathbf{z}, \mathbf{t})$. Note that $\varphi(\mathbf{x}, \mathbf{x_c}, \mathbf{y}, \mathbf{y_c}, \mathbf{z}, \mathbf{0}) = \psi(\mathbf{x}, \mathbf{x_c}, \mathbf{y}, \mathbf{y_c}, \mathbf{z})$, so a solution where all transition variables are 0 corresponds to a configuration of M'. To see that we have not introduced any shortcut between configurations that are not valid machine transitions, notice that in any solution of φ, at most a single transition variable can be 1. Therefore none of the transitional solutions belonging to different transitions can be adjacent. Furthermore, out of the solutions that have a transition variable set to 1, only the first and the last correspond to a valid configuration. Therefore none of the intermediate solutions (between the starting and the ending configuration of a transition) can be adjacent to a solution with all transition variables set to 0.

Finally, we define the assignment $\mathbf{s}$ to be the one corresponding to the initial configuration of M' with w on the tape, and $\mathbf{t}$ to be the assignment corresponding to the accepting configuration—where the state is q_{accept}, and the tape has only the # symbol at position 0. This completes the reduction.

The formula φ is a CNF-formula where clause size is unbounded. We use the same algorithm for producing structural expressions for k-clauses as in the proof of Lemma 3.5 to get a 3-CNF-formula. By Lemma 3.2 and Corollary 3.3, ST-CONN and CONN for $\mathcal{S}_3$ are PSPACE-complete. ∎

By Lemma 3.4 and Corollary 3.3, this completes the proof of the hardness part of the dichotomies for CONN and ST-CONN (Theorems 2.8 and 2.9).

Finally, we show that 3-CNF-formulas can have exponential diameter by inductively constructing a path of length at least $2^{\frac{n}{2}}$ on n variables and then identifying it with the solution space of a 3-CNF-formula with $O(n^2)$ clauses. By Lemma 3.4 and Corollary 3.3, this implies the hardness part of the diameter dichotomy (Theorem 2.10).

LEMMA 3.7. *For n even, there is a* 3-CNF*-formula φ_n with n variables and $O(n^2)$ clauses, such that $G(\varphi_n)$ is a path of length greater than $2^{\frac{n}{2}}$.*

Proof. The construction is in two steps: We first exhibit an induced subgraph G_n of the n-dimensional hypercube with large diameter. We then construct a 3-CNF-formula φ_n so that $G_n = G(\varphi_n)$.

The graph G_n is a path of length $2^{\frac{n}{2}}$. We construct it using induction. For $n = 2$, we take $V(G_2) = \{(0,0),(0,1),(1,1)\}$ which has diameter 2. Assume that we have constructed G_{n-2} with $2^{\frac{n-2}{2}}$ vertices, and with distinguished vertices $\mathbf{s_{n-2}}, \mathbf{t_{n-2}}$ such that the shortest path from $\mathbf{s_{n-2}}$ to $\mathbf{t_{n-2}}$ in G_{n-2} has length $2^{\frac{n-2}{2}}$. We now describe the set $V(G_n)$. For each vertex $\mathbf{v} \in V(G_{n-2})$, $V(G_n)$ contains two vertices $(\mathbf{v},0,0)$ and $(\mathbf{v},1,1)$. Note that the subgraph induced by these vertices alone consists of two disconnected copies of G_{n-2}. To connect these two components, we add the vertex $\mathbf{m} = (\mathbf{t},0,1)$ (which is connected to $(\mathbf{t},0,0)$ and $(\mathbf{t},1,1)$ in the induced subgraph). Note that the resulting graph G_n is connected, but any path from $(\mathbf{u},0,0)$ to $(\mathbf{v},1,1)$ must pass through $\mathbf{m}$. Further note that by induction the graph G_n is also a path. The vertices $\mathbf{s_n} = (\mathbf{s_{n-2}},0,0)$ and $\mathbf{t_n} = (\mathbf{s_{n-2}},1,1)$ are diametrically opposite ends of this path. The path length is at least $2 \cdot 2^{\frac{n-2}{2}} + 2 > 2^{\frac{n}{2}}$. Also $\mathbf{s_2} = (0,0)$, $\mathbf{s_n} = (\mathbf{s_{n-2}},0,0)$, $\mathbf{t_n} = (\mathbf{s_{n-2}},1,1)$ and hence $\mathbf{s_n} = (0,\ldots,0), \mathbf{t_n} = (0,\ldots,0,1,1)$.

We construct a sequence of 3-CNF-formulas $\varphi_n(x_1,\ldots,x_n)$ so that $G_n = G(\varphi_n)$. Let $\varphi_2(x_1,x_2) = \bar{x}_1 \vee x_2$. Assume we have $\varphi_{n-2}(x_1,\ldots,x_{n-2})$. We add two variables x_{n-1} and x_n and the clauses

$$\varphi_{n-2}(x_1,\ldots,x_{n-2}), \quad \bar{x}_{n-1} \wedge x_n,$$

$$x_{n-1} \vee \bar{x}_n \vee \bar{x}_i \qquad \text{for } \ i \le n-4, \tag{3.2}$$

$$x_{n-1} \vee \bar{x}_n \vee x_i \qquad \text{for } \ i = n-3, n-2. \tag{3.3}$$

Note that a clause in (3.2) is just the implication $(\bar{x}_{n-1} \wedge x_n) \to \bar{x}_i$. Thus clauses (3.2), (3.3) enforce the condition that $x_{n-1} = 0, x_n = 1$ implies that $(x_1,\ldots,x_{n-2}) = \mathbf{t_{n-2}} = (0,\ldots,0,1,1)$. ∎

4. The easy case of the dichotomy: Tight sets of relations.

4.1. Schaefer sets of relations. We begin by showing that all Schaefer sets of relations are tight. Schaefer relations are characterized by closure properties. We say that an r-ary relation R is closed under some k-ary operation $\alpha : \{0,1\}^k \to \{0,1\}$ if for every $\mathbf{a^1}, \mathbf{a^2}, \ldots, \mathbf{a^k} \in R$, the tuple $(\alpha(a_1^1, a_1^2, \ldots, a_1^k), \ldots, \alpha(a_r^1, \ldots, a_r^k))$ is in R. We denote this tuple by $\alpha(\mathbf{a^1}, \ldots, \mathbf{a^k})$.

We will use the following lemma about closure properties on several occasions.

LEMMA 4.1. *If a logical relation R is closed under an operation $\alpha : \{0,1\}^k \to \{0,1\}$ such that $\alpha(1,1,\ldots,1) = 1$ and $\alpha(0,0,\ldots,0) = 0$ (a.k.a. an idempotent operation), then every connected component of $G(R)$ is closed under α.*

Proof. Consider $\mathbf{a^1}, \ldots, \mathbf{a^k} \in R$, such that they all belong to the same connected component of $G(R)$. It suffices to prove that $\mathbf{a} = \alpha(\mathbf{a^1}, \ldots, \mathbf{a^k})$ is in the same connected component of $G(R)$. To that end we will first prove that for any $\mathbf{s}, \mathbf{t} \in R$ if there

is a path from $\mathbf{s}$ to $\mathbf{t}$ in $G(R)$, then there is a path from $\alpha(\mathbf{b^1}, \ldots, \mathbf{b^{i-1}}, \mathbf{s}, \mathbf{b^{i+1}}, \ldots, \mathbf{b^k})$ to $\alpha(\mathbf{b^1}, \ldots, \mathbf{b^{i-1}}, \mathbf{t}, \mathbf{b^{i+1}}, \ldots, \mathbf{b^k})$ for any $\mathbf{b^1}, \ldots, \mathbf{b^k} \in R$. This observation implies that there is a path from $\mathbf{a^1} = \alpha(\mathbf{a^1}, \mathbf{a^1}, \ldots, \mathbf{a^1})$ to $\alpha(\mathbf{a^1}, \mathbf{a^2}, \mathbf{a^1}, \ldots, \mathbf{a^1})$, from there to $\alpha(\mathbf{a^1}, \mathbf{a^2}, \mathbf{a^3}, \mathbf{a^1}, \ldots, \mathbf{a^1})$ and so on, to $\alpha(\mathbf{a^1}, \mathbf{a^2}, \ldots, \mathbf{a^k}) = \mathbf{a}$. Thus $\mathbf{a}$ is in the same connected component of $G(R)$ as $\mathbf{a^1}$.

Let the path from $\mathbf{s}$ to $\mathbf{t}$ be $\mathbf{s} = \mathbf{s^1} \to \mathbf{s^2} \to \ldots \mathbf{s^m} = \mathbf{t}$. For every $j \in \{1, 2, \ldots, m-1\}$, the tuples $\alpha(\mathbf{b^1}, \ldots, \mathbf{b^{i-1}}, \mathbf{s^j}, \mathbf{b^{i+1}}, \ldots, \mathbf{b^m})$ and $\alpha(\mathbf{b^1}, \ldots, \mathbf{b^{i-1}}, \mathbf{s^{j+1}}, \mathbf{b^{i+1}}, \ldots, \mathbf{b^m})$ differ in at most one position (the position in which $\mathbf{s^j}$ and $\mathbf{s^{j+1}}$ are different); therefore they belong to the same component of $G(R)$. Thus $\alpha(\mathbf{b^1}, \ldots, \mathbf{b^{i-1}}, \mathbf{s^1}, \mathbf{b^{i+1}}, \ldots, \mathbf{b^m})$ and $\alpha(\mathbf{b^1}, \ldots, \mathbf{b^{i-1}}, \mathbf{s^m}, \mathbf{b^{i+1}}, \ldots, \mathbf{b^m})$ belong to the same component. □

We are ready to prove that all Schaefer relations are tight.

LEMMA 4.2. *Let R be a logical relation.*

1. *If R is bijunctive, then R is componentwise bijunctive.*
2. *If R is Horn, then R is* OR*-free.*
3. *If R is dual Horn, then R is* NAND*-free.*
4. *If R is affine, then R is componentwise bijunctive,* OR*-free, and* NAND*-free.*

Proof. The case of bijunctive relations follows immediately from Lemma 4.1 and the fact that a relation is bijunctive if and only if it is closed under the ternary majority operation maj, which is idempotent.

The cases of Horn and dual Horn are symmetric. Suppose an r-ary Horn relation R is not OR-free. Then there exist $i, j \in \{1, \ldots, r\}$ and constants $t_1, \ldots, t_r \in \{0, 1\}$ such that the relation $R(t_1, \ldots, t_{i-1}, x, t_{i+1}, \ldots, t_{j-1}, y, t_{j+1}, \ldots, t_r)$ on variables x and y is equivalent to $x \vee y$, i.e.,

$$R(t_1, \ldots, t_{i-1}, x, t_{i+1}, \ldots, t_{j-1}, y, t_{j+1}, \ldots, t_r) = \{01, 11, 10\}.$$

Thus the tuples $\mathbf{t^{00}}, \mathbf{t^{01}}\mathbf{t^{10}}, \mathbf{t^{11}}$ defined by $(t_i^{ab}, t_j^{ab}) = (a, b)$ and $t_k^{ab} = t_k$ for every $k \notin \{i, j\}$, where $a, b, \in \{0, 1\}$ satisfy $\mathbf{t^{10}}, \mathbf{t^{11}}, \mathbf{t^{01}} \in R$ and $\mathbf{t^{00}} \notin R$. However, since every Horn relation is closed under $\wedge$, it follows that $\mathbf{t^{01}} \wedge \mathbf{t^{10}} = \mathbf{t^{00}}$ must be in R, which is a contradiction.

For the affine case, a small modification of the last step of the above argument shows that an affine relation also is OR-free; therefore, dually, it is also NAND-free. Namely, since a relation R is affine if and only if it is closed under ternary $\oplus$, it follows that $\mathbf{t^{01}} \oplus \mathbf{t^{11}} \oplus \mathbf{t^{10}} = \mathbf{t^{00}}$ must be in R.

Since the connected components of an affine relation are both OR-free and NAND-free, the subgraphs that they induce are hypercubes, which are also bijunctive relations. Therefore an affine relation is also componentwise bijunctive. □

These containments are proper. For instance, $R_{1/3} = \{100, 010, 001\}$ is componentwise bijunctive, but not bijunctive as $\text{maj}(100, 010, 001) = 000 \notin R_{1/3}$.

4.2. Structural properties of tight sets of relations. In this section, we explore some structural properties of the solution graphs of tight sets of relations. These properties provide simple algorithms for CONN($\mathcal{S}$) and ST-CONN($\mathcal{S}$) for tight sets $\mathcal{S}$, and also guarantee that for such sets, the diameter of $G(\varphi)$ of CNF($\mathcal{S}$)-formula φ is linear.

LEMMA 4.3. *Let $\mathcal{S}$ be a set of componentwise bijunctive relations and φ a* CNF($\mathcal{S}$)*-formula. If $\mathbf{a}$ and $\mathbf{b}$ are two solutions of φ that lie in the same component of $G(\varphi)$, then $d_\varphi(\mathbf{a}, \mathbf{b}) = |\mathbf{a} - \mathbf{b}|$, i.e., no distance expands.*

Proof. Consider first the special case in which every relation in $\mathcal{S}$ is bijunctive. In this case, φ is equivalent to a 2-CNF-formula and so the space of solutions of φ is closed under majority. We show that there is a path in $G(\varphi)$ from $\mathbf{a}$ to $\mathbf{b}$

such that along the path only the assignments on variables with indices from the set $D = \{i | a_i \neq b_i\}$ change. This implies that the shortest path is of length $|D|$ by induction on $|D|$. Consider any path $\mathbf{a} \to \mathbf{u^1} \to \cdots \to \mathbf{u^r} \to \mathbf{b}$ in $G(\varphi)$. We construct another path by replacing $\mathbf{u^i}$ by $\mathbf{v^i} = \text{maj}(\mathbf{a}, \mathbf{u^i}, \mathbf{b})$ for $i = 1, \ldots, r$ and removing repetitions. This is a path because for any i $\mathbf{v^i}$ and $\mathbf{v^{i+1}}$ differ in at most one variable. Furthermore, $\mathbf{v^i}$ agrees with $\mathbf{a}$ and $\mathbf{b}$ for every i for which $a_i = b_i$. Therefore, along this path only variables in D are flipped.

For the general case, we show that every component F of $G(\varphi)$ is the solution space of a 2-CNF-formula φ'. Let $R \in \mathcal{S}$ be a relation with two components, R_1, R_2, each of which are bijunctive. Consider a clause in φ of the form $R(x_1, \ldots, x_k)$. The projection of F onto $x_1, \ldots, x_k$ is itself connected and must satisfy R. Hence it lies within one of the two components R_1, R_2; assume it is R_1. We replace $R(x_1, \ldots, x_k)$ by $R_1(x_1, \ldots, x_k)$. Call this new formula φ_1. $G(\varphi_1)$ consists of all components of $G(\varphi)$ whose projection on $x_1, \ldots, x_k$ lies in R_1. We repeat this for every clause. Finally we are left with a formula φ' over a set of bijunctive relations. Hence φ' is bijunctive and $G(\varphi')$ is a component of $G(\varphi)$. So the claim follows from the bijunctive case. □

COROLLARY 4.4. *Let $\mathcal{S}$ be a set of componentwise bijunctive relations. Then*

1. *for every $\varphi \in \text{CNF}(\mathcal{S})$ with n variables, the diameter of each component of $G(\varphi)$ is bounded by n;*
2. *st-Conn$(\mathcal{S})$ is in* P*;*
3. *Conn$(\mathcal{S})$ is in* coNP*.*

Proof. The bound on diameter is an immediate consequence of Lemma 4.3.

The following algorithm solves st-Conn$(\mathcal{S})$ given vertices $\mathbf{s}, \mathbf{t} \in G(\varphi)$. Start with $\mathbf{u} = \mathbf{s}$. At each step, find a variable x_i so that $u_i \neq t_i$ and such that if we flip x_i, the assignment would still be satisfying. Repeat until $\mathbf{t}$ is reached. If at any stage no such variable exists, then declare that $\mathbf{s}$ and $\mathbf{t}$ are not connected. If the $\mathbf{s}$ and $\mathbf{t}$ are disconnected, the algorithm is bound to fail. So assume that they are connected. Correctness is proved by induction on $d = |\mathbf{s} - \mathbf{t}|$. It is clear that the algorithm works when $d = 1$. Assume that the algorithm works for $d - 1$. If s and t are connected and are distance d apart, Lemma 4.3 implies there is a path of length d between them in $G(\varphi)$. In particular, the algorithm will find a variable x_i to flip. The resulting assignment is at distance $d - 1$ from $\mathbf{t}$, so now we proceed by induction.

Next we prove that Conn$(\mathcal{S}) \in$ coNP. A short certificate that the graph is not connected is a pair of assignments $\mathbf{s}$ and $\mathbf{t}$ which are solutions from different components. To verify that they are disconnected it suffices to run the algorithm for st-Conn. □

We consider sets of OR-free relations. Define the *coordinatewise partial order* $\leq$ on Boolean vectors as follows: $\mathbf{a} \leq \mathbf{b}$ if $a_i \leq b_i$ for each i. A monotone path between $\mathbf{a}$ and $\mathbf{b}$ is a path in $G(\varphi)$, $\mathbf{a} \to \mathbf{u^1} \to \cdots \to \mathbf{u^r} \to \mathbf{b}$ such that $\mathbf{a} \leq \mathbf{u^1} \leq \cdots \leq \mathbf{u^r} \leq \mathbf{b}$.

LEMMA 4.5. *Let $\mathcal{S}$ be a set of* OR*-free relations and φ a* CNF$(\mathcal{S})$*-formula. Every component of $G(\varphi)$ contains a minimum solution with respect to the coordinatewise order; moreover, every solution is connected to the minimum solution in the same component via a monotone path.*

Proof. We call a satisfying assignment locally minimal if it has no neighboring satisfying assignments that are smaller than it. We will show that there is exactly one such assignment in each component of $G(\varphi)$.

Suppose there are two distinct locally minimal assignments $\mathbf{u}$ and $\mathbf{u'}$ in some component of $G(\varphi)$. Consider the path between them where the maximum Hamming weight of assignments on the path is minimized. If there are many such paths, pick

one where the smallest number of assignments have the maximum Hamming weight. Denote this path by $\mathbf{u} = \mathbf{u^1} \to \mathbf{u^2} \to \cdots \to \mathbf{u^r} = \mathbf{u'}$. Let $\mathbf{u^i}$ be an assignment of largest Hamming weight in the path. Then $\mathbf{u^i} \neq \mathbf{u}$ and $\mathbf{u^i} \neq \mathbf{u'}$, since $\mathbf{u}$ and $\mathbf{u'}$ are locally minimal. The assignments $\mathbf{u^{i-1}}$ and $\mathbf{u^{i+1}}$ differ in exactly 2 variables, say, in x_1 and x_2. So $\{u_1^{i-1}u_2^{i-1},\ u_1^i u_2^i,\ u_1^{i+1}u_2^{i+1}\} = \{01, 11, 10\}$. Let $\hat{\mathbf{u}}$ be such that $\hat{u}_1 = \hat{u}_2 = 0$, and $\hat{u}_i = u_i$ for $i > 2$. If $\hat{\mathbf{u}}$ is a solution, then the path $\mathbf{u^1} \to \mathbf{u^2} \to \cdots \to \mathbf{u^{i-1}} \to \hat{\mathbf{u}} \to \mathbf{u^{i+1}} \to \cdots \to \mathbf{u^r}$ contradicts the way we chose the original path. Therefore, $\hat{\mathbf{u}}$ is not a solution. This means that there is a clause that is violated by it but is satisfied by $\mathbf{u^{i-1}}$, $\mathbf{u^i}$, and $\mathbf{u^{i+1}}$. So the relation corresponding to that clause is not OR-free, which is a contradiction.

The unique locally minimal solution in a component is its minimum solution, because starting from any other assignment in the component, it is possible to keep moving to neighbors that are smaller, and the only time it becomes impossible to find such a neighbor is when the locally minimal solution is reached. Therefore, there is a monotone path from any satisfying assignment to the minimum in that component. □

COROLLARY 4.6. *Let $\mathcal{S}$ be a set of* OR*-free relations. Then*

1. *for every $\varphi \in \mathrm{CNF}(\mathcal{S})$ with n variables, the diameter of each component of $G(\varphi)$ is bounded by $2n$;*
2. ST-CONN$(\mathcal{S})$ *is in* P*;*
3. CONN$(\mathcal{S})$ *is in* coNP*.*

Proof. Given solutions $\mathbf{s}$ and $\mathbf{t}$ in the same component of $G(\varphi)$, there is a monotone path from each to the minimal solution $\mathbf{u}$ in the component. This gives a path from $\mathbf{s}$ to $\mathbf{t}$ of length at most $2n$. To check if $\mathbf{s}$ and $\mathbf{t}$ are connected, we just check that the minimal assignments reached from $\mathbf{s}$ and $\mathbf{t}$ are the same. □

Sets of NAND-free relations are handled dually to OR-free relations. In this case there is a maximum solution in every connected component of $G(\phi)$ and every solution is connected to it via a monotone path. Finally, putting everything together, we complete the proofs of all our dichotomy theorems.

COROLLARY 4.7. *Let $\mathcal{S}$ be a tight set of relations. Then*

1. *for every $\varphi \in \mathrm{CNF}(\mathcal{S})$ with n variables, the diameter of each component of $G(\varphi)$ is bounded by $2n$;*
2. ST-CONN$(\mathcal{S})$ *is in* P*;*
3. CONN$(\mathcal{S})$ *is in* coNP*.*

4.3. The complexity of CONN for tight sets of relations. We pinpoint the complexity of CONN$(\mathcal{S})$ for the tight cases which are not Schaefer, using a result of Juban [19].

LEMMA 4.8. *For $\mathcal{S}$ tight, but not Schaefer,* CONN$(\mathcal{S})$ *is* coNP*-complete.*

Proof. The problem ANOTHER-SAT$(\mathcal{S})$ is as follows: Given a formula φ in CNF$(\mathcal{S})$ and a solution $\mathbf{s}$, does there exist a solution $\mathbf{t} \neq \mathbf{s}$? Juban [19, Theorem 2] shows that if $\mathcal{S}$ is not Schaefer, then ANOTHER-SAT is NP-complete. He also shows [19, Corollary 1] that if $\mathcal{S}$ is not Schaefer, then the relation $x \neq y$ is expressible from $\mathcal{S}$ through substitutions.

Since $\mathcal{S}$ is not Schaefer, ANOTHER-SAT$(\mathcal{S})$ is NP-complete. Let $\varphi, \mathbf{s}$ be an instance of ANOTHER-SAT on variables $x_1, \ldots, x_n$. We define a CNF$(\mathcal{S})$-formula ψ on the variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ as

$$\psi(x_1, \ldots, x_n, y_1, \ldots, y_n) = \varphi(x_1, \ldots, x_n) \wedge_i (x_i \neq y_i).$$

It is easy to see that $G(\psi)$ is connected if and only if $\mathbf{s}$ is the unique solution to φ. □

We are left with the task of determining the complexity of CONN($\mathcal{S}$) for the case when $\mathcal{S}$ is a Schaefer set of relations. In Lemmas 4.9 and 4.10 we show that CONN($\mathcal{S}$) is in P if $\mathcal{S}$ is affine or bijunctive. This leaves the case of Horn and dual Horn, which we discuss at the end of this section.

LEMMA 4.9. *If $\mathcal{S}$ is a bijunctive set of relations, then there is a polynomial-time algorithm for* CONN($\mathcal{S}$).

Proof. Consider a formula $\phi(x_1, \ldots, x_n)$ in CNF($\mathcal{S}$). Since $\mathcal{S}$ is a bijunctive set of relations ϕ can be written as a 2-CNF-formula. Since satisfiability of 2-CNF-formulas is decidable in polynomial time, it is easy to decide for a given variable x_i whether there exist solutions in which it takes a particular value in $\{0, 1\}$. The variables which can take only one value are assigned that value. Without loss of generality we can assume that the resulting 2-CNF formula is $\psi(x_1, \ldots, x_m)$.

Consider the graph of implications of ψ defined in the following way: The vertices are the literals $x_1, \ldots, x_m, \bar{x}_1, \ldots, \bar{x}_m$. There is a directed edge from literal l_1 to literal l_2 if and only if ψ contains a clause containing l_2 and the negation of l_1, which we denote by $\bar{l}_1$ (if l_1 is a negated variable $\bar{x}$, then $\bar{l}_1$ denotes x). The directed edge represents the fact that in a satisfying assignment if the literal l_1 is assigned true, then the literal l_2 is also assigned true. We will show that $G(\psi)$ is disconnected if and only if the graph of implications contains a directed cycle. This property can be checked in polynomial time.

Suppose the graph of implications contains a directed cycle of literals $l_1 \to l_2 \to l_3 \to \cdots \to l_k \to l_1$. By the construction, the graph also contains a directed cycle on the negations of these literals, but in the opposite direction: $\bar{l}_k \to \bar{l}_{k-1} \to \cdots \to \bar{l}_2 \to \bar{l}_1 \to \bar{l}_k$. There is a satisfying assignment $\mathbf{s}$ in which l_1 is assigned 1, and also a satisfying assignment $\mathbf{t}$ in which $\bar{l}_1$ is assigned 1. By the implications, in $\mathbf{s}$ the literals $l_1, l_2, \ldots, l_k$ are assigned 1, and in $\mathbf{t}$ $\bar{l}_1, \bar{l}_2, \ldots, \bar{l}_k$ are assigned 1. Suppose there is a path from $\mathbf{s}$ to $\mathbf{t}$. Then let l_i be the first literal in the cycle whose value changes along the path from $\mathbf{s}$ to $\mathbf{t}$. Then there is a satisfying assignment in which l_i is assigned 0, whereas all other literals on the cycle are assigned 1. On the other hand, this cannot be a satisfying assignment because the edge (l_{i-1}, l_i) implies that there is a clause containing only l_i and the negation of l_{i-1}, and this clause is violated by the assignment. This is a contradiction, and therefore there can be no path from $\mathbf{s}$ to $\mathbf{t}$.

Next, suppose the graph of implications contains no directed cycle, and $G(\psi)$ is disconnected. Let $\mathbf{s}$ and $\mathbf{t}$ be satisfying assignments from different connected components of $G(\psi)$ that are at minimum Hamming distance. Let U be the set of variables on which $\mathbf{s}$ and $\mathbf{t}$ differ. There are two literals corresponding to each variable, and let $U^{\mathbf{s}}$ and $U^{\mathbf{t}}$ denote, respectively, the literals that are true in $\mathbf{s}$ and in $\mathbf{t}$. Since the directed graph induced by $U^{\mathbf{s}}$ in the implications graph contains no directed cycle, there exists a literal $l \in U^{\mathbf{s}}$ without an incoming edge from another literal in $U^{\mathbf{s}}$. Otherwise, by following the incoming edges we would find a cycle in the graph induced by $U^{\mathbf{s}}$. In addition, for every literal $l' \notin U^{\mathbf{s}}$ that is assigned true by $\mathbf{s}$, there is no edge from l' to l because that would contradict the fact that $\mathbf{t}$ is also a satisfying assignment (by the definition of U, l' is assigned true by $\mathbf{t}$). Therefore, with respect to $\mathbf{s}$ the literal l does not appear in any clause in which it is implied, i.e., in which it is the only satisfying literal. Thus, the value of the corresponding variable can be flipped and the resulting assignment is still satisfying. This assignment is in the same component as $\mathbf{s}$ but it is closer to $\mathbf{t}$, which contradicts our choice of $\mathbf{s}$ and $\mathbf{t}$. □

LEMMA 4.10. *If $\mathcal{S}$ is an affine set of relations, then there is a polynomial-time algorithm for* CONN($\mathcal{S}$).

Proof. An affine formula can be described as the set of solutions of a linear system of equations. For any solution, if only a variable that appears in at least one of the equations is flipped, the resulting assignment is not a solution. Therefore it suffices to check whether the system has more than one solution (after variables that don't appear in any equation are removed), which is easily done by checking the rank of the matrix obtained from the Gaussian elimination algorithm. □

We are left with characterizing the complexity of CONN for sets of Horn relations and for sets of dual Horn relations. In the conference version [16] of the present paper, we had conjectured that if $\mathcal{S}$ is Horn or dual Horn, then CONN($\mathcal{S}$) is in P, but this was disproved by Makino, Tamaki, and Yamamoto [24]. They showed that CONN($\{R_2\}$) is coNP-complete, where $R_2 = \{0,1\}^3 \backslash \{110\}$; hence there exist Horn (and by symmetry also dual Horn) sets of relations for which CONN is coNP-complete. Their proof is via a reduction from POSITIVE NOT-ALL-EQUAL 3-SAT, which as seen earlier is SAT($\{R_{\text{NAE}}\}$), where $R_{\text{NAE}} = \{0,1\}^3 \setminus \{000, 111\}$. This problem is also known as 3-hypergraph 2-colorability,

The relation R_2 is a 3-clause with one positive literal. We will describe a natural set of Horn relations first introduced in [14], which cannot be used to express R_2. We show that for this set there is a polynomial-time algorithm for CONN.

DEFINITION 4.11. *A logical relation R is* implicative hitting set-bounded− *(*IHSB−*) if it is the set of solutions of a Horn formula in which all clauses of size greater than* 2 *have only negative literals. Similarly, R is* implicative hitting set-bounded+ *(*IHSB+*) if it is the set of solutions of a dual Horn formula in which all clauses of size greater than* 2 *have only positive literals.*

These types of logical relations can be characterized by closure properties. A relation R is IHSB− if and only if it is closed under $\mathbf{a} \wedge (\mathbf{b} \vee \mathbf{c})$; in other words if $\mathbf{a}, \mathbf{b}, \mathbf{c} \in R$, where R is of arity r, then $\mathbf{a} \wedge (\mathbf{b} \vee \mathbf{c}) = (a_1 \wedge (b_1 \vee c_1), a_2 \wedge (b_2 \vee c_2), \ldots, a_r \wedge (b_r \vee c_r)) \in R$. A relation R is IHSB+ if and only if it is closed under $\mathbf{a} \vee (\mathbf{b} \wedge \mathbf{c})$. While the definition may at first look unnatural, it comes from Post's classification of Boolean functions (see [5]). One of the consequences of this classification is that IHSB− relations cannot express all Horn relations, and in particular R_2, even in the sense of Schaefer's expressibility. For the purposes of structural expressibility we can define an even larger class of relations which cannot structurally express R_2 (unless P = coNP).

DEFINITION 4.12. *A logical relation R is* componentwise IHSB− (IHSB+) *if every connected component of $G(R)$ is IHSB− (IHSB+).*

By Lemma 4.1, every relation that is IHSB− (IHSB+) is also componentwise IHSB− (IHSB+). Of course, the class of componentwise IHSB− relations is much broader and in fact includes relations that are not even Horn, such as $R_{1/3}$. However, in the following lemma we consider only componentwise IHSB− (IHSB+) relations which are Horn (dual Horn). We will say that a set of relations $\mathcal{S}$ is componentwise IHSB− (IHSB+) if every relation in $\mathcal{S}$ is componentwise IHSB− (IHSB+).

LEMMA 4.13. *If $\mathcal{S}$ is a set of relations that are Horn (dual Horn) and componentwise IHSB− (IHSB+), then there is a polynomial-time algorithm for* CONN($\mathcal{S}$).

Proof. First we consider the case in which every relation in $\mathcal{S}$ is IHSB−. The formula can be written as a conjunction of Horn clauses, such that clauses of length greater than 2 have only negative literals. Let all unit clauses be assigned and propagated—their variables take the same value in all satisfying assignments. The

resulting formula is also IHSB− and has two kinds of clauses: 2-clauses with one positive and one negative literal, and clauses of size 2 or more with only negative literals. The assignment of zero to all variables is satisfying. There is more than one connected component if and only if there is another assignment that is locally minimal by Lemma 4.5. A locally minimal satisfying assignment is such that if any of the variables assigned 1 is changed to 0 the resulting assignment is not satisfying. Thus all variables assigned 1 appear in at least one 2-clause with one positive and one negative literal for which both variables are assigned 1. We say that such an assignment certifies the disconnectivity.

To describe the algorithm, we first define the following implication graph G. The vertices are the set of variables. There is a directed edge (x_i, x_j) if and only if $(x_j \vee \bar{x}_i)$ is a clause in the IHSB− representation. Let $S_1, \ldots, S_m$ be the sets of variables in clauses with only negative literals. For every variable x_i let T_i denote the set of variables reachable from x_i in the directed graph. If $x_i \in T_i$, then x_i lies in a directed cycle. Note that if x_i is set to 1, then every variable in T_i must also be set to 1. The algorithm rejects if and only if there exists a variable x_i such that $x_i \in T_i$ and T_i does not contain S_j for any $j \in \{1, \ldots, m\}$. We show that this happens if and only if the solution graph is disconnected. Note that the algorithm runs in polynomial time.

Assume that the graph of solutions is disconnected and consider the satisfying assignment $\mathbf{s}$ that certifies disconnectivity. Let U be the set of variables x_i such that $s_i = 1$. Since every variable in U appears in at least one 2-clause for which both variables are from U, the directed graph induced by U is such that every vertex has an incoming edge. By starting at any vertex in U and following the incoming edge backwards until we repeat some vertex, we find a cycle in the subgraph induced by U. For any variable x_i in such a cycle it holds that $x_i \in T_i$. Further $T_i \subseteq U$, since setting x_i to 1 forces all variables in T_i to be 1. Also T_i cannot contain S_j for any j, else the corresponding clause would not be satisfied by $\mathbf{s}$. Thus the algorithm rejects whenever the solution graph is disconnected.

Conversely, if the algorithm rejects, there exists a variable x_i such that $x_i \in T_i$ and T_i does not contain S_j for any $j \in \{1, \ldots, m\}$. Consider the assignment in which all variables from T_i are assigned 1, and the rest are assigned 0. We will show that this assignment is satisfying and it is a certificate for disconnectivity. Clauses which contain only negated variables are satisfied since $S_j \not\subset T_i$ for all j. Now consider a clause of the form $(x_j \vee \bar{x}_k)$ and note that there is a directed edge (x_k, x_j). If $x_k = 0$, this is satisfied. If $x_k = 1$, then $x_k \in T_i$, and hence $x_j \in T_i$ because of the edge (x_k, x_j). But then x_j is set to 1, so the clause is satisfied. To show that this solution is minimal, consider trying to set $x_k \in T_i$ to 0. There is an incoming edge (x_j, x_k) for some $x_j \in T_i$, and hence a clause $(x_k \vee \bar{x}_j)$, which will become unsatisfied if we set $x_k = 0$. Thus we have a certificate for the space being disconnected.

Next, consider a formula $\phi(x_1, \ldots, x_n)$ in CNF($\mathcal{S}$). We reduce the connectivity question to one for a formula with IHSB− relations. Since satisfiability of Horn formulas is decidable in polynomial time and every connected component of a Horn relation is a Horn relation by Lemma 4.1, it is easy to decide for a given clause and a given connected component of its corresponding relation (the relation obtained after identifying repeated variables) whether there exists a solution for which the variables in this clause are assigned a value in the specified connected component. If there exists a clause for which there is more than one connected component for which solutions exist, then the space of solutions is disconnected. This follows from the fact that the projection of $G(\phi)$ onto the hypercube corresponding to the variables appearing in

this clause is disconnected. Therefore we can assume that the relation corresponding to every clause has a single connected component. Since that component is IHSB− the relation itself is IHSB−. □

It is still open whether CONN is coNP-complete for every remaining Horn set of relations, i.e., every set of Horn relations that contains at least one relation that is not componentwise IHSB−. Following the same line of reasoning as in the proof of our structural expressibility theorem, we are able to show that one of the paths of length 4 defined in section 3.2, namely, $M(\bar{x}_1, \bar{x}_2, x_3)$, can be expressed structurally from every such set of relations. Thus the trichotomy would be established if one shows that CONN($\{M(\bar{x}_1, \bar{x}_2, x_3)\}$) is coNP-hard.

5. Discussion and open problems. In section 2, we conjectured a trichotomy for CONN($\mathcal{S}$). In view of the results established here, what remains is to pinpoint the complexity of CONN($\mathcal{S}$) when $\mathcal{S}$ is Horn but not componentwise IHSB−, and when $\mathcal{S}$ is dual Horn but not componentwise IHSB+. We conjecture that for those cases CONN($\mathcal{S}$) is coNP-complete.

We mentioned that one could also consider CNF($\mathcal{S}$)-formulas without constants, and the extension of our results to this setting is still an open problem. A more interesting and challenging direction is the extension of our results to larger domains.

Finally, our techniques may shed light on other connectivity-related problems, such as approximating the diameter and counting the number of components, or proving hardness of other configuration connectivity problems. An example of the latter appears in recent work of Ito et al. [18].

REFERENCES

[1] D. ACHLIOPTAS AND F. RICCI-TERSENGHI, *On the solution-space geometry of random constraint satisfaction problems*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006, pp. 130–139.

[2] D. ACHLIOPTAS, P. BEAME, AND M. MOLLOY, *Exponential bounds for DPLL below the satisfiability threshold*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 132–133.

[3] D. ACHLIOPTAS, A. NAOR, AND Y. PERES, *Rigorous location of phase transitions in hard optimization problems*, Nature, 435 (2005), pp. 759–764.

[4] G. BIROLI, R. MONASSON, AND M. WEIGT, *A variational description of the ground state structure in random satisfiability problems*, Eur. Phys. J. B, 14 (2000), pp. 551–568.

[5] E. BÖHLER, N. CREIGNOU, S. REITH, AND H. VOLLMER, *Playing with Boolean blocks, Part* II: *Constraint satisfaction problems*, ACM SIGACT-Newsletter, 35 (2004), pp. 22–35.

[6] P. S. BONSMA AND L. CERECEDA, *Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances*, in Mathematical Foundations of Computer Science, Springer, Berlin, 2007, pp. 738–749.

[7] G. BRIGHTWELL AND P. WINKLER, *Gibbs measures and dismantlable graphs*, J. Combin. Theory Ser. B, 78 (2000), pp. 141–166.

[8] A. BULATOV, *A dichotomy theorem for constraint satisfaction problems on a* 3*-element set*, J. ACM, 53 (2006), pp. 66–120.

[9] L. CERECEDA, J. VAN DEN HEUVEL, AND M. JOHNSON, *Finding paths between* 3*-colourings*, submitted (2007).

[10] N. CREIGNOU, *A dichotomy theorem for maximum generalized satisfiability problems*, J. Comput. System Sci., 51 (1995), pp. 511–522.

[11] N. CREIGNOU AND H. DAUDÉ, *Combinatorial sharpness criterion and phase transition classification for random CSPs*, Inform. and Comput., 190 (2004), pp. 220–238.

[12] N. CREIGNOU AND M. HERMANN, *Complexity of generalized satisfiability counting problems*, Inform. and Comput., 125 (1996), pp. 1–12.

[13] N. CREIGNOU AND B. ZANUTTINI, *A complete classification of the complexity of propositional abduction*, SIAM J. Comput., 36 (2006), pp. 207–229.

[14] N. CREIGNOU, S. KHANNA, AND M. SUDAN, *Complexity Classification of Boolean Constraint*

Satisfaction Problems, SIAM Monogr. Discrete Math. Appl. 7, SIAM, Philadelphia, 2001.

[15] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[16] P. Gopalan, P. G. Kolaitis, E. Maneva, and C. H. Papadimitriou, *The connectivity of boolean satisfiability: Computational and structural dichotomies*, in Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, 2006, pp. 346–357.

[17] R. Hearne and E. Demaine, *The nondeterministic constraint logic model of computation: Reductions and applications*, in Proceedings of the 29th International Colloquium on Automata, Languages and Programming, 2002, pp. 401–413.

[18] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno, *On the complexity of reconfiguration problems*, in Proceedings of the 19th Annual International Symposium on Algorithms and Computation (ISAAC 2008), 2008, pp. 28–39.

[19] L. Juban, *Dichotomy theorem for the generalized unique satisfiability problem*, in Proceedings of the 12th International Symposium Fundamentals of Computation Theory, 1999, pp. 327–337.

[20] D. Kavvadias and M. Sideri, *The inverse satisfiability problem*, SIAM J. Comput., 28 (1998), pp. 152–163.

[21] S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson, *The approximability of constraint satisfaction problems*, SIAM J. Comput., 30 (2001), pp. 1863–1920.

[22] L. Kirousis and P. Kolaitis, *The complexity of minimal satisfiability problems*, Inform. and Comput., 187 (2003), pp. 20–39.

[23] R. Ladner, *On the structure of polynomial time reducibility*, J. ACM, 22 (1975), pp. 155–171.

[24] K. Makino, S. Tamaki, and M. Yamamoto, *On the boolean connectivity problem for horn relations*, in Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT), 2007, pp. 187–200.

[25] E. Maneva, E. Mossel, and M. J. Wainwright, *A new look at survey propagation and its generalizations*, J. ACM, 54 (2007), pp. 2–41.

[26] M. Mézard and R. Zecchina, *Random k-satisfiability: From an analytic solution to an efficient algorithm*, Phys. Rev. E, 66 (2002), article 056126.

[27] M. Mézard, T. Mora, and R. Zecchina, *Clustering of solutions in the random satisfiability problem*, Phys. Rev. Lett., 94 (2005), article 197205.

[28] M. Mézard, G. Parisi, and R. Zecchina, *Analytic and algorithmic solution of random satisfiability problems*, Science, 297 (2002), pp. 812–815.

[29] M. Molloy, *Models for random constraint satisfaction problems*, SIAM J. Comput., 32 (2003), pp. 935–949.

[30] S. Reith and H. Vollmer, *Optimal satisfiability for propositional calculi and constraint satisfaction problems*, Inform. and Comput., 186 (2003), pp. 1–19.

[31] T. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 216–226.

[32] B. Selman, H. Kautz, and B. Cohen, *Local search strategies for satisfiability testing*, in Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, D. S. Johnson and M. A. Trick, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 26, AMS, Providence, RI, 1996, pp. 521–532.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2356–2381

THE UNDECIDABILITY OF THE INFINITE RIBBON PROBLEM: IMPLICATIONS FOR COMPUTING BY SELF-ASSEMBLY*

LEONARD ADLEMAN†, JARKKO KARI‡, LILA KARI§, DUSTIN REISHUS†, AND PETR SOSIK¶

Abstract. Self-assembly, the process by which objects autonomously come together to form complex structures, is omnipresent in the physical world. Recent experiments in self-assembly demonstrate its potential for the parallel creation of a large number of nanostructures, including possibly computers. A systematic study of self-assembly as a mathematical process has been initiated by L. Adleman and E. Winfree. The individual components are modeled as square tiles on the infinite two-dimensional plane. Each side of a tile is covered by a specific "glue," and two adjacent tiles will stick iff they have matching glues on their abutting edges. Tiles that stick to each other may form various two-dimensional "structures" such as squares and rectangles, or may cover the entire plane. In this paper we focus on a special type of structure, called a ribbon: a non-self-crossing rectilinear sequence of tiles on the plane, in which successive tiles are adjacent along an edge and abutting edges of consecutive tiles have matching glues. We prove that it is undecidable whether an arbitrary finite set of tiles with glues (infinite supply of each tile type available) can be used to assemble an infinite ribbon. While the problem can be proved undecidable using existing techniques if the ribbon is required to start with a given "seed" tile, our result settles the "unseeded" case, an open problem formerly known as the "unlimited infinite snake problem." The proof is based on a construction, due to R. Robinson, of a special set of tiles that allow only aperiodic tilings of the plane. This construction is used to create a special set of directed tiles (tiles with arrows painted on the top) with the "strong plane-filling property"—a variation of the "plane-filling property" previously defined by J. Kari. A construction of "sandwich" tiles is then used in conjunction with this special tile set, to reduce the well-known undecidable tiling problem to the problem of the existence of an infinite directed zipper (a special kind of ribbon). A "motif" construction is then introduced that allows one tile system to simulate another by using geometry to represent glues. Using motifs, the infinite directed zipper problem is reduced to the infinite ribbon problem, proving the latter undecidable. An immediate consequence of our result is the undecidability of the existence of arbitrarily large structures self-assembled using tiles from a given tile set.

Key words. DNA computing, molecular computing, self-assembly, tiling, undecidability

AMS subject classifications. 68Q80, 37B15, 03D35, 92B05

DOI. 10.1137/080723971

*Received by the editors May 12, 2008; accepted for publication (in revised form) December 12, 2008; published electronically March 20, 2009. This paper is based on "On the decidability of self-assembly of infinite ribbons," by L. Adleman, J. Kari, L. Kari, and D. Reishus, which appeared in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), 2002, pp. 530–537. © 2002 IEEE.

http://www.siam.org/journals/sicomp/38-6/72397.html

†Department of Computer Science, University of Southern California, 3710 McClintock Ave., RTH 501, Los Angeles, CA 90089-2905 (adleman@usc.edu, reishus@usc.edu). The first author's research was supported by NASA/JPL, NSF, ONR, and DARPA grants.

‡Department of Mathematics, University of Turku, Turku FI-20014, Finland (jkari@utu.fi). This author's research was supported by NSF grant CCR 97-33101 and Academy of Finland grant 211967.

§Department of Computer Science, University of Western Ontario, London, ON, N6A 5B7, Canada (lila@csd.uwo.ca). This author's research was supported by a Canada Research Chair Award and a discovery grant of the Natural Sciences and Engineering Research Council of Canada.

¶Institute of Computer Science, Silesian University in Opava, 746 01 Opava, Czech Republic, and Department of Artificial Intelligence, Polytechnical University of Madrid, 28660 Madrid, Spain (petr.sosik@fpf.slu.cz). This author's research was supported by grant 201/06/0567 of the Czech Science Foundation and the Ramón y Cajal Program of the Ministry of Science and Technology of Spain.

1. Introduction. Self-assembly, the process by which objects autonomously come together to form complex structures, is omnipresent in the physical world. Atoms bind to each other by chemical bonds to form molecules, molecules may form crystals or macromolecules, and cells interact to form biological organisms. Recently it has been suggested that complex self-assembly processes will ultimately be used in circuit fabrication, nanorobotics, DNA computation, and amorphous computing. Indeed, in electronics, engineering, medicine, material science, manufacturing, and other disciplines, there is a continuous drive toward miniaturization. Unfortunately, current "top-down" techniques such as lithography may not be capable of efficiently creating structures with features the size of molecules or atoms. Self-assembly provides a "bottom-up" approach by which such fine structures might be created.

Experimental research on self-assembly includes simulation of one-dimensional cellular automata by using macroscopic plastic tiles that assemble at an oil/water interface [38], studies of self-directed growth of hybrid organic molecules on a silicon substrate [31], self-assembly of regular lipid hollow icosahedra [14], self-assembly of patterns of lead on a copper surface as templates for fabricating nanostructures [35], formation of electrical networks by self-assembly of small plastic and metal objects [22], [49], self-assembly of molecular machines [20], the construction by self-assembly of a molecular-thickness transistor [43], and DNA nanomachines made by self-assembly (e.g., molecular switches [30], tweezers [53], [33], walkers [44], autonomous DNA motors [47], [9], [23]), as well as algorithmic self-assembled Sierpinski triangles [41], self-assembled cloneable DNA octahedra [45], stiff self-assembled tetrahedra for three-dimensional electronic circuits [21], and DNA origami [40].

Investigations into DNA computing [2] and amorphous computing [1] have pointed out a strong connection between self-assembly and computation. While [52], [29], [36], [50] have shown the potential of DNA self-assembly for computation, [51] has experimentally demonstrated the self-assembly of periodic two-dimensional arrays from DNA "tiles," in which "binding" is regulated by the "sticky" ends of the DNA tiles. Reference [32] has demonstrated the execution of logical operations (cumulative XOR) by self-assembly of DNA triple cross-over molecules and [10] has demonstrated the potential for programmable self-assembled computing devices.

A systematic study of self-assembly as a computational process has been initiated in [3]. The individual components are therein modeled as square tiles on the infinite two-dimensional plane. The tiles cannot be rotated. Each side of a tile is covered by a specific "glue," and two adjacent tiles will stick iff they have matching glues on their abutting edges. As such, tiles have been previously studied in the context of classic questions about tilings of the plane [48]. The tiling problem, for example (proved undecidable in [11], [37]), asks whether a given set of tiles (supply of each tile type unlimited) can be used to correctly tile the entire plane. However, the new requirements of experimental self-assembly of tiles into required nanoscale shapes pose new types of questions. What is the minimal number of tiles required for the self-assembly of a desired shape [5], [19], [13], [46], [25], [39]? What is the running time of such a self-assembly [5], [4]? Do the results still hold if one generalizes the model by making the "sticking" reversible, or by defining different bond strengths and requiring more than one bond for sticking to occur [42], [4], [6], [5]? Do reversible self-assemblies achieve equilibrium [6]? What is the optimal initial concentration of tile types that guarantees the fastest assembly of a required shape [5]? Can a self-assembled shape recover from the loss of arbitrary many tiles [12]? What are possible computational primitives for algorithmic self-assembly [8]?

One of the interesting problems of self-assembly is whether or not a given set of

tiles allows uncontrollable growth, that is, whether arbitrarily large structures can be produced. It turns out that this question is equivalent to asking whether or not, given a set of tiles, there exists an infinite ribbon that can be formed with tiles from this set. A *ribbon* is a non-self-crossing sequence of tiles on the plane, in which successive tiles are adjacent along an edge and abutting edges of consecutive tiles have matching glues. If a given "seed" tile is specified, the problem of existence of an infinite ribbon starting at that tile has been proved undecidable [16], [15], [17]. However, when no such seed is specified, existing proof techniques failed to produce an answer and the problem was declared open in [17]. The unseeded problem is also relevant given that, in some physical simulations of self-assembly [38], the growth mechanism was deemed incompatible with computations that use a chosen input. (More recently, in [18], successful experimental "seeded" self-assembly of fixed-length Sierpinski-patterned ribbons was reported. The assemblies, which use DNA tiles, start their growth from DNA "origami seeds"—special DNA tiles obtained by using the origami method [40].) Here we prove that the unseeded problem is also undecidable. This result settles an open problem known hitherto as the "unlimited infinite snake problem" [17]. Our results are placed within the framework of the "classical" tiling by self-assembly, in that our model does not take into account physical phenomena that may affect experimental DNA self-assembly such as concentrations of tiles, different glue strengths, and glue-matching errors.

The paper is organized as follows. Section 2 introduces the notions of tile, glue, tile system, sticking, ribbon, zipper, valid tiling of the plane, and directed tiles. In particular, a zipper is a ribbon with the additional requirement that even nonconsecutive tiles that touch must have matching glues at their abutting edges.

Section 3 starts with defining a directed tile system with the strong plane-filling property: in such a system, any infinite directed zipper is forced to follow a specific plane-filling self-similar path, and, moreover, an infinite directed zipper is always guaranteed to exist. The construction of a directed tile system with the above property uses a 3×3 block construction based on directed tiles defined in [27], [26], which, in turn, resemble tiles devised by Robinson in [37] to produce only aperiodic tilings of the plane. These tiles were augmented in [26], [27] with directions that forced any directed tiled path to form a self-similar plane-filling Hilbert curve that recursively fills arbitrarily large squares by filling each of their quadrants. The original construction used the condition that no glue mismatch be present in the 3×3 neighborhood of any tile on the path, to force the path to follow the desired curve. Here, the additional 3×3 block construction is needed to ensure that the weaker condition of no glue mismatch between any two tiles on the directed tiled path is enough to force its course.

Section 4 then proves the undecidability of the existence of an infinite directed zipper by reducing the tiling problem to it. The reduction is based on a construction of sandwich tiles, which have directed tiles from the directed tile system with the strong plane-filling property on top and undirected tiles from a given tile system on the bottom. The next result of the section proves the undecidability of the existence of an infinite ribbon by reducing the undecidable infinite directed zipper problem to it. The proof uses a "ribbon motif" construction that simulates each directed zipper tile by a ribbon of undirected tiles following its contours, and it uses geometry (bumps and dents) to simulate zipper-tile glues.

Sections 5 and 6 point to the implications of the undecidability of existence of infinite ribbons for the problem of self-assembly of arbitrarily large supertiles, and they summarize our results.

2. Notation and definitions. A tile is an oriented unit square. The north, east, south, and west edges of the tile are each labeled with a symbol called a *glue* from a finite alphabet X. Tiles can be placed on the plane but not rotated. The positions of the tiles on the plane are indexed by $\mathbb{Z}^2$, the set of pairs of integers.

Formally, a *tile* t is a quadruple $t = (t_N, t_E, t_S, t_W) \in X^4$, where X is a finite set. The components t_N, t_E, t_S, t_W will be called the glues on the north, east, south, and west edges of the tile, respectively. A *tile system* T is a finite subset of X^4. For all tiles $t = (t_N, t_E, t_S, t_W)$ and $t' = (t'_N, t'_E, t'_S, t'_W)$, t *sticks* on the north to t' iff $t_N = t'_S$. Sticking on the east, south, and west is defined similarly.

The directions $\mathcal{D} = \{N, E, S, W\}$ are functions from $\mathbb{Z}^2$ to $\mathbb{Z}^2$: $N(x, y) = (x, y+1)$, $E(x, y) = (x+1, y)$, $S(x, y) = (x, y-1)$, $W(x, y) = (x-1, y)$. We say that the positions (x, y) and (x', y') are *adjacent* iff $(x', y') \in \{N(x, y), E(x, y), S(x, y), W(x, y)\}$. In addition, (x, y) *abuts* (x', y') on the north iff $(x', y') = N(x, y)$, and similarly for the other directions.

Given a tile system T, a *T-tiling of the plane* is a total function $f : \mathbb{Z}^2 \longrightarrow T$. The tiling f is *valid at position* (x, y) iff $f(x, y)$ sticks on the north to $f(N(x, y))$, $f(x, y)$ sticks on the east to $f(E(x, y))$, $f(x, y)$ sticks on the south to $f(S(x, y))$, and $f(x, y)$ sticks on the west to $f(W(x, y))$. That is to say, f is valid at (x, y) iff the tile at position (x, y) sticks on the appropriate sides to all the tiles that are at positions adjacent to it. The tiling is *valid* iff it is valid at every position $(x, y) \in \mathbb{Z}^2$. The well-known *tiling problem* posed by Wang asks the following: Given a tile system T, does there exist a valid T-tiling of the plane? The tiling problem was first proved undecidable by Berger [11], and the result was improved by Robinson [37]. One can generalize the notion of T-tiling of the entire plane by defining a *partial T-tiling* as a partial function $g : \mathbb{Z}^2 \longrightarrow T$. A partial tiling g is *valid on a region* $R \subseteq \mathrm{dom}(g)$ iff, for all $(x_1, y_1), (x_2, y_2) \in R$, if (x_1, y_1) abuts (x_2, y_2) on the north (east, south, west), then $g(x_1, y_1)$ sticks to $g(x_2, y_2)$ on the north (east, south, west). The partial tiling g is called *valid* iff it is valid on the entire $\mathrm{dom}(g)$.

A *path* is a function $P : I \longrightarrow \mathbb{Z}^2$, where I is a set of consecutive integers and for all $(i, i+1 \in I)$, $P(i)$ and $P(i+1)$ are adjacent. That is, a path is a sequence of adjacent positions on the plane. For all $i \in I$, we denote $P(i)$ as (x_i, y_i). We say that (x, y) is on P iff $(x, y) \in \mathrm{range}(P)$. For all tile systems T, a *T-tiled path* is a pair (P, r), where P is a path and r is a function $r : \mathrm{range}(P) \longrightarrow T$.

A T-tiled path (P, r) is a *T-ribbon* iff

(a) P is one-to-one, and

(b) for all $(i, i+1 \in \mathrm{dom}(P))$, if (x_{i+1}, y_{i+1}) abuts (x_i, y_i) on the north (south, east, west), then $r(x_{i+1}, y_{i+1})$ sticks to $r(x_i, y_i)$ on the north (south, east, west).

Informally, a tiled path is a ribbon iff it does not cross itself and the glue between tiles at consecutive positions along the path match.

A T-ribbon (P, r) is a *T-zipper* iff for all $(x, y), (x', y')$ on P, if (x', y') abuts (x, y) on the north (south, east, west), then $r(x', y')$ sticks to $r(x, y)$ on the north (south, east, west). Note that the notion of a zipper is more restrictive than the notion of a ribbon in that a zipper requires all of its tiles (consecutive or not) in adjacent positions to stick.

A *directed tile system* is a pair (T, d), where T is a tile system and $d : T \longrightarrow \{N, E, S, W\}$ is a function from the tile system to the direction functions. A directed tile system can be thought of as a tile system in which each tile has an arrow painted on it pointing north, east, south, or west.

A (T, d)-*directed tiled path* is a pair (P, r), where (P, r) is a T-tiled path and for all $(i, i+1 \in \mathrm{dom}(P))$ we have that $(x_{i+1}, y_{i+1}) = d(r(x_i, y_i))(x_i, y_i)$. A directed tiled path is a tiled path in which each tile points to its successor on the path. If (P, r) is a (T, d)-directed tiled path and (P, r) is a T-ribbon (zipper), then (P, r) is a (T, d)-directed ribbon (zipper).

A path, (directed) tiled path, (directed) ribbon, or (directed) zipper is called finite, semi-infinite, or infinite iff the domain of the associated path is finite, has a least or greatest element and is infinite, or is $\mathbb{Z}$, respectively.

To prove the undecidability of existence of infinite ribbons, we shall first prove the undecidability of existence of infinite directed zippers. Afterwards, a "ribbon motif" construction will be employed to prove the result for ribbons: the construction simulates a directed zipper by a ribbon motif of smaller tiles that goes around the contours of the zipper tiles, and it uses geometry to simulate zipper-tile glues.

In order to prove the undecidability of existence of infinite directed zippers, we shall use the undecidability of the tiling problem in conjunction with a sandwich construction that makes use of the existence of a directed tile system with the so-called *strong plane-filling property.*

DEFINITION 2.1. *A directed tile system* (T, d) *has the* strong plane-filling property *iff*

(i) *there exists an infinite* (T, d)*-directed zipper, and*
(ii) *for all infinite* (T, d)*-directed zippers* (P, r)*, for all natural numbers* n *there exists* (x, y) *such that* $(x + i, y + j)$ *is on* P *for* $i = 0, 1, 2, \ldots, n$ *and* $j = 0, 1, 2, \ldots, n$.

If a directed zipper (or ribbon or path) (P, r) satisfies the property in Definition 2.1(ii), we also say that it *covers arbitrarily large squares.*

Definition 2.1 states that in a directed tile system with the strong plane-filling property any infinite directed zipper is forced to be plane-filling (by filling arbitrarily large squares) and, moreover, that such an infinite directed zipper always exists.

3. A directed tile system with the strong plane-filling property. An essential element of our subsequent proofs will be the main result of this section, Theorem 3.1, namely the construction of a directed tile system that satisfies the strong plane-filling property of Definition 2.1. In fact, our directed tiles satisfy an even stronger requirement than (ii): any infinite directed tiled path is either plane-filling or it has a glue mismatch between two of its tiles. In particular, using our tiles, an infinite directed tiled path may not even form a loop without encountering a glue mismatch along the way. Consequently, every infinite directed tiled path that does not fill the plane must necessarily contain two neighboring tiles with a glue mismatch. The constructed tiles therefore satisfy a stronger version of Definition 2.1, thus satisfying a stronger property than the original plane-filling property defined in [26, 27].

In [27], any infinite directed tiled path was forced to be plane-filling unless there was a mismatch of glues inside the 3×3 neighborhood of some tile of the path. As described in subsections 3.2 and 3.3, directed tiles with this weaker plane-filling property were explicitly constructed by augmenting Robinson's aperiodic tiles [37] with directions. With these tiles, all directed tiled paths consisting of tiles without errors in their 3×3 neighborhood formed self-similar plane-filling curves of the kind used by Hilbert [24] to show that the unit square is a continuous image of the unit segment. The constructed tiles filled arbitrarily large squares recursively, by first filling the individual quadrants of a square. The process was guided by the direction associated

with each tile, which forced the construction to proceed along the self-similar Hilbert curve that traversed each quadrant of a square and linked the quadrants to each other.

In the present application we need the stronger variant of the plane-filling property introduced in Definition 2.1. In our case the infinite directed tiled path must be forced to be plane-filling even under the weak assumption that there is no glue mismatch between any two tiles belonging to it. It turns out that such a directed tile system is obtained by taking 3×3 blocks of the tiles in [27], as described in subsection 3.4. For this particular set of tiles, our 3×3 block construction ensures that the weak requirement of absence of glue mismatches between any two tiles belonging to the directed tiled path is sufficient to determine its uniqueness. Namely, we can prove that the constructed directed tile system has the property that, starting with any arbitrary tile, the only way to build an infinite error-free directed zipper is by forming a Hilbert curve that covers arbitrarily large squares.

3.1. Generalized tile systems. We begin by generalizing the notion of neighborhood from section 2, where the "neighbors" of a position were the positions at its north, east, south, and west.

DEFINITION 3.1. *A* neighborhood vector *is a* k*-tuple*

$$V = (\bar{x_1}, \bar{x_2}, \ldots, \bar{x_k}), \quad \bar{x_i} \in \mathbb{Z}^2, \quad 1 \leq i \leq k.$$

The neighbors *of a position* $\bar{x} \in \mathbb{Z}^2$ *are the positions* $\bar{x} + \bar{x_i}$ *for* $1 \leq i \leq k$.

The classical *von Neumann neighborhood* is defined by the vector

$$V_{vN} = [(0,0), (1,0), (0,1), (-1,0), (0,-1)].$$

In the von Neumann neighborhood, each position has five neighbors: itself and its four adjacent positions, as defined in section 2. In fact, all notions defined in section 2 are based on the von Neumann neighborhood.

Another particular neighborhood of interest, the *Moore neighborhood*, is defined as follows:

$$V_M = [(-1,1), (0,1), (1,1), (-1,0), (0,0), (1,0), (-1,-1), (0,-1), (1,-1)].$$

The Moore neighborhood of a position is a 3×3 block centered at that position, where the eight directions of the compass are used when we refer to the eight surrounding positions.

In contrast, in the generalized sense of Definition 3.1, the neighbors of a position need not be adjacent to it. Under this generalized definition, the neighborhood of a position may even have holes in it; i.e., adjacent positions may not belong to its neighborhood, while positions further away may. For example, $V = [(-2,2), (0,2), (2,2), (-1,1), (1,1), (-2,0), (0,0), (2,0), (-1,-1), (1,-1), (-2,-2), (0,-2), (2,-2)]$ defines a checkerboard type of neighborhood, where the neighbors of the center "white" position are all the "white" positions in the 5×5 square centered at it.

We shall now use the generalized notion of neighborhood to define a generalized tile system and its corresponding notion of valid tiling.

DEFINITION 3.2. *A* generalized tile system *is a triple* (T, V, R)*, where* T *is a finite set,* $V = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k)$*,* $\bar{x}_i \in \mathbb{Z}^2, 1 \leq i \leq k$*, is a neighborhood vector, and* $R \subseteq T^k$ *is a* k*-ary relation on* T.

As usual, a (partial) (T, V, R)-tiling (shortly T-tiling if the other elements are obvious from the context) is a (partial) function $\psi : \mathbb{Z}^2 \xrightarrow{o} T$.

DEFINITION 3.3. *A (T, V, R)-tiling ψ is* valid *at $\bar{x} \in \mathrm{dom}(\psi)$ iff there exists $r \in R$ such that, for all i, $1 \leq i \leq k$, either $(\bar{x} + \bar{x}_i) \in \mathrm{dom}(\psi)$ and $\psi(\bar{x} + \bar{x}_i) = r(i)$ or otherwise $\psi(\bar{x} + \bar{x}_i)$ is undefined.*

A (T, V, R)-tiling ψ is valid on a region $S \subseteq \mathrm{dom}(\psi)$ iff $\psi|_S$ is valid at every point $\bar{x} \in S$. We say that ψ is valid iff it is valid on $\mathrm{dom}(\psi)$. If ψ is valid on $\mathbb{Z}^2$, then ψ is called a valid (T, V, R)-tiling of the plane.

As before, we can define a *generalized directed tile system* (T, V, R, d), where (T, V, R) is a generalized tile system and $d : T \longrightarrow \{N, E, S, W\}$ is a function from the generalized directed tile system to the direction functions.

3.2. Microtiles. The starting point of our construction of a directed tile system (T_0, d_0) with the strong plane-filling property will be a generalized directed tile system $(T_\mu, V_\mu, R_\mu, d_\mu)$ constructed in [27]. The elements of T_μ are called *microtiles*. Instead of glues, they use labeled arrows to define their correspondence so that arrow heads match arrow tails on the edges of tiles. A T_μ-tiling will sometimes be called *microtiling*.

There are five different types of microtiles in T_μ (Figure 1): (a) single and (b) double crosses, (c) single, (d) double, and (e) mixed arms. The arms can be horizontal or vertical arms. There are also labeled diagonal arrows assigned to microtiles in [27]. They play a role in proofs of Lemmata 3.1 and 3.2. Since these proofs are not reproduced here, we omit the description of the diagonal arrows.

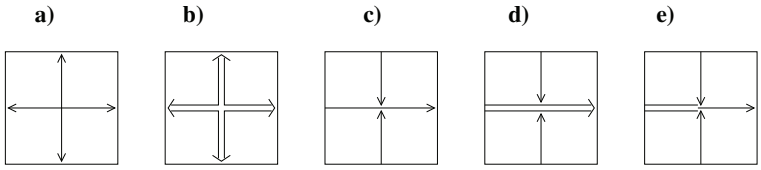


FIG. 1. *The five types of microtiles:* (a) *single cross,* (b) *double cross,* (c) *single arm,* (d) *double arm, and* (e) *mixed arm. Diagonal arrows are not shown. Reprinted from the Journal of Computer and System Sciences, Volume* 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

A cross, either single or double, has its arrow heads labeled from $\{SE, SW, NE, NW\}$, but all four arrow heads must have the same label. The principal arrow of an arm (single, double, or mixed) has a label from $\{SE, SW, NE, NW\}$, with the same label on both ends. The side arrows of mixed arms have labels as described in Figure 2. A single or double horizontal arm has its upper (lower) arrow labeled SX (NX, respectively) for $X \in \{E, W\}$. A single or double vertical arm has its left (right) arrow labeled YE (YW, respectively) for $Y \in \{N, S\}$. To conclude the description of microtiles, we note that more labels will be described in the next section.

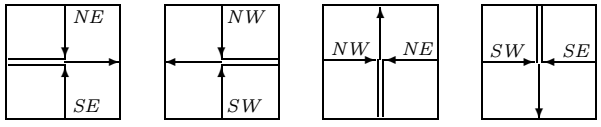


FIG. 2. *The possible labels of the two side arrows on a mixed arm. Reprinted from the Journal of Computer and System Sciences, Volume* 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

DEFINITION 3.4 (see [27]). *A* microtile system *is a generalized directed tile system $(T_\mu, V_\mu, R_\mu, d_\mu)$, where*

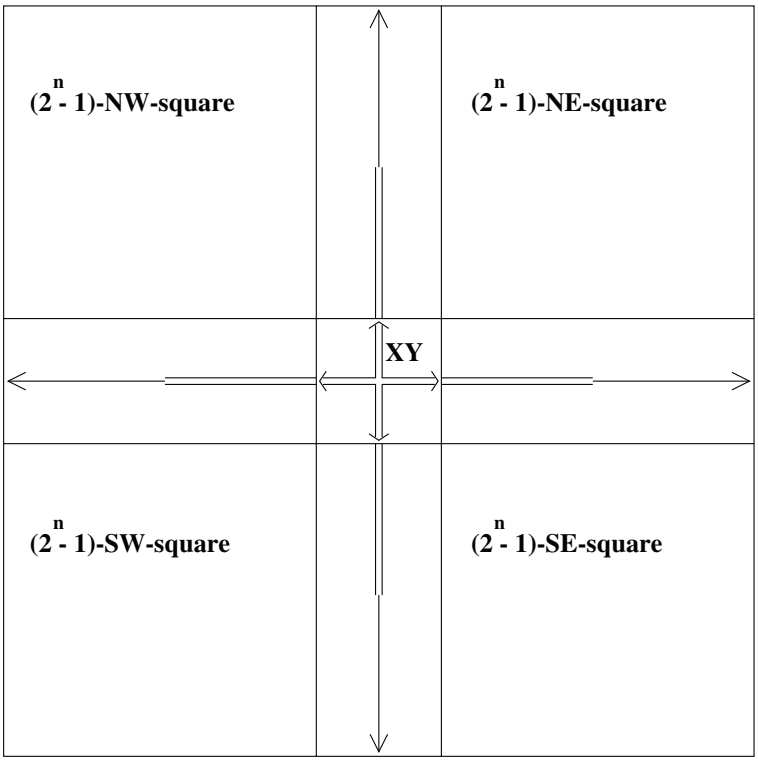


FIG. 3. *Constructing the* $(2^{n+1}-1)$*-*XY*-square. Reprinted from the Journal of Computer and System Sciences, Volume* 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

- T_μ *is the set of tiles with labeled arrows described above;*
- V_μ *is the Moore neighborhood;*
- R_μ *is the set of all* $v \in T_\mu^9$ *such that in the* 3×3 *tiling* $\{(V_\mu(i), v(i)) \mid 1 \le i \le 9\}$ *all arrow heads meet on the edges of tiles matching arrow tails with the same labels and vice versa.*

The labeling of microtiles described above allows that, for each $n \in \mathbb{N}$, four special squares called (2^n-1)-XY*-squares* can be defined recursively.

DEFINITION 3.5. *A* (2^n-1)-XY-square*,* $n \ge 1$*,* $XY \in \{NE, NW, SE, SW\}$*, is a microtiling* $f_{XY} : S_n \longrightarrow T_\mu$*, where* $S_n = \{(x+i, y+j) \mid 1 \le i, j \le 2^n - 1\}$ *for some* $(x, y) \in \mathbb{Z}^2$*. It is iteratively constructed in the following way.*

A single cross labeled XY *is a* 1-XY*-square. A* $(2^{n+1}-1)$-XY-square *consists of four* (2^n-1)-$X'Y'$*-squares,* $X'Y' \in \{NW, NE, SW, SE\}$*, ordered as in Figure* 3*. These squares are separated by a double cross labeled* XY *at the position* $(x+2^n, y+2^n)$*, called the* central cross *of the square, and rows of arms radiating from it. The arms are double near the central cross, but halfway the mixed arms make them single.*

The iterative construction is illustrated in Figure 3. By the construction of a microtile system, all (2^n-1)-XY-squares are valid microtilings. Figure 4 provides an example of a 7-XY-square. We may sometimes refer to a (2^n-1)-XY-square as (2^n-1)-square if XY is arbitrary. Proofs of the following results can be found in [27].

LEMMA 3.1 (see [27]). *Let* $f : S^{(1)} \longrightarrow T_\mu$*,* $g : S^{(2)} \longrightarrow T_\mu$ *be two* (2^n-1)*-squares with the same* n*. If* $f|_{S^{(1)} \cap S^{(2)}} = g|_{S^{(1)} \cap S^{(2)}}$ *and there exists* $(x, y) \in S^{(1)} \cap S^{(2)}$ *such*

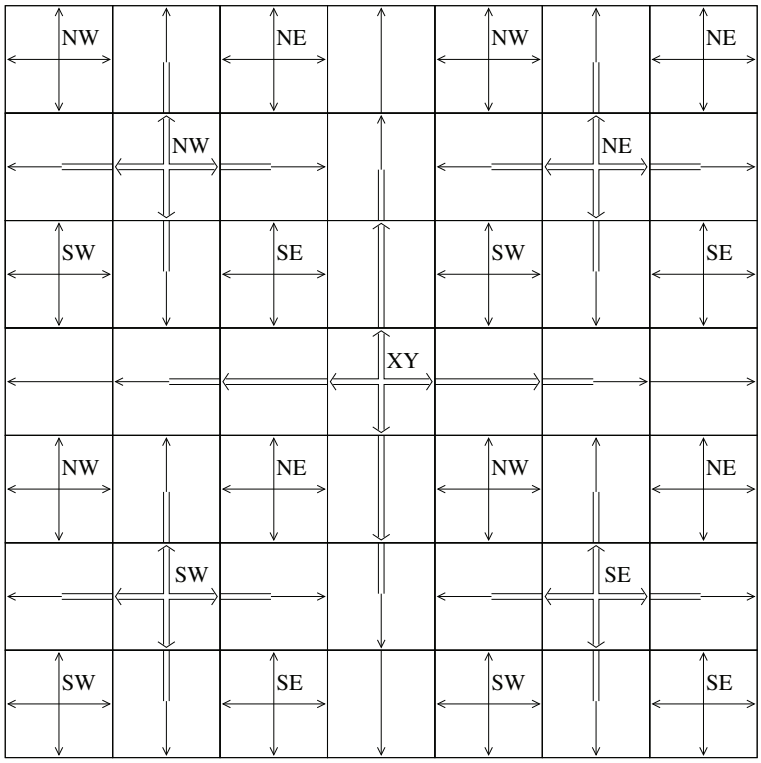


FIG. 4. *The 7-XY-square with the labels on the crosses. Only the principal arrows of the arms are drawn. Reprinted from the Journal of Computer and System Sciences, Volume* 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

that $f(x, y)$ is a single cross, then $S^{(1)} = S^{(2)}$.

In other words, if $S^{(1)}$ and $S^{(2)}$ have a nonempty common overlap containing a single cross, then they are identical.

LEMMA 3.2 (see [27]). *Let $f : \mathbb{Z}^2 \longrightarrow T_\mu$ be a microtiling of the plane and assume there exists a $(2^n - 1)$-XY-square $f_{XY} : S_n \longrightarrow T_\mu$ such that $f|_{S_n} = f_{XY}$ and, moreover, f is valid on S_n. Let $XY = SW$ (NW, NE, SE). Then*

(i) *the microtile just outside the NE (SE, SW, NW, respectively) corner of the square is a double cross;*

(ii) *the row of arms radiating Y-wise (X-wise) from this double cross has the length $2^n - 1$ and consists of $2^{n-1} - 1$ horizontal (vertical) double arms, followed by a horizontal (vertical) mixed arm, and then followed by $2^{n-1} - 1$ horizontal (vertical) single arms;*

(iii) *the microtile just outside the SE (NE, NW, SW, respectively) corner is a horizontal arm and the microtile just outside the NW (SW, SE, NE, respectively) corner is a vertical arm.*

In Figure 4, for example, the double cross right outside the NE corner of the 3-*SW*-square is the double cross microtile labeled XY, with the row of microtiles radiating westwise and southwise as required.

3.3. Minitiles. Another type of tile needed in our construction is a *minitile* [27]. Each minitile will be a 2×2 block of microtiles that has *exactly one single cross* in

its *upper right corner*; see Figure 8(center).

DEFINITION 3.6. *Consider the microtile system* $(T_\mu, V_\mu, R_\mu, d_\mu)$. *A* minitile *is a quadruple* $a = (\mu_1^a, \mu_2^a, \mu_3^a, \mu_4^a) \in T_\mu^4$ *such that* μ_1^a *is a single cross,* $\mu_2^a, \mu_3^a, \mu_4^a$ *are not, and the microtiling* $\{((1,1), \mu_1^a), ((1,0), \mu_2^a), ((0,0), \mu_3^a), ((0,1), \mu_4^a)\}$ *is valid.*

Therefore, by the above definition, all the arrow heads and tails between $\mu_1^a, \mu_2^a, \mu_3^a, \mu_4^a$ match. Let T_m be a set of minitiles; then a T_m-tiling will be called *minitiling*. If $f : \mathbb{Z}^2 \xrightarrow{o} T_m$ is a minitiling, then its corresponding microtiling $m(f) : \mathbb{Z}^2 \xrightarrow{o} T_\mu$ is defined as follows. If $f(x, y) = a = (\mu_1^a, \mu_2^a, \mu_3^a, \mu_4^a)$, then

$$m(f)(2x+1, 2y+1) = \mu_1^a,$$
$$m(f)(2x+1, 2y) = \mu_2^a,$$
$$m(f)(2x, 2y) = \mu_3^a,$$
$$m(f)(2x, 2y+1) = \mu_4^a.$$

Next, directions are added to minitiles. As we restricted ourselves to 2×2 blocks having exactly one single cross located in the upper right corner, attaching directions to minitiles will amount to attaching directions to single-cross microtiles. Let us describe first the types of directed minitiled paths (or simply paths, if no danger of confusion exists) these directions will define. The paths are constructed recursively through squares of size $2^n \times 2^n$ minitiles.

DEFINITION 3.7. *For each* $n \geq 0$ *we define recursively an* A_{2^n} *(*B_{2^n}, C_{2^n}, D_{2^n}*) -minitiled path as follows. For* $n = 0$, *a minitile with its single cross labeled* X *is an* X_1*-path for* $X \in \{A, B, C, D\}$.

For each $n \geq 0$, *a* $A_{2^{n+1}}$ *(*$B_{2^{n+1}}$, $C_{2^{n+1}}$, $D_{2^{n+1}}$*)-path is built recursively from four* X_{2^n}*-paths,* $X \in \{A, B, C, D\}$, *as described in Figure* 5.

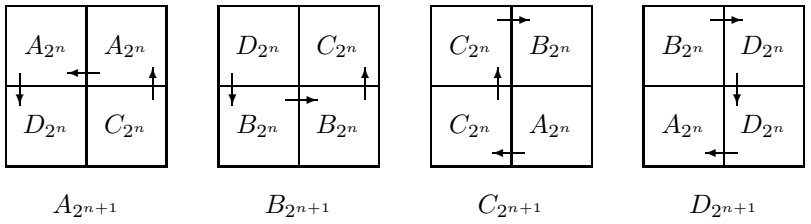


FIG. 5. *Constructing paths through squares of* $2^{n+1} \times 2^{n+1}$ *minitiles. Reprinted from the Journal of Computer and System Sciences, Volume* 48, *Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182, 1994, *with permission from Elsevier.*

For example, the A_4-path starts in the lower right corner of the square, visits all its minitiles, and ends in the lower left corner (see Figure 6). Paths created using Definition 3.7 form the familiar Hilbert curve. One can easily verify the following result.

LEMMA 3.3. *For each* $n \geq 0$ *the* A_{2^n} *(*B_{2^n}, C_{2^n}, D_{2^n}*)-path starts in the SE (NW, SE, NW, respectively) corner, ends in the SW (NE, NE, SW, respectively) corner, visits all minitiles of the underlying* $2^n \times 2^n$ *minitile square, and does not visit any minitiles outside of the underlying square.*

Let $f_{XY} : S_{n+1} \longrightarrow T_\mu$ be a $(2^{n+1} - 1)$-XY-square of microtiles, where $S_{n+1} = \{(x+i, y+j) \mid 1 \leq i, j \leq 2^{n+1}-1\}$. Then, by the iterative construction in Definition 3.5, those (and only those) microtiles at positions $(x+2i-1, y+2j-1)$, $1 \leq i, j \leq 2^n$, are

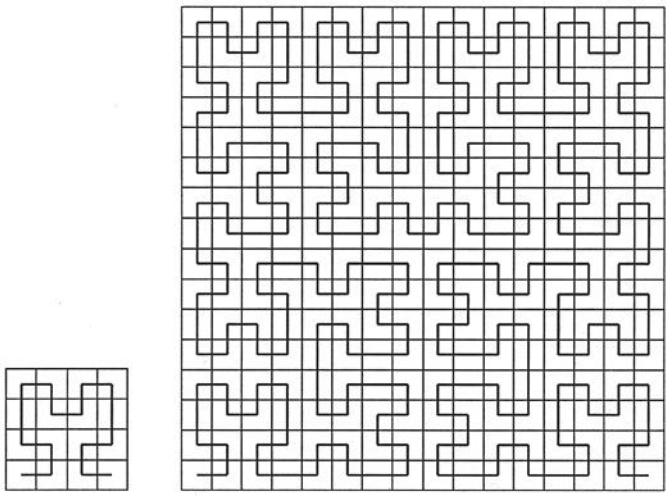


FIG. 6. *The paths A_4 and A_{16}. Reprinted from the Journal of Computer and System Sciences, Volume* 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

TABLE 1

The labeling of mixed arms. The four rows of the table represent (i) *the direction of the principal arrow of the arm,* (ii) *the label of the principal arrow,* (iii) *the label of the arrow situated clockwise relative to the principal arrow, and* (iv) *the label of the arrow situated counterclockwise relative to the principal arrow.*

(i)	E	W	N	S	E	W	N	S	E	W	N	S	E	W	N	S
(ii)	A	A	A	A	B	B	B	B	C	C	C	C	D	D	D	D
(iii)	C	A	A	D	B	D	C	B	A	C	B	C	D	B	D	A
(iv)	A	D	A	C	C	B	D	B	B	C	C	A	D	A	B	D

single crosses. Since, moreover, f_{XY} is a valid microtiling, all 2×2 squares of S_{n+1} with single crosses in their upper right corner form minitiles. The single crosses in the leftmost column and the bottommost row of S_{n+1} can be easily completed by adding another column and a row of microtiles to f_{XY} so that they again form valid 2×2 squares of microtiles. Therefore, by Definition 3.6, we obtain a (partial) minitiling $f : \mathbb{Z}^2 \xrightarrow{o} T_m$ from which the square f_{XY} arose, i.e., $m(f)|_{S_{n+1}} = f_{XY}$.

If a minitile has its single cross in S_{n+1}, we say that it is *attached* to S_{n+1}. Recall that there are exactly $2^n \times 2^n$ single crosses in S_{n+1}; therefore there will be $2^n \times 2^n = 4^n$ minitiles attached to it. The directions will be defined in such a way that the path these minitiles form is exactly an A_{2^n}-, B_{2^n}-, C_{2^n}-, or D_{2^n}-path.

In order to control which of the four possible paths the directions define in a specific square, additional labels are assigned to arrows in microtiles [27]. Each arrow (single or double) can be given any label from the set $\{A, B, C, D\}$, the only restrictions we impose on crosses and mixed arms. As usual, in each cross all four arrow heads must have the same label. The central cross of each $(2^{n+1} - 1)$-square will determine the type of path the directions define on the square (see Lemma 3.6). The labeling of mixed arms is crucial for the composition of A-, B-, C-, or D-squares and is specified in Table 1. As usual, in a valid microtiling the meeting arrow heads and arrow tails must have the same label.

One can verify that the labels in Table 1 correspond exactly to the construction

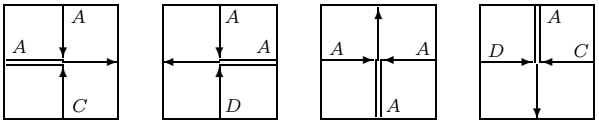


FIG. 7. *The labeling of mixed arms whose principal arrow has the label A.*

of the paths in Figure 5. For example, the mixed arms in the first four columns in Table 1 are illustrated in Figure 7. The first of these mixed arms has its principal arrow labeled A. This indicates that the mixed arm is a connector in a $(2^{n+1}-1)$-square whose central cross is labeled A and which therefore contains an $A_{2^{n+1}}$-path. The vertical arrow incoming from the north (south) is labeled A (C, respectively). These facts indicate that the (2^n-1)-*NE*-square on top of the mixed arm contains an A_{2^n}-path, and that the (2^n-1)-*SE*-square below it contains a C_{2^n}-path, as they should, according to the recursive construction of A_{2^n}-paths in Figure 5.

TABLE 2

The definition of directions assigned to minitiles (or to their respective single crosses). Reprinted from the Journal of Computer and System Sciences, Volume 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

	The direction of a single cross is
N	if its NE neighbor is a double cross with label C or a vertical arm whose left edge has a side arrow with label A or B,
E	if its NE neighbor is a double cross with label B or a horizontal arm whose lower edge has a side arrow with label C or D,
S	if its SW neighbor is a double cross with label D or a vertical arm whose right edge has a side arrow with label A or B,
W	if its SW neighbor is a double cross with label A or a horizontal arm whose upper edge has a side arrow with label C or D.

The definitions of directions are summarized in Table 2. It has been shown in [27] that if $f_{XY} : S_n \longrightarrow T_\mu$ is a (2^n-1)-XY-square, then for each single cross whose Moore neighborhood is contained in S_n there exists exactly one rule in Table 2 that can be applied. All the above defined elements forming the directed paths, i.e., the new labels A, B, C, D and the directions of minitiles, are unified together by means of the neighborhood relation R_m. Now we can complete the formal definition of a minitile system, based on the above described microtile system $(T_\mu, V_\mu, R_\mu, d_\mu)$.

DEFINITION 3.8. *A* minitile system *is a generalized directed tile system* (T_m, V_m, R_m, d_m) *such that*

(i) T_m *is the set of all minitiles as defined in Definition* 3.6*;*

(ii) V_m *is the Moore neighborhood;*

(iii) R_m *is the set of all* 9*-tuples* $r \in T_m^9$ *such that all the following conditions are met:*

  (a) *the underlying* 6×6 *microtiling* $m(\{(V_m(i), r(i)) \mid 1 \le i \le 9\})$ *is valid;*

  (b) *exactly one rule in Table* 2 *is applicable to the single cross* $\mu_1^{r(5)}$ *of the central minitile* $r(5)$;

  (c) *exactly one of the following four conditions holds:* $d_m(r(2)) = S$, $d_m(r(4)) = E$, $d_m(r(6)) = W$, $d_m(r(8)) = N$;

(iv) $d_m(a) = d_\mu(\mu_1^a)$ *for each* $a = (\mu_1^a, \mu_2^a, \mu_3^a, \mu_4^a) \in T_m$.

Recall that a 9-tuple $r \in R_m$ corresponds to a 3×3 block of minitiles. The condition (a) states that a necessary condition for the minitiling f to be valid at

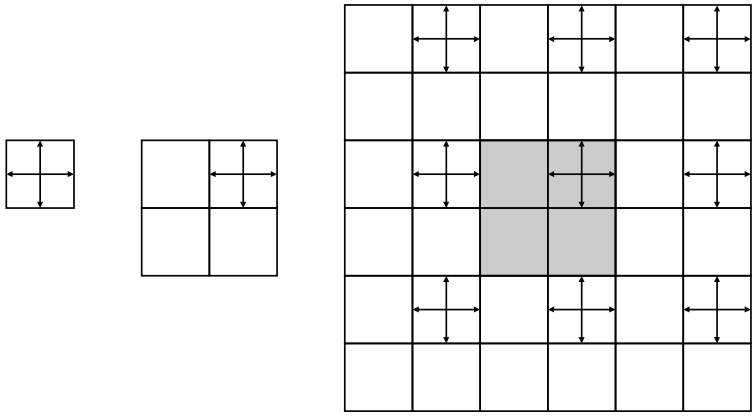


FIG. 8. *Left: a microtile; center: a minitile is a* 2×2 *block of microtiles having exactly one single cross, in its upper right corner; right: a macrotile (shaded); the nonshaded minitiles define the "glues" of the macrotile.*

$(x, y) \in \mathbb{Z}^2$ is that its corresponding microtiling $m(f)$ be valid at $(2x + i, 2y + j)$ for all $-1 \leq i, j \leq 2$. In other words, inside the 6×6 block of microtiles corresponding to the 3×3 Moore neighborhood of (x, y), all arrow heads match meeting arrow tails and their labels match as well, including the new labels A, B, C, D.

The condition (b) adds another necessary requirement that the direction of the minitile at (x, y) must be uniquely defined via Table 2. Finally, by the condition (c) a minitiling is not valid at (x, y) unless exactly one of the four adjacent minitiles at its N, S, E, W has its direction pointing towards it.

Recall that, due to Definition 3.3, if some minitiles in the Moore neighborhood of (x, y) are missing, we consider the minitiling valid at (x, y) if the missing positions can be filled with minitiles such that conditions (iii)(a)–(c) would be met.

3.4. Macrotiles: The 3×3 block construction. In this section we finalize the construction of a directed tile system (T_0, d_0) with the strong plane-filling property. We will use the generalized directed tile system of minitiles (T_m, V_m, R_m, d_m) constructed so far to obtain the directed tile system of *macrotiles* (T_0, d_0). When dealing with (T_0, d_0) we revert to the usual notion of tiles sticking by glues as defined in section 2.

The idea of the construction is as follows. Consider all the validly minitiled Moore neighborhoods as defined by R_m. With each such 9-tuple, which intuitively is a 3×3 block of minitiles, we associate a *macrotile* (see Figure 8). The position of the macrotile in a plane is identical with the position of its center minitile. The glues of a macrotile will be the 2×3 and 3×2 subblocks of minitiles corresponding to each side. Two adjacent macrotiles will stick if they have the same glues at their abutting edges (see Figure 9).

DEFINITION 3.9. *Consider the minitile system* (T_m, V_m, R_m, d_m). *A* macrotile system *is a directed tile system* (T_0, d_0), $T_0 \subseteq X_0^4$, *where the following hold:*

(i) $X_0 = T_m^6$.

(ii) T_0 *is constructed as follows. For each validly minitiled Moore neighborhood* $t \in R_m$, *there exists a* macrotile $t^* \in T_0$ *with glues*
$t_N^* = (t(1), t(2), t(3), t(4), t(5), t(6))$,
$t_S^* = (t(4), t(5), t(6), t(7), t(8), t(9))$,

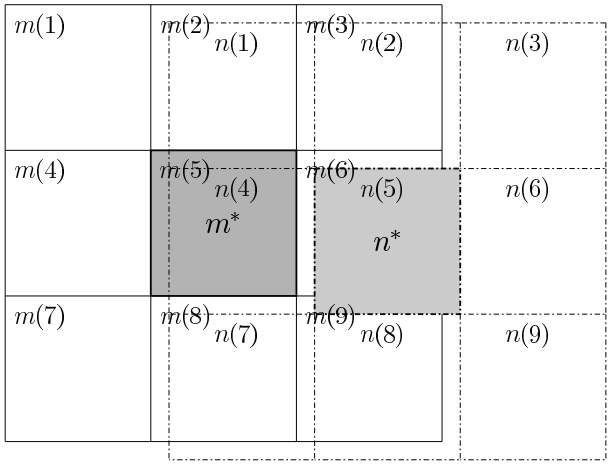


FIG. 9. *Two adjacent macrotiles m^* and n^* (shaded) stick iff their glues at the abutting edges are equal. This means that the overlapping 2×3 portions of their Moore neighborhoods coincide, i.e., $m(2) = n(1)$, $m(3) = n(2)$, $m(5) = n(4)$, etc.*

$t^*_E = (t(2), t(3), t(5), t(6), t(8), t(9))$,
$t^*_W = (t(1), t(2), t(4), t(5), t(7), t(8))$.

(iii) $d_0(t^*) = d_m(t(5))$.

If intuitively a macrotile corresponds to a 3×3 block of minitiles, its north glue is the 2×3 top subblock, the south glue is the bottom 2×3 subblock, the west glue the left 3×2 subblock, and the east glue the right 3×2 subblock of minitiles.

The direction of a macrotile t^* is the direction of its center minitile $t(5)$ which we will denote also by $c(t^*)$. Similarly as with minitiles, we say that t^* is *attached* to a $(2^n - 1)$-square $S_n \longrightarrow T_\mu$ if the single cross of $c(t^*)$ is located within S_n.

Two adjacent macrotiles will stick iff the glues on their abutting edges are identical. For example, two macrotiles at positions (x, y), $(x + 1, y)$ will stick if, when viewed as 3×3 blocks centered at (x, y), respectively, $(x + 1, y)$, their overlapping minitiles coincide. (See Figure 9.)

DEFINITION 3.10. *Consider the macrotile system (T_0, d_0) and a T_0-tiling $f : \mathbb{Z}^2 \xrightarrow{o} T_0$.*

(i) *The* overlap map *$o(f) : \mathbb{Z}^2 \xrightarrow{o} 2^{T_m}$ is a function defined as follows. For all $\bar{x} \in \mathrm{dom}(f)$, if $f(\bar{x}) = t^*$, then $t(i) \in o(f)(\bar{x} + V_m(i))$.*

(ii) *The* projection map *$p(f) : \mathbb{Z}^2 \xrightarrow{o} T_m$ is a function defined as follows: If $(x, y) \in \mathrm{dom}(f)$ and $f(x, y) = t^*$, then $p(f)(x, y) = c(t^*)$.*

In other words, the *overlap map* of a set of macrotiles situated on the plane will place at each position all the minitiles that would occupy it if we were to view each macrotile at (x, y) as a 3×3 block centered at (x, y) and covering a 3×3 neighborhood around it. The *projection map* replaces each macrotile at (x, y) with the minitile $c(t^*)$ at its center. Note that, while $\mathrm{dom}(p(f)) = \mathrm{dom}(f), \mathrm{dom}(f) \subseteq \mathrm{dom}(o(f))$.

Consider the following special case of the overlap map: for all $(x, y) \in \mathbb{Z}^2$, either $o(f)$ is undefined or $\mathrm{card}(o(f)) = 1$. Such a one-to-one overlap map is called a *perfect overlap*. The following results are straightforward.

LEMMA 3.4. *Let $f : \mathbb{Z}^2 \xrightarrow{o} T_0$ be a T_0-tiling with $\mathrm{dom}(f) = \{(x, y), (x', y')\}$ and the positions (x, y) and (x', y') be adjacent; then $f(x, y)$ will stick to $f(x', y')$ iff $o(f)$ is a perfect overlap.*

In other words, two macrotiles situated at adjacent positions will *stick* (in the glue-match-at-abutting-edges sense defined in section 2) iff the overlap map they define is one-to-one.

Let us introduce the following notation for the set of positions on the plane belonging to the Moore neighborhood of $\bar{x} \in \mathbb{Z}^2$: $N_M(\bar{x}) = \{\bar{x} + V_M(i) \mid 1 \leq i \leq 9\}$.

LEMMA 3.5. *Let $f : \mathbb{Z}^2 \xrightarrow{o} T_0$ be a T_0-tiling and let $\bar{x}, \bar{y} \in \mathrm{dom}(f)$. Further let f contain a T_0-ribbon (P, r) such that $\{\bar{x}, \bar{y}\} \subseteq \mathrm{range}(P)$ and*

$$N_M(\bar{x}) \cap N_M(\bar{y}) \subseteq \bigcap_{\bar{z} \in \mathrm{range}(P)} N_M(\bar{z}).$$

Then the overlap map $o(f|_{\{\bar{x},\bar{y}\}})$ is perfect.

Informally, let two macrotiles x^* and y^* be connected with a (short) ribbon such that the intersection of the Moore neighborhoods of x^* and y^* is included in (and hence is equal to) the intersection of the Moore neighborhoods of all tiles in the ribbon. Then, by Lemma 3.4, each two adjacent tiles in the ribbon have perfect overlap. By transitivity, we deduce that the intersection of overlap maps of all the tiles in the ribbon is one-to-one and hence so is the common overlap map of x^* and y^*.

The following lemma states that each infinite directed zipper of macrotiles covers arbitrarily large squares.

LEMMA 3.6. *Let $n \in \mathbb{N}$ and let (P, r) be a T_0-directed zipper, $P : I \longrightarrow \mathbb{Z}^2$, $r : \mathrm{range}(P) \longrightarrow T_0$, such that there exists $i \in I$ with $\min(I) + 4^n \leq i \leq \max(I) - 4^n$. Denote $P(i) = (x_i, y_i)$ and $r(x_i, y_i) = t^*$. Then the following hold:*

(i) *There exists a $(2^{n+1} - 1)$-square $f_{XY} : S_{n+1} \longrightarrow T_\mu$ such that $(2x_i + 1, 2y_i + 1) \in S_{n+1}$, $S_{n+1} \subseteq \mathrm{dom}(m(p(r)))$ and $f_{XY} = m(p(r))|_{S_{n+1}}$.*

(ii) *Let $Q = \{(k, l) \in \mathbb{Z}^2 \mid (2k+1, 2l+1) \in S_{n+1}\}$. If the central cross of S_{n+1} has label A (B, C, D), then $p(r|_Q)$ is an A_{2^n} (B_{2^n}, C_{2^n}, D_{2^n}, respectively)-path.*

(iii) *The overlap map $o(r|_Q)$ is a perfect overlap.*

Informally, consider a directed zipper of macrotiles that passes through a macrotile t^* and extends for at least 4^n positions preceding and the 4^n positions succeeding the position of t^*. Then (i) the single cross of the minitile $c(t^*)$ belongs to a $(2^{n+1} - 1)$-square $f_{XY} : S_{n+1} \longrightarrow T_\mu$. Moreover, (ii) the projection of the portion of the directed zipper consisting of the macrotiles attached to S_{n+1} is an A_{2^n}-, B_{2^n}-, C_{2^n}-, or D_{2^n}-directed minitiled path if the central cross microtile of the square has label A, B, C, or D, respectively. Last, (iii) the overlap map of that portion of the directed zipper is a perfect overlap.

Proof. The proof is by induction on n.

$n = 0$. Assume there is no zipper error involving $4^0 = 1$ macrotile before and one macrotile after t^*. Then the required $2^{n+1} - 1 = 2^1 - 1 = 1$-square consists of only the single cross at the upper right corner of the minitile $c(t^*)$ and the statement is vacuously true.

Assuming the statement holds for $n - 1$, we prove it for n. Assume there are no zipper errors involving any of the 4^n macrotiles that precede or succeed t^*. By induction hypothesis (I.H.) (i), the minitile $c(t^*)$ is attached to a $(2^n - 1)$-square $S^{(1)} \longrightarrow T_\mu$ with the required properties. Assume that it is a $(2^n - 1)$-SW-square (the other cases are analogous.)

Properties of the square $S^{(1)}$.

Let $Q_1 = \{(k, l) \in \mathbb{Z}^2 \mid (2k + 1, 2l + 1) \in S^{(1)}\}$ be the square of macrotiles attached to $S^{(1)}$. By I.H. (iii), the overlap map of $r|_{Q_1}$ is perfect (one-to-one). The underlying microtiling $m(o(r|_{Q_1}))$ covers $S^{(1)}$ and the Moore neighborhood of all its

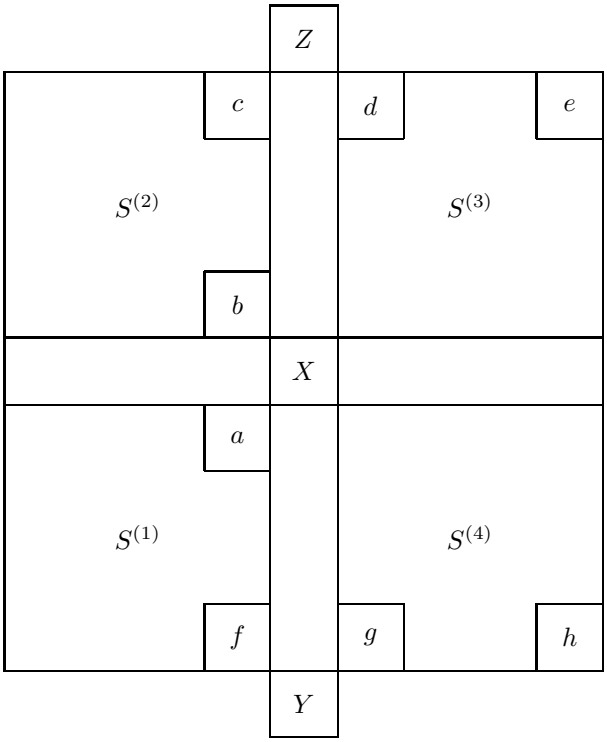


FIG. 10. *The* $(2^{n+1}-1)$*-square constructed during the proof of Lemma* 3.6. *Reprinted from the Journal of Computer and System Sciences, Volume* 48*, Jarkko Kari, Reversibility and surjectivity problems of cellular automata, pages* 149–182*,* 1994*, with permission from Elsevier.*

tiles (actually, it is even larger) and, by Definitions 3.8 and 3.9, is valid. Then, by Lemma 3.2, the microtile just outside the upper right corner of $S^{(1)}$ is a double cross, denoted by X in Figure 10. The double cross can have any of the four labels A, B, C, D. Assume it has label C.

By Lemma 3.2 again, the microtiles exactly on top of $S^{(1)}$ are horizontal arms (first double arms, then one mixed arm, then single arms) radiating westwise, with the same label C. Then, by Table 1, the lower edge of the mixed arm has a side arrow labeled C (column 10, row 4 of the table). Then the central cross of $S^{(1)}$ has the same label C, due to the column of arms radiating from it northwise to the mentioned mixed arm (see Definition 3.5). This means the path through $S^{(1)}$ is, by I.H. (ii), a C-path which, by Lemma 3.3, starts at a single cross f and ends at a single cross a (Figure 10).

Adding the square $S^{(2)}$*.*

As a has at its NE a double cross labeled C, according to Table 2, the direction of a is N. This means that the path of macrotiles, passing through the macrotile a^* with the center single cross a, will proceed to b^*. Denote by b the single cross of $c(b^*)$ (Figure 10).

Recall that for the macrotile t^* attached to $S^{(1)}$ we assumed that it is preceded and succeeded by at least 4^n macrotiles on the path P. Since there are exactly 4^{n-1} macrotiles attached to $S^{(1)}$, the distance of t^* and b^* on the path P is at most 4^{n-1}, by Lemma 3.3. Hence the I.H. can be applied to b^* because the number of macrotiles preceding and succeeding b^* on P is at least $4^n - 4^{n-1} = 3 \cdot 4^{n-1} > 4^{n-1}$. By I.H. (i),

b belongs to a (2^n-1)-square $S^{(2)} \longrightarrow T_\mu$.

Observe that a is not in $S^{(2)}$ (otherwise, by Lemma 3.1, $S^{(1)}$ and $S^{(2)}$ would coincide, which is impossible as b is in $S^{(2)} - S^{(1)}$). Hence $S^{(1)}$ and $S^{(2)}$ are disjoint. By I.H. (ii), $S^{(2)}$ contains an A-, B-, C-, or D-path. As $c(b^*)$ is the first minitile of the path, and b is located just above a, by Lemma 3.3 the path must be either an A-path or C-path. In both cases b is located in the SE corner of $S^{(2)}$, and we conclude that $S^{(2)}$ is above $S^{(1)}$.

As $S^{(2)}$ has all the properties conferred by I.H., we repeat for $S^{(2)}$ the steps previously made for $S^{(1)}$. In particular, let $Q_2 = \{(k,l) \in \mathbb{Z}^2 |\ (2k+1, 2l+1) \in S^{(2)}\}$. By I.H. (iii), the overlap map $o(r|_{Q_2})$ of the macrotiles attached to $S^{(2)}$ is one-to-one. Hence, the microtiling $m(o(r|_{Q_2}))$ covering $S^{(2)}$ and its Moore neighborhood is valid.

By definition of mixed arms in Figure 2, the mixed arm under $S^{(2)}$ has an upper edge with a side arrow labeled NW. As there is a column of arms connecting this arrow with the central microtile of $S^{(2)}$, we have that $S^{(2)}$ is a (2^n-1)-NW-square. By Lemma 3.2 we have to the right of $S^{(2)}$ a column of vertical arms with a mixed arm in the middle. By Table 1, the left edge of this mixed arm has a side arrow labeled C, and hence the central cross of $S^{(2)}$ is also labeled C. By I.H. (ii), the path through $S^{(2)}$ is a C-path. By Lemma 3.3, the path enters through b^* and exits through c^* (Figure 10).

We know already that, when taken separately, the overlap maps of the partial tilings with macrotiles attached to $S^{(1)}$, respectively, $S^{(2)}$, are one-to-one. When considering the overlap map associated with $S^{(1)}$ *and* $S^{(2)}$, we have to inspect the cases when there are two macrotiles, x^* attached to $S^{(1)}$ and y^* attached to $S^{(2)}$, such that their Moore neighborhoods overlap. The possible cases are those described in Figure 11 and the symmetrical ones. Recall that, by Lemma 3.3, the zipper (P, r) visits all the macrotiles attached to $S^{(1)} \cup S^{(2)}$. Hence any adjacent pair of them sticks and any tiled path forms a ribbon. Then, in each of the cases in Figure 11 the figure also shows the short ribbon required by Lemma 3.5. (These ribbons have nothing to do with the zipper (P, r).) Hence, by Lemma 3.5, the common overlap map of x^* and y^* is perfect. This further implies that the overlap map $o(r|_{Q_1 \cup Q_2})$ of all the macrotiles attached to $S^{(1)}$ and $S^{(2)}$ is one-to-one.

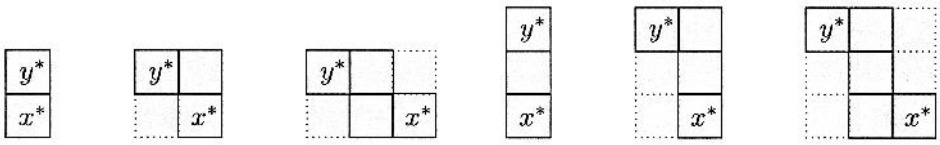


FIG. 11. *Pairs of macrotiles attached to $S^{(1)}$ and $S^{(2)}$ with nonempty common overlap.*

Adding the square $S^{(3)}$.

By Lemma 3.2(ii) applied to $S^{(2)}$, there is a row of vertical arms labeled C to the right of $S^{(2)}$ which ends just to the right of the single cross c. By Lemma 3.2(iii) Z must be a horizontal arm whose lower edge has its side arrow labeled C. As Z is the NE neighbor of c, by Table 2, the direction of c is E. We can reason similarly as in the case of tile b in $S^{(2)}$ to conclude that d is the first single cross in a B-path through a (2^n-1)-NE-square $S^{(3)} \longrightarrow T_\mu$ situated at the right side of $S^{(2)}$. The path ends at tile e (Figure 10).

Reasoning as in the case of $S^{(1)}$ and $S^{(2)}$ we can conclude that the overlap map of macrotiles associated with $S^{(2)}$ and $S^{(3)}$ is one-to-one. To conclude that the overlap

map associated with $S^{(1)}$ *and* $S^{(2)}$ *and* $S^{(3)}$ is one-to-one, we have to consider pairs of macrotiles, x^* attached to $S^{(1)}$ and y^* attached to $S^{(3)}$, such that their Moore neighborhoods overlap. One can easily verify that there are at most nine such pairs. Similarly as in the case of common overlap of $S^{(1)}$ and $S^{(2)}$, for each of these pairs there exists a short ribbon satisfying the requirements of Lemma 3.5. By reasoning similar to the previous case, the overlap map of x^* and y^* is one-to-one and so is the common overlap map of macrotiles attached to $S^{(1)} \cup S^{(2)} \cup S^{(3)}$.

Adding the square $S^{(4)}$.

Consider the overlap map of macrotiles attached to $S^{(1)}$, $S^{(2)}$, and $S^{(3)}$ which forms a valid minitiling. Let f^* be the macrotile with the center single cross f, where the C-path through $S^{(1)}$ starts. The overlap map extends one minitile to the east and south of f^*, hence covering both microtiles g and Y. By Lemma 3.2, Y is a horizontal arm whose upper edge has a side arrow labeled C. As Y is the SW neighbor of the single cross g, it implies, by Table 2, that g has to have direction W.

Moreover, by the nonexistence of a zipper error on the path segment considered, and the definition of the overlap map, there is no tiling error in the overlap map of $S^{(1)}$, $S^{(2)}$, and $S^{(3)}$. Recall that, by Definition 3.8, in a valid minitiling each single cross has only one other single cross pointing towards it. Consequently, g is the unique predecessor of f on the path; therefore g^* uniquely precedes f^*.

According to I.H. (i), g is known to belong to a $(2^n - 1)$-square $S^{(4)} \longrightarrow T_\mu$. In the same way as before, we conclude that $S^{(4)}$ is situated at the right side of $S^{(1)}$, and the path through $S^{(4)}$ is an A-path starting at h and ending at g that visits all its single crosses. We can also reason as before to prove that the overlap map of macrotiles involved in $S^{(1)}$, $S^{(2)}$, $S^{(3)}$, *and* $S^{(4)}$ is one-to-one.

Conclusion.

Altogether, by Definition 3.5 and Lemma 3.2, the squares $S^{(1)}$, $S^{(2)}$, $S^{(3)}$, and $S^{(4)}$ form a $(2^{n+1} - 1)$-square $f_{XY} : S_{n+1} \longrightarrow T_\mu$. All its single crosses are visited by the projection of the path and the macrotile t^* is on this path, as required in the statement (i) of the lemma.

Furthermore, by Definition 3.7, the A-, C-, C-, and B-paths through f_{XY} form a $C_{2^{n+1}}$-path. As we have started with the assumption that the central cross X of S_{n+1} is labeled C, the statement (ii) of the lemma holds, too.

Finally, we have also shown that the common overlap map of the partial macrotiling with macrotiles attached to $S^{(1)}$, $S^{(2)}$, $S^{(3)}$, and $S^{(4)}$ is a perfect overlap. This verifies the statement (iii) of the lemma and concludes the induction step. The other cases (X labeled by A, B, D, and $S^{(1)}$ having other label than SW) are analogous. □

THEOREM 3.1. *There exists a directed macrotile system* (T_0, d_0) *with the strong plane-filling property.*

Proof. Consider the macrotile system (T_0, d_0) from Definition 3.9. Observe that there exists an infinite directed zipper of macrotiles from T_0, namely the zipper (P, r) constructed inductively in the proof of Lemma 3.6. Moreover, by the statement of Lemma 3.6(i), any infinite path following the directions, which in addition has no zipper errors, covers arbitrarily large squares. □

4. Undecidability of existence of infinite ribbons. We shall now use Theorem 3.1 to prove the undecidability of existence of an infinite directed zipper.

THEOREM 4.1. *The set* $\{(T, d) | (T, d)$ *is a directed tile system and there exists an infinite* (T, d)*-directed zipper* $\}$ *is undecidable.*

Proof. Reduce the undecidable tiling problem to our problem.

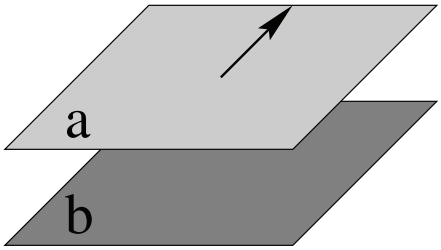


FIG. 12. *A sandwich tile $\sigma(a,b)$ consists of a directed tile $a \in T_0$ placed on top of a tile $b \in T_1$. The direction of $\sigma(a,b)$ is $d(\sigma(a,b)) = d_0(a)$, the direction of its top tile.*

By Theorem 3.1 there exists a directed tile system (T_0, d_0) with the strong plane-filling property.

Let T_1 be a tile system. Consider the following "sandwich tiles," each consisting of a tile from T_0 placed on top of a tile from T_1. That is, create a new directed tile system (T, d) of sandwich tiles such that $T = \{\sigma(a,b) | a \in T_0 \text{ and } b \in T_1\}$, where, for all $a = (a_N, a_E, a_S, a_W) \in T_0$ and $b = (b_N, b_E, b_S, b_W) \in T_1$, the sandwich tile $\sigma(a,b) = ((a_N, b_N), (a_E, b_E), (a_S, b_S), (a_W, b_W))$ and the direction of the sandwich tile is $d(\sigma(a,b)) = d_0(a)$ (see Figure 12).

Hence, a sandwich tile $\sigma(a,b)$ sticks on the north (east, south, west) to $\sigma(a',b')$ iff a sticks on the north (east, south, west) to a' and b sticks on the north (east, south, west) to b'.

We will show now that there exists a valid T_1-tiling of the plane iff there exists an infinite (T,d)-directed zipper.

"$\Rightarrow$" Assume that there exists a valid T_1-tiling of the plane $f : \mathbb{Z}^2 \longrightarrow T_1$. Consider an infinite (T_0, d_0)-directed zipper (P, r) (which exists, since (T_0, d_0) has the strong plane-filling property). Consider the T-tiled path (P, q) of sandwich tiles such that, for all (x, y) on P, $q(x, y) = \sigma(r(x, y), f(x, y))$. Informally, the directed tiled path of sandwich tiles consists of the infinite (T_0, d_0)-directed zipper on its top and the corresponding partial valid T_1-tiling $f|_{\text{range}(P)}$ on the bottom. It is clear that in fact (P, q) is an infinite (T, d)-directed zipper of sandwich tiles.

"$\Leftarrow$" Assume that there exists an infinite (T, d)-directed zipper (P, q) of sandwich tiles. Then, let (P, r) be its top layer; i.e., for all (x, y) on P, $r(x, y) = a$ iff $q(x, y) = \sigma(a, b)$ for some $b \in T_1$. Then clearly, (P, r) is an infinite (T_0, d_0)-directed zipper. Hence, as (T_0, d_0) has the strong plane-filling property, P contains "arbitrarily large squares": for all n there exists (x, y) such that, $(x+i, y+j)$ is on P for $i = 0, 1, 2, \ldots, n$ and $j = 0, 1, 2, \ldots, n$. Let (P, z) be the bottom layer of (P, q), i.e., the T_1-tiled path such that, for all (x, y) on P, $z(x, y) = b$ iff $q(x, y) = \sigma(a, b)$. Then clearly, (P, z) is in fact an infinite T_1-zipper. It follows that $\text{range}(P) = \text{dom}(z)$ contains arbitrarily large squares and, moreover, for all n, $z : \text{range}(P) \longrightarrow T_1$ is a partial tiling valid on a square of size n. It now follows from the König infinity lemma [28] that there exists a valid T_1-tiling of the plane. □

Having proved the undecidability of existence of an infinite directed zipper, we shall use this result to show that the existence of an infinite ribbon is also undecidable.

THEOREM 4.2. *The set $\{T | T$ is a tile system and there exists an infinite T-ribbon$\}$ is undecidable.*

Proof. Reduce the set of Theorem 4.1 to this set.

Given a directed tile system (T, d), we will construct a tile system T' such that there exists an infinite (T, d)-directed zipper iff there exists an infinite T'-ribbon.

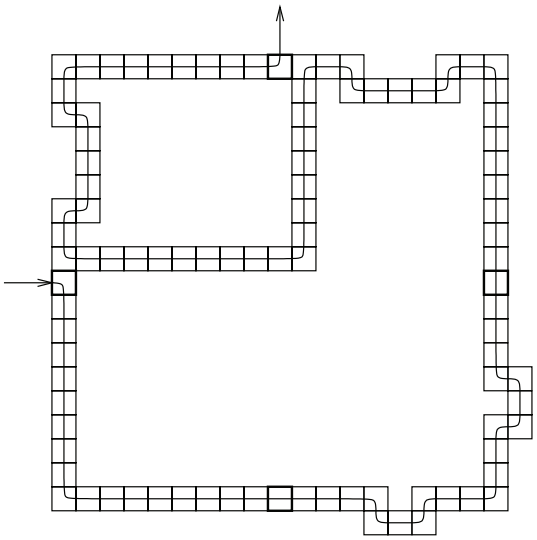


FIG. 13. *A ribbon motif with input tile on the west and output tile on the north.*

Recall that a zipper differs from a ribbon in that in a ribbon only consecutive tiles are required to have matching glues on abutting edges, while on a zipper even nonconsecutive neighboring tiles have to have matching glues on their abutting edges.

The construction is as follows. For each directed tile $t \in T$, construct three T'-*motifs*. A motif is a finite T'-ribbon (P, r) of special form. We construct these ribbon motifs as follows (see Figure 13).

Each motif is essentially a tiled path that outlines the contours of a square. There are dents and bumps on the north, east, south, and west sides of the motif. In addition, if (P, r) is a motif, then the first position on P is midway along one side of the motif. This side is called the input side of the motif and the tile at the first position is called the input tile of the motif. The last position on P is midway along a side of the motif different from the input side. This side is called the output side of the motif and the tile at the last position is called the output tile of the motif.

Given a directed tile $t \in T$, we construct the three possible motifs with output side $d(t)$. We call these three motifs the "variants" of t (see Figure 14). On each variant, we put a bump on the east (south) side, to encode the glue on the east (south) side of t. We also put a dent on the west (north) side, to encode the glue on the west (north) side of t. The dents and bumps are designed so that if, for example, t_1 sticks on the north to t_2, then the dents on the north of t_1 variant motifs fit the bumps on the south of t_2 variants (see Figure 15). If, however, t_1 does not stick on the north to t_2, then the north sides of t_1 variant motifs will overlap the bumps on the south of t_2 variants. Since overlaps are not allowed in ribbons, if t_1 does not stick on the north to t_2, no T'-ribbon can have a t_2 variant motif directly north of a t_1 variant motif.

The glues on tiles in T' are chosen so that each tile can occur in exactly one variant motif, and in each variant motif it occurs exactly once. To this aim, the two edges of each ribbon tile connecting it with the preceding (succeeding) tile on a variant motif are labeled so that this is the only sticking that can form along those edges. The only exception to this rule are the input (output) tiles of motifs, which have the edges connecting them to the preceding (succeeding) tiles labeled differently, as detailed below.

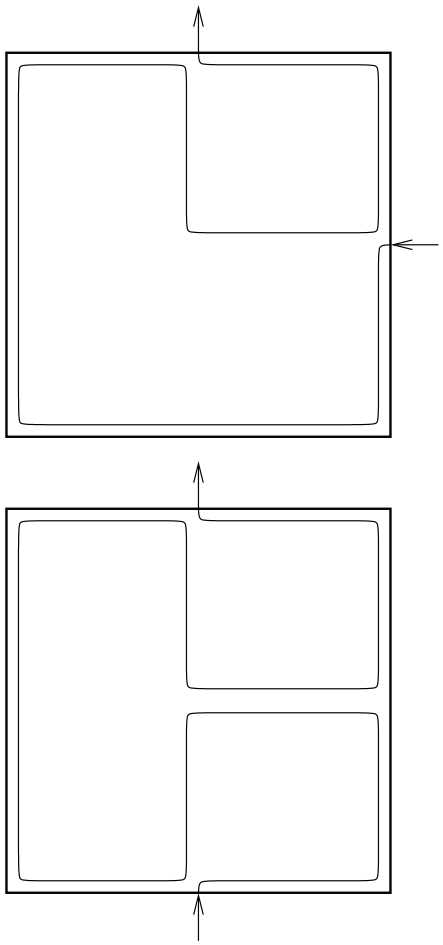


FIG. 14. *The two remaining variant motifs corresponding to a directed zipper tile with direction north (the first one is depicted in Figure* 13*). The bumps and dents are omitted from the variant motifs for clarity.*

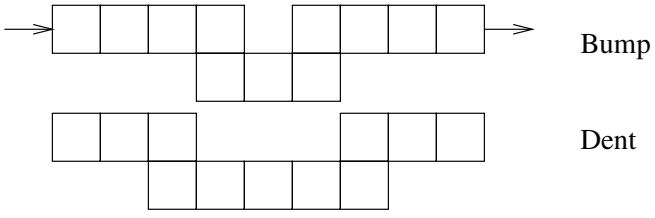


FIG. 15. *The bump-and-dent pair that simulates a glue. Each glue is assigned a unique position on the side of a motif. If the glues on two directed zipper tiles situated at adjacent positions match, then the bump on the first motif will be placed exactly at the necessary position to fit in the dent of the second motif.*

So far, the bumps and dents were used to simulate the glues of zipper tiles. We now simulate the direction of a zipper tile by incorporating it in the glues of the input and output tiles of its variant motifs. We namely use four new glues, *West-to-East*, *East-to-West*, *North-to-South*, and *South-to-North* as follows. If the direction of a directed zipper tile t_1 is north, i.e., $d(t_1) = N$, then

(a) the output ribbon-tiles of all three variant motifs of t_1 will have their north edge labeled *South-to-North*, and

(b) the input tile of the variant motif of t_1 with a west (east, south) input side will have its west (east, south) edge labeled *West-to-East* (*East-to-West*, *South-to-North*).

We label in a similar way the appropriate edges of the input and output tiles of other motifs, namely those edges that are meant to connect the motifs to each other. This labeling ensures that the variant motifs will be connected to each other in the proper order dictated by the direction of the originating directed zipper tiles.

Last, we want to ensure that only motifs of the kind described above can form, and that motifs can connect to each other only through their input and output tiles. To this aim, we have two new different "null" glues: *null(1)* and *null(2)*. We label, for all the above constructed ribbon tiles, the so-far-unlabeled north and west edges with *null(1)* and the unlabeled east and south edges with *null(2)*. Because *null(1)* matches only *null(1)*, and *null(1)* is used only on the west and north edges, the edges of tiles labeled with these glues will not stick to any other tiles along those edges. The same is true for *null(2)*. These "nonstick" glues ensure that the ribbon will only follow the intended motif and will not fill it in, or stick to anything outside it, except at the input and output tiles.

To summarize, given the directed tile system (T, d) we can construct the tile system T' consisting of the ribbon tiles used in all variant motifs of tiles in T, as described above.

Assume that there exists an infinite directed (T, d)-zipper (P, r). One can easily construct an infinite T'-ribbon (P', r'). The idea is that, for each position (x_i, y_i) on P, the tile $r(x_i, y_i)$ is replaced by one of its variant motifs. The variant chosen is one with input side opposite $d(r(x_{i-1}, y_{i-1}))$.

Conversely, assume that there exists an infinite T'-ribbon, (P', r'). It is clear from our choice of ribbon tiles and glues that (P', r') must consist of an infinite sequence of motifs. Hence we can construct an infinite (T, d)-directed zipper by replacing each motif with the tile t of which it is a variant. □

Theorem 4.2 proves the undecidability of existence of an infinite ribbon. This settles the open problem [17], stating the following: "Problem 4.1. Given a tiling system T, is there an infinite T-snake within the infinite grid $G = \mathbb{Z} \times \mathbb{Z}$?"

In the terminology of [17], a tiling system T is exactly as defined in section 2, a finite set of tiles, i.e., of squares with colored edges that cannot be rotated and with infinitely many copies of each tile available. The grid G is the integer grid of positions in the plane. An infinite T-snake is a sequence of tiles on the plane in which successive tiles are adjacent along an edge and touching edges have the same color; i.e., infinite T-snakes are (possibly self-crossing) 2-way infinite ribbons where identical tiles must be present at the crossing sites. In general [17], [34], an infinite snake problem asks, given a tiling system T, and some portion P of the plane, whether there is an infinite T-snake that lies entirely within P. Reference [17] proves that, given T and a strip of width $k \in N$, the existence of an infinite T-snake that lies entirely within the strip is decidable. Given a tile system T, and a specific tile $t_0 \in T$, the problem of whether there exists an infinite T-snake that contains t_0 is proved undecidable in [17]

using methods in [16] (the case of a 1-way infinite snake starting at t_0 was proven undecidable in [15]). If the special "seed" tile is not specified, like in Problem 4.1, [17] conjectures the problem undecidable but states that "[...] it seems that this would be difficult to prove. We have not been able to adjust the proof techniques of other undecidability results for this purpose."

Theorem 4.2, by showing the undecidability of existence of infinite non-self-crossing snakes, proves Problem 4.1 undecidable.

5. Undecidability of self-assembly of arbitrarily large supertiles. Let us return to the discussion of self-assembly. Supertiles are constructed by an incremental process starting from a single tile and proceeding by addition of single tiles that "stick" to the hitherto built structure. The problem we are addressing is whether or not, given a tile system, an infinite supertile can self-assemble with tiles from that system. To formalize our notions, we have the following definition.

DEFINITION 5.1. *A shape is a function $f : I \longrightarrow \mathbb{Z}^2$ such that I is a set of consecutive natural numbers, $0 \in I$, and the following condition holds. For all $i \in I$, if $i > 0$, then there exists $j \in I$ such that $j < i$ and $f(i)$ and $f(j)$ are adjacent.*

A shape thus describes a connected region of the plane. The size of a shape f is the cardinality of its range, i.e., the number of positions it contains, regardless of how many times they are visited. If we fill in each position of a shape with tiles from a given tile system T, we obtain the notion of a T-supertile.

DEFINITION 5.2. *For all tile systems T, a T-supertile is a pair (f, g), where f is a shape and $g : range(f) \longrightarrow T$ is a function such that the following condition holds. For all $i \in dom(f)$, if $i > 0$, then there exists $j \in dom(f)$ such that $j < i$ and $f(i)$ abuts $f(j)$ on the north (east, south, west) and $g(f(i))$ sticks on the north (east, south, west) to $g(f(j))$.*

The size of a T-supertile (f, g) is the size of its corresponding shape f. Two T-supertiles (f, g) and (f', g') are equivalent iff there exists $i, j \in \mathbb{Z}$ such that for all $(x, y) \in \mathbb{Z}^2$, $(x, y) \in \text{range}(f)$ iff $(x+i, y+j) \in \text{range}(f')$ and for all $(x, y) \in \text{range}(f)$, $g(x, y) = g'(x+i, y+j)$. That is, the T-supertile (f', g') is equivalent to the T-supertile (f, g) iff (f', g') can be obtained by translating (f, g) on the plane.

Note that the definitions of shape and supertile may differ in other papers. The ideas are similar, but the details may not be the same. The problem stated at the beginning of this section is settled by the following result.

THEOREM 5.1. *The following sets are undecidable:*

$S_1 = \{T | T$ is a tile system and there exists an infinite T-supertile$\}$;

$S_2 = \{T | T$ is a tile system and there exists infinitely many nonequivalent finite T-supertiles$\}$.

Proof. It is easily shown that sets S_1 and S_2 are identical to the set $\{T | T$ is a tile system and there exists an infinite T-ribbon $\}$. Hence Theorem 5.1 follows from Theorem 4.2. □

6. Conclusion. In this paper we prove the undecidability of the "unseeded version" of the problem of distinguishing tile systems that allow infinite ribbons to self-assemble from those that do not. The proof includes the construction of a special set of directed tiles with the so-called strong plane-filling property, whereby any directed zipper (a special kind of ribbon) is forced to be plane-filling and, moreover, such an infinite directed zipper always exists.

This result settles an open problem formerly known as the "unlimited infinite snake problem."

We also prove the undecidability of the "unseeded version" of the problem of distinguishing tile systems that allow the self-assembly of infinite supertiles from those that do not.

We introduce a "motif" construction that allows one tile system to simulate another by using geometry to represent glues. This construction may be useful in other contexts.

Acknowledgments. We thank David Harel for pointing out that the "infinite ribbon problem" had been open since 1992 under the name "unlimited infinite snake problem," and Dale Myers and Yael Etzion-Petruschka for clarifications of [17]. We thank Qi Cheng, Ashish Goel, Ming-Deh Huang, Pablo Moisset de Espanes, and Paul Rothemund for discussion and comments.

REFERENCES

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss, *Amorphous computing*, Comm. ACM, 43 (2000), pp. 74–82.

[2] L. Adleman, *Molecular computation of solutions to combinatorial problems*, Science, 266 (1994), pp. 1021–1024.

[3] L. Adleman, *Towards a Mathematical Theory of Self-Assembly*, Technical report 00-722, Department of Computer Science, University of Southern California, Los Angeles, CA, 2000.

[4] L. Adleman, Q. Cheng, A. Goel, and M. Huang, *Running time and program size for self-assembled squares*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), 2001, pp. 740–748.

[5] L. Adleman, Q. Cheng, A. Goel, M. Huang, D. Kempe, P. Moisset de Espanes, and P. Rothemund, *Combinatorial optimization problems in self-assembly*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), 2002, pp. 23–32.

[6] L. Adleman, Q. Cheng, A. Goel, M. Huang, and H. Wasserman, *Linear self-assemblies: Equilibria, entropy, and convergence rates*, in Proceedings of the Sixth International Conference on Difference Equations Augsburg, Germany 2001: New Progress in Difference Equations, B. Aulbach, S. Elaydi, and G. Ladas, eds., CRC Press, Boca Raton, FL, 2004, pp. 51–60.

[7] L. Adleman, J. Kari, L. Kari, and D. Reishus, *On the decidability of self-assembly of infinite ribbons*, in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), 2002, pp. 530–537.

[8] R. Barish, P. Rothemund, and E. Winfree, *Two computational primitives for algorithmic self-assembly: Copying and counting*, NanoLetters, 5 (2005), pp. 2586–2592.

[9] J. Bath, S. Green, and A. Turberfield, *A free-running DNA motor powered by a nicking enzyme*, Angew. Chem. Int. Edn., 44 (2005), pp. 4358–4361.

[10] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro, *Programmable and autonomous computing machines made of biomolecules*, Nature, 414 (2001), pp. 430–434.

[11] R. Berger, *The undecidability of the domino problem*, Mem. Amer. Math. Soc. 66 (1966), pp. 1–72.

[12] H. Chen, A. Goel, C. Luhrs, and E. Winfree, *Self-assembling tile systems that heal from small fragments*, in Preliminary Proceedings of DNA Computing 13, M. Garzon and H. Yan, eds., 2007, pp. 30–46.

[13] M. Cook, P. Rothemund, and E. Winfree, *Self-assembled circuit patterns*, in Proceedings of DNA Computing 9, Lecture Notes in Comput. Sci. 2943, J. Chen and J. Reif, eds., Springer, Berlin, 2004, pp. 91–107.

[14] M. Dubois, B. Demé, T. Gulik-Krzywlcki, J.-C. Dedieu, C. Vautrin, S. Désert, E. Perez, and T. Zemb, *Self-assembly of regular hollow icosahedra in salt-free catanionic solutions*, Nature, 411 (2001), pp. 672–675.

[15] H. Ebbinghaus, *Domino threads and complexity*, in Computation Theory and Logic, Lecture Notes in Comput. Sci. 270, E. Borger, ed., Springer, Berlin, 1987, pp. 131–142.

[16] Y. Etzion, *On the Solvability of Domino Snake Problems*, M.Sc. thesis, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1991.

[17] Y. Etzion-Petruschka, D. Harel, and D. Myers, *On the solvability of domino snake problems*, Theoretic. Comput. Sci., 131 (1994), pp. 243–269.
[18] K. Fujibayashi, R. Hariadi, S. Park, E. Winfree, and S. Murata, *Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern*, NanoLetters, 8 (2008), pp. 1791–1797.
[19] A. Goel and P. Moisset de Espanes, *Towards minimum size self-assembled counters*, in Proceedings of DNA Computing 13, M. Garzon and H. Yan, eds., Lecture Notes in Comput. Sci. 4848, Springer, Berlin, 2008, pp. 46–53.
[20] M. Gomez-Lopez, J. Preece, and J. Stoddart, *The art and science of self-assembling molecular machines*, Nanotechnology, 7 (1996), pp. 183–192.
[21] R. Goodman, I. Schaap, C. Tardin, C. Erben, R. Berry, C. Schmidt, and A. Turberfield, *Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication*, Science, 310 (2005), pp. 1661–1665.
[22] D. Gracias, J. Tien, T. Breen, C. Hsu, and G. Whitesides, *Forming electrical networks in three dimensions by self-assembly*, Science, 289 (2000), pp. 1170–1173.
[23] S. Green, L. Lubrich, and A. Turberfield, *DNA hairpins: Fuel for autonomous DNA devices*, Biophys. J., 91 (2006), pp. 2966–2975.
[24] D. Hilbert, *Über die stetige Abbildung einer Linie auf ein Flächtenstück*, Math. Ann., 38 (1891), pp. 459–460.
[25] M. Kao and R. Schweller, *Reducing tile complexity for self-assembly through temperature programming*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006, pp. 571–580.
[26] J. Kari, *Reversibility of 2D cellular automata is undecidable*, Phys. D, 45 (1990), pp. 379–385.
[27] J. Kari, *Reversibility and surjectivity problems of cellular automata*, J. Comput. System Sci., 48 (1994), pp. 149–182.
[28] D. König, *Über eine Schlussweise aus dem Endlichen ins Unendliche*, Acta Litt. Sci. Szeged, 3 (1927), pp. 121–130.
[29] M. Lagoudakis and T. LaBean, *2D DNA self-assembly for satisfiability*, in DNA Based Computers V, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 54, E. Winfree and D. Gifford, eds., AMS, Providence, RI, 2000, pp. 141–154.
[30] T. Liedl and F. Simmel, *Switching the conformation of a DNA molecule with a chemical oscillator*, NanoLetters, 5 (2005), pp. 1894–1898.
[31] G. Lopinski, D. Wayner, and R. Wolkow, *Self-directed growth of molecular nano-structures on silicon*, Nature, 406 (2000), pp. 48–51.
[32] C. Mao, T. LaBean, J. Reif, and N. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules*, Nature, 407 (2000), pp. 493–496.
[33] B. Muller, A. Reuter, F. Simmel, and D. Lamb, *Single-pair FRET characterization of DNA tweezers*, NanoLetters, 6 (2006), pp. 2814–2820.
[34] D. Myers, *Decidability of the tiling connectivity problem*, abstract 79T-E42, Notices Amer. Math. Soc., 195 (1979), A-441.
[35] R. Plass, J. Last, N. Bartelt, and G. Kellogg, *Nanostructures: Self-assembled domain patterns*, Nature, 412 (2001), p. 875.
[36] J. Reif, *Local parallel biomolecular computation*, in DNA Based Computers III, DIMACS Ser. Discrete. Math. Theoret. Comput. Sci. 48, H. Rubin and D. Wood, eds., AMS, Providence, RI, 1999, pp. 217–254.
[37] R. Robinson, *Undecidability and nonperiodicity for tilings of the plane*, Invent. Math., 12 (1971), pp. 177–209.
[38] P. Rothemund, *Using lateral capillary forces to compute by self-assembly*, Proc. Natl. Acad. Sci. USA, 97 (2000), pp. 984–989.
[39] P. Rothemund, *Theory and Experiments in Algorithmic Self-Assembly*, Ph.D. thesis, University of Southern California, Los Angeles, CA, 2001.
[40] P. Rothemund, *Folding DNA to create nanoscale shapes and patterns*, Nature, 440 (2006), pp. 297–302.
[41] P. Rothemund, N. Papadakis, and E. Winfree, *Algorithmic self-assembly of DNA Sierpinski triangles*, PLoS Biol., 2 (2004), e424.
[42] P. Rothemund and E. Winfree, *The program-size complexity of self-assembled squares*, in Proceedings of the ACM Symposium on Theory of Computing (STOC), 2001, pp. 459–468.
[43] J. Schön, H. Meng, and Z. Bao, *Self-assembled monolayer organic field-effect transistors*, Nature, 413 (2000), pp. 713–715.
[44] W. Sherman and N. Seeman, *A precisely controlled DNA biped walking device*, NanoLetters, 4 (2004), pp. 1203–1207.
[45] W. Shih, J. Quispe, and G. Joyce, *A 1.7-kilobase single-stranded DNA that folds into a*

nanoscale octahedron, Nature, 427 (2004), pp. 618–621.

[46] D. Soloveichik and E. Winfree, *Complexity of self-assembled shapes*, SIAM J. Comput., 36 (2007), pp. 1544–1569.

[47] Y. Tian, Y. He, Y. Peng, and C. Mao, *A DNA enzyme that walks processively and autonomously along a one-dimensional track*, Angew. Chem. Int. Edn., 44 (2005), pp. 4355–4358.

[48] H. Wang, *Proving theorems by pattern recognition*, Bell System Tech. J., 40 (1961), pp. 1–42.

[49] G. Whitesides, J. Mathias, and C. Seto, *Molecular self-assembly and nanochemistry: A chemical strategy for the synthesis of nanostructures*, Science, 254 (1991), pp. 1312–1319.

[50] E. Winfree, *Algorithmic Self-Assembly of DNA*, Ph.D. thesis, California Institute of Technology, Pasadena, CA, 1998.

[51] E. Winfree, F. Liu, L. Wenzler, and N. Seeman, *Design and self-assembly of two-dimensional DNA crystals*, Nature, 394 (1998), pp. 539–544.

[52] E. Winfree, X. Yang, and N. Seeman, *Universal computation via self-assembly of DNA: Some theory and experiments*, in DNA Based Computers II, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 44, L. Landweber and E. Baum, eds., AMS, Providence, RI, 1999, pp. 191–213.

[53] B. Yurke, A. Turberfield, A. Mills, Jr., F. Simmel, and J. Neumann, *A DNA-fuelled molecular machine made of DNA*, Nature, 406 (2000), pp. 605–608.

SIAM J. COMPUT.
Vol. 38, No. 6, pp. 2382–2410

DYNAMIC PROGRAMMING OPTIMIZATION OVER RANDOM DATA: THE SCALING EXPONENT FOR NEAR-OPTIMAL SOLUTIONS*

DAVID J. ALDOUS†, CHARLES BORDENAVE‡, AND MARC LELARGE§

Abstract. A very simple example of an algorithmic problem solvable by dynamic programming is to maximize, over $A \subseteq \{1, 2, \ldots, n\}$, the objective function $|A| - \sum_i \xi_i \mathbb{1}(i \in A, i+1 \in A)$ for given $\xi_i > 0$. This problem, with random (ξ_i), provides a test example for studying the relationship between optimal and near-optimal solutions of combinatorial optimization problems. We show that, amongst solutions differing from the optimal solution in a small proportion δ of places, we can find near-optimal solutions whose objective function value differs from the optimum by a factor of order δ^2 but not of smaller order. We conjecture this relationship holds widely in the context of dynamic programming over random data, and Monte Carlo simulations for the Kauffman–Levin NK model are consistent with the conjecture. This work is a technical contribution to a broad program initiated in [D. J. Aldous and A. G. Percus, *Proc. Natl. Acad. Sci. USA*, 100 (2003), pp. 11211–11215] of relating such scaling exponents to the algorithmic difficulty of optimization problems.

Key words. dynamic programming, local weak convergence, Markov chain, near-optimal solutions, optimization, probabilistic analysis of algorithms, scaling exponent

AMS subject classifications. 68Q25, 90C39, 60J05

DOI. 10.1137/070709037

1. Introduction and motivation.

1.1. Near-optimal solutions in combinatorial optimization. Consider a combinatorial optimization problem which is "size n" in the sense that a feasible solution $\mathbf{x} = (x_i, 1 \le i \le n)$ consists of n elements (e.g., edges of a graph; binary digits) subject to some constraints, and the objective function $f(\mathbf{x})$ is akin to a sum over i of costs or rewards associated with each x_i. In such a setting one can define the relative distance between the structure of a feasible solution $\mathbf{x}$ and the optimal solution $\mathbf{x}^*$ by

$$\delta_n(\mathbf{x}) = n^{-1}|\{i : x_i \neq x_i^*\}|,$$

and the relative difference in objective function is $n^{-1}|f(\mathbf{x}) - f(\mathbf{x}^*)|$. So the quantity

$$\varepsilon_n(\delta) := \min\{n^{-1}|f(\mathbf{x}) - f(\mathbf{x}^*)| \ : \delta_n(\mathbf{x}) \ge \delta\} \tag{1}$$

measures how close we can get to the optimal value using feasible solutions which have nonnegligibly different structure from the optimal solution. A program initiated in [3] is to study this quantity for combinatorial optimization problems over *random* data. In this setting $\varepsilon_n(\delta)$ becomes a random variable, but in many cases one expects that

*Received by the editors November 26, 2007; accepted for publication (in revised form) December 17, 2008; published electronically March 20, 2009.
http://www.siam.org/journals/sicomp/38-6/70903.html
†Department of Statistics, University of California, Berkeley, CA 94270 (aldous@stat.berkeley.edu). This author's research was supported by NSF grant DMS0704159.
‡Université de Toulouse & CNRS, Institut de Mathématiques de Toulouse, F-31062 Toulouse, France (charles.bordenave@math.univ-toulouse.fr). This author's research was supported by NSF grant CCF-050023.
§INRIA-ENS, 75005 Paris, France (marc.lelarge@ens.fr).

as $n \to \infty$ there is a *deterministic* limit function $\varepsilon(\delta)$. Motivation for this program is a conjecture that (within some suitable class of problems)

$$\varepsilon(\delta) \asymp \delta^{\alpha} \text{ as } \delta \to 0$$

for some *scaling exponent* α, whose value is robust under model details, and that for "algorithmically easy" problems we have $\alpha = 2$ (which of course mimics the behavior we expect by calculus for smooth functions $f : \mathbb{R}^d \to \mathbb{R}$) whereas for "algorithmically hard" problems we have $\alpha > 2$. Here is the previous evidence in support of this conjecture.

(i) *Traveling salesman problem* and *minimum matching problem* [3]. In the random link (mean-field) model, a cavity method analysis (nonrigorous but generally regarded as accurate) enables one to compute $\varepsilon(\delta)$ numerically and to observe scaling exponent $\alpha = 3$. In the random Euclidean model, Monte Carlo simulations suggest the same $\alpha = 3$.

(ii) *Minimum spanning tree.* Here we expect $\alpha = 2$. This is proved in [2] for the $d \geq 2$ dimensional random Euclidean model and also for a "disordered lattice" model.

The purpose of this paper is to consider some problems which are algorithmically easy to solve via dynamic programming, and where we therefore expect $\alpha = 2$. We first give a trivial but instructive case (section 1.2) and then describe a prototypical "interesting" case, the Kauffman–Levin NK model (section 1.3). Here both a heuristic argument and simulations suggest $\alpha = 2$, but we do not have a proof. Our main focus is on giving a complete analysis of a simple nontrivial model (section 1.4), where we are required to pick a subset $A \subseteq [n] := \{1, 2, \ldots, n\}$ of items with a reward of 1 per item picked and independent and identically distributed (i.i.d.) costs ξ_i incurred if both items i and $i + 1$ are picked. Theorem 2 establishes $\alpha = 2$ for this specific model. In these dynamic programming examples and the minimum spanning tree example, the key structural property is that the near-optimal solutions attaining the minimum in (1) differ from the optimal solution via only "local changes," each local change affecting only a number of items which remains $O(1)$ as $\delta \to 0$. It is natural to speculate that this structural property corresponds quite generally to the $\alpha = 2$ case.

Related work. We do not know any other lines of research in theoretical computer science which are close to the topic of this paper. A recent survey of average-case complexity of NP problems is given in [7]. Interest in the average-case gap between optimal and second-optimal solutions arises in several contexts; see, e.g., [5]. Closer in spirit is the statistical physics of disordered systems, where for low temperatures the Gibbs distribution on configurations concentrates on near-minimal-cost configurations. In the context of random energy models (the precise analogue of optimization over random data), two random picks from the Gibbs distribution over the same random choice of energy are called *replicas*, and study of such replicas and their overlaps is a central theme of the *replica method* [15, 17]. So that topic studies the structural difference between two *typical* near-optimal configurations, whereas we study the *maximal* (over all near-optimal configurations) structural difference from the optimal configuration. Our mathematical arguments are much less sophisticated than those in statistical physics, but there are some intriguing parallels, described briefly in section 5.2.

1.2. A trivial example. Let $(X_i, i \geq 1)$ be i.i.d. real-valued random variables with continuous density $h(x)$ and $\mathrm{E}X < \infty$. For each n consider the problem of finding

$$M_n = \max_{A \subseteq [n]} \sum_{i \in A} (X_i - 1).$$

The maximum is obviously obtained by choosing $A = \{i : X_i > 1\}$ and then as $n \to \infty$

$$n^{-1} M_n \to \mathrm{E}(X_1 - 1)^+ \text{ a.s.}$$

Fix $0 < \delta < 1$. It is also obvious that the subset A' that minimizes

$$M'_n = \max_{A' \subseteq [n]} \sum_{i \in A'} (X_i - 1)$$

$$\text{subject to } |A' \triangle A| \geq \delta n$$

is the subset $A' = A \triangle D$, where D is the set of indices of the $\lceil \delta n \rceil$ smallest values of $|X_i - 1|$. So as $n \to \infty$

$$n^{-1}(M_n - M'_n) \to_{L_1} \varepsilon(\delta) := \int_{1-a(\delta)}^{1+a(\delta)} |x - 1| h(x)\, dx,$$

where $a(\delta)$ is defined by

$$\delta = \int_{1-a(\delta)}^{1+a(\delta)} h(x)\, dx.$$

So by continuity of $h(x)$, and assuming $0 < h(1) < \infty$, as $\delta \downarrow 0$ we have

$$a(\delta) \sim \tfrac{\delta}{2h(1)}; \quad \varepsilon(\delta) \sim a^2(\delta) h(1) \sim \tfrac{\delta^2}{4h(1)}, \tag{2}$$

which is the desired "scaling exponent = 2" result.

Discussion. (i) This example illustrates a feature that arises in other examples, that proving $\alpha = 2$ reduces to showing that the density of a certain measure at a certain point is finite and nonzero. In nontrivial examples the measure in question arises in the *analysis* of the problem rather than the statement of the problem: see Lemma 19 below and Proposition 8 of [2].

(ii) In this example we could see the form of the best near-optimal solution by inspection, but a systematic method is to use Lagrange multipliers. In this example, introduce a parameter $\theta > 0$ and consider for each n

$$A_\theta := \arg\max_A \left(\sum_{i \in A} (X_i - 1) + \theta |A \triangle A^*| \right),$$

where $A^* = \{i : X_i > 1\}$ is the optimal solution. By inspection the solution is

$$A_\theta = \{i : \ 1 - \theta \leq X_i \leq 1 \text{ or } 1 + \theta \leq X_i\}.$$

Although now $|A_\theta \triangle A^*|$ is random, we can use the law of large numbers to obtain existence of the limits

$$\delta(\theta) := \lim_{n \to \infty} n^{-1} |A^* \triangle A_\theta| = \int_{1-\theta}^{1+\theta} h(x)\, dx,$$

$$\varepsilon(\theta) := \lim_{n \to \infty} n^{-1} \left(\sum_{i \in A^*} (X_i - 1) - \sum_{i \in A_\theta} (X_i - 1) \right) = \int_{1-\theta}^{1+\theta} |x - 1|\, dx.$$

By the interpretation of Lagrange multipliers, this is an implicit function representation of ε as a function of δ and rederives the limit (2) above.

$$\begin{array}{ll} 01100011010010111010001001101010101101111000101011010 & \mathbf{x}^N \\ 01100011\underline{011}0101110100010011010\underline{10001110100}0000101011010 & \mathbf{y} \end{array}$$

FIG. 1. *Excursions of lengths* $l = 3$ *and* 11. *Here* $K = 2$.

1.3. The NK model. The Kauffman–Levin NK model of random fitness landscape has attracted extensive literature in statistical physics [10, 19] and has been studied by probabilists [9, 11]. For our version of the model we fix $K \geq 2$. We seek to minimize, over binary sequences $\mathbf{x} = (x_1, \ldots, x_N)$, the objective function $H_N(\mathbf{x}) = \sum_{i=1}^{N-K} W_i(x_i, x_{i+1}, \ldots, x_{i+K})$, where the values $(W_i(b_0, b_1, \ldots, b_K) : i \geq 1, \mathbf{b} \in \{0,1\}^{K+1})$ are independent exponential(1) random variables. This is algorithmically easy via dynamic programming. Write $\mathbf{x}^N$ for the minimizing sequence. By subadditivity there is an a.s. limit $N^{-1}H_N(\mathbf{x}^N) \to c_K$. For a general sequence $\mathbf{y} = \mathbf{y}^N$ write

$$\delta_N(\mathbf{y}) = N^{-1}|\{1 \leq i \leq N-K : (y_i, \ldots, y_{i+K}) \neq (x_i^N, \ldots, x_{i+K}^N)\}|,$$

$$\varepsilon_N(\mathbf{y}) = N^{-1}(H_N(\mathbf{y}) - H_N(\mathbf{x}^N))$$

and then set

$$\varepsilon_N(\delta) = \min\{\varepsilon_N(\mathbf{y}) : \ \delta_N(\mathbf{y}) \geq \delta\}. \tag{3}$$

We expect existence of a deterministic limit

$$\varepsilon(\delta) = \ \text{a.s.-}\lim_{N\to\infty} \varepsilon_N(\delta).$$

A heuristic analysis. The purpose of this section is to give a heuristic argument for $\varepsilon(\delta) \asymp \delta^2$. Given i and $l \geq K+1$, consider the set of sequences $\mathbf{y}$ such that

$$(y_j, \ldots, y_{j+K}) = (x_j^N, \ldots, x_{j+K}^N) \quad \forall j \notin [i+1, i+l],$$

$$(y_j, \ldots, y_{j+K}) \neq (x_j^N, \ldots, x_{j+K}^N) \quad \forall j \in [i+1, i+l].$$

Over this set, let $D_{i,l}$ be the minimum of $H_N(\mathbf{y}) - H_N(\mathbf{x}^N)$ and let $\mathbf{y}^{(i,l)}$ be the minimizing sequence. The distribution of $D_{i,l}$ essentially depends only on l, not on i or N; write $f_l(0+)$ for its density at $0+$. Let us assume

$$\sum_{l \geq K+1} l^2 f_l(0+) = A < \infty. \tag{4}$$

It is intuitively clear how to choose a sequence $\mathbf{y}$ which minimizes $\varepsilon_N(\mathbf{y})$ for a given δ. Just fix a small $\eta > 0$ and create a sequence of "excursion" away from $\mathbf{x}^N$ as follows. For each pair (i,l) such that $D_{i,l} < \eta l$, choose $\mathbf{y}$ to equal $\mathbf{y}^{(i,l)}$ on the sites $[i+K+1, i+l]$; set $\mathbf{y} = \mathbf{x}^N$ elsewhere. See Figure 1.

With this scheme, δ will be the mean length of possible excursions starting from a given site, that is,

$$\delta \sim \sum_{l \geq K+1} l \cdot \eta l f_l(0+).$$

TABLE 1

Monte Carlo simulations with $K = 3, N = 10,000$; 1000 repeats. These are exact optimizations done by introducing a Lagrange multiplier θ which penalizes matching $(K+1)$-tuples. We find $c_3 = 0.3065$.

θ	δ	ε	ε/δ^2	$\mathrm{E}L_\delta$
0.002	0.0397	$4.85 \cdot 10^{-5}$	0.0308	10.9
0.004	0.0774	$2.00 \cdot 10^{-4}$	0.0334	11.0
0.008	0.147	$7.69 \cdot 10^{-4}$	0.0354	11.3
0.016	0.266	$2.75 \cdot 10^{-3}$	0.0388	11.8

And ε is the mean increment of H_N associated with possible excursions starting from a given site, that is,

$$\varepsilon \sim \sum_{l \geq K+1} (\eta l/2) \cdot \eta l f_l(0+).$$

In other words $\delta \sim A\eta$, $\varepsilon \sim A\eta^2/2$, giving $\varepsilon \sim (2A)^{-1}\delta^2$, which is the desired "scaling exponent = 2" result.

Why should assumption (4) be true? Well, for large l we expect central limit behavior: $D_l \approx \mathrm{Normal}(\mu l, \sigma^2 l)$ for some $\mu > 0$ and $0 < \sigma^2 < \infty$. This in turn suggests that $f_l(0+)$ should decrease at least geometrically fast in l.

Note that the optimizing $\mathbf{y}^N$ in (3) will have (in the $N \to \infty$ limit) some distribution L_δ of excursion lengths. The heuristic argument predicts that as $\delta \downarrow 0$ we have $L_\delta \xrightarrow{d} L$, where the limit distribution has $\mathrm{P}(L = l) \propto l f_l(0+)$ and $\mathrm{E}L < \infty$.

Simulations (Table 1) with $K = 3$ are consistent with both the predicted scaling exponent 2 and the prediction of existence of a $\delta \downarrow 0$ limit distribution L for excursion lengths. Making a rigorous proof seems difficult, and so we turn to a simpler example.

1.4. Main model and results. Let $(\xi_i, i \geq 1)$ be i.i.d. copies of a strictly positive random variable ξ, and write $G(x) = \mathrm{P}(\xi \leq x)$. Define the *benefit* function

$$f_n(A) = \left(|A| - \sum_{i=1}^{n-1} \xi_i \mathbb{1}(i \in A, i+1 \in A) \right), \quad A \subseteq \{1, 2, \ldots, n\}, \tag{5}$$

where $\mathbb{1}(B) = \mathbb{1}_B$ denotes the indicator random variable associated with an event B. Intuitively, we choose a set A of items, getting reward 1 from each item chosen but paying cost ξ_i if we choose both i and $i+1$; we seek to maximize benefit = reward − cost. So we shall study

$$M_n := \max_{A \subseteq \{1,2,\ldots,n\}} f_n(A). \tag{6}$$

To simplify exposition we will assume

$$G \text{ has bounded continuous density } g \text{ with } g(\tfrac{1}{2}) > 0, \tag{7}$$

which implies

$$0 < G(\tfrac{1}{2}) < 1, \tag{8}$$

though we suspect that Theorems 1 and 2 remain true under some much weaker nondegeneracy assumptions. See section 5.1 for further remarks.

We will first prove the following.

THEOREM 1. *There exists* $\frac{1}{2} \le c \le 1$ *such that, a.s. and in* L^1,

$$\lim_{n\to\infty} n^{-1} M_n = c.$$

The constant c *is given by the forthcoming formula* (31). *If* ξ *is an exponential random variable with parameter* $\lambda > 0$, *then*

$$c = (1 - e^{-\lambda})^{-1} - \lambda^{-1}.$$

We record the explicit value of c only in the exponential case, but one could use formula (31) to obtain explicit values for other standard distributions.

We now formalize the setup in the introduction. The optimization problem (6) has a solution, a random subset $A_n^{\text{opt}} \subseteq \{1, 2, \ldots, n\}$, and Corollary 4 will show the solution is unique with probability $\to 1$ as $n \to \infty$. Define the random variable:

$$\varepsilon_n(\delta) := \min\left\{ n^{-1}(f_n(A_n^{\text{opt}}) - f_n(B)) : |B \triangle A_n^{\text{opt}}| \ge \delta n \right\}, \tag{9}$$

where the minimum is over all subsets $B \subset \{1, \ldots, n\}$ such that the symmetric difference with A_n^{opt} is at least δn. Our main result is the following.

THEOREM 2. $\bar{\varepsilon}(\delta) := \lim_n \mathrm{E}\varepsilon_n(\delta)$ *exists for all* $0 < \delta < 1$, *and*

$$\limsup_{\delta \downarrow 0} \delta^{-2} \bar{\varepsilon}(\delta) < \infty, \tag{10}$$

$$\liminf_{\delta \downarrow 0} \delta^{-2} \bar{\varepsilon}(\delta) \ > 0. \tag{11}$$

We now outline the key ideas in the proof and the organization of this paper.

Dynamic programming over i.i.d. data is essentially just study of a related Markov chain (section 2.2), and in our model there are simple *inclusion criteria* for whether item i is in the optimal solution. The inclusion criterion involves two Markov chains (one looking left, one looking right) and the cost ξ_i (Table 2 and Lemma 5). By considering the related infinite-time *stationary* Markov chain and using the same inclusion criteria, we can define a random subset $A^{\text{opt}} \subset \mathbb{Z}$ interpretable as the solution of an infinite optimization problem (section 2.3). The $n \to \infty$ limit benefit in Theorem 1 is just the mean benefit per item using A^{opt} in the infinite problem (section 2.4).

Study of $\varepsilon_n(\delta)$ is an "optimization under constraint" problem, most naturally handled via introduction of a Lagrange multiplier θ. So the B_n^{opt} attaining the maximum in (9) can be studied as above by introducing a more complicated Markov chain parametrized by θ (section 4.1), finding the inclusion criteria (Table 3), formulating the parallel optimization under constraint problem, and observing that $\bar{\varepsilon}(\delta)$ is representable via functions $\delta(\theta), \varepsilon(\theta)$ defined in terms of the stationary distribution of the more complicated Markov chain (Proposition 12). Without trying to write details, it seems intuitively clear that the methodology above could be implemented in more general dynamic programming models such as the NK model of section 1.3. However, to complete the argument we need to analyze the $\theta \to 0$ behavior of the functions $\delta(\theta), \varepsilon(\theta)$. Even in our simple model, we do not have any useful explicit expression for the needed stationary distribution, so we proceed via inequalities rather than using the exact formulas. For the upper bound (10) we just identify a "local configuration" which can be replaced by a different local configuration at small extra cost (section 3). For the lower bound, we decompose the process into blocks by breaking at certain special configurations, and then we get bounds on the chance that B_n^{opt} differs from

	g	$-$	$(g+1)$	$-$				$-$	$(d-1)$	$-$	d
A_1	$\circ$	$-$	$\bullet$	$-$	$\circ$	$\ldots$	$\bullet$	$-$	$\circ$	$-$	$\bullet$
A_2	$\bullet$	$-$	$\circ$	$-$	$\bullet$	$\ldots$	$\circ$	$-$	$\bullet$	$-$	$\circ$

FIG. 2. *Included items marked* $\bullet$*, excluded items marked* $\circ$.

A_n^{opt} on a block and bounds on the mean decrease in benefit if it does differ (section 4.5). But these arguments rely on the particular combinatorial structure of our special model. It is not clear how readily they can be extended to general models.

2. Analysis of optimal solutions.

2.1. Nonuniqueness. In the case $n = 2$, if $\xi_1 > 1$, then both $\{1\}$ and $\{2\}$ attain the maximum value 1 of the optimization problem (6): the optimizing set is not unique. Corollary 4 shows that, provided some $\xi_i + \xi_{i+1} < 1$ is less than 1, the optimizing set A_n^{opt} is unique, and by assumption (8) this proviso holds with probability $\to 1$ as $n \to \infty$. After this section we generally ignore the possibility of nonuniqueness.

We start with some terminology that will also be used later. For an integer interval $[g, d]$ with $d - g + 1$ even, the two *complementary alternating subsets* A_1, A_2 are as shown in Figure 2.

LEMMA 3. *Let* $n \geq 2$. *For almost all realizations of* $\xi_1, \ldots, \xi_{n-1}$*, the following are equivalent:*

(a) *The subset maximizing* (6) *is not unique.*

(b) n *is even and the only optimal solutions are the two complementary alternating subsets of* $[1, n]$.

(c) n *is even and* $M_n = n/2$.

Proof. Either of (b), (c) implies (a), so it is enough to show (a) implies (b) and (c). Suppose distinct subsets B_1 and B_2 attain the maximum. Then a.s. the values of ξ_i used in the optimal sum are identical, that is,

$$\{i : \ (i, i+1) \subset B_1\} = \{i : \ (i, i+1) \subset B_2\} := \mathcal{S}, \text{ say.} \tag{12}$$

First suppose $\mathcal{S}$ is empty. Then each of B_1 and B_2 has only isolated elements. But amongst such sets, the maximum of (6) is attained (for n odd) uniquely by the alternating subset giving $M_n = (n+1)/2$, or (for n even) only by the complementary alternating subsets. So $\mathcal{S}$ empty implies (b) and (c). For general $\mathcal{S}$, take some $i \in B_1 \triangle B_2$, and then take the maximal interval $i \in [g, d] \subset [1, n]$ which is disjoint from $\mathcal{S}$. Repeating the argument above, the restrictions of B_1 and B_2 to $[g, d]$ must be complementary alternating subsets. If $[g, d] \neq [1, n]$, then either $d + 1$ or $g - 1$ is in $\mathcal{S}$—say $d + 1$—and so $d + 1 \in B_1 \cap B_2$. But exactly one of B_1, B_2 contains d, contradicting (12). So $[g, d] = [1, n]$, and so $\mathcal{S}$ is empty. □

COROLLARY 4. *If* $\xi_i + \xi_{i+1} < 1$ *for some* $1 \leq i \leq n-2$*, then a.s.* A_n^{opt} *is unique.*

Proof. Fix i with $\xi_i + \xi_{i+1} < 1$ and let B be the alternating subset of $[1, n]$ containing i and $i+2$. Replacing B by $B \cup \{i+1\}$ increases the benefit by $1 - \xi_i - \xi_{i+1} > 0$, so B cannot be optimal, and the result follows from Lemma 3. □

2.2. Dynamic programming. Finding the maximum value and the maximizing subset of (6) is algorithmically easy by dynamic programming, as follows. Define

$$V_{n,i}^L = \max_{i \in A \subseteq \{1,\ldots,i-1,i\}} \left(|A| - \sum_{j=1}^{i-1} \xi_j \mathbb{1}(j \in A, j+1 \in A) \right), \tag{13}$$

$$W_{n,i}^L = \max_{i \notin A \subseteq \{1,\dots,i-1,i\}} \left(|A| - \sum_{j=1}^{i-1} \xi_j \mathbb{1}(j \in A, j+1 \in A) \right), \tag{14}$$

which differ in that the former requires $i \in A$ and the latter requires $i \notin A$. The superscripts L here and R later indicate *left* and *right*. Note that in fact $V_{n,i}^L, W_{n,i}^L$ above and $X_{n,i}^L$ below do not depend on n, but the notation is useful to distinguish from the limit process X_i^L later.

From (13), (14) we see $V_{n,1}^L = 1, W_{n,1}^L = 0$, and by induction over $1 \le i$

$$V_{n,i+1}^L = 1 + \max(V_{n,i}^L - \xi_i, W_{n,i}^L),$$

$$W_{n,i+1}^L = \max(V_{n,i}^L, W_{n,i}^L),$$

the two terms in the max indicating the choice of using or not using element i. Then $M_n = \max(V_{n,n}^L, W_{n,n}^L)$, and by examining which max term is used at each stage leading to M_n we can recover the optimizing subset A_n^{opt}.

We now describe an alternative, more useful way to obtain A_n^{opt}. First, consider the evolution rule for the process

$$X_{n,i}^L := V_{n,i}^L - W_{n,i}^L \tag{15}$$

as i increases; the rule is

$$\begin{aligned} X_{n,i+1}^L &= 1 + \max(0, X_{n,i}^L - \xi_i) - \max(0, X_{n,i}^L) \\ &= 1 + \max(-X_{n,i}^L, -\xi_i)\mathbb{1}(X_{n,i}^L \ge 0). \end{aligned} \tag{16}$$

One can check by induction that $0 \le X_{n,i}^L \le 1$ and thus rewrite the recursion as

$$X_{n,i+1}^L = \max(1 - X_{n,i}^L, 1 - \xi_i).$$

For n fixed we define the right processes analogously:

$$V_{n,i}^R = \max_{i \in A \subseteq \{i,i+1,\dots,n\}} \left(|A| - \sum_{j=i}^{n-1} \xi_j \mathbb{1}(j \in A, j+1 \in A) \right), \tag{17}$$

$$W_{n,i}^R = \max_{i \notin A \subseteq \{i,i+1,\dots,n\}} \left(|A| - \sum_{j=i}^{n-1} \xi_j \mathbb{1}(j \in A, j+1 \in A) \right), \tag{18}$$

with $V_{n,n}^R = 1, W_{n,n}^R = 0$. Observe that the evolution rule for the process

$$X_{n,i}^R := V_{n,i}^R - W_{n,i}^R \tag{19}$$

as i decreases does not depend on n. In fact, we have

$$X_{n,i-1}^R = \max(1 - X_{n,i}^R, 1 - \xi_{i-1}). \tag{20}$$

The point is that we can determine the optimizing random set A_n^{opt} in terms of the quantities above. Fix i, consider the quantities $(X_{n,i}^L, V_{n,i}^L, W_{n,i}^L)$, ξ_i, and $(X_{n,i+1}^R, V_{n,i+1}^R, W_{n,i+1}^R)$, and drop subscripts. We have four choices of whether to include

TABLE 2
Inclusion criteria for $i, i+1$ in A_n^{opt}.

$-i-(i+1)-$	Absolute benefit	Relative benefit	When used
$-\bullet--\bullet-$	$V^L+V^R-\xi$	$X^L+X^R-\xi$	$\xi<\min(X^L,X^R)$
$-\bullet--\circ-$	V^L+W^R	X^L	if $X^R<\min(X^L,\xi)$
$-\circ--\bullet-$	W^L+V^R	X^R	if $X^L<\min(X^R,\xi)$
$-\circ--\circ-$	W^L+W^R	0	never

(marked as $\bullet$ in Table 2) or exclude (marked as $\circ$ in Table 2) items i and $i+1$ in the optimal set A_n^{opt}. For each choice, the table shows the absolute benefit of that choice and then the relative benefit (relative to the choice to exclude both items). For each i the optimal A_n^{opt} will contain, in positions $(i, i+1)$, the combination with the largest relative benefit, and the final column indicates the criteria for use of each combination. (The case of nonuniqueness of A_n^{opt}, Lemma 3, is the case where X_i^L and X_i^R alternate between 0 and 1 throughout the interval $[1,n]$, and where we have equalities $X_i^L = X_{i+1}^R < \xi_i$. Outside this case, one of the three strict inequalities holds. We ignore the nonuniqueness possibility in the summary below.)

We summarize the argument above as follows.

LEMMA 5. *For each n define $X_{n,i}^L, 1\le i\le n$, and $X_{n,i}^R, 1\le i\le n$, by*

$$X_{n,1}^L = 1; \quad X_{n,i+1}^L = \max(1-X_{n,i}^L, 1-\xi_i), \quad 1\le i\le n-1, \tag{21}$$

$$X_{n,n}^R = 1; \quad X_{n,i-1}^R = \max(1-X_{n,i}^R, 1-\xi_{i-1}), \quad 2\le i\le n. \tag{22}$$

Then A_n^{opt} is the random subset of $\{1,2,\dots,n\}$ specified by the following: for each $1\le i\le n-1$,

$$\text{if } \xi_i < \min(X_{n,i}^L, X_{n,i+1}^R), \quad \text{then} \quad i\in A_n^{opt}, \quad i+1\in A_n^{opt},$$

$$\text{if } X_{n,i+1}^R < \min(X_{n,i}^L, \xi_i), \quad \text{then} \quad i\in A_n^{opt}, \quad i+1\notin A_n^{opt},$$

$$\text{if } X_{n,i}^L < \min(X_{n,i+1}^R, \xi_i), \quad \text{then} \quad i\notin A_n^{opt}, \quad i+1\in A_n^{opt}.$$

Let us emphasize two points:

- whether or not $i\in A_n^{\mathrm{opt}}$ depends only on the three random variables $X_{n,i}^L$, $\xi_i, X_{n,i+1}^R$;
- the only place where the value of n enters is as the boundary condition $X_{n,n}^R = 1$.

In the next section, we show how to define a corresponding stationary process

$$((X_i^L, \xi_i, X_{i+1}^R),\ -\infty < i < \infty).$$

By applying the specification in Lemma 5 to this process, we will define a set $A^{\mathrm{opt}}\subseteq\mathbb{Z}$ which will be shown (Lemma 8) to be the limit of A_n^{opt}. As a consequence, we will be able to derive the limit of M_n/n.

2.3. A stationary Markov chain and the infinite limit problem. The recursion (21) specifies a Markov chain on the continuous state space $[0,1]$ with transitions

$$x\to \max(1-x, 1-\xi), \tag{23}$$

where ξ has distribution function G. Write $F(x) = \mathrm{P}(X^L \le x)$ for a stationary distribution function for this chain. Then

$$\begin{aligned} F(x) &= \mathrm{P}(\max(1 - X^L, 1 - \xi) \le x) \\ &= \mathrm{P}(\min(X^L, \xi) > 1 - x) \\ &= \overline{G}(1-x)\overline{F}(1-x), \end{aligned}$$

where for any distribution function F we write $\overline{F}(x) = 1 - F(x)$. Iterating this identity once gives

$$F(x) = \overline{G}(1-x)\left(1 - \overline{G}(x)\overline{F}(x)\right),$$

and solving this equation gives

$$F(x) = \frac{\overline{G}(1-x)\overline{G}(x)}{1 - \overline{G}(x)\overline{G}(1-x)}. \tag{24}$$

The assumption (7) that G has a density implies that F has a density, so in what follows we do not need to distinguish carefully between weak and strict inequalities for random variables with these distributions.

Now consider the infinite line graph, with vertices $-\infty < i < \infty$ and with i.i.d. edge-costs ξ_i on edge $(i, i+1)$ such that $\mathrm{P}(\xi_0 + \xi_1 < 1) > 0$, which is ensured by the condition $\overline{G}(1/2) < 1$.

LEMMA 6. *The recursion*

$$X^L_{i+1} = \max(1 - X^L_i, 1 - \xi_i), \quad -\infty < i < \infty, \tag{25}$$

defines uniquely a joint distribution for $((\xi_i, X^L_i), -\infty < i < \infty)$ *in which* (X^L_i) *is the stationary Markov chain with transition kernel* (23) *and stationary distribution* (24). *And*

$$X^L_i = \phi(\ldots, \xi_{i-2}, \xi_{i-1}) \tag{26}$$

for a certain function ϕ *not depending on* i.

Proof. Having proved existence and uniqueness of the stationary distribution at (24), it remains only to prove the measurability property (26). Iterating (25) once shows

$$1 - \xi_i \le X^L_{i+1} \le \max(1 - \xi_i, \xi_{i-1}). \tag{27}$$

So outside the event $\{1 - \xi_i < \xi_{i-1}\}$ the value of X^L_{i+1} depends only on (ξ_{i-1}, ξ_i) and not on the value of X^L_i. So inductively on $Q \ge 1$ there exists a measurable function ϕ_Q such that

$$X^L_1 = \phi_Q(\xi_{-2Q-1}, \xi_{-2Q}, \ldots, \xi_0) \text{ outside } \cap_{q=-Q}^{0} \{1 - \xi_{2q} < \xi_{2q-1}\}.$$

Now (26) follows because $\mathrm{P}\left(\cap_{q=-Q}^{0}\{1 - \xi_{2q} < \xi_{2q-1}\}\right) = \left(\mathrm{P}(\xi_0 + \xi_1 > 1)\right)^{Q+1} \to 0$. □

If we define an "i decreasing" process by

$$X^R_i = \phi(\ldots, \xi_{i+2}, \xi_{i+1}, \xi_i), \tag{28}$$

then (X_i^R) satisfies the analogous recursion

$$X_i^R = \max(1 - X_{i+1}^R, 1 - \xi_i), \quad -\infty < i < \infty, \tag{29}$$

and is distributed as the same stationary Markov chain. Hence we have a rigorous definition of a unique (in distribution) stationary process $((X_i^L, \xi_i, X_{i+1}^R),\ -\infty < i < \infty)$ satisfying (25), (29) which we will call the *triple process*. Note that from (26), (28)

(30) for each i the three random variables X_i^L, ξ_i, X_{i+1}^R are independent.

LEMMA 7. *Let $(X_i^L, \xi_i, X_{i+1}^R),\ -\infty < i < \infty$, be the stationary triple process. Then there is a random subset A^{opt} of $\mathbb{Z}$ specified by the following: for each $-\infty < i < \infty$,*

$$\text{if } \xi_i < \min(X_i^L, X_{i+1}^R), \quad \text{then} \quad i \in A^{opt}, \quad i+1 \in A^{opt},$$

$$\text{if } X_{i+1}^R < \min(X_i^L, \xi_i), \quad \text{then} \quad i \in A^{opt}, \quad i+1 \notin A^{opt},$$

$$\text{if } X_i^L < \min(X_{i+1}^R, \xi_i), \quad \text{then} \quad i \notin A^{opt}, \quad i+1 \in A^{opt}.$$

Proof. We need only check that the definition of A^{opt} is consistent, in that the criterion for item i to be excluded should be the same whether we look at the pair $(i, i+1)$ or the pair $(i-1, i)$. (Of course this is intuitively clear from the consistency in the finite setting of Lemma 5, but let us give an algebraic verification anyway.) We need to check

$$\{X_i^L < \min(X_{i+1}^R, \xi_i)\} \stackrel{?}{=} \{X_i^R < \min(X_{i-1}^L, \xi_{i-1})\}.$$

Using the recursions (29), (25) for X_i^R and X_i^L, we need to check

$$\{\max(1-X_{i-1}^L, 1-\xi_{i-1}) < \min(X_{i+1}^R, \xi_i)\} \stackrel{?}{=} \{\max(1-X_{i+1}^R, 1-\xi_i) < \min(X_{i-1}^L, \xi_{i-1})\}.$$

But these are equal by applying the transformation $u \to 1 - u$ to the right-hand side. □

Because the rule defining A^{opt} is translation-invariant, the augmented triple process

$$((X_i^L, \xi_i, X_{i+1}^R, \mathbb{1}(i \in A^{\text{opt}})),\ -\infty < i < \infty)$$

is also stationary. The next lemma shows this process is the limit of the corresponding finite-n process. The mode of convergence can be viewed as a very elementary case of *local weak convergence* [4] of random graphical structures. In other words, it asserts that relative to a random time-origin the finite processes approximate the limit process.

LEMMA 8. *Let U_n be uniform on $\{1, \dots, n\}$. As $n \to \infty$*

$$((X_{n,U_n+i}^L, \xi_{U_n+i}, X_{n,U_n+i+1}^R, \mathbb{1}(U_n + i \in A_n^{opt})),\ -\infty < i < \infty)$$

$$\stackrel{d}{\to}\ ((X_i^L, \xi_i, X_{i+1}^R, \mathbb{1}(i \in A^{opt})),\ -\infty < i < \infty),$$

where the left-hand side is defined arbitrarily for $U_n + i \notin \{1, \dots, n\}$ and where convergence in distribution is with respect to the usual product topology on infinite sequence space.

Proof. Because the X's are bounded and the ξ's are i.i.d., the sequence of processes is tight in the product topology. Write

$$((\hat{X}_i^L, \hat{\xi}_i, \hat{X}_{i+1}^R, \mathbb{1}(i \in \hat{A}^{\rm opt})),\ -\infty < i < \infty)$$

for a subsequential weak limit. Clearly $(\hat{\xi}_i) \stackrel{d}{=} (\xi_i)$. Because for each n the process $(X_{n,i}^L, \xi_i)$ satisfies recursion (21), the limit $(\hat{X}_i^L, \hat{\xi}_i)$ satisfies this recursion, and so by the "uniqueness of joint distribution" assertion of Lemma 6, $(\hat{X}_i^L, \hat{\xi}_i) \stackrel{d}{=} (X_i^L, \xi_i)$. Applying the same argument to X^R we deduce

$$((X_{U_n+i}^L, \xi_{U_n+i}, X_{n,U_n+i+1}^R),\ -\infty < i < \infty) \stackrel{d}{\to} ((X_i^L, \xi_i, X_{i+1}^R),\ -\infty < i < \infty).$$

For fixed i_0 the event $i_0 \in A^{\rm opt}$ is a function of the limit process, the function implied by Lemma 7, and by a standard fact [6, Theorem 5.2] it is enough to check that this function is a.s. continuous with respect to the limit process. But this just requires that the probability of an *equality* between some two of $X_{i_0}^L, \xi_{i_0}, X_{i_0+1}^R$ should be zero, which follows from their independence (30) and existence of densities (7), (24). □

2.4. Proof of Theorem 1. Because

$$M_n = \sum_{i=1}^{n} \mathbb{1}(i \in A_n^{\rm opt}) - \sum_{i=1}^{n-1} \xi_i \mathbb{1}(i \in A_n^{\rm opt}, i+1 \in A_n^{\rm opt}),$$

we can write

$$n^{-1}\mathrm{E}M_n = \mathrm{P}(U_n \in A_n^{\rm opt}) - \mathrm{E}\xi_{U_n}\mathbb{1}(U_n \in A_n^{\rm opt}, U_n + 1 \in A_n^{\rm opt})\mathbb{1}(U_n \neq n)$$

and then by Lemma 8

$$n^{-1}\mathrm{E}M_n \to c := \mathrm{P}(0 \in A^{\rm opt}) - \mathrm{E}\xi_0 \mathbb{1}(0 \in A^{\rm opt}, 1 \in A^{\rm opt}).$$

Note that clearly $c \leq 1$; the other inequality $c \geq 1/2$ holds because the subset $\{1, 3, 5, \ldots\}$ is a feasible choice.

We now exploit the *method of bounded differences* [12] in a very routine way. We observe that $M_n = m_n(\xi_1, \ldots, \xi_n)$ for a certain function m_n with the property

> changing any one argument of $m_n(z_1, \ldots, z_n)$ changes the value of $m_n(\cdot)$ by at most 1

This property holds because $A_n^{\rm opt}$ will never contain a pair $(i, i+1)$ for which $\xi_i > 1$. And this property implies the well-known Azuma–Hoeffding inequality of the form (see, e.g., [16])

$$\mathrm{P}(|M_n - \mathrm{median}(M_n)| \geq t) \leq 4\exp(-\tfrac{t^2}{4n}).$$

It is now routine to use this large deviation inequality to establish the a.s. and L^1 convergence of $n^{-1}M_n$ to c.

To evaluate c, abbreviate (X_0^L, ξ_0, X_1^R) to (X^L, ξ, X^R) and use the Lemma 7 definition of $A^{\rm opt}$ to write

$$\begin{aligned}
\mathrm{P}(0 \in A^{\rm opt}) &= 1 - \mathrm{P}(X^L < \min(X^R, \xi)) \\
&= 1 - \tfrac{1}{2}(1 - \mathrm{P}(\xi < \min(X^L, X^R))) \text{ by symmetry} \\
&= \tfrac{1}{2} + \tfrac{1}{2}\mathrm{P}(\xi < \min(X^L, X^R))
\end{aligned}$$

and then

$$c = \tfrac{1}{2} + \tfrac{1}{2}\mathrm{P}(\xi < \min(X^L, X^R)) - \mathrm{E}\xi\mathbb{1}(\xi < \min(X^L, X^R)). \tag{31}$$

Recall that X^L, ξ, and X^R are independent and that X^L and X^R have common distribution F given in terms of G by (24). So (31) constitutes a formula for c in terms of the underlying distribution function G of ξ.

We now evaluate c in the special case where ξ has the exponential(λ) distribution:

$$\overline{G}(x) = e^{-\lambda x}, \quad 0 < x < \infty,$$

so that, from formula (24), we have

$$F(x) = \frac{e^{-\lambda(1-x)}(1 - e^{-\lambda x})}{1 - e^{-\lambda}} = \frac{e^{\lambda x} - 1}{e^{\lambda} - 1}.$$

We deduce

$$\begin{aligned}
\mathrm{P}(\xi < \min(X^L, X^R)) &= \int_0^1 \lambda e^{-\lambda u} P^2(X^L > u)\, du \\
&= \frac{\lambda}{(e^\lambda - 1)^2} \int_0^1 e^{-\lambda u} \left(e^\lambda - e^{\lambda u}\right)^2 du \\
&= \frac{\lambda}{(e^\lambda - 1)^2} \left(e^{2\lambda} \int_0^1 e^{-\lambda u} du - 2e^\lambda + \int_0^1 e^{\lambda u} du \right) \\
&= \frac{e^{2\lambda} - 2\lambda e^\lambda - 1}{(e^\lambda - 1)^2},
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}\xi\mathbb{1}(\xi < \min(X^L, X^R)) &= \int_0^1 u\lambda e^{-\lambda u} P^2(X^L > u)\, du \\
&= \frac{\lambda}{(e^\lambda - 1)^2} \int_0^1 u e^{-\lambda u} \left(e^\lambda - e^{\lambda u}\right)^2 du \\
&= \frac{\lambda}{(e^\lambda - 1)^2} \left(e^{2\lambda} \int_0^1 u e^{-\lambda u} du - e^\lambda + \int_0^1 u e^{\lambda u} du \right) \\
&= \frac{1}{\lambda(e^\lambda - 1)^2} \left(e^{2\lambda} - (\lambda^2 + 2)e^\lambda + 1 \right).
\end{aligned}$$

Combining,

$$\begin{aligned}
c &= \frac{1}{2} + \frac{\frac{\lambda}{2}(e^{2\lambda} - 2\lambda e^\lambda - 1) - (e^{2\lambda} - (\lambda^2 + 2)e^\lambda + 1)}{\lambda(e^\lambda - 1)^2} \\
&= \frac{1}{1 - e^{-\lambda}} - \frac{1}{\lambda}.
\end{aligned}$$

3. The upper bound in Theorem 2. Local weak convergence (Lemma 8 above and Lemma 11 below) provides one sense in which the $n \to \infty$ limit of the solution $A_n^{\rm opt}$ of the size-n optimization problem is $A^{\rm opt}$. A logically different sense is provided by coupling, as follows. Part of the stationary triple process is the doubly infinite i.i.d. sequence $(\ldots, \xi_{-1}, \xi_0, \xi_1, \xi_2, \ldots)$. For each n use these same random variables $\xi_1, \ldots, \xi_n$ to construct $A_n^{\rm opt}$. Because of boundary effects it is not always true that $A^{\rm opt} \cap [1, n] = A_n^{\rm opt}$. But we expect the sets to coincide "away from the boundary," and Lemma 9(b) below provides one expression of this equality. We call this technique *localization*.

3.1. Optimality properties of $A^{\rm opt}$. Lemma 7 gave a concise definition of $A^{\rm opt}$ but did not explicitly identify its optimality properties. Lemma 9 below will relate $A^{\rm opt}$ to certain finite optima and thereby allow us to deduce some explicit properties.

The benefit function $f_n(A)$ and its maximum value M_n defined at (5), (6) refer to subsets of $[1, n]$, and it is convenient to make the corresponding definitions for an arbitrary interval $[\ell, m]$:

$$f_{[\ell,m]}(A) := |A| - \sum_{i=\ell}^{m-1} \xi_i \mathbb{1}(i \in A, i+1 \in A), \quad A \subseteq \{\ell, \ell+1, \ldots, m\}, \tag{32}$$

$$M_{[\ell,m]} := \max_{A \subseteq \{\ell, \ell+1, \ldots, m\}} f_{[\ell,m]}(A), \tag{33}$$

and denote by $A_{[\ell,m]}^{\rm opt}$ the corresponding optimizing set.

LEMMA 9. (a) *If $\xi_{i-1} + \xi_i \le 1$, then $i \in A^{opt}$.*

(b) *If $\ell < m$ and $\xi_{\ell-1} + \xi_\ell \le 1$ and $\xi_{m-1} + \xi_m \le 1$, then $A_{[\ell,m]}^{opt}$ is unique and*

$$A^{opt} \cap [\ell, m] = A_{[\ell,m]}^{opt}. \tag{34}$$

If, furthermore, $[\ell, m] \subseteq [1, n]$, then $A_n^{opt} \cap [\ell, m] = A_{[\ell,m]}^{opt}$ (interpreting $\xi_0 = 0$ if $\ell = 1$).

(c) *If both $i, i+1 \in A^{opt}$, then $\xi_i \le 1$.*

(d) *If $\xi_i + \xi_{i+1} > 1$, then $i, i+1$ and $i+2$ together cannot belong to A^{opt}.*

(e) *Let $k \ge 2$. If $[g, g+2k-1]$ is an interval such that $\xi_g > \xi_{g+1} > \cdots > \xi_{g+2k-1} > \xi_{g+2k}$ and*

$$\xi_j + \xi_{j+1} > 1, \quad g \le j \le g + 2k - 2,$$

then $A^{opt} \cap [g, g+2k-1]$ must be one of the two complementary alternating sequences in $[g, g+2k-1]$.

Proof. (a) If $\xi_{i-1} + \xi_i \le 1$, then $X_i^L \ge 1 - \xi_{i-1} \ge \xi_i$, and hence from the Lemma 7 definition we see that for any possible value of X_{i+1}^R, we have $i \in A^{\rm opt}$.

(b) First note that both ℓ and m are in $A_{[\ell,m]}^{\rm opt}$; otherwise adding each element would increase $f_{[\ell,m]}(A_{[\ell,m]}^{\rm opt})$ by at least $1 - \xi_\ell$ and $1 - \xi_{m-1}$, respectively. Next note that by rewriting Lemma 5 (which concerns the special case $[\ell, m] = [1, n]$) for general $[\ell, m]$, we have a construction of $A_{[\ell,m]}^{\rm opt}$ in terms of processes $X_{[\ell,m],i}^L$ and $X_{[\ell,m],i}^R$ for $\ell \le i \le m$ defined by the recursions analogous to (21), (22). By (a) both ℓ and m are in $A^{\rm opt}$. We have now shown $X_{[\ell,m],\ell}^L = 1 - \xi_{\ell-1} = X_\ell^L$ and $X_{[\ell,m],m-1}^R = 1 - \xi_{m-1} = X_{m-1}^R$; because the restricted and unrestricted processes have the same boundary conditions and satisfy the same recursions over $[l, m]$ they must agree throughout the interval. Finally, because both endpoints ℓ and m are in $A_{[\ell,m]}^{\rm opt}$

it cannot fit the "complementary alternating sequences" criteria for nonuniqueness (Lemma 3). The same argument works for A_n^{opt}.

One could prove (c), (d), (e) algebraically from the definition of A^{opt}, but it is more intuitive to exploit the finite optimality criterion as follows. From assumption (8) there are infinitely many ℓ with $\xi_{\ell-1}+\xi_\ell \le 1$, and so for any given i there is a (random) long interval $[\ell, m]$ containing i for which by (b) $A^{\mathrm{opt}} \cap [\ell, m] = A^{\mathrm{opt}}_{[\ell,m]}$. In other words, the restriction of A^{opt} to $[\ell, m]$ is the solution of the finite optimization problem (32), and we can derive its properties by considering the effect of local changes. Now (c) and (d) follow from the following observations:

(for (c)): if $i, i+1$ are in A^{opt}, then removing $i+1$ will give a relative benefit of at least $\xi_i - 1$.

(for (d)): if $i, i+1, i+2$ are all in A^{opt}, then removing $i+1$ will give a relative benefit of $\xi_i + \xi_{i+1} - 1$.

For (e), consider $j \in [g, g+2k]$. By (d) we cannot have $\{j, j+1, j+2\} \subset A^{\mathrm{opt}}$. If j and $j+1$ but not $j+2$ are in A^{opt}, then deleting $j+1$ while adding $j+2$ would increase the benefit by at least $\xi_j - \xi_{j+2} > 0$, which is impossible. It follows that we cannot have $\{j, j+1\} \subset A^{\mathrm{opt}}$. Thus $A^{\mathrm{opt}} \cap [g, g+2k-1]$ contains only isolated elements. It is now easy to check that one can change $A^{\mathrm{opt}} \cap [g, g+2k-1]$ into one of the alternating sequences on $[g, g+2k-1]$ in such a way that the cardinality does not decrease, and the end items $g, g+2k-1$ change (if at all) only from included to excluded. Thus the change can only increase the benefit; appealing to the uniqueness property (b) in a larger interval establishes (e). □

3.2. Proof of upper bound. In this section we prove the bound

$$\limsup_{\delta \downarrow 0} \delta^{-2} \limsup_n \mathrm{E}\varepsilon_n(\delta) < \infty \tag{35}$$

via a simple construction of near-optimal sets. We first describe a particular configuration. Let $g, d \in \mathbb{Z}$ such that $d - g = 2k$ for some $k \ge 2$, and consider the sets A and B below:

	g	$-$	$(g+1)$	$-$							$(g+2k-1)$	$-$	$g+2k$
A	$\bullet$	$-$	$\circ$	$-$	$\bullet$	$-$	$\circ$	$\dots$	$\bullet$	$-$	$\circ$	$-$	$\bullet$
B	$\bullet$	$-$	$\bullet$	$-$	$\circ$	$-$	$\bullet$	$\dots$	$\circ$	$-$	$\bullet$	$-$	$\bullet$

where $|A \triangle B| = 2k-1$ and the difference between the benefits of A and B is

$$\begin{aligned} f_{[g,g+2k]}(A) - f_{[g,g+2k]}(B) &= (k+1) - ((k+2) - \xi_g - \xi_{g+2k-1}) \\ &= \xi_g + \xi_{g+2k-1} - 1. \end{aligned} \tag{36}$$

Now fix $k \ge 2$ and $\alpha > 0$ such that $\alpha k < 1/2$. Consider the event Ω_g defined by

$$\xi_g > \xi_{g+1} > \xi_{g+2} > \cdots > \xi_{g+2k-2} > \xi_{g+2k-1} > \tfrac{1}{2} > \xi_{g+2k};$$

$$\xi_{g-1} + \xi_g < 1; \quad \xi_{g+2k-1} + \xi_{g+2k} < 1.$$

By assumption (7) this event has nonzero probability. If this event occurs, Lemma 9(a) shows that A^{opt} contains g and $g+2k$, and then Lemma 9(e) implies that $A^{\mathrm{opt}} \cap [g, g+2k]$ is the set A above. By applying Lemma 9(b) to $[\ell, m] = [g, g+2k]$ we have the analogue in the finite n setting: if Ω_g occurs for $[g, g+2k] \subset [1, n]$, then

$A_n^{\text{opt}} \cap [g, g+2k]$ is the set A above. So if we change A_n^{opt} by replacing pattern A by pattern B on such an interval, then from (36) the decrease in benefit equals $\xi_g + \xi_{g+2k-1} - 1 > 0$. Now define

$$\Omega_g^{(\alpha)} = \Omega_g \cap \{1 < \xi_g + \xi_{g+2k-1} < 1 + 2k\alpha\},$$

$$q(\alpha) = \mathrm{P}(\Omega_g^{(\alpha)}),$$

$$r(\alpha) = \mathrm{E}(\xi_g + \xi_{g+2k-1} - 1)\mathbb{1}(\Omega_g^{(\alpha)})$$

so that $r(\alpha)$ is the unconditional mean increase in benefit from the possible change, now performed only if event $\Omega_g^{(\alpha)}$ happens. Using assumption (7) we see that $(\xi_g + \xi_{g+2k-1})$ restricted to $\Omega_g^{(\alpha)}$ has a continuous density which is nonzero at 1, which easily implies that for fixed k

$$q(\alpha) \sim \bar{q}\alpha, \quad r(\alpha) \sim \bar{r}\alpha^2 \text{ as } \alpha \downarrow 0 \tag{37}$$

for constants $\bar{q}, \bar{r} \in (0, \infty)$.

Given n and the optimal set A_n^{opt}, construct a near-optimal set $B_n^{(\alpha)}$ as follows. Let $g_1 = 1$ and let

$$[g_1, g_1 + 2k], \quad [g_2, g_2 + 2k], \quad [g_3, g_3 + 2k], \ldots, [g_{j_n}, g_{j_n} + 2k]$$

be the adjacent disjoint intervals in $[1, n]$ containing $2k + 1$ integers. For each such $g = g_j$, if event $\Omega_g^{(\alpha)}$ occurs, then on $[g, g + 2k]$ replace pattern A by pattern B.

Letting $n \to \infty$ and using the weak law of large numbers, we get

$$\tfrac{1}{n}|B_n^{(\alpha)} \triangle A_n^{\text{opt}}| \to 2kq(\alpha)/(2k+1) \text{ in probability,}$$

$$\tfrac{1}{n}(f_n(A_n^{\text{opt}}) - f_n(B_n^{(\alpha)})) \to r(\alpha)/(2k+1) \text{ in probability.}$$

If $\frac{1}{n}|B_n^{(\alpha)} \triangle A_n^{\text{opt}}| \le kq(\alpha)/(2k+1)$, then redefine $B_n^{(\alpha)}$ to be the empty set. Then (taking $k = 3$ for concreteness)

$$\tfrac{1}{n}|B_n^{(\alpha)} \triangle A_n^{\text{opt}}| \ge 3q(\alpha)/7,$$

$$\lim_n \tfrac{1}{n}(\mathrm{E}f_n(A_n^{\text{opt}}) - \mathrm{E}f_n(B_n^{(\alpha)})) = r(\alpha)/7.$$

The upper bound (35) now follows from the $\alpha \to 0$ asymptotics (37).

4. Proof of Theorem 2: The lower bound.

4.1. Analysis of near-optimal solutions: The quintuple process. Throughout section 4 we fix a constant $\tau > 0$ such that

$$G\left(\tfrac{1}{2} - \tau\right) > 0. \tag{38}$$

Such a constant exists by assumption (7). To study near-optimal solutions, fix a Lagrange multiplier θ such that

$$0 < \theta < \tau. \tag{39}$$

We will derive the existence of, and derive an exact expression for, the function $\bar{\varepsilon}(\delta) = \lim_n \mathrm{E}\varepsilon_n(\delta)$ when δ is sufficiently small. The expression is an implicit function

representation $\bar{\varepsilon}(\delta(\theta)) = \varepsilon(\theta)$ via two functions $\varepsilon(\theta), \delta(\theta)$ defined (49), (50) in terms of the stationary distribution of a certain *quintuple process*.

We study the modified optimization problem in which we get an extra reward θ for choosing an item which is not in A_n^{opt} or for not choosing an item which is in A_n^{opt}:

$$\max_{A\subseteq[n]} \left(|A| - \sum_{i=1}^{n} \xi_i \mathbb{1}(i \in A, i+1 \in A) + \theta |A \triangle A_n^{\mathrm{opt}}| \right). \tag{40}$$

To study this we modify (13), (14) to

$$\widetilde{V}_{n,i}^L = \max_{i\in A\subseteq\{1,2,\ldots,i\}} \left(|A| - \sum_{j=1}^{i-1} \xi_j \mathbb{1}(j, j+1 \in A) + \theta \, |(A \triangle A_n^{\mathrm{opt}}) \cap \{1,2,\ldots,i\}| \right), \tag{41}$$

$$\widetilde{W}_{n,i}^L = \max_{i\notin A\subseteq\{1,2,\ldots,i\}} \left(|A| - \sum_{j=1}^{i-1} \xi_j \mathbb{1}(j, j+1 \in A) + \theta \, |(A \triangle A_n^{\mathrm{opt}}) \cap \{1,2,\ldots,i\}| \right). \tag{42}$$

We also define $\widetilde{M}_n = \max(\widetilde{V}_{n,n}^L, \widetilde{W}_{n,n}^L)$ and write B_n^{opt} for the corresponding optimizing set. Note that these quantities depend on θ. Analogous to the definition (15) of $X_{n,i}^L$ we define

$$Z_{n,i}^L := \widetilde{V}_{n,i}^L - \widetilde{W}_{n,i}^L.$$

Then as the analogue of (16) we can obtain the recursion

$$Z_{n,i+1}^L = 1 - \min(Z_{n,i}^L, \xi_i)\mathbb{1}(Z_{n,i}^L > 0) + \theta J_{n,i+1},$$

where

$$Z_{n,1}^L = 1 + \theta J_{n,1},$$

$$J_{n,i} = \mathbb{1}(i \notin A_n^{\mathrm{opt}}) - \mathbb{1}(i \in A_n^{\mathrm{opt}}).$$

Recall from section 2.3 the stationary triple process $((X_i^L, \xi_i, X_{i+1}^R), -\infty < i < \infty)$ and define

$$J_i = \mathbb{1}(i \notin A^{\mathrm{opt}}) - \mathbb{1}(i \in A^{\mathrm{opt}}).$$

Just as the stationary triple process is interpretable (Lemma 8) as an $n \to \infty$ limit of the process $(X_{n,i}^L, \xi_i, X_{n,i+1}^R)$, we want to define a process which will be the limit of $(Z_{n,i}^L, X_{n,i}^L, \xi_i, X_{n,i+1}^R)$. So define a *quadruple process* $(Z_i^L, X_i^L, \xi_i, X_{i+1}^R)$ to be a process such that

(i) $(X_i^L, \xi_i, X_{i+1}^R)$ evolves as the triple process,

(ii) Z_i^L satisfies the recursion

$$Z_{i+1}^L = 1 - \min(Z_i^L, \xi_i)\mathbb{1}(Z_i^L > 0) + \theta J_{i+1}. \tag{43}$$

Recall $0 < \theta < \tau$.

LEMMA 10. *The quadruple process* $((Z_i^L, X_i^L, \xi_i, X_{i+1}^R), -\infty < i < \infty)$ *has a unique stationary distribution, for which*

$$Z_i^L = \psi(\dots, \xi_{i-2}, \xi_{i-1}, \xi_i, X_{i+1}^R) \tag{44}$$

for a certain function ψ *not depending on* i. *On the event* $\{\xi_{i-1} + \xi_i \le 1 - \tau\}$, *we have*

$$X_{i+1}^L = 1 - \xi_i, Z_{i+1}^L = 1 - \xi_i + \theta J_{i+1}. \tag{45}$$

Proof. Recursion (43) implies $Z_{i+1}^L \ge 1 - \xi_i + \theta J_{i+1}$. Thus iterating once (43) and using this last inequality, we obtain

$$1 - \xi_i + \theta J_{i+1} \le Z_{i+1}^L \le 1 - \min(1 - \xi_{i-1} + \theta J_i, \xi_i)\mathbb{1}(1 - \xi_{i-1} + \theta J_i > 0) + \theta J_{i+1}.$$

Thus, on the event $\{\xi_{i-1} + \xi_i \le 1 - \theta\}$ we have $Z_{i+1}^L = 1 - \xi_i + \theta J_{i+1}$ and also, by (27), we have $X_{i+1}^L = 1 - \xi_i$, establishing (45). Assumption (7) implies that the event $\{\xi_{i-1} + \xi_i \le 1 - \tau\}$ occurs for infinitely many $i < 0$, so in particular $K := \max\{i < 0 : \xi_{i-1} + \xi_i \le 1 - \tau\}$ is finite. By the recursion (43) we can write Z_0^L in the form

$$Z_0^L = \psi^1(\xi_{K+1}, \xi_{K+2}, \dots, \xi_{-1}; Z_{K+1}^L; J_{K+2}, J_{K+3}, \dots, J_0)$$

for some function ψ^1. Then by (45) with $Z_i^L = Z_{K+1}^L$ we can rewrite as

$$Z_0^L = \psi^2(\xi_K, \xi_{K+1}, \xi_{K+2}, \dots, \xi_{-1}; J_{K+1}, J_{K+2}, J_{K+3}, \dots, J_0).$$

By the definition of A^{opt}, each J_i is a function of X_i^L, ξ_i, X_{i+1}^R, and then from the recursions for X_i^L and X_i^R

$$Z_0^L = \psi^3(\xi_K, \xi_{K+1}, \xi_{K+2}, \dots, \xi_0; X_{K+1}^L, X_1^R).$$

By (45) with $X_i^L = X_{K+1}^L$ this is of the form

$$Z_0^L = \psi(\dots, \xi_{-2}, \xi_{-1}, \xi_0, X_1^R).$$

Now (44) defines a stationary version of the quadruple process. □

Just as $X_{n,i}^R$ was the "looking right" analogue of the "looking left" process $X_{n,i}^L$, we can define a "looking right" process $Z_{n,i}^R$ analogous to $Z_{n,i}^L$ as follows. Define

$$\widetilde{V}_{n,i}^R = \max_{i \in A \subseteq \{i, i+1, \dots, n\}} \left(|A| - \sum_{j=i}^{n-1} \xi_j \mathbb{1}(j, j+1 \in A) + \theta \, |(A \triangle A^{\text{opt}}) \cap \{i, i+1, \dots, n\}| \right), \tag{46}$$

$$\widetilde{W}_{n,i}^R = \max_{i \notin A \subseteq \{i, i+1, \dots, n\}} \left(|A| - \sum_{j=i}^{n-1} \xi_j \mathbb{1}(j, j+1 \in A) + \theta \, |(A \triangle A^{\text{opt}}) \cap \{i, i+1, \dots, n\}| \right). \tag{47}$$

Then the difference $Z_{n,i}^R = \widetilde{V}_{n,i}^R - \widetilde{W}_{n,i}^R$ satisfies the recursion

$$Z_{n,i}^R = 1 - \min(Z_{n,i+1}^R, \xi_i)\mathbb{1}(Z_{n,i+1}^R > 0) + \theta J_{n,i}; \quad Z_{n,n}^R = 1 + \theta J_{n,n}.$$

TABLE 3
Inclusion criteria for $i, i+1$ in B_n^{opt}.

$-i-(i+1)-$	Absolute benefit	Relative benefit	When used
$-\bullet--\bullet-$	$\widetilde{V}^L + \widetilde{V}^R - \xi$ $+\theta(\mathbb{1}_{i\notin A^{\rm opt}} + \mathbb{1}_{i+1\notin A^{\rm opt}})$	$Z^L + Z^R - \xi$ $-\theta(J_i + J_{i+1})$	$\xi < \min(Z^L - \theta J_i,$ $Z^R - \theta J_{i+1})$
$-\bullet--\circ-$	$\widetilde{V}^L + \widetilde{W}^R$ $+\theta(\mathbb{1}_{i\notin A^{\rm opt}} + \mathbb{1}_{i+1\in A^{\rm opt}})$	$Z^L - \theta J_i$	$(Z^R - \theta J_{i+1})^+$ $< \min(Z^L - \theta J_i, \xi)$
$-\circ--\bullet-$	$\widetilde{W}^L + \widetilde{V}^R$ $+\theta(\mathbb{1}_{i\in A^{\rm opt}} + \mathbb{1}_{i+1\notin A^{\rm opt}})$	$Z^R - \theta J_{i+1}$	$(Z^L - \theta J_i)^+$ $< \min(Z^R - \theta J_{i+1}, \xi)$
$-\circ--\circ-$	$\widetilde{W}^L + \widetilde{W}^R$ $+\theta(\mathbb{1}_{i\in A^{\rm opt}} + \mathbb{1}_{i+1\in A^{\rm opt}})$	0	otherwise

Recall that $B_n^{\rm opt}$ attains $\max_{A\subseteq\{1,\dots,n\}}(|A| - \sum_{i=1}^{n-1}\xi_i \mathbb{1}(i\in A, i+1\in A) + \theta|A \triangle A_n^{\rm opt}|)$. As in section 2.2, we can write down the benefits of each of the four possible choices for including or excluding items i and $i+1$, and thereby obtain criteria for which combination is used in $B_n^{\rm opt}$. See Table 3, in which $(Z_{n,i}^L, \xi_i, Z_{n,i+1}^R)$ is abbreviated to (Z^L, ξ, Z^R) and the n subscript is dropped. It should now be clear that the stationary quadruple process can be extended to a *stationary quintuple process*

$$(Z_i^L, X_i^L, \xi_i, X_{i+1}^R, Z_{i+1}^R), \quad -\infty < i < \infty,$$

in which Z^R satisfies the recursion

$$Z_i^R = 1 - \min(Z_{i+1}^R, \xi_i)\mathbb{1}(Z_{i+1}^R > 0) + \theta J_i, \quad -\infty < i < \infty,$$

satisfied by $Z_{n,i}^R$. By "reflection symmetry" between Z^R and Z^L, the functional relationship (44) holds for Z^R in reflected form with the same function ψ:

$$Z_i^R = \psi(\dots, \xi_{i+1}, \xi_i, \xi_{i-1}, X_{i-1}^L). \tag{48}$$

We can now use the stationary quintuple process to define a random subset $B^{\rm opt} \subset \mathbb{Z}$ by specifying that, for each pair $(i, i+1)$, we use the one of the four choices which has the largest relative benefit in Table 3. Analogously to Lemma 7 one can check that this definition is consistent. The local weak convergence property (Lemma 8) extends to the present setting as follows.

LEMMA 11. *Let U_n be uniform on $\{1,\dots,n\}$. As $n\to\infty$*

$$\begin{aligned}&((Z_{n,U_n+i}^L, X_{n,U_n+i}^L, \xi_{U_n+i}, X_{n,U_n+i+1}^R, Z_{n,U_n+i+1}^R,\\ &\qquad \mathbb{1}(U_n+i\in A_n^{opt}), \mathbb{1}(U_n+i\in B_n^{opt})))_{-\infty<i<\infty}\\ &\xrightarrow{d} ((Z_i^L, X_i^L, \xi_i, X_{i+1}^R, Z_{i+1}^R, \mathbb{1}(i\in A^{opt}), \mathbb{1}(i\in B^{opt})))_{-\infty<i<\infty}.\end{aligned}$$

Proof. The proof repeats the proof of Lemma 8, using (44), (48) to incorporate the (Z^L, Z^R) terms. In order to incorporate the $B^{\rm opt}$ component, we need to check that the function $\mathbb{1}(0\in B^{\rm opt})$ is a.s. continuous with respect to the stationary distribution of $(Z_0^L, X_0^L, \xi_0, X_1^R, Z_1^R)$. From Table 3, we get that $\{0\in B^{\rm opt}\} = \{Z_0^L - \theta J_0 > \min(\xi_0, \max(Z_1^R - \theta J_1, 0)\}$. Hence, it requires that the probability of an equality between some of two $Z_0^L - \theta J_0, \xi_0, Z_1^R - \theta J_1$ is zero. We check only that $\mathrm{P}(Z_0^L - \theta J_0 = \xi_0) = 0$. The recursion satisfied by Z_0^L reads $Z_0^L - \theta J_0 =$

$1 - \min(Z_{-1}^L, \xi_{-1})\mathbb{1}(Z_{-1}^L > 0)$. Thus, arguing as in the proof of Lemma 10, $Z_0^L - \theta J_0$ is a function of $(Z_{K+1}^L, \xi_{K+1}, \ldots, \xi_{-1}, J_{K+1}, J_{K+2}, \ldots, J_{-1})$ with $K = \max\{i < 0 : \xi_{i-1} + \xi_i \le 1 - \tau\}$. Since $Z_{K+1}^L = 1 - \xi_K + \theta J_{K+1}$ and $J_i \in \{-1, 1\}$, we deduce by recursion that there exists a pair of integers (i_0, n) with $K \le i_0 \le -1$ and $-K \le n \le K$ such that $Z_0^L \in \{1 - \xi_{i_0} + n\theta, \xi_{i_0} + n\theta\}$. The independence of ξ_i and ξ_0 for $i < 0$ and assumption (7) imply that $\mathrm{P}(Z_0^L - \theta J_0 = \xi_0) = 0$. □

Now define

$$\delta(\theta) = \mathrm{P}(\{0 \in A^{\mathrm{opt}}\} \triangle \{0 \in B^{\mathrm{opt}}\}), \tag{49}$$

$$\begin{aligned}\varepsilon(\theta) = {} & \mathrm{P}(0 \in A^{\mathrm{opt}}) - \mathrm{E}\xi_0 \mathbb{1}(0 \in A^{\mathrm{opt}}, 1 \in A^{\mathrm{opt}}) - \mathrm{P}(0 \in B^{\mathrm{opt}}) \\ & + \mathrm{E}\xi_0 \mathbb{1}(0 \in B^{\mathrm{opt}}, 1 \in B^{\mathrm{opt}}).\end{aligned} \tag{50}$$

So $\delta(\theta)$ is the proportion of items at which A^{opt} and B^{opt} differ, and $\varepsilon(\theta)$ is the difference in mean benefit per item between A^{opt} and B^{opt}. By Lemma 11,

$$\begin{aligned}\frac{1}{n}\mathrm{E}|A_n^{\mathrm{opt}} \triangle B_n^{\mathrm{opt}}| &= \mathrm{E}|\mathbb{1}(U_n \in A_n^{\mathrm{opt}}) - \mathbb{1}(U_n \in B_n^{\mathrm{opt}})| \\ &\to \mathrm{P}(\{0 \in A^{\mathrm{opt}}\} \triangle \{0 \in B^{\mathrm{opt}}\}) = \delta(\theta),\end{aligned} \tag{51}$$

and similarly the mean benefits satisfy

$$n^{-1}(\mathrm{E}f_n(A_n^{\mathrm{opt}})) - \mathrm{E}f_n(B_n^{\mathrm{opt}})) \to \varepsilon(\theta). \tag{52}$$

PROPOSITION 12. *Let $\widetilde{M}_n = f_n(B_n^{opt})$ be the benefit associated with B_n^{opt}; then a.s. and in L^1*

$$\lim_{n\to\infty} n^{-1}|B_n^{opt} \triangle A_n^{opt}| = \delta(\theta), \tag{53}$$

$$\lim_{n\to\infty} n^{-1}(M_n - \widetilde{M}_n) = \varepsilon(\theta). \tag{54}$$

Moreover for any choice B_n' satisfying (53) *in L^1, the associated benefit $M_n' = f_n(B_n')$ satisfies*

$$\liminf_n n^{-1}\mathrm{E}(M_n - M_n') \ge \varepsilon(\theta).$$

Proof. The convergence assertions (53), (54) follow from (51), (52) and the same concentration argument used in the proof of Theorem 1; we will not repeat the details. By construction, for any B_n' the associated reward M_n' satisfies

$$M_n' + \theta|B_n' \triangle A_n^{\mathrm{opt}}| \le \widetilde{M}_n + \theta|B_n^{\mathrm{opt}} \triangle A_n^{\mathrm{opt}}|.$$

Then because both (B_n^{opt}) and (B_n') satisfy (53), we see that

$$\mathrm{E}M_n' \le \mathrm{E}\widetilde{M}_n + o(n). \qquad \square$$

Discussion. For $0 < \theta < \tau$ and for $\delta = \delta(\theta)$, Proposition 12 implies that the limit $\bar{\varepsilon}(\delta) = \lim_n \mathrm{E}\varepsilon_n(\delta)$ exists and that

$$\bar{\varepsilon}(\delta(\theta)) = \varepsilon(\theta).$$

So to prove Theorem 2 it should be enough to prove

$$\delta(\theta) \sim \alpha\theta, \quad \varepsilon(\theta) \sim \beta\theta^2 \text{ as } \theta \to 0 \tag{55}$$

for positive constants α, β. Now the definitions (49), (50) enable us to rewrite (using Table 3) $\delta(\theta)$ and $\varepsilon(\theta)$ in terms of the stationary distribution $(Z_0^L, X_0^L, \xi_0, X_1^R, Z_1^R)$ of the quintuple process, as

$$\delta(\theta) = \mathrm{P}\left(\{X_0^L > \min(X_1^R, \xi_0)\} \triangle \{Z_0^L - \theta J_0 > \min((Z_1^R - \theta J_1)^+, \xi_0)\}\right), \tag{56}$$

$$\begin{aligned}\varepsilon(\theta) = {} & \mathrm{P}(X_0^L > \min(X_1^R, \xi_0)) - \mathrm{P}(Z_0^L - \theta J_0 > \min((Z_1^R - \theta J_1)^+, \xi_0)) \\ & - \mathrm{E}\xi_0\left(\mathbb{1}(\xi_0 < \min(X_0^L, X_1^R)) - \mathbb{1}(\xi_0 < \min(Z_0^L - \theta J_0, Z_1^R - \theta J_1))\right).\end{aligned}$$

So if we had an explicit formula for the stationary distribution $(Z_0^L, X_0^L, \xi_0, X_1^R, Z_1^R)$, then we could derive an explicit formula for $\delta(\theta)$ and $\varepsilon(\theta)$ and seek to prove (55) by calculus. But we do not have such an explicit formula—note the independence property (30) of the triple process does not hold for the quintuple process—and we have not completely succeeded in that program. We could prove the $\delta(\theta) \sim \alpha\theta$ part of (55), though we use only the weaker upper bound, proved by a simpler argument in section 4.2. To handle $\varepsilon(\theta)$ we show how to rewrite $\delta(\theta)$ and $\varepsilon(\theta)$ in a different way (Proposition 18) that allows us to derive inequalities, which will establish the stated form of Theorem 2.

4.2. Existence of the limit function $\bar{\varepsilon}(\delta)$. There is a minor technical point we deal with first. We expect intuitively that the function $\delta(\theta)$ should be continuous monotone, but neither property is obvious. If there were small values of δ which were not of the form $\delta = \delta(\theta)$ for some θ, then we cannot use Proposition 12 to establish existence of a limit $\bar{\varepsilon}(\delta)$. Instead we outline an argument (reusing previous methods) to prove more abstractly (Lemma 13) that the limit $\bar{\varepsilon}(\delta)$ always exists. We could have started the proof of Theorem 2 this way, but we wanted to emphasize the Lagrange multiplier approach as more useful for calculation.

LEMMA 13. *$\bar{\varepsilon}(\delta) := \lim_n \mathrm{E}\varepsilon_n(\delta)$ exists for each $0 < \delta < 1$.*

Note that $\varepsilon_n(\delta)$ is a priori nondecreasing in δ, and hence $\bar{\varepsilon}(\cdot)$ is nondecreasing.

Outline proof. Fix $0 < \delta < 1$. Let $B_n^{(\delta)}$ attain the minimum in the definition (9) of $\varepsilon_n(\delta)$. Set $\bar{\varepsilon}_*(\delta) = \liminf_n \mathrm{E}\varepsilon_n(\delta)$. There exists a subsequence (of the subsequence of n attaining the liminf) in which the local weak convergence (Lemma 8) of A_n^{opt} to A^{opt} extends to joint convergence of $B_n^{(\delta)}$ to some limit random set $B^{(\delta)}$. The analogues of (49), (50) with $B^{(\delta)}$ in place of B^{opt} equal δ and $\bar{\varepsilon}_*(\delta)$. For arbitrary n, start with the restriction (B_n^*, say) of $B^{(\delta)}$ to $[1, n]$ and then show that by modifying B_n^* near the endpoints we can construct B_n^{**} satisfying $|B_n^{**} \triangle A_n^{\mathrm{opt}}| \geq \delta n$ and $\mathrm{E}[n^{-1}(f_n(A_n^{\mathrm{opt}}) - f_n(B_n^{**}))] \to \bar{\varepsilon}_*(\delta)$. □

The following lemma (to be proved in section 4.4) allows us to complete the proof of Theorem 2.

LEMMA 14. *There exist positive constants C_1, C_2 such that, for all $0 < \theta < \tau$,*

$$\delta(\theta) \leq C_1\theta, \tag{57}$$

$$\varepsilon(\theta) \geq C_2\theta^2. \tag{58}$$

We now finish the proof of Theorem 2. Recall that Proposition 12 showed $\bar{\varepsilon}(\delta(\theta)) = \varepsilon(\theta)$, and that (Lemma 13) $\bar{\varepsilon}(\cdot)$ is a nondecreasing function. Using (57)

$$\bar{\varepsilon}(C_1\theta) \geq \bar{\varepsilon}(\delta(\theta)) = \varepsilon(\theta) \geq C_2\theta^2,$$

and setting $\delta = C_1\theta$ gives $\bar{\varepsilon}(\delta) \geq C_2\delta^2/C_1^2$. This establishes the lower bound (11) and completes the proof of Theorem 2.

4.3. A cycle formula representation.

LEMMA 15. *If $\xi_{i-1} + \xi_i < 1 - \tau$, then $i \in A^{opt}$ and $i \in B^{opt}$.*

Proof. Suppose $\xi_{i-1} + \xi_i < 1 - \tau$. Lemma 9(a) showed $i \in A^{\mathrm{opt}}$. Recall that B_n^{opt} maximizes (40). If $i \notin A$, then the increase in the benefit at (40) obtained by including i is at least $1 - \xi_i - \xi_{i-1} - \theta$, so by our standing assumption (39) the increase is positive, and so $i \in B_n^{\mathrm{opt}}$. Letting $n \to \infty$ and using Lemma 11 gives the same conclusion for B^{opt}. □

We next need a lemma (analogous to Lemma 9(b)) giving conditions under which we can "localize" A^{opt} and B^{opt} by forcing them to coincide with the optimal sets A_n^{opt} and B_n^{opt} for the optimization problem on $[1, n]$ for suitable n, which we now write as $t - 1$.

LEMMA 16. *Let $t \geq 2$. Suppose $\xi_{i-1} + \xi_i < 1 - \tau$ for each of $i = 0, 1, t-1, t$. Then the following hold:*

(a) *A^{opt} and B^{opt} contain $\{0, 1, t-1, t\}$.*

(b) *The restrictions of A^{opt} and B^{opt} to $[1, t-1]$ coincide with A_{t-1}^{opt} and B_{t-1}^{opt}.*

(c) *For any $B \subseteq \{1, 2, \ldots, t-1\}$, either $B = A_{t-1}^{opt}$ or $f_{t-1}(B) < f_{t-1}(A_{t-1}^{opt})$.*

(d) *In particular, either $A_{t-1}^{opt} = B_{t-1}^{opt}$ or $f_{t-1}(A_{t-1}^{opt}) > f_{t-1}(B_{t-1}^{opt})$.*

Proof. (a) follows from Lemma 15. Observe that A_{t-1}^{opt} and B_{t-1}^{opt} contain 1 and $t - 1$, because $\xi_1 < 1 - \tau$ and $\xi_{t-2} < 1 - \tau$. If we consider the solutions $A_{[\ell,m]}^{\mathrm{opt}}$, $B_{[\ell,m]}^{\mathrm{opt}}$ for some interval $[\ell, m]$ strictly containing $[0, t]$, then they contain 1 and $t - 1$ by the argument for Lemma 15. Thus by optimality the restrictions of $A_{[\ell,m]}^{\mathrm{opt}}$ and $B_{[\ell,m]}^{\mathrm{opt}}$ to $[1, t-1]$ must coincide with A_{t-1}^{opt} and B_{t-1}^{opt}. So (b) follows from weak convergence, Lemma 11. And (c) follows from the uniqueness result, Lemma 3. □

We start by quoting a standard form (cf. [8, Exercise 6.3.4]) of Kac's identity for stationary processes.

LEMMA 17. *Let $(\Xi_i, -\infty < i < \infty)$ be a stationary ergodic sequence on some state space, let $\mathrm{P}(\Xi_1 \in \bar{D}) > 0$, and let $h(\Xi_1)$ be real-valued and integrable. For any $t_0 \geq 1$, define $T = t_0 \min\{i \geq 2 : \Xi_{it_0} \in \bar{D})$. Then*

$$\mathrm{E}h(\Xi_1) = \mathrm{E}\left[\mathbb{1}(\Xi_1 \in \bar{D}) \sum_{i=1}^{T-1} h(\Xi_i)\right].$$

We apply this to $\Xi_i = (Z_i^L, X_i^L, \xi_i, \xi_{i-1}, \xi_{i-2}, X_{i+1}^R, Z_{i+1}^R)$, $t_0 = 3$, and

$$D := \{\xi_{-1} + \xi_0 < 1 - \tau,\ \xi_0 + \xi_1 < 1 - \tau\} = \{\Xi_1 \in \bar{D}\} \tag{59}$$

for suitable $\bar{D}$, making the T in Lemma 17 be

$$T = 3\min\{t \geq 2 : \xi_{3t-2} + \xi_{3t-1} < 1 - \tau,\ \xi_{3t-1} + \xi_{3t} < 1 - \tau\}. \tag{60}$$

Now definition (49) says $\delta(\theta) = \mathrm{E}h(\Xi_0)$ for

$$h(\Xi_0) = \mathbb{1}(\{0 \in A^{\mathrm{opt}}\} \triangle \{0 \in B^{\mathrm{opt}}\}).$$

So $\sum_{i=1}^{T-1} h(\Xi_i)$ equals the cardinality of $A^{\mathrm{opt}} \triangle B^{\mathrm{opt}}$ restricted to $[1, T-1]$. On the event D, Lemma 16 identifies this restriction as $A^{\mathrm{opt}}_{T-1} \triangle B^{\mathrm{opt}}_{T-1}$, so Kac's identity gives (61) below. Similarly, definition (50) says $\varepsilon(\theta) = \mathrm{E}h(\Xi_0)$ for

$$h(\Xi_0) = \mathbb{1}(0 \in A^{\mathrm{opt}}) - \xi_0 \mathbb{1}(0 \in A^{\mathrm{opt}}, 1 \in A^{\mathrm{opt}}) - \mathbb{1}(0 \in B^{\mathrm{opt}}) + \xi_0 \mathbb{1}(0 \in B^{\mathrm{opt}}, 1 \in B^{\mathrm{opt}}),$$

and on the event D the sum $\sum_{i=1}^{T-1} h(\Xi_i)$ equals the difference $f_{T-1}(A^{\mathrm{opt}}_{T-1}) - f_{T-1}(B^{\mathrm{opt}}_{T-1})$ between the benefits. This establishes (62), and the final assertion (63) follows from Lemma 16(d). To summarize:

PROPOSITION 18. *Let D be the event* (59) *and let T be the random time* (60). *Then*

$$\delta(\theta) = \mathrm{E}[\mathbb{1}_D \times |A^{opt}_{T-1} \triangle B^{opt}_{T-1}|], \tag{61}$$

$$\varepsilon(\theta) = \mathrm{E}[\mathbb{1}_D \times (f_{T-1}(A^{opt}_{T-1}) - f_{T-1}(B^{opt}_{T-1}))], \tag{62}$$

$$\text{on } D, \text{ either } A^{opt}_{T-1} = B^{opt}_{T-1} \text{ or } f_{T-1}(A^{opt}_{T-1}) - f_{T-1}(B^{opt}_{T-1}) > 0. \tag{63}$$

4.4. An integration lemma. Let us rewrite the difference in (62) as

$$W(\theta) := f_{T-1}(A^{\mathrm{opt}}_{T-1}) - f_{T-1}(B^{\mathrm{opt}}_{T-1})$$

to emphasize its dependence on θ; and note D does not depend on θ. The key ingredient in the proof of the lower bound is the following lemma, to be proved in section 4.5.

LEMMA 19. *There exists $C_3 > 0$ such that for all $0 < \theta < \tau$, for all $k \geq 0$, and $x > 0$,*

$$\mathrm{P}(T \geq k,\ 0 < \mathbb{1}_D W(\theta) < x) \leq C_3 x (k+1) \mathrm{P}(T \geq k).$$

Taking $k = 0$ in this lemma we get

$$\mathrm{P}(0 < \mathbb{1}_D W(\theta) < x) \leq C_3 x. \tag{64}$$

Recall a simple integration lemma (for a more general result see [2, Lemma 6(a)]).

LEMMA 20. *Let $V \geq 0$ be a real-valued random variable such that*

$$\mathrm{P}(0 < V < x) \leq Cx, \quad 0 < x < \infty.$$

Then

$$\mathrm{E}V \geq \frac{[\mathrm{P}(V > 0)]^2}{2C}.$$

By (64) and Lemma 20, we get

$$\begin{aligned} \varepsilon(\theta) &= \mathrm{E}(\mathbb{1}_D W(\theta)) \text{ by (62)} \\ &\geq \frac{[\mathrm{P}(W(\theta)\mathbb{1}_D > 0)]^2}{2C_3}. \end{aligned} \tag{65}$$

To finish the proof of (58), we need the following lemma.

LEMMA 21. *There exists a positive constant C_4 such that, for all $0 < \theta < \tau$,*

$$\mathrm{P}(W(\theta)\mathbb{1}_D > 0) \geq C_4\theta. \tag{66}$$

Proof. By assumption (7) we may assume that the constant τ at (38) is such that

$$\inf_{1/2-2\tau<x<1/2\tau} g(x) > 0, \tag{67}$$

where g is the density function for ξ_i. Consider the following event:

$$\Omega(\theta) = \{\xi_{-1} \in (0, 1/2),\ \xi_0 \in (0, 1/2-\tau),\ \xi_1 \in (1/2-\tau, 1/2),$$
$$\xi_2 \in (1-\xi_1-\theta, 1-\xi_1),\ \xi_3 \in (0, 1/2-2\tau)\}.$$

Using (67) there exists $C_4 > 0$ such that

$$\mathrm{P}(\Omega(\theta)) \geq C_4\theta.$$

Assume this event $\Omega(\theta)$ happens. Then $\xi_{-1}+\xi_0 \leq 1-\tau$, $\xi_0+\xi_1 \leq 1-\tau$, $1-\theta < \xi_1+\xi_2 < 1$, and $\xi_2+\xi_3 \leq 1-\tau$. So D happens and, using Lemma 9(a), we have $\{1,2,3\} \in A^{\mathrm{opt}}$, and by Lemma 16(b) the same holds true for A^{opt}_{T-1}. Still assuming $\Omega(\theta)$ occurs, we see that for $B = A^{\mathrm{opt}}_{T-1}\backslash\{2\}$, we have $f_{T-1}(A^{\mathrm{opt}}_{T-1}) - f_{T-1}(B) = 1-\xi_1-\xi_2 \in (0,\theta)$ and therefore $f_{T-1}(B) + \theta|A^{\mathrm{opt}}_{T-1} \triangle B| > f_{T-1}(A^{\mathrm{opt}}_{T-1})$, implying $0 < W(\theta)$ by (63). In particular

$$\mathrm{P}(W(\theta)\mathbb{1}_D > 0) \geq \mathrm{P}(\Omega(\theta)) \geq C_4\theta,$$

and we have proved assertion (66). □

From (65) and (66), we directly get the second assertion (58) of Lemma 14. We now show how to obtain the first assertion of Lemma 14. Recall that, by definition, we have

$$f_{T-1}(B^{\mathrm{opt}}_{T-1}) + \theta|A^{\mathrm{opt}}_{T-1} \triangle B^{\mathrm{opt}}_{T-1}| \geq f_{T-1}(A^{\mathrm{opt}}_{T-1});$$

hence we get $\theta T > \theta|A^{\mathrm{opt}}_{T-1} \triangle B^{\mathrm{opt}}_{T-1}| \geq W(\theta)$. In particular, by Proposition 18, we have $D \cap \{W(\theta) > 0\} \subset D \cap \{\theta T > W(\theta) > 0\}$. Also, by Lemma 19, we have

$$\begin{aligned}
\delta(\theta) &\leq \mathrm{E}[T\mathbb{1}_D\mathbb{1}(W(\theta) > 0)] \text{ by (61)} \\
&\leq \sum_j j\mathrm{P}(T \geq j, \theta j > W(\theta) > 0) \\
&\leq C_3\theta \sum_j j^2(j+1)\mathrm{P}(T \geq j) \\
&\leq C_3\theta\mathrm{E}[(T+1)^4],
\end{aligned}$$

and $T/3$ has a geometric distribution so that assertion (57) of Lemma 14 follows.

4.5. Proof of Lemma 19. Write $W = W(\theta)$. Consider the random collection

$$\mathcal{B}(T-1) := \{B \subseteq \{1,2,\dots,T-1\} :\ B \neq A^{\mathrm{opt}}_{T-1}, 1 \in B, T-1 \in B\}.$$

By Proposition 18

$$\text{on } D, \text{ either } W = 0 \text{ or } W \geq \min_{B \in \mathcal{B}(T-1)} (f_{T-1}(A^{\mathrm{opt}}_{T-1}) - f_{T-1}(B)) > 0. \tag{68}$$

Our first goal is to derive a lower bound (Lemma 24) for the right-hand side of (68) in terms of the ξ_i's. Until the end of the proof of Lemma 24, we are working on the event D.

Let $C = \arg\min_{B \in \mathcal{B}(T-1)} (f_{T-1}(A^{\mathrm{opt}}_{T-1}) - f_{T-1}(B))$ be the optimal perturbation of A^{opt} on $[1, T-1]$. For any subinterval $\mathcal{I} = [\ell, m] \subseteq [1, T-1]$ write $\mathcal{I}_e = [\max(\ell - 1, 1), \min(m+1, T-1)]$. Decompose $A^{\mathrm{opt}} \triangle C$ as $\cup_i \mathcal{I}_i$, where the $\mathcal{I}_i$'s are disjoint maximal intervals of $A^{\mathrm{opt}} \triangle C$. Then

$$\begin{aligned} f_{T-1}(A^{\mathrm{opt}}) - f_{T-1}(C) &= \sum_i \left(f_{(\mathcal{I}_i)_e}(A^{\mathrm{opt}} \cap (\mathcal{I}_i)_e) - f_{(\mathcal{I}_i)_e}(C \cap (\mathcal{I}_i)_e) \right) \\ &= \sum_i \left(f_{T-1}(A^{\mathrm{opt}}_{T-1}) - f_{T-1}(C_i) \right), \end{aligned}$$

where $C_i = (A^{\mathrm{opt}}_{T-1} \cap \mathcal{I}_i{}^c) \cup (C \cap (\mathcal{I}_i)_e)$. This implies that $A^{\mathrm{opt}} \triangle C$ is a *single* subinterval $\mathcal{I}$ of $[1, T-1]$.

We now look at the possible perturbations of A^{opt} on the interval $[0, T]$. Recall that we are working on the event D, and that A^{opt} contains $0, 1, T-1, T$. Let $L_0, L_1, \dots, L_K$ be the maximal subintervals $[a, b] \subseteq A^{\mathrm{opt}} \cap [0, T]$ for which $b > a$, that is, with at least two elements. So we can partition $[0, T]$ as $L_0 \cup S_0 \cup L_1 \cup S_1 \cup \ldots \cup L_K$, where the S_k's are the complementary intervals. We call the L_k's *lakes* and the S_k's *switches*.

LEMMA 22. *Let $L = [a, b]$ be a lake. For any set $B \in \mathcal{B}(T-1)$ such that $B \cap L^c = A^{opt} \cap L^c$ and hence $B \cap L \neq A^{opt} \cap L$, we have*

$$f_{T-1}(A^{opt}) - f_{T-1}(B) \geq \min \left\{ 1 - \xi_a, 1 - \xi_{b-1}, \min_{a \leq i \leq b-1} 1 - \xi_{i-1} - \xi_i \right\} > 0. \tag{69}$$

Proof. First suppose B is obtained by removing from A^{opt} a single item. If this item is a, we have $f_{T-1}(A^{\mathrm{opt}}) - f_{T-1}(B) = 1 - \xi_a$; if it is b, we have $f_{T-1}(A^{\mathrm{opt}}) - f_{T-1}(B) = 1 - \xi_{b-1}$, and if it is $i \in (a, b)$, then we have $f_{T-1}(A^{\mathrm{opt}}) - f_{T-1}(B) = 1 - \xi_{i-1} - \xi_i$. So by optimality of A^{opt}_{T-1} the first inequality in (69) holds for these B, and Lemma 16 implies the last inequality in (69). Now recall that Lemma 9(c) shows $\min_{a \leq i \leq b-1} 1 - \xi_i \geq 0$. So construct a general B by removing items from A^{opt} one by one, and for items after the first the benefit can only decrease. So the first inequality holds generally. □

LEMMA 23. *Let $S = [a, b]$ be a switch and $S_e = [a-1, b+1]$. For any set B such that $B \cap S_e^c = A^{opt} \cap S_e^c$ and $B \cap S \neq A^{opt} \cap S$, we have*

$$\begin{aligned} f_{S_e}(A^{opt}) - f_{S_e}(B) \geq \min \Bigg\{ & \min_{a-1 \leq i < j \leq b} \xi_i + \xi_j - 1, \min_{a-1 \leq i \leq b} \xi_i, \min_{a \leq i \leq b} \xi_i - \xi_{a-2}, \\ & \min_{a \leq i \leq b} \xi_i - \xi_{b+1}, 1 - \xi_{a-2} - \xi_{b+1} \Bigg\}. \end{aligned}$$

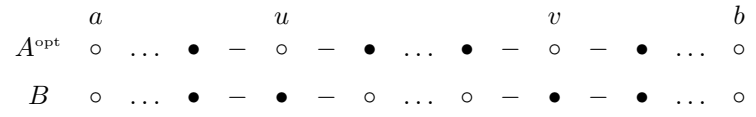


FIG. 3. *Case* $[\circ \dots \circ]$, *where* $a \le u \le v \le b$. *Benefit change* $= \xi_{u-1} + \xi_v - 1$.

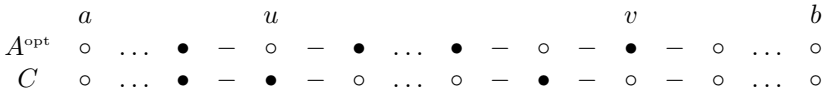


FIG. 4. *Case* $[\circ \dots \bullet]$, *where* $a \le u \le v \le b - 1$. *Benefit change* $= \xi_{u-1}$.

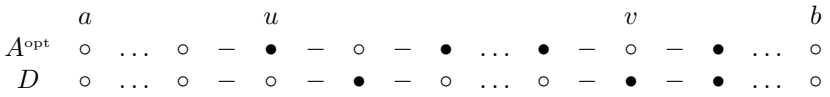


FIG. 5. *Case* $[\bullet \dots \circ]$, *where* $a + 1 \le u \le v \le b$. *Benefit change* $= \xi_v$.

TABLE 4

$[u, v]$	$[a-1, \circ]$	$[a-1, \bullet]$	$[\circ, b+1]$	$[\bullet, b+1]$	$[a-1, b+1]$
benefit change	$\xi_v - \xi_{a-2}$	$1 - \xi_{a-2}$	$\xi_u - \xi_{b+1}$	$1 - \xi_{b+1}$	$1 - \xi_{a-2} - \xi_{b+1}$

Proof. By construction a switch starts and ends with items not in A^{opt}, and the two items before and after the switch are in A^{opt}. Moreover, Table 2 shows that two adjacent items cannot both be not in A^{opt}, so the items in a switch $[a, b]$ must strictly alternate between in and not in A^{opt}, as illustrated in Figure 3.

We first consider a set B obtained from A^{opt} by flipping all items in some subinterval $[u, v]$ of $[a, b]$. There are four cases, corresponding to whether the endpoints u, v are in or not in A^{opt}. We exhibit three cases in Figures 3, 4, and 5, labeled as, e.g., $[\bullet \dots \circ]$, together with the value of the benefit change $f_{S_e}(A^{\text{opt}}) - f_{S_e}(B)$. In the fourth case $[\bullet \dots \bullet]$, the benefit change equals 1.

We also need to consider cases where the flipped subinterval $[u, v]$ has $u = a - 1$ or $v = b + 1$ or both. There are five cases, indicated in Table 4.

Now consider any subset B satisfying the hypothesis of Lemma 23. Decompose $A^{\text{opt}} \triangle B$ into disjoint maximal intervals $\mathcal{I}_i$. It is easy to check that the benefit change between A^{opt} and B is just the sum of the separate benefit changes between A^{opt} and A^{opt} with interval $\mathcal{I}_i$ flipped. Thus the minimum over B is attained by one of the cases we have considered, establishing the lemma. ∎

LEMMA 24. *Set* $w = \min_{1 \le i < j \le T-1} \{|\xi_i + \xi_j - 1|; \xi_i; |1 - \xi_i|; |\xi_i - \xi_j|\}$. *On the event* D, *either* $W = 0$ *or* $W \ge w$.

Proof. We need only consider the case $W > 0$. Recall that $C = \arg\min_{B \in \mathcal{B}(T-1)} (f_{T-1}(A^{\text{opt}}_{T-1}) - f_{T-1}(B))$ is such that $A^{\text{opt}} \triangle C$ is a *single* subinterval $\mathcal{I}$ of $[1, T-1]$. It is enough to show that C satisfies the assumptions of Lemmas 22 (for some lake) or the assumptions of Lemma 23 (for some switch), for then the lower bound w follows from the lower bounds in those lemmas.

We argue by contradiction: if false, then $\mathcal{I}$ intersects some lake and some adjacent switch, say L_k and S_k (the case of L_k and S_{k-1} is similar). So there exists $a < b < c$ such that $b = \sup L_k$ and $\mathcal{I} = [a, c]$. Now check the following:

If $c \in A^{\mathrm{opt}}$, then $f(B) - f(C) = 1$, for $B := C \cup \{b, b+2, b+4, \dots, c\} \backslash \{b+1, b+3, \dots, c-1\}$.

If $c \notin A^{\mathrm{opt}}$, then $f(B) - f(C) = \xi_c$, for $B := C \cup \{b, b+2, b+4, \dots, c-1\} \backslash \{b+1, b+3, \dots, c\}$.

Either case contradicts the optimality of C. □

We may now complete the proof of Lemma 19. The key point is that the bound w in Lemma 24 does not depend on θ. From Lemma 24,

$$\begin{aligned}
&\mathrm{P}(T \geq 3k,\ 0 < W(\theta)\mathbb{1}_D < x) \\
&\leq \mathrm{P}(T \geq 3k,\ D; 0 < w < x) \leq \mathrm{P}(T \geq 3k,\ w < x) \\
&\leq \mathrm{P}\left(T \geq 3k,\ \min_{1 \leq i < j \leq T-1} |\xi_i + \xi_j - 1| < x\right) + \mathrm{P}\left(T \geq 3k,\ \min_{1 \leq i \leq T-1} \xi_i < x\right) \\
&\quad + \mathrm{P}\left(T \geq 3k,\ \min_{1 \leq i \leq T-1} |\xi_i - 1| < x\right) + \mathrm{P}\left(T \geq 3k,\ \min_{1 \leq i < j \leq T-1} |\xi_i - \xi_j| < x\right).
\end{aligned}$$

The four terms on the right-hand side are treated similarly: we will just study the final term and will prove that there exists $C > 0$ independent of k such that

$$\mathrm{P}\left(\min_{1 \leq i < j \leq T-1} |\xi_i - \xi_j| < x \,\middle|\, T \geq 3k\right) \leq C(k+1)x. \tag{70}$$

The effect of conditioning on the event $\{T \geq 3k\}$ is that each nonoverlapping triple $(\xi_{3m}, \xi_{3m+1}, \xi_{3m+2})$ is conditioned to satisfy either $\{\xi_{3m} + \xi_{3m+1} \geq 1 - \tau\} \cup \{\xi_{3m+1} + \xi_{3m+2} \geq 1 - \tau\}$ or $\{\xi_{3m} + \xi_{3m+1} < 1 - \tau,\ \xi_{3m+1} + \xi_{3m+2} < 1 - \tau\}$ (for $m = T$). It follows that, for any $i < j$,

$$\mathrm{P}((\xi_i, \xi_j) \in \cdot | T > j) \leq a^{-2} \mathrm{P}((\xi_i, \xi_j) \in \cdot), \tag{71}$$

where

$$\begin{aligned}
a = \min \big(&\mathrm{P}(\{\xi_0 + \xi_1 \geq 1 - \tau\} \cup \{\xi_1 + \xi_2 \geq 1 - \tau\}),\ \\
&\mathrm{P}(\xi_0 + \xi_1 < 1 - \tau, \xi_1 + \xi_2 < 1 - \tau)\big).
\end{aligned}$$

From assumption (7) the density of $\xi_j - \xi_i$ is bounded by some constant b, and so

$$\begin{aligned}
&\mathrm{P}\left(\min_{1 \leq i < j \leq T-1} |\xi_i - \xi_j| < x \,\middle|\, T \geq 3k\right) \\
&\leq \sum_{i<j} \mathrm{P}(|\xi_i - \xi_j| < x, T \geq j | T \geq 3k) \\
&= \sum_{i<j} \mathrm{P}(|\xi_i - \xi_j| < x | T \geq \max(j+1, 3k)) \mathrm{P}(T \geq j+1 | T \geq 3k) \\
&\leq b a^{-2} x \sum_{j \geq 3k-1} (j-1) \mathrm{P}(T \geq j+1 | T \geq 3k) \\
&\leq b a^{-2} x \sum_{j \geq k} 3(j+1) \mathrm{P}(T \geq 3j | T \geq 3k)
\end{aligned}$$

$$= ba^{-2}x \sum_{j \geq k} 3(j+1)\mathrm{P}(T \geq 3(j-k))$$

$$\leq ba^{-2}x(k\mathrm{E}[T] + \mathrm{E}[T(T+1)]),$$

where we used the fact that $T/3$ has a geometric distribution. This concludes the proof of Lemma 19.

5. Final remarks.

5.1. Technical assumptions on G. We stated a single assumption (7) on G. What we actually used was three consequences of this assumption:

- $\mathrm{P}(\xi < 1/2) > 0$, which implies $\mathrm{P}(\xi_i + \xi_{i+1} < 1) > 0$, was used in Lemma 15 and thereby throughout section 4 (because it implies $i \in A^{\mathrm{opt}}$) to implement "localization" arguments.
- $\mathrm{P}(\xi \leq 1/2) < 1$ was used in section 3.2 to show $P(\Omega_g) > 0$. Note that if $\mathrm{P}(\xi \leq 1/2) = 1$, then the optimization problem is degenerate in that the optimal $A_n^{\mathrm{opt}} = \{1, 2, \ldots, n\}$.
- $\xi_1 + \xi_2$ has density bounded below in some interval $(1, 1+\eta)$, which was used in section 3.2 to obtain (37).

The latter two are used only in a convenient way to exhibit one near-optimal set. The "localization" arguments essentially just require one to find some event of positive probability involving $(\xi_{-k}, \ldots, \xi_k)$ which forces items 0 and 1 to be in (or not in) A^{opt}. Lemma 15 is just a simple way to exhibit such an event. So we expect Theorem 2 to remain true under much weaker assumptions on G.

5.2. Parallels with the cavity method. The arguments in this paper in the context of i.i.d.-DP (dynamic programming) may be compared with the more sophisticated arguments from the statistical physics *cavity method* [14], as reformulated in more probabilistic language in [1, 4], whose prototype example we take to be the analysis of the traveling salesman problem (TSP) in the "mean-field" model of geometry where there are n points and each of the $\binom{n}{2}$ interpoint links has random length. Of course *algorithmically* DP and TSP are quite different, but there are striking parallels between the analysis of optimal solutions of i.i.d.-DP and mean-field-TSP, as follows:

- There are $n \to \infty$ limits for the random data; in DP this is just the obvious infinite i.i.d. sequence, while for mean-field-TSP it is a certain random infinite tree.
- The "inclusion criterion" for i.i.d.-DP involves X_i^L, X_{i+1}^R and the edge-cost ξ_i. Finite-n TSP has of course no simple inclusion criteria, but in the $n \to \infty$ limit of mean-field-TSP there is an analogous criterion for inclusion of an edge (i, j) in terms of quantities Z_i^L, Z_j^R and the edge-length ξ_{ij}. Each Z is interpreted (cf. (19) for DP) as the difference between costs of two optimal solutions (subject to different local constraints) on one side of the tree.
- The distribution we use for X in i.i.d.-DP, the stationary distribution of a Markov chain, is the solution of an equation with abstract structure $X \stackrel{d}{=} h(\xi, X^1)$. The distribution we use for Z in mean-field-TSP, by a recursion on the limit tree, is the solution of an equation with abstract structure $Z \stackrel{d}{=} h(\xi; Z^1, Z^2, Z^3, \ldots)$, where the Z^j's are i.i.d. copies of the unknown distribution Z.

These parallels provide a glimpse of how the analogue of Theorem 1, a formula for the asymptotic expected cost in mean-field-TSP, may be derived (the original nonrigorous

argument was in [13]; a rigorous proof was given only recently via more combinatorial methods [18]). The analogue of Theorem 2 for mean-field-TSP, using Lagrange multipliers as in this paper, and leading to a nonrigorous argument that the scaling exponent equals 3, was given in [3].

Acknowledgment. We thank Vlada Limic for discussions regarding the NK model and Guilhem Semerjian for explaining to us an alternative proof of the $\delta(\theta) \sim \alpha\theta$ part of (55).

REFERENCES

[1] D. J. Aldous, *The $\zeta(2)$ limit in the random assignment problem*, Random Structures Algorithms, 18 (2001), pp. 381–418.
[2] D. J. Aldous, C. Bordenave, and M. Lelarge, *Near-minimal spanning trees: A scaling exponent in probability models*, Ann. Inst. H. Poincaré Probab. Stat., 44 (2008), pp. 962–976.
[3] D. J. Aldous and A. G. Percus, *Scaling and universality in continuous length combinatorial optimization*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 11211–11215.
[4] D. J. Aldous and J. M. Steele, *The objective method: Probabilistic combinatorial optimization and local weak convergence*, in Probability on Discrete Structures, Encyclopaedia Math. Sci. 110, Springer-Verlag, Berlin, 2004, pp. 1–72.
[5] R. Beier and B. Vöcking, *Typical properties of winners and losers in discrete optimization*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM, New York, 2004, pp. 343–352.
[6] P. Billingsley, *Convergence of Probability Measures*, John Wiley & Sons, New York, 1968.
[7] A. Bogdanov and L. Trevisan, *Average-case complexity*, Found. Trends Theor. Comput. Sci., 2 (2006), pp. 1–106.
[8] R. Durrett, *Probability: Theory and Examples*, 3rd ed., Brooks/Cole, Pacific Grove, CA, 2005.
[9] R. Durrett and V. Limic, *Rigorous results for the NK model*, Ann. Probab., 31 (2003), pp. 1713–1753.
[10] S. A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press, New York, 1993.
[11] V. Limic and R. Pemantle, *More rigorous results on the Kauffman-Levin model of evolution*, Ann. Probab., 32 (2004), pp. 2149–2178.
[12] C. McDiarmid, *On the method of bounded differences*, in Surveys in Combinatorics, 1989 (Norwich, 1989), London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK, 1989, pp. 148–188.
[13] M. Mézard and G. Parisi, *A replica analysis of the travelling salesman problem*, J. Physique, 47 (1986), pp. 1285–1296.
[14] M. Mézard and G. Parisi, *The cavity method at zero temperature*, J. Statist. Phys., 111 (2003), pp. 1–34.
[15] M. Mézard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond*, World Sci. Lecture Notes Phys. 9, World Scientific, Teaneck, NJ, 1987.
[16] M. Talagrand, *A new look at independence*, Ann. Probab., 24 (1996), pp. 1–34.
[17] M. Talagrand, *Spin Glasses: A Challenge for Mathematicians. Cavity and Mean Field Models*, Ergeb. Math. Grenzgeb. (3), Springer-Verlag, Berlin, 2003.
[18] J. Wästlund, *The Travelling Salesman Problem in the Stochastic Mean Field Model*, http://www.math.chalmers.se/∼wastlund/.
[19] E. D. Weinberger, *Local properties of Kauffman's N-k model: A tunably rugged energy landscape*, Phys. Rev. A, 44 (1991), pp. 6399–6413.